

Studies in Computational Intelligence 47

S. Sumathi · S. Esakkirajan

Fundamentals of Relational Database Management Systems

 Springer

Studies in Computational Intelligence, Volume 47

Editor-in-chief

Prof. Janusz Kacprzyk

Systems Research Institute

Polish Academy of Sciences

ul. Newelska 6

01-447 Warsaw

Poland

E-mail: kacprzyk@ibspan.waw.pl

Further volumes of this series
can be found on our homepage:
springer.com

Vol. 29. Sai Sumathi, S.N. Sivanandam
*Introduction to Data Mining and its
Application*, 2006
ISBN 978-3-540-34350-9

Vol. 30. Yukio Ohsawa, Shusaku Tsumoto (Eds.)
Chance Discoveries in Real World Decision Making,
2006
ISBN 978-3-540-34352-3

Vol. 31. Ajith Abraham, Crina Grosan, Vitorino
Ramos (Eds.)
Stigmergic Optimization, 2006
ISBN 978-3-540-34689-0

Vol. 32. Akira Hirose
Complex-Valued Neural Networks, 2006
ISBN 978-3-540-33456-9

Vol. 33. Martin Pelikan, Kumara Sastry, Erick
Cantú-Paz (Eds.)
*Scalable Optimization via Probabilistic
Modeling*, 2006
ISBN 978-3-540-34953-2

Vol. 34. Ajith Abraham, Crina Grosan, Vitorino
Ramos (Eds.)
Swarm Intelligence in Data Mining, 2006
ISBN 978-3-540-34955-6

Vol. 35. Ke Chen, Lipo Wang (Eds.)
Trends in Neural Computation, 2007
ISBN 978-3-540-36121-3

Vol. 36. Ildar Batyrshin, Janusz Kacprzyk, Leonid
Sheremetov, Lotfi A. Zadeh (Eds.)
*Preception-based Data Mining and Decision Making
in Economics and Finance*, 2006
ISBN 978-3-540-36244-9

Vol. 37. Jie Lu, Da Ruan, Guangquan Zhang (Eds.)
E-Service Intelligence, 2007
ISBN 978-3-540-37015-4

Vol. 38. Art Lew, Holger Mauch
Dynamic Programming, 2007
ISBN 978-3-540-37013-0

Vol. 39. Gregory Levitin (Ed.)
*Computational Intelligence in Reliability
Engineering*, 2007
ISBN 978-3-540-37367-4

Vol. 40. Gregory Levitin (Ed.)
*Computational Intelligence in Reliability
Engineering*, 2007
ISBN 978-3-540-37371-1

Vol. 41. Mukesh Khare, S.M. Shiva Nagendra (Eds.)
*Artificial Neural Networks in Vehicular Pollution
Modelling*, 2007
ISBN 978-3-540-37417-6

Vol. 42. Bernd J. Krämer, Wolfgang A. Halang (Eds.)
Contributions to Ubiquitous Computing, 2007
ISBN 978-3-540-44909-6

Vol. 43. Fabrice Guillet, Howard J. Hamilton (Eds.)
Quality Measures in Data Mining, 2007
ISBN 978-3-540-44911-9

Vol. 44. Nadia Nedjah, Luiza de Macedo
Mourelle, Mario Neto Borges, Nival Nunes
de Almeida (Eds.)
Intelligent Educational Machines, 2007
ISBN 978-3-540-44920-1

Vol. 45. Vladimir G. Ivancevic, Tijana T. Ivancevic
*Neuro-Fuzzy Associative Machinery for
Comprehensive Brain and Cognition Modelling*,
2007
ISBN 978-3-540-47463-0


Vol. 46. Valentina Zharkova, Lakhmi C. Jain (Eds.)
*Artificial Intelligence in Recognition and
Classification of Astrophysical and Medical
Images*, 2007
ISBN 978-3-540-47511-8

Vol. 47. S. Sumathi, S. Esakkirajan
*Fundamentals of Relational Database Management
Systems*, 2007
ISBN 978-3-540-48397-7

S. Sumathi
S. Esakkirajan

Fundamentals of Relational Database Management Systems

With 312 Figures and 30 Tables

 Springer

Dr. S. Sumathi

Assistant Professor

Department of Electrical and Electronics Engineering

PSG College of Technology

P.O. Box 1611

Peelamedu

Coimbatore 641 004

Tamil Nadu, India

E-mail: ss_eeein@yahoo.com

S. Esakkirajan

Lecturer

Department of Electrical and Electronics Engineering

PSG College of Technology

P.O. Box 1611

Peelamedu

Coimbatore 641 004

Tamil Nadu, India

E-mail: ser@mail.psgtech.ac.in

Library of Congress Control Number: 2006935984

ISSN print edition: 1860-949X

ISSN electronic edition: 1860-9503

ISBN-10 3-540-48397-7 Springer Berlin Heidelberg New York

ISBN-13 978-3-540-48397-7 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2007

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Cover design: deblik, Berlin

Typesetting by SPi using a Springer L^AT_EX macro package

Printed on acid-free paper SPIN: 11820970 89/SPi 5 4 3 2 1 0

Preface

Information is a valuable resource to an organization. Computer software provides an efficient means of processing information, and database systems are becoming increasingly common means by which it is possible to store and retrieve information in an effective manner. This book provides comprehensive coverage of fundamentals of database management system. This book is for those who wish a better understanding of relational data modeling, its purpose, its nature, and the standards used in creating relational data model.

Relational databases are the most popular database management systems in the world and are supported by a variety of vendor implementations. Majority of the practical tasks in industry require applying relatively not complex algorithms to huge amounts of well-structured data. The efficiency of the application depends on the quality of data organization. Advances in database technology and processing offer opportunities for using information flexibility and efficiently when data is organized and stored in relational structures. The relational DBMS is a success in the commercial market place with respect to business data processing and related applications. This success is a result of cost effective application development combined with high data consistency. The success has led to the use of relational DBMS technology in other application environments requesting its traditional virtues, while at the same time adding new requirements.

SQL is the standard computer language used to communicate with relational database management systems. Chapter 4 gives an introduction to SQL with illustrative examples. The limitations of SQL and how to overcome that limitations using PL/SQL are discussed in Chap. 5.

The current trends in hardware like RAID technology made relational DBMSs to support high transmission rates, very high availability, and a soft real-time transaction a cost effective possibility. The basics of RAID technology, different levels of RAID are discussed in this book.

Object-oriented databases are also becoming important. As object-oriented programming continues to increase in popularity, the demand for

such databases will grow. Due to this reason a separate chapter is being devoted to object-oriented DBMS and object-relational DBMS.

This text discusses a number of new technologies and challenges in database management systems like Genome Database Management System, Mobile Database Management System, Multimedia Database Management System, Spatial Database Management Systems, and XML.

Finally, there is no substitute for experience. To ensure that every student can have experience for creating data models and database design, list of projects along with codes in VB and Oracle are given. The goal in providing the list of projects is to ensure that students should have at least one commercial product at their disposal.

About the Book

The book is meant for wide range of readers from College, University Students who wish to learn basics as well as advanced concepts in Database Management System. It can also be meant for the programmers who may be involved in the programming based on the Oracle and Visual Basic applications.

Database Management System, at present is a well-developed field, among academicians as well as between program developers. The principles of Database Management System are dealt in depth with the information and the useful knowledge available for computing processes. The various approaches to data models and the relative advantages of relational model are given in detail.

Relational databases are the most popular database management systems in the world and are supported by a variety of vendor implementations. The solutions to the problems are programmed using Oracle and the results are given. The overview of Oracle and Visual Basic is provided for easy reference to the students and professionals. This book also provides introduction to commercial DBMS, pioneers in DBMS, and dictionary of DBMS terms in appendix.

The various worked out examples and the solutions to the problems are well balanced pertinent to the RDBMS Projects, Labs, and for College and University Level Studies.

This book provides data models, database design, and application-oriented structures to help the reader to move in to the database management world. The book also presents application case studies on a wide range of connected fields to facilitate the reader for better understanding. This book can be used from Under Graduation to Post-Graduate Level. Some of the projects done are also added in the book. The book contains solved example problems, review questions, and solutions.

This book can be used as a ready reference guide for computer professionals who are working in DBMS field. Most of the concepts, solved problems and

applications for wide variety of areas covered in this book, which can fulfill as an advanced academic book.

We hope that the reader will find this book a truly helpful guide and a valuable source of information about the database management principles for their numerous practical applications.

Salient Features

The salient features of this book includes:

- Detailed description on relational database management system concepts
- Variety of solved examples
- Review questions with solutions
- Worked out results to understand the concepts of relational database management Systems using Oracle Version 8.0.
- Application case studies and projects on database management system in various fields like Transport Management, Hospital Management, and Academic Institution Management, Hospital Management, Railway Management and Election Voting System.

Organization of the Book

The book covers 14 chapters altogether. The fundamentals of relational database management systems are discussed with basic principles, advanced concepts, and recent challenges. The application case studies are also discussed.

The chapters are organized as follows:

- Chapter 1 gives an overview of database management system, Evolution of Database Management System, ANSI/SPARK data model, Two-tier, Three-tier and Multi-tier database architecture.
- The preliminaries of the Entity Relation (ER) data model are described in Chap. 2. Different types of entities, attributes and relations are discussed with examples. Mapping from ER model to relational model, Enhanced ER model, which includes generalization, specialization, are given with relevant examples.
- Chapter 3 deals with relational data model. In this chapter E.F. Codd rule, basic definition of relation, cardinality of the relation, arity of the relation, constraints in relation are given with suitable examples. Relational algebra, tuple relational calculus, domain relational calculus and different operations involved are explained with lucid examples. This chapter also discusses the features of QBE with examples.
- Chapter 4 exclusively deals with Structured Query Language. The data definition language, data manipulation language and the data control language were explained with suitable examples. Views, imposition of constraints in a relation are discussed with examples.

- Chapter 5 deals with PL/SQL. The shortcomings of SQL and how they are overcome in PL/SQL, the structure of PL/SQL are given in detail. The iterative control like FOR loop, WHILE loop are explained with examples. The concept of CURSOR and the types of CURSORS are explained with suitable examples. The concept of PROCEDURE, FUNCTION, and PACKAGE are explained in detail. The concept of EXCEPTION HANDLING and the different types of EXCEPTION HANDLING are given with suitable examples. This chapter also gives an introduction to database triggers and the different types of triggers.
- Chapter 6 deals with various phases in database design. The concept of database design tools and the different types of database design tools are given in this chapter. Functional dependency, normalization are also discussed in this chapter. Different types of functional dependency, normal forms, conversion from one normal form to the other are explained with examples. The idea of denormalization is also introduced in this chapter.
- Chapter 7 gives details on transaction processing. Detailed description about deadlock condition and two phase locking are given through examples. This chapter also discusses the concept of query optimization, architecture of query optimizer and query optimization through Genetic Algorithm.
- Chapter 8 deals with database security and recovery. The need for database security, different types of database security is explained in detail. The different types of database failures and the method to recover the database is given in this chapter. ARIES recovery algorithm is explained in a simple manner in this chapter.
- Chapter 9 discusses the physical database design. The different types of File organization like Heap file, sequential file, and indexed file are explained in this chapter. The concept of B tree and B⁺ tree are explained with suitable example. The different types of data storage devices are discussed in this chapter. Advanced data storage concept like RAID, different levels of RAID, hardware and software RAID are explained in detail.
- Advanced concepts like data mining, data warehousing, and spatial database management system are discussed in Chap. 10. The data mining concept and different types of data mining systems are given in this chapter. The performance issues, data integration, data mining rules are explained in this chapter.
- Chapter 11 throws light on the concept of object-oriented and object Relational DBMS. The benefits of object-oriented programming, object-oriented programming languages, characteristics of object-oriented database, application of OODBMS are discussed in detail. This chapter also discusses the features of ORDBMS, comparison of ORDBMS with OODBMS.
- Chapter 12 deals with distributed and parallel database management system. The features of distributed database, distributed DBMS architecture, distributed database design, distributed concurrency control are discussed

in depth. This chapter also discusses the basics of parallel database management, parallel database architecture, parallel query optimization.

- Recent challenges in DBMS are given in Chap. 13 which includes genome database management, mobile database management, spatial database management system and XML. In genome database management, the concept of genome, genetic code, genome directory system project is discussed. In mobile database, mobile database center, mobile database architecture, mobile transaction processing, distributed database for mobile are discussed in detail. In spatial database, spatial data types, spatial database modeling, querying spatial data, spatial DBMS implementation are analyzed. In XML, the origin of XML, XML family, XSL, XML, and database applications are discussed.
- Few projects related to bus transport management system, hospital management, course administration system, Election voting system, library management system and railway management system are implemented using Oracle as front end and Visual Basic as back end are discussed in Chap. 14. This chapter also gives an idea of how to do successful projects in DBMS.
- Four appendices given in this book includes dictionary of DBMS terms, overview of commands in SQL, pioneers in DBMS, commercial DBMS. Dictionary of DBMS terms gives the definition of commonly used terms in DBMS. Overview of commands in SQL gives the commonly used commands and their function. Pioneers in DBMS introduce great people like E.F. Codd, Peter Chen who have contributed for the development of database management system. Commercial DBMS introduces some of the popular commercial DBMS like System R, DB2 and Informix.
- The bibliography is given at the end after the appendix chapter.

About the Authors

S. Sumathi, B.E. in Electronics and Communication Engineering and Masters degree in Applied Electronics, Government College of Technology, Coimbatore, TamilNadu and Ph.D. in the area of Data Mining, is currently working as Assistant Professor in the Department of Electrical and Electronics Engineering, PSG College of Technology, Coimbatore with teaching and research experience of 16 years. She received the prestigious Gold Medal from the Institution of Engineers Journal Computer Engineering Division, for the research paper titled, “Development of New Soft Computing Models for Data Mining” and also Best project award for UG Technical Report, “Self-Organized Neural Network Schemes: As a Data mining tool”. She received Dr. R. Sundramoorthy award for Outstanding Academic of PSG College of Technology in the year 2006. She has guided a project which received Best M.Tech Thesis award from Indian Society for Technical Education, New Delhi. In appreciation of publishing various technical articles she has received

National and International Journal Publication Awards. She has also prepared manuals for Electronics and Instrumentation Laboratory and Electrical and Electronics Laboratory of EEE Department, PSG College of Technology, Coimbatore, has organized second National Conference on Intelligent and Efficient Electrical Systems and has conducted short-term courses on “Neuro Fuzzy System Principles and Data Mining Applications.” She has published several research articles in National and International Journals/Conferences and guided many UG and PG projects. She has also reviewed papers in National/International Journals and Conferences. She has published three books on “Introduction to Neural Networks with Matlab,” “Introduction to Fuzzy Systems with Matlab,” and “Introduction to Data Mining and its Applications.” The research interests include neural networks, fuzzy systems and genetic algorithms, pattern recognition and classification, data warehousing and data mining, operating systems and parallel computing, etc.

S. Esakkirajan has a B.Tech Degree from Cochin University of Science and Technology, Cochin and M.E. Degree from PSG College of Technology, Coimbatore, with a Rank in M.E. He has received Alumni Award in his M.E. He has presented papers in International and National Conferences. His research areas include database management system, neural network, genetic algorithm, and digital image processing.

Acknowledgment

The authors are always thankful to the Almighty for perseverance and achievements.

Sumathi and Esakkirajan wish to thank Mr. Rangaswamy, Managing Trustee, PSG Institutions, Mr. C.R. Swaminathan, Chief Executive, and Dr. R. Rudramoorthy, Principal, PSG College of Technology, Coimbatore, for their whole-hearted cooperation and great encouragement given in this successful endeavor. The authors appreciate and acknowledge Mr. Karthikeyan, Mr. Ponson, Mr. Manoj Kumar, Mr. Afsar Ahmed, Mr. Harikumar, Mr. Abdus Samad, Mr. Antony and Mr. Balumahendran who have been with them in their endeavors with their excellent, unforgettable help, and assistance in the successful execution of the work.

Dr. Sumathi owe much to her daughter Priyanka, who has helped her and to the support rendered by her husband, brother, and family. Mr. Esakkirajan like to thank his wife Akila, who shouldered a lot of extra responsibilities and did this with the long-term vision, depth of character, and positive outlook that are truly befitting of her name. He like to thank his father Sankaralingam for providing moral support and constant encouragement.

DEDICATED TO ALMIGHTY

Contents

1	Overview of Database Management System	1
1.1	Introduction	1
1.2	Data and Information	2
1.3	Database	2
1.4	Database Management System	3
1.4.1	Structure of DBMS	3
1.5	Objectives of DBMS	4
1.5.1	Data Availability	4
1.5.2	Data Integrity	4
1.5.3	Data Security	4
1.5.4	Data Independence	5
1.6	Evolution of Database Management Systems	5
1.7	Classification of Database Management System.....	6
1.8	File-Based System	7
1.9	Drawbacks of File-Based System	8
1.9.1	Duplication of Data	8
1.9.2	Data Dependence	8
1.9.3	Incompatible File Formats	8
1.9.4	Separation and Isolation of Data	9
1.10	DBMS Approach	9
1.11	Advantages of DBMS	10
1.11.1	Centralized Data Management	10
1.11.2	Data Independence	10
1.11.3	Data Inconsistency	10
1.12	Ansi/Spark Data Model	11
1.12.1	Need for Abstraction	11
1.12.2	Data Independence	12
1.13	Data Models	13
1.13.1	Early Data Models	14
1.14	Components and Interfaces of Database Management System.....	14

1.14.1	Hardware	14
1.14.2	Software	15
1.14.3	Data	16
1.14.4	Procedure	16
1.14.5	People Interacting with Database	16
1.14.6	Data Dictionary	20
1.14.7	Functional Components of Database System Structure	21
1.15	Database Architecture	22
1.15.1	Two-Tier Architecture	22
1.15.2	Three-tier Architecture	24
1.15.3	Multitier Architecture	24
1.16	Situations where DBMS is not Necessary	26
1.17	DBMS Vendors and their Products	26
2	Entity–Relationship Model	31
2.1	Introduction	31
2.2	The Building Blocks of an Entity–Relationship Diagram	32
2.2.1	Entity	32
2.2.2	Entity Type	32
2.2.3	Relationship	32
2.2.4	Attributes	32
2.2.5	ER Diagram	33
2.3	Classification of Entity Sets	34
2.3.1	Strong Entity	34
2.3.2	Weak Entity	34
2.4	Attribute Classification	35
2.4.1	Symbols Used in ER Diagram	35
2.5	Relationship Degree	39
2.5.1	Unary Relationship	39
2.5.2	Binary Relationship	40
2.5.3	Ternary Relationship	40
2.5.4	Quaternary Relationships	40
2.6	Relationship Classification	41
2.6.1	One-to-Many Relationship Type	41
2.6.2	One-to-One Relationship Type	41
2.6.3	Many-to-Many Relationship Type	41
2.6.4	Many-to-One Relationship Type	42
2.7	Reducing ER Diagram to Tables	42
2.7.1	Mapping Algorithm	42
2.7.2	Mapping Regular Entities	43
2.7.3	Converting Composite Attribute in an ER Diagram to Tables	44
2.7.4	Mapping Multivalued Attributes in ER Diagram to Tables	45

2.7.5	Converting “Weak Entities” in ER Diagram to Tables	45
2.7.6	Converting Binary Relationship to Table	46
2.7.7	Mapping Associative Entity to Tables	47
2.7.8	Converting Unary Relationship to Tables	49
2.7.9	Converting Ternary Relationship to Tables	50
2.8	Enhanced Entity–Relationship Model (EER Model)	51
2.8.1	Supertype or Superclass	51
2.8.2	Subtype or Subclass	52
2.9	Generalization and Specialization	52
2.10	ISA Relationship and Attribute Inheritance	53
2.11	Multiple Inheritance	53
2.12	Constraints on Specialization and Generalization	54
2.12.1	Overlap Constraint	54
2.12.2	Disjoint Constraint	55
2.12.3	Total Specialization	55
2.12.4	Partial Specialization	56
2.13	Aggregation and Composition	56
2.14	Entity Clusters	57
2.15	Connection Traps	58
2.15.1	Fan Trap	59
2.15.2	Chasm Trap	59
2.16	Advantages of ER Modeling	60
3	Relational Model	65
3.1	Introduction	65
3.2	CODD’S Rules	65
3.3	Relational Data Model	67
3.3.1	Structural Part	67
3.3.2	Integrity Part	67
3.3.3	Manipulative Part	68
3.3.4	Table and Relation	69
3.4	Concept of Key	69
3.4.1	Superkey	69
3.4.2	Candidate Key	70
3.4.3	Foreign Key	70
3.5	Relational Integrity	70
3.5.1	Entity Integrity	70
3.5.2	Null Integrity	71
3.5.3	Domain Integrity Constraint	71
3.5.4	Referential Integrity	71
3.6	Relational Algebra	72
3.6.1	Role of Relational Algebra in DBMS	72
3.7	Relational Algebra Operations	72
3.7.1	Unary and Binary Operations	72

3.7.2	Rename operation (ρ)	76
3.7.3	Union Operation	77
3.7.4	Intersection Operation	78
3.7.5	Difference Operation	79
3.7.6	Division Operation	80
3.7.7	Cartesian Product Operation	82
3.7.8	Join Operations	83
3.8	Advantages of Relational Algebra	89
3.9	Limitations of Relational Algebra	89
3.10	Relational Calculus	90
3.10.1	Tuple Relational Calculus	90
3.10.2	Set Operators in Relational Calculus	92
3.11	Domain Relational Calculus (DRC)	97
3.11.1	Queries in Domain Relational Calculus:	98
3.11.2	Queries and Domain Relational Calculus Expressions	98
3.12	QBE	102
4	Structured Query Language	111
4.1	Introduction	111
4.2	History of SQL Standard	112
4.2.1	Benefits of Standardized Relational Language	113
4.3	Commands in SQL	113
4.4	Datatypes in SQL	114
4.5	Data Definition Language (DDL)	117
4.6	Selection Operation	121
4.7	Projection Operation	122
4.8	Aggregate Functions	124
4.8.1	COUNT Function	124
4.8.2	MAX, MIN, and AVG Aggregate Function	127
4.9	Data Manipulation Language	135
4.9.1	Adding a New Row to the Table	136
4.9.2	Updating the Data in the Table	137
4.9.3	Deleting Row from the Table	138
4.10	Table Modification Commands	138
4.10.1	Adding a Column to the Table	139
4.10.2	Modifying the Column of the Table	141
4.10.3	Deleting the Column of the Table	142
4.11	Table Truncation	143
4.11.1	Dropping a Table	145
4.12	Imposition of Constraints	146
4.12.1	NOT NULL Constraint	147
4.12.2	UNIQUE Constraint	149
4.12.3	Primary Key Constraint	151
4.12.4	CHECK Constraint	154

4.12.5	Referential Integrity Constraint	155
4.12.6	ON DELETE CASCADE	159
4.12.7	ON DELETE SET NULL	161
4.13	Join Operation	163
4.13.1	Equijoin	165
4.14	Set Operations	166
4.14.1	UNION Operation	166
4.14.2	INTERSECTION Operation	168
4.14.3	MINUS Operation	169
4.15	View	169
4.15.1	Nonupdatable View	172
4.15.2	Views from Multiple Tables	176
4.15.3	View From View	179
4.15.4	VIEW with CHECK Constraint	186
4.15.5	Views with Read-only Option	187
4.15.6	Materialized Views	191
4.16	Subquery	192
4.16.1	Correlated Subquery	194
4.17	Embedded SQL	201
5	PL/SQL	213
5.1	Introduction	213
5.2	Shortcomings in SQL	213
5.3	Structure of PL/SQL	214
5.4	PL/SQL Language Elements	215
5.5	Data Types	222
5.6	Operators Precedence	223
5.7	Control Structure	224
5.8	Steps to Create a PL/SQL Program	226
5.9	Iterative Control	228
5.10	Cursors	231
5.10.1	Implicit Cursors	232
5.10.2	Explicit Cursor	234
5.11	Steps to Create a Cursor	235
5.11.1	Declare the Cursor	235
5.11.2	Open the Cursor	236
5.11.3	Passing Parameters to Cursor	237
5.11.4	Fetch Data from the Cursor	237
5.11.5	Close the Cursor	237
5.12	Procedure	243
5.13	Function	247
5.14	Packages	252
5.15	Exceptions Handling	255
5.16	Database Triggers	264
5.17	Types of Triggers	267

6	Database Design	283
6.1	Introduction	283
6.2	Objectives of Database Design	285
6.3	Database Design Tools	286
6.3.1	Need for Database Design Tool	286
6.3.2	Desired Features of Database Design Tools	286
6.3.3	Advantages of Database Design Tools	287
6.3.4	Disadvantages of Database Design Tools	287
6.3.5	Commercial Database Design Tools	287
6.4	Redundancy and Data Anomaly	288
6.4.1	Problems of Redundancy	288
6.4.2	Insertion, Deletion, and Updation Anomaly	288
6.5	Functional Dependency	289
6.6	Functional Dependency Inference Rules (Armstrong's Axioms)	292
6.7	Closure of Set of Functional Dependencies	294
6.7.1	Closure of a Set of Attributes	294
6.7.2	Minimal Cover	295
6.8	Normalization	296
6.8.1	Purpose of Normalization	296
6.9	Steps in Normalization	296
6.10	Unnormal Form to First Normal Form	298
6.11	First Normal Form to Second Normal Form	300
6.12	Second Normal Form to Third Normal Form	301
6.13	Boyce–Codd Normal Form (BCNF)	304
6.14	Fourth and Fifth Normal Forms	307
6.14.1	Fourth Normal Form	307
6.14.2	Fifth Normal Form	311
6.15	Denormalization	311
6.15.1	Basic Types of Denormalization	311
6.15.2	Table Denormalization Algorithm	312
7	Transaction Processing and Query Optimization	319
7.1	Transaction Processing	319
7.1.1	Introduction	319
7.1.2	Key Notations in Transaction Management	320
7.1.3	Concept of Transaction Management	320
7.1.4	Lock-Based Concurrency Control	326
7.2	Query Optimization	332
7.2.1	Query Processing	333
7.2.2	Need for Query Optimization	333
7.2.3	Basic Steps in Query Optimization	334
7.2.4	Query Optimizer Architecture	335
7.2.5	Basic Algorithms for Executing Query Operations	341

7.2.6	Query Evaluation Plans	344
7.2.7	Optimization by Genetic Algorithms	346
8	Database Security and Recovery	353
8.1	Database Security	353
8.1.1	Introduction	353
8.1.2	Need for Database Security	354
8.1.3	General Considerations.....	354
8.1.4	Database Security System	356
8.1.5	Database Security Goals and Threats	356
8.1.6	Classification of Database Security	357
8.2	Database Recovery	368
8.2.1	Different Types of Database Failures	368
8.2.2	Recovery Facilities.....	368
8.2.3	Main Recovery Techniques.....	370
8.2.4	Crash Recovery	370
8.2.5	ARIES Algorithm	371
9	Physical Database Design	381
9.1	Introduction	381
9.2	Goals of Physical Database Design.....	382
9.2.1	Physical Design Steps.....	382
9.2.2	Implementation of Physical Model.....	383
9.3	File Organization.....	384
9.3.1	Factors to be Considered in File Organization.....	384
9.3.2	File Organization Classification	384
9.4	Heap File Organization.....	385
9.4.1	Uses of Heap File Organization	385
9.4.2	Drawback of Heap File Organization.....	385
9.4.3	Example of Heap File Organization.....	386
9.5	Sequential File Organization	386
9.5.1	Sequential Processing of File.....	387
9.5.2	Draw Back	387
9.6	Hash File Organization.....	387
9.6.1	Hashing Function	387
9.6.2	Bucket	388
9.6.3	Choice of Bucket	389
9.6.4	Extendible Hashing	391
9.7	Index File Organization	392
9.7.1	Advantage of Indexing	392
9.7.2	Classification of Index	392
9.7.3	Search Key	393
9.8	Tree-Structured Indexes	394
9.8.1	ISAM.....	394

9.8.2	B-Tree	394
9.8.3	Building a B ⁺ Tree	394
9.8.4	Bitmap Index	396
9.9	Data Storage Devices	397
9.9.1	Factors to be Considered in Selecting Data Storage Devices	397
9.9.2	Magnetic Technology	397
9.9.3	Fixed Magnetic Disk	398
9.9.4	Removable Magnetic Disk	398
9.9.5	Floppy Disk	398
9.9.6	Magnetic Tape	398
9.10	Redundant Array of Inexpensive Disk	398
9.10.1	RAID Level 0 + 1	399
9.10.2	RAID Level 0	400
9.10.3	RAID Level 1	401
9.10.4	RAID Level 2	401
9.10.5	RAID Level 3	402
9.10.6	RAID Level 4	403
9.10.7	RAID Level 5	404
9.10.8	RAID Level 6	405
9.10.9	RAID Level 10	406
9.11	Software-Based RAID	406
9.12	Hardware-Based RAID	407
9.12.1	RAID Controller	407
9.12.2	Types of Hardware RAID	408
9.13	Optical Technology	409
9.13.1	Advantages of Optical Disks	409
9.13.2	Disadvantages of Optical Disks	409
10	Data Mining and Data Warehousing	415
10.1	Data Mining	415
10.1.1	Introduction	415
10.1.2	Architecture of Data Mining Systems	416
10.1.3	Data Mining Functionalities	417
10.1.4	Classification of Data Mining Systems	417
10.1.5	Major Issues in Data Mining	418
10.1.6	Performance Issues	419
10.1.7	Data Preprocessing	420
10.1.8	Data Mining Task	423
10.1.9	Data Mining Query Language	425
10.1.10	Architecture Issues in Data Mining System	426
10.1.11	Mining Association Rules in Large Databases	427
10.1.12	Mining Multilevel Association From Transaction Databases	430

10.1.13	Rule Constraints	433
10.1.14	Classification and Prediction	434
10.1.15	Comparison of Classification Methods	436
10.1.16	Prediction	441
10.1.17	Cluster Analysis	442
10.1.18	Mining Complex Types of Data	449
10.1.19	Applications and Trends in Data Mining	453
10.1.20	How to Choose a Data Mining System	456
10.1.21	Theoretical Foundations of Data Mining	458
10.2	Data Warehousing	461
10.2.1	Goals of Data Warehousing	461
10.2.2	Characteristics of Data in Data Warehouse	462
10.2.3	Data Warehouse Architectures	462
10.2.4	Data Warehouse Design	465
10.2.5	Classification of Data Warehouse Design	467
10.2.6	The User Interface	471
11	Objected-Oriented and Object Relational DBMS	477
11.1	Objected oriented DBMS	477
11.1.1	Introduction	477
11.1.2	Object-Oriented Programming Languages (OOPLs)	479
11.1.3	Availability of OO Technology and Applications	481
11.1.4	Overview of OODBMS Technology	482
11.1.5	Applications of an OODBMS	487
11.1.6	Evaluation Criteria	491
11.1.7	Evaluation Targets	519
11.1.8	Object Relational DBMS	525
11.1.9	Object-Relational Model	526
11.1.10	Aggregation and Composition in UML	529
11.1.11	Object-Relational Database Design	530
11.1.12	Comparison of OODBMS and ORDBMS	537
12	Distributed and Parallel Database Management Systems	559
12.1	Distributed Database	559
12.1.1	Features of Distributed vs. Centralized Databases	561
12.2	Distributed DBMS Architecture	562
12.2.1	DBMS Standardization	562
12.2.2	Architectural Models for Distributed DBMS	563
12.2.3	Types of Distributed DBMS Architecture	564
12.3	Distributed Database Design	565
12.3.1	Framework for Distributed Database Design	566
12.3.2	Objectives of the Design of Data Distribution	567
12.3.3	Top-Down and Bottom-Up Approaches to the Design of Data Distribution	568
12.3.4	Design of Database Fragmentation	568

12.4	Semantic Data Control	572
12.4.1	View Management	572
12.4.2	Views in Centralized DBMSs	573
12.4.3	Update Through Views	573
12.4.4	Views in Distributed DBMS	574
12.4.5	Data Security	574
12.4.6	Centralized Authorization Control	575
12.4.7	Distributed Authorization Control	575
12.4.8	Semantic Integrity Control	576
12.4.9	Distributed Semantic Integrity Control	577
12.5	Distributed Concurrency Control	578
12.5.1	Serializability Theory	578
12.5.2	Taxonomy of Concurrency Control Mechanism	578
12.5.3	Locking-Based Concurrency Control	580
12.5.4	Timestamp-Based Concurrency Control Algorithms	582
12.5.5	Optimistic Concurrency Control Algorithms	583
12.5.6	Deadlock Management	583
12.6	Distributed DBMS Reliability	586
12.6.1	Reliability Concepts and Measures	586
12.6.2	Failures in Distributed DBMS	588
12.6.3	Basic Fault Tolerance Approaches and Techniques	590
12.6.4	Distributed Reliability Protocols	590
12.7	Parallel Database	592
12.7.1	Database Server and Distributed Databases	593
12.7.2	Main Components of Parallel Processing	595
12.7.3	Functional Aspects	597
12.7.4	Various Parallel System Architectures	599
12.7.5	Parallel DBMS Techniques	602
13	Recent Challenges in DBMS	611
13.1	Genome Databases	612
13.1.1	Introduction	612
13.1.2	Basic Idea of Genome	612
13.1.3	Building Block of DNA	612
13.1.4	Genetic Code	614
13.1.5	GDS (Genome Directory System) Project	614
13.1.6	Conclusion	619
13.2	Mobile Database	619
13.2.1	Concept of Mobile Database	619
13.2.2	General Block Diagram of Mobile Database Center	620
13.2.3	Mobile Database Architecture	620
13.2.4	Modes of Operations of Mobile Database	622
13.2.5	Mobile Database Management	622
13.2.6	Mobile Transaction Processing	623
13.2.7	Distributed Database for Mobile	624

13.3	Spatial Database	626
13.3.1	Spatial Data Types	627
13.3.2	Spatial Database Modeling	628
13.3.3	Discrete Geometric Spaces	628
13.3.4	Querying	629
13.3.5	Integrating Geometry into a Query Language	630
13.3.6	Spatial DBMS Implementation	631
13.4	Multimedia Database Management System	632
13.4.1	Introduction	632
13.4.2	Multimedia Data	632
13.4.3	Multimedia Data Model	633
13.4.4	Architecture of Multimedia System	635
13.4.5	Multimedia Database Management System Development	636
13.4.6	Issues in Multimedia DBMS	636
13.5	XML	637
13.5.1	Introduction	637
13.5.2	Origin of XML	637
13.5.3	Goals of XML	638
13.5.4	XML Family	638
13.5.5	XML and HTML	638
13.5.6	XML Document	639
13.5.7	Document Type Definitions (DTD)	640
13.5.8	Extensible Style Sheet Language (XSL)	640
13.5.9	XML Namespaces	641
13.5.10	XML and Database Applications	643
14	Projects in DBMS	645
14.1	List of Projects	645
14.2	Overview of the Projects	645
14.2.1	Front-End: Microsoft Visual Basic	645
14.2.2	Back-End: Oracle 9i	646
14.2.3	Interface: ODBC	646
14.3	First Project: Bus Transport Management System	647
14.3.1	Description	647
14.3.2	Features of the Project	647
14.3.3	Source Code	649
14.4	Second Project: Course Administration System	656
14.4.1	Description	656
14.4.2	Source Code	656
14.5	Third Project: Election Voting System	666
14.5.1	Description	666
14.5.2	Source Code	666
14.6	Fourth Project: Hospital Management System	673
14.6.1	Description	673
14.6.2	Source Code	674

14.7	Fifth Project: Library Management System	680
14.7.1	Description	680
14.7.2	Source Code	680
14.8	Sixth Project: Railway Management System	690
14.8.1	Description	690
14.8.2	Source Code	690
14.9	Some Hints to Do Successful Projects in DBMS	696
A Dictionary of DBMS Terms		699
B Overview of Commands in SQL		721
C Pioneers in DBMS		727
C.1	About Dr. Edgar F. Codd	728
C.2	Ronald Fagin	736
C.2.1	Abstract of Ronald Fagin's Article	737
D Popular Commercial DBMS		739
D.1	System R	739
D.1.1	Introduction to System R	739
D.1.2	Keywords Used	739
D.1.3	Architecture and System Structure	740
D.1.4	Relational Data Interface	742
D.1.5	Data Manipulation Facilities in SEQUEL	743
D.1.6	Data Definition Facilities	745
D.1.7	Data Control Facilities	746
D.2	Relational Data System	749
D.3	DB2	752
D.3.1	Introduction to DB2	752
D.3.2	Definition of DB2 Data Structures	753
D.3.3	DB2 Stored Procedure	753
D.3.4	DB2 Processing Environment	755
D.3.5	DB2 Commands	757
D.3.6	Data Sharing in DB2	759
D.3.7	Conclusion	760
D.4	Informix	760
D.4.1	Introduction to Informix	760
D.4.2	Informix SQL and ANSI SQL	761
D.4.3	Software Dependencies	762
D.4.4	New Features in Version 7.3	763
D.4.5	Conclusion	766
Bibliography		767

Abbreviations









ACM	Association of Computing Machinery
ACID	Atomicity, Consistency, Isolation, and Durability
ANSI	American National Standard Institute
ANSI/SPARK	American National Standard Institute/Standards Planning And Requirements Committee
API	Application Program Interface
ARIES	Algorithms for Recovery and Isolation Exploiting -Semantics
ASCII	American Standard Code for Information Interchange
ASP	Active Server Page
BCNF	Boyce-Codd Normal Form
BLOB	Binary Large Object
CAD/CAM	Computer Aided Design/Computer Aided Manufacturing
CAEP	Classification by Aggregating Emerging Patterns
CASE	Computer Aided Software Engineering
CLOB	Character Large Object
CD	Compact Disk
CD-ROM	Compact Disk Read Only Memory
CD-RW	Compact Disk ReWritable
CLARA	Clustering LARge Application
CLARANS	Clustering Large Application based upon Randomized Search
CODASYL	Conference On Data System Language
CPT	Conditional Probability Table
CSS	Cascade Style Sheet
CURE	Clustering Using Representatives
CURSOR	Current Set of Records
DB	Database
DB2	Database 2 (an IBM Relational DBMS)
DBMS	Database Management System
DBA	Database Administrator

XXIV Abbreviations

DBTG	Database Task Group
DCL	Data Control Language
DD	Data Dictionary
DDBMS	Distributed Database Management Systems
DDL	Data Description Language
DKNF	Domain Key Normal Form
DLM	Distributed Lock Manager
DL/I	Data Language I
DM	Data Manager
DML	Data Manipulation Language
DOM	Document Object Model
DRC	Domain Relational Calculus
DSS	Decision Support System
DTD	Document Type Definition
DW	Data Warehouse
ER Model	Entity Relationship Model
EER Model	Enhanced Entity Relationship Model
ERD	Entity Relationship Diagram
FD	Functional Dependency
GDS	Genome Directory System
GIS	Geographical Information System
GLS	Global Language Support
GMOD	Generic Model Organism Database
GUAM	Generalized Update Access Method
GUI	Graphical User Interface
HGP	Human Genome Project
HTML	Hyper Text Markup Language
NAS	Network Attached Storage
IBM	International Business Machines
IDE	Integrated Development Environment
IMS	Information Management System
ISAM	Indexed Sequential Access Method
ISO	International Standard Organization
JDBC	Java Database Connectivity
LAN	Local Area Network
MARS	Multimedia Analysis and Retrieval System
MMDDBMS	Multimedia Database Management System
MM	Media Manager
MOLAP	Multidimensional Online Analytical Processing
MPEG	Motion Picture Expert Group
MTL	Multimedia Transaction Language
ODBC	Open Database Connectivity
ODMG	Object Database Management Group

OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
OMG	Object Management Group
OOPL	Object Oriented Programming Language
ORDBMS	Object Relational Database Management System
OODBMS	Object Oriented Database Management System
OS	Operating System
PAM	Partitioning Around Medoids
PCTE	Portable Common Tool Environment
PL/SQL	Programming Language/Structured Query Language
QBE	Query By Example
RAID	Redundant Array of Inexpensive/Independent Disk
RDBMS	Relational Database Management System
ROLAP	Relational Online Analytical Processing
SCSI	Small Computer System Interface
SEQUEL	Structured Query English Language
SGML	Standard Generalized Markup Language
SQL	Structured Query Language
SQL/DS	Structured Query Language/Data System
TM	Transaction Manager
TRC	Tuple Relational Calculus
UML	Unified Modeling Language
VB	Visual Basic
VSAM	Virtual Storage Access Method
WORM	Write Once Read Many
WWW	World Wide Web
W3C	World Wide Web Consortium
XML	Extended Markup Language
XSL	Extensible Style Sheet Language
2PL	Two Phase Lock
4GL	Fourth Generation Language
1:1	One-to-One
1:M	One-to-Many
1NF	First Normal Form
2NF	Second Normal Form
3NF	Third Normal Form
4NF	Fourth Normal Form
5NF	Fifth Normal Form

List of Symbols

Symbol	Meaning
π	Projection operator
σ	Selection operator
\cup	Union operator
\cap	Intersection operator
\times	Cartesian product operator
\bowtie	Join operator
\ltimes	Left outer join operator
\rtimes	Right outer join operator
$\bowtie\!\!\!\!\!\times$	Full outer join operator
$\ltimes\!\!\!\!\!\times$	Semi join operator
ρ	Rename operator
\forall	Universal quantifier
\exists	Existential quantifier
	Entity
	Attribute
	Multivalued attribute
	Relationship
	Associative entity
	Identifying relationship type
	Derived attribute
	Weak entity type

Overview of Database Management System

Learning Objectives. This chapter provides an overview of database management system which includes concepts related to data, database, and database management system. After completing this chapter the reader should be familiar with the following concepts:

- Data, information, database, database management system
- Need and evolution of DBMS
- File management vs. database management system
- ANSI/SPARK data model
- Database architecture: two-, three-, and multitier architecture

1.1 Introduction

Science, business, education, economy, law, culture, all areas of human development “work” with the constant aid of data. Databases play a crucial role within science research: the body of scientific and technical data and information in the public domain is massive and factual data are fundamental to the progress of science. But the progress of science is not the only process affected by the way people use databases. Stock exchange data are absolutely necessary to any analyst; access to comprehensive databases of large scale is an everyday activity of a teacher, an educator, an academic or a lawyer. There are databases collecting all sorts of different data: nuclear structure and radioactive decay data for isotopes (the Evaluated Nuclear Structure Data File) and genes sequences (the Human Genome Database), prisoners’ DNA data (“DNA offender database”), names of people accused for drug offenses, telephone numbers, legal materials and many others. In this chapter, the basic idea about database management system, its evolution, its advantage over conventional file system, database system structure is discussed.

1.2 Data and Information

Data are raw facts that constitute building block of information. Data are the heart of the DBMS. It is to be noted that all the data will not convey useful information. Useful information is obtained from processed data. In other words, data has to be interpreted in order to obtain information. Good, timely, relevant information is the key to decision making. Good decision making is the key to organizational survival.

Data are a representation of facts, concepts, or instructions in a formalized manner suitable for communication, interpretation, or processing by humans or automatic means. The data in DBMS can be broadly classified into two types, one is the collection of information needed by the organization and the other is “metadata” which is the information about the database. The term “metadata” will be discussed in detail later in this chapter.

Data are the most stable part of an organization’s information system. A company needs to save information about employees, departments, and salaries. These pieces of information are called data. Permanent storage of data are referred to as persistent data. Generally, we perform operations on data or data items to supply some information about an entity. For example library keeps a list of members, books, due dates, and fines.

1.3 Database

A database is a well-organized collection of data that are related in a meaningful way, which can be accessed in different logical orders. Database systems are systems in which the interpretation and storage of information are of primary importance. The database should contain all the data needed by the organization as a result, a huge volume of data, the need for long-term storage of the data, and access of the data by a large number of users generally characterize database systems. The simplified view of database system is shown in Fig. 1.1. From this figure, it is clear that several users can access the data in an

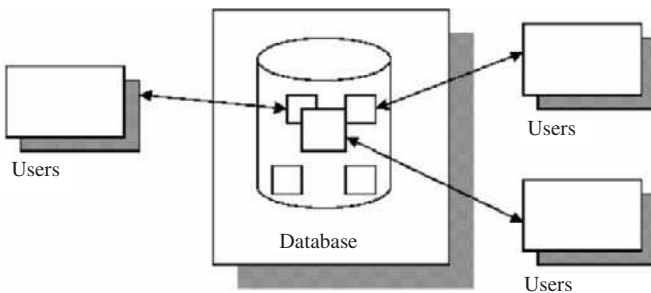


Fig. 1.1. Simplified database view

organization still the integrity of the data should be maintained. A database is integrated when same information is not recorded in two places.

1.4 Database Management System

A database management system (DBMS) consists of collection of interrelated data and a set of programs to access that data. It is software that is helpful in maintaining and utilizing a database.

A DBMS consists of:

- A collection of interrelated and persistent data. This part of DBMS is referred to as database (DB).
- A set of application programs used to access, update, and manage data. This part constitutes data management system (MS).
- A DBMS is general-purpose software i.e., not application specific. The same DBMS (e.g., Oracle, Sybase, etc.) can be used in railway reservation system, library management, university, etc.
- A DBMS takes care of storing and accessing data, leaving only application specific tasks to application programs.

DBMS is a complex system that allows a user to do many things to data as shown in Fig. 1.2. From this figure, it is evident that DBMS allows user to input data, share the data, edit the data, manipulate the data, and display the data in the database. Because a DBMS allows more than one user to share the data; the complexity extends to its design and implementation.

1.4.1 Structure of DBMS

An overview of the structure of database management system is shown in Fig. 1.3. A DBMS is a software package, which translates data from its logical representation to its physical representation and back.

The DBMS uses an application specific database description to define this translation. The database description is generated by a database designer

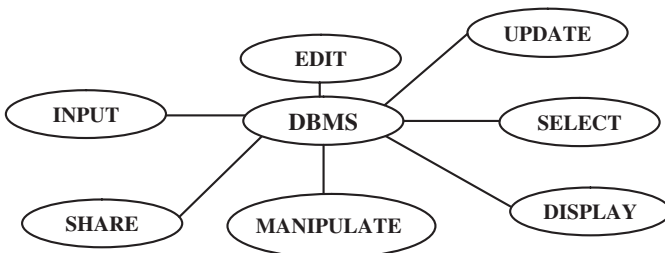


Fig. 1.2. Capabilities of database management system

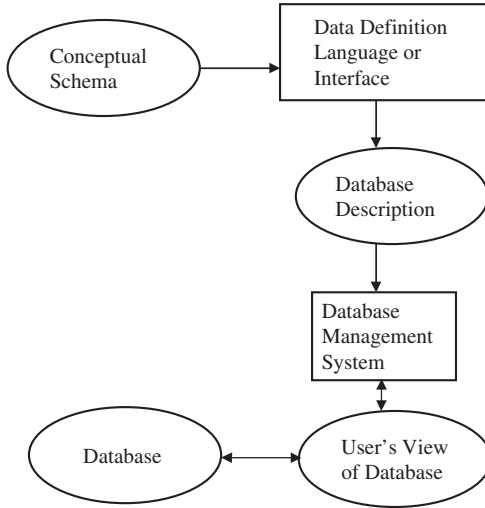


Fig. 1.3. Structure of database management system

from his or her conceptual view of the database, which is called the Conceptual Schema. The translation from the conceptual schema to the database description is performed using a data definition language (DDL) or a graphical or textual design interface.

1.5 Objectives of DBMS

The main objectives of database management system are data availability, data integrity, data security, and data independence.

1.5.1 Data Availability

Data availability refers to the fact that the data are made available to wide variety of users in a meaningful format at reasonable cost so that the users can easily access the data.

1.5.2 Data Integrity

Data integrity refers to the correctness of the data in the database. In other words, the data available in the database is a reliable data.

1.5.3 Data Security

Data security refers to the fact that only authorized users can access the data. Data security can be enforced by passwords. If two separate users are accessing a particular data at the same time, the DBMS must not allow them to make conflicting changes.

1.5.4 Data Independence

DBMS allows the user to store, update, and retrieve data in an efficient manner. DBMS provides an “abstract view” of how the data is stored in the database.

In order to store the information efficiently, complex data structures are used to represent the data. The system hides certain details of how the data are stored and maintained.

1.6 Evolution of Database Management Systems

File-based system was the predecessor to the database management system. Apollo moon-landing process was started in the year 1960. At that time, there was no system available to handle and manage large amount of information. As a result, North American Aviation which is now popularly known as Rockwell International developed software known as Generalized Update Access Method (GUAM). In the mid-1960s, IBM joined North American Aviation to develop GUAM into Information Management System (IMS). IMS was based on Hierarchical data model. In the mid-1960s, General Electric released Integrated Data Store (IDS). IDS were based on network data model. Charles Bachmann was mainly responsible for the development of IDS. The network database was developed to fulfill the need to represent more complex data relationships than could be modeled with hierarchical structures. Conference on Data System Languages formed Data Base Task Group (DBTG) in 1967. DBTG specified three distinct languages for standardization. They are Data Definition Language (DDL), which would enable Database Administrator to define the schema, a subschema DDL, which would allow the application programs to define the parts of the database and Data Manipulation Language (DML) to manipulate the data.

The network and hierarchical data models developed during that time had the drawbacks of minimal data independence, minimal theoretical foundation, and complex data access. To overcome these drawbacks, in 1970, Codd of IBM published a paper titled “A Relational Model of Data for Large Shared Data Banks” in *Communications of the ACM*, vol. 13, No. 6, pp. 377–387, June 1970. As an impact of Codd’s paper, System R project was developed during the late 1970 by IBM San Jose Research Laboratory in California. The project was developed to prove that relational data model was implementable. The outcome of System R project was the development of Structured Query Language (SQL) which is the standard language for relational database management system. In 1980s IBM released two commercial relational database management systems known as DB2 and SQL/DS and Oracle Corporation released Oracle. In 1979, Codd himself attempted to address some of the failings in his original work with an extended version of the relational model called RM/T in 1979 and RM/V2 in 1990. The attempts to provide a data model

that represents the “real world” more closely have been loosely classified as Semantic Data Modeling.

In recent years, two approaches to DBMS are more popular, which are Object-Oriented DBMS (OODBMS) and Object Relational DBMS (OR-DBMS).

The chronological order of the development of DBMS is as follows:

- Flat files – 1960s–1980s
- Hierarchical – 1970s–1990s
- Network – 1970s–1990s
- Relational – 1980s–present
- Object-oriented – 1990s–present
- Object-relational – 1990s–present
- Data warehousing – 1980s–present
- Web-enabled – 1990s–present

Early 1960s. Charles Bachman at GE created the first general purpose DBMS Integrated Data Store. It created the basis for the network model which was standardized by CODASYL (Conference on Data System Language).

Late 1960s. IBM developed the Information Management System (IMS). IMS used an alternate model, called the Hierarchical Data Model.

1970. Edgar Codd, from IBM created the Relational Data Model. In 1981 Codd received the Turing Award for his contributions to database theory. Codd Passed away in April 2003.

1976. Peter Chen presented Entity-Relationship model, which is widely used in database design.

1980. SQL developed by IBM, became the standard query language for databases. SQL was standardized by ISO.

1980s and 1990s. IBM, Oracle, Informix and others developed powerful DBMS.

1.7 Classification of Database Management System

The database management system can be broadly classified into (1) Passive Database Management System and (2) Active Database Management System:

1. *Passive Database Management System.* Passive Database Management Systems are program-driven. In passive database management system the users query the current state of database and retrieve the information currently available in the database. Traditional DBMS are passive in the sense that they are explicitly and synchronously invoked by user or application program initiated operations. Applications send requests for operations to be performed by the DBMS and wait for the DBMS to confirm and return any possible answers. The operations can be definitions and updates of the schema, as well as queries and updates of the data.

2. *Active Database Management System.* Active Database Management Systems are data-driven or event-driven systems. In active database management system, the users specify to the DBMS the information they need. If the information of interest is currently available, the DBMS actively monitors the arrival of the desired information and provides it to the relevant users. The scope of a query in a passive DBMS is limited to the past and present data, whereas the scope of a query in an active DBMS additionally includes future data. An active DBMS reverses the control flow between applications and the DBMS instead of only applications calling the DBMS, the DBMS may also call applications in an active DBMS.

Active databases contain a set of active rules that consider events that represent database state changes, look for TRUE or FALSE conditions as the result of a database predicate or query, and take an action via a data manipulation program embedded in the system. *Alert* is extension architecture at the IBM Almaden Research, for experimentation with active databases.

1.8 File-Based System

Prior to DBMS, file system provided by OS was used to store information. In a file-based system, we have collection of application programs that perform services for the end users. Each program defines and manages its own data.

Consider University database, the University database contains details about student, faculty, lists of courses offered, and duration of course, etc. In File-based processing for each database there is separate application program which is shown in Fig. 1.4.

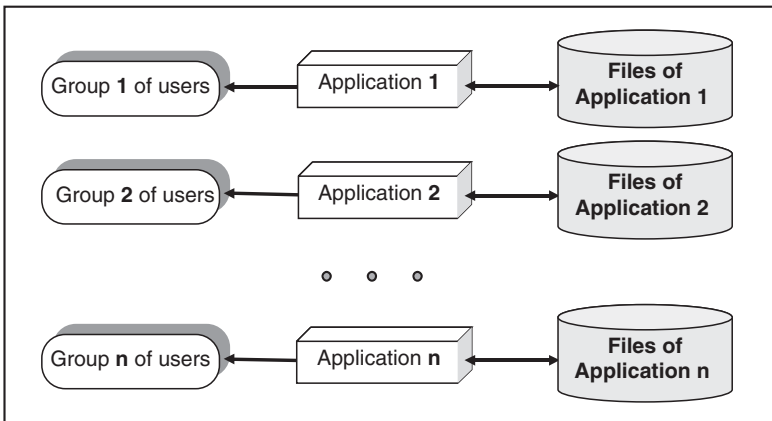


Fig. 1.4. File-based System

One group of users may be interested in knowing the courses offered by the university. One group of users may be interested in knowing the faculty information. The information is stored in separate files and separate applications programs are written.

1.9 Drawbacks of File-Based System

The limitations of file-based approach are duplication of data, data dependence, incompatible file formats, separation, and isolation of data.

1.9.1 Duplication of Data

Duplication of data means same data being stored more than once. This can also be termed as data redundancy. Data redundancy is a problem in file-based approach due to the decentralized approach. The main drawbacks of duplication of data are:

- Duplication of data leads to wastage of storage space. If the storage space is wasted it will have a direct impact on cost. The cost will increase.
- Duplication of data can lead to loss of data integrity; the data are no longer consistent. Assume that the employee detail is stored both in the department and in the main office. Now the employee changes his contact address. The changed address is stored in the department alone and not in the main office. If some important information has to be sent to his contact address from the main office then that information will be lost. This is due to the lack of decentralized approach.

1.9.2 Data Dependence

Data dependence means the application program depends on the data. If some modifications have to be made in the data, then the application program has to be rewritten. If the application program is independent of the storage structure of the data, then it is termed as data independence. Data independence is generally preferred as it is more flexible. But in file-based system there is program-data dependence.

1.9.3 Incompatible File Formats

As file-based system lacks program data independence, the structure of the file depends on the application programming language. For example, the structure of the file generated by FORTRAN program may be different from the structure of a file generated by “C” program. The incompatibility of such files makes them difficult to process jointly.

1.9.4 Separation and Isolation of Data

In file-based approach, data are isolated in separate files. Hence it is difficult to access data. The application programmer must synchronize the processing of two files to ensure that the correct data are extracted. This difficulty is more if data has to be retrieved from more than two files.

The draw backs of conventional file-based approach are summarized later:

1. We have to store the information in a secondary memory such as a disk. If the volume of information is large; it will occupy more memory space.
2. We have to depend on the addressing facilities of the system. If the database is very large, then it is difficult to address the whole set of records.
3. For each query, for example the address of the student and the list of electives that the student has chosen, we have to write separate programs.
4. While writing several programs, lot of variables will be declared and it will occupy some space.
5. It is difficult to ensure the integrity and consistency of the data when more than one program accesses some file and changes the data.
6. In case of a system crash, it becomes hard to bring back the data to a consistent state.
7. “Data redundancy” occurs when identical data are distributed over various files.
8. Data distributed in various files may be in different formats hence it is difficult to share data among different application (Data Isolation).

1.10 DBMS Approach

DBMS is software that provides a set of primitives for defining, accessing, and manipulating data. In DBMS approach, the same data are being shared by different application programs; as a result data redundancy is minimized. The DBMS approach of data access is shown in Fig. 1.5.

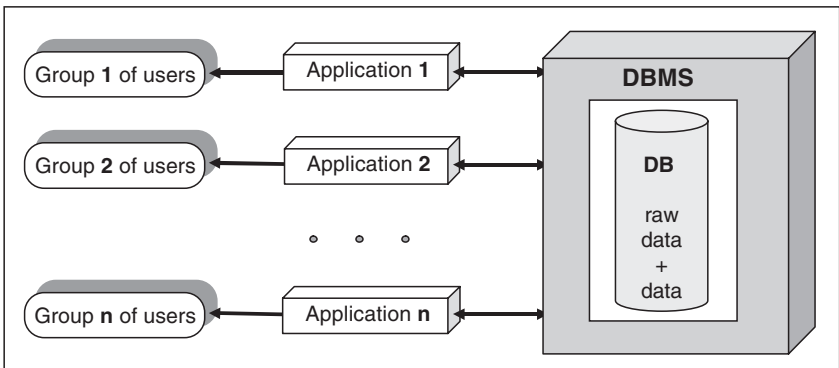


Fig. 1.5. Data access through DBMS

1.11 Advantages of DBMS

There are many advantages of database management system. Some of the advantages are listed later:

1. Centralized data management.
2. Data Independence.
3. System Integration.

1.11.1 Centralized Data Management

In DBMS all files are integrated into one system thus reducing redundancies and making data management more efficient.

1.11.2 Data Independence

Data independence means that programs are isolated from changes in the way the data are structured and stored. In a database system, the database management system provides the interface between the application programs and the data. Physical data independence means the applications need not worry about how the data are physically structured and stored. Applications should work with a logical data model and declarative query language.

If major changes were to be made to the data, the application programs may need to be rewritten. When changes are made to the data representation, the data maintained by the DBMS is changed but the DBMS continues to provide data to application programs in the previously used way.

Data independence is the immunity of application programs to changes in storage structures and access techniques. For example if we add a new attribute, change index structure then in traditional file processing system, the applications are affected. But in a DBMS environment these changes are reflected in the catalog, as a result the applications are not affected. Data independence can be physical data independence or logical data independence.

Physical data independence is the ability to modify physical schema without causing the conceptual schema or application programs to be rewritten.

Logical data independence is the ability to modify the conceptual schema without having to change the external schemas or application programs.

1.11.3 Data Inconsistency

Data inconsistency means different copies of the same data will have different values. For example, consider a person working in a branch of an organization. The details of the person will be stored both in the branch office as well as in the main office. If that particular person changes his address, then the “change of address” has to be maintained in the main as well as the branch office.

For example the “change of address” is maintained in the branch office but not in the main office, then the data about that person is inconsistent.

DBMS is designed to have data consistency. Some of the qualities achieved in DBMS are:

1. Data redundancy → Reduced in DBMS.
2. Data independence → Activated in DBMS.
3. Data inconsistency → Avoided in DBMS.
4. Centralizing the data → Achieved in DBMS.
5. Data integrity → Necessary for efficient Transaction.
6. Support for multiple views → Necessary for security reasons.
 - Data redundancy means duplication of data. Data redundancy will occupy more space hence it is not desirable.
 - Data independence means independence between application program and the data. The advantage is that when the data representation changes, it is not necessary to change the application program.
 - Data inconsistency means different copies of the same data will have different values.
 - Centralizing the data means data can be easily shared between the users but the main concern is data security.
 - The main threat to data integrity comes from several different users attempting to update the same data at the same time. For example, “The number of booking made is larger than the capacity of the aircraft/train.”
 - Support for multiple views means DBMS allows different users to see different “views” of the database, according to the perspective each one requires. This concept is used to enhance the security of the database.

1.12 Ansi/Spark Data Model (American National Standard Institute/ Standards Planning and Requirements Committee)

The distinction between the logical and physical representation of data were recognized in 1978 when ANSI/SPARK committee proposed a generalized framework for database systems. This framework provided a three-level architecture, three levels of abstraction at which the database could be viewed.

1.12.1 Need for Abstraction

The main objective of DBMS is to store and retrieve information efficiently; all the users should be able to access same data. The designers use complex data structure to represent the data, so that data can be efficiently stored and retrieved, but it is not necessary for the users to know physical database storage details. The developers hide the complexity from users through several levels of abstraction.

1.12.2 Data Independence

Data independence means the internal structure of database should be unaffected by changes to physical aspects of storage. Because of data independence, the Database administrator can change the database storage structures without affecting the users view.

The different levels of data abstraction are:

1. Physical level or internal level
2. Logical level or conceptual level
3. View level or external level

Physical Level

It is concerned with the physical storage of the information. It provides the internal view of the actual physical storage of data. The physical level describes complex low-level data structures in detail.

Logical Level

Logical level describes what data are stored in the database and what relationships exist among those data.

Logical level describes the entire database in terms of a small number of simple structures. The implementation of simple structure of the logical level may involve complex physical level structures; the user of the logical level does not need to be aware of this complexity. Database administrator use the logical level of abstraction.

View Level

View level is the highest level of abstraction. It is the view that the individual user of the database has. There can be many view level abstractions of the same data. The different levels of data abstraction are shown in Fig. 1.6.

Database Instances

Database change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an instance of the database.

Database Schema

The overall design of the database is called the database schema. A schema is a collection of named objects. Schemas provide a logical classification of objects in the database. A schema can contain tables, views, triggers, functions, packages, and other objects.

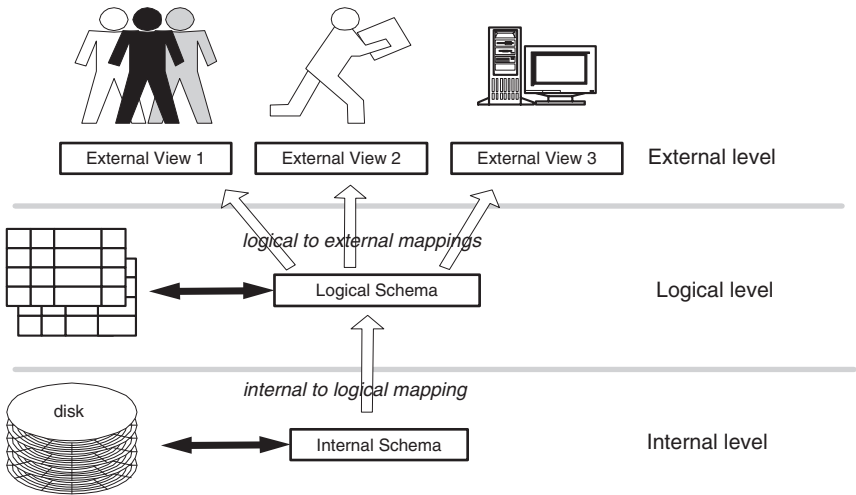
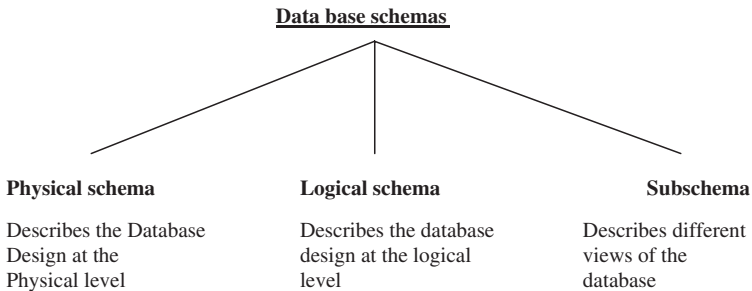


Fig. 1.6. ANSI/SPARK data model

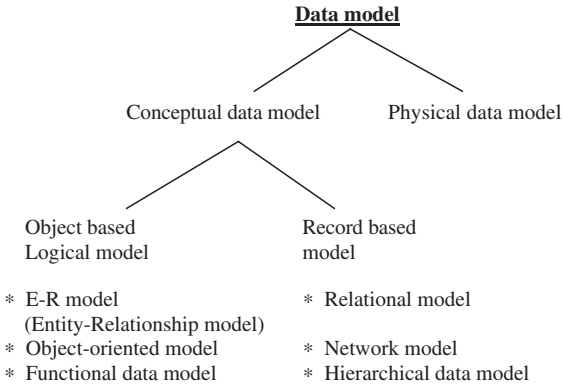
A schema is also an object in the database. It is explicitly created using the CREATE SCHEMA statement with the current user recorded as the schema owner. It can also be implicitly created when another object is created, provided the user has IMPLICIT_SCHEMA authority.



1.13 Data Models

Data model is collection of conceptual tools for describing data, relationship between data, and consistency constraints. Data models help in describing the structure of data at the logical level. Data model describe the structure of the database. A data model is the set of conceptual constructs available for defining a schema. The data model is a language for describing the data and database, it may consist of abstract concepts, which must be translated by the

designer into the constructs of the data definition interface, or it may consist of constructs, which are directly supported by the data definition interface. The constructs of the data model may be defined at many levels of abstraction.



1.13.1 Early Data Models

Three historically important data models are the hierarchical, network, and relational models. These models are relevant for their contributions in establishing the theory of data modeling and because they were all used as the basis of working and widely used database systems. Together they are often referred to as the “basic” data models. The hierarchical and network models, developed in the 1960s and 1970s, were based on organizing the primitive data structures in which the data were stored in the computer by adding connections or links between the structures. As such they were useful in presenting the user with a well-defined structure, but they were still highly coupled to the underlying physical representation of the data. Although they did much to assist in the efficient access of data, the principle of data independence was poorly supported.

1.14 Components and Interfaces of Database Management System

A database management system involves five major components: data, hardware, software, procedure, and users. These components and the interface between the components are shown in Fig. 1.7.

1.14.1 Hardware

The hardware can range from a single personal computer, to a single mainframe, to a network of computers. The particular hardware depends on the

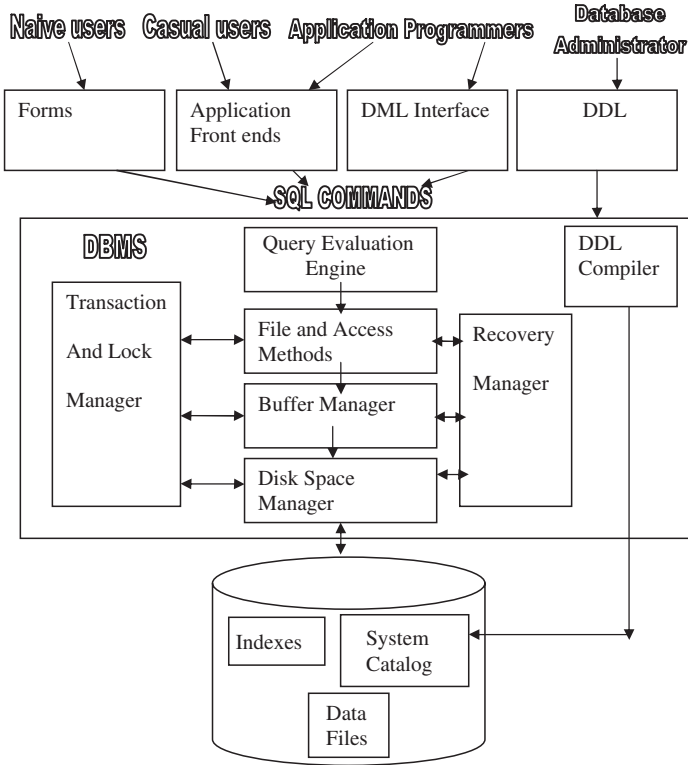


Fig. 1.7. Database management system components and interfaces

requirements of the organization and the DBMS used. Some DBMSs run only on particular operating systems, while others run on a wide variety of operating systems. A DBMS requires a minimum amount of main memory and disk space to run, but this minimum configuration may not necessarily give acceptable performance.

1.14.2 Software

The software includes the DBMS software, application programs together with the operating systems including the network software if the DBMS is being used over a network. The application programs are written in third-generation programming languages like “C,” COBOL, FORTRAN, Ada, Pascal, etc. or using fourth-generation language such as SQL, embedded in a third-generation language. The target DBMS may have its own fourth-generation tools which allow development of applications through the provision of nonprocedural query languages, report generators, graphics generators, and application generators. The use of fourth-generation tools can improve productivity significantly and produce programs that are easier to maintain.

1.14.3 Data

A database is a repository for data which, in general, is both integrated and shared. Integration means that the database may be thought of as a unification of several otherwise distinct files, with any redundancy among those files partially or wholly eliminated. The sharing of a database refers to the sharing of data by different users, in the sense that each of those users may have access to the same piece of data and may use it for different purposes. Any given user will normally be concerned with only a subset of the whole database. The main features of the data in the database are listed later:

1. The data in the database is well organized (structured)
2. The data in the database is related
3. The data are accessible in different orders without great difficulty

The data in the database is persistent, integrated, structured, and shared.

Integrated Data

A data can be considered to be a unification of several distinct data files and when any redundancy among those files is eliminated, the data are said to be integrated data.

Shared Data

A database contains data that can be shared by different users for different application simultaneously. It is important to note that in this way of sharing of data, the redundancy of data are reduced, since repetitions are avoided, the possibility of inconsistencies is reduced.

Persistent Data

Persistent data are one, which cannot be removed from the database as a side effect of some other process. Persistent data have a life span that is not limited to single execution of the programs that use them.

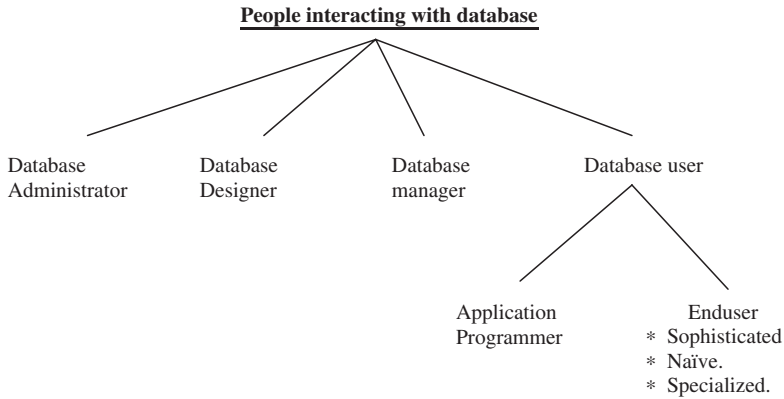
1.14.4 Procedure

Procedures are the rules that govern the design and the use of database. The procedure may contain information on how to log on to the DBMS, start and stop the DBMS, procedure on how to identify the failed component, how to recover the database, change the structure of the table, and improve the performance.

1.14.5 People Interacting with Database

Here people refers to the people who manages the database, database administrator, people who design the application program, database designer and the people who interacts with the database, database users.

A DBMS is typically run as a back-end server in a local or global network, offering services to clients directly or to Application Servers.



Database Administrator

Database Administrator is a person having central control over data and programs accessing that data. The database administrator is a manager whose responsibilities are focused on management of technical aspects of the database system. The objectives of database administrator are given as follows:

1. To control the database environment
2. To standardize the use of database and associated software
3. To support the development and maintenance of database application projects
4. To ensure all documentation related to standards and implementation is up-to-date

The summarized objectives of database administrator are shown in Fig. 1.8.

The control of the database environment should exist from the planning right through to the maintenance stage. During application development the database administrator should carry out the tasks that ensure proper control of the database when an application becomes operational. This includes review of each design stage to see if it is feasible from the database point of view. The database administrator should be responsible for developing standards to apply to development projects. In particular these standards apply to system analysis, design, and application programming for projects which are going to use the database. These standards will then be used as a basis for training systems analysts and programmers to use the database management system efficiently.

Responsibilities of Database Administrator (DBA)

The responsibility of the database administrator is to maintain the integrity, security, and availability of data. A database must be protected from

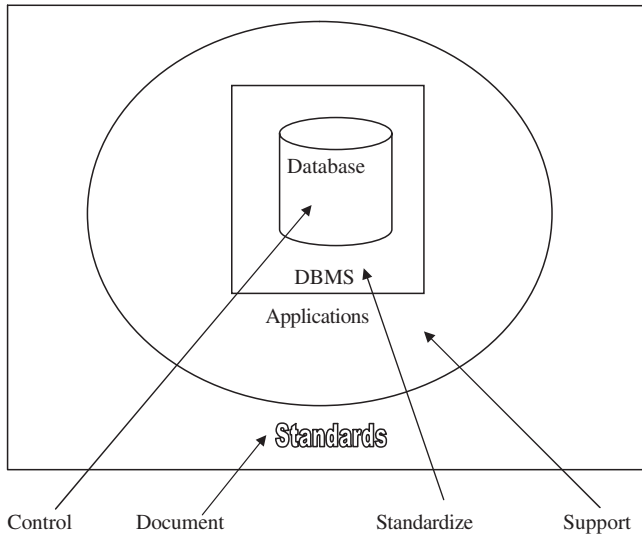


Fig. 1.8. Objectives of database administration

accidents, such as input or programming errors, from malicious use of the database and from hardware or software failures that corrupt data. Protection from accidents that cause data inaccuracy is a part of maintaining data integrity. Protecting the database from unauthorized or malicious use is termed as database security. The responsibilities of the database administrator are summarized as follows:

1. Authorizing access to the database.
2. Coordinating and monitoring its use.
3. Acquiring hardware and software resources as needed.
4. Backup and recovery. DBA has to ensure regular backup of database, in case of damage, suitable recovery procedure are used to bring the database up with little downtime as possible.

Database Designer

Database designer can be either logical database designer or physical database designer. Logical database designer is concerned with identifying the data, the relationships between the data, and the constraints on the data that is to be stored in the database. The logical database designer must have thorough understanding of the organizations data and its business rule.

The physical database designer takes the logical data model and decides the way in which it can be physically implemented. The logical database designer is responsible for mapping the logical data model into a set of tables and integrity constraints, selecting specific storage structure, and designing

security measures required on the data. In a nutshell, the database designer is responsible for:

1. Identifying the data to be stored in the database.
2. Choosing appropriate structure to represent and store the data.

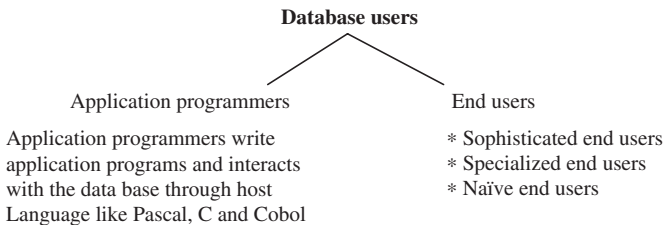
Database Manager

Database manager is a program module which provides the interface between the low level data stored in the database and the application programs and queries submitted to the system:

- The database manager would translate DML statement into low level file system commands for storing, retrieving, and updating data in the database.
- *Integrity enforcement.* Database manager enforces integrity by checking consistency constraints like the bank balance of customer must be maintained to a minimum of Rs. 300, etc.
- *Security enforcement.* Unauthorized users are prohibited to view the information stored in the data base.
- *Backup and recovery.* Backup and recovery of database is necessary to ensure that the database must remain consistent despite the fact of failures.

Database Users

Database users are the people who need information from the database to carry out their business responsibility. The database users can be broadly classified into two categories like application programmers and end users.



Sophisticated End Users

Sophisticated end users interact with the system without writing programs. They form requests by writing queries in a database query language. These are submitted to query processor. Analysts who submit queries to explore data in the database fall in this category.

Specialized End Users

Specialized end users write specialized database application that does not fit into data-processing frame work. Application involves knowledge base and expert system, environment modeling system, etc.

Naive End Users

Naïve end user interact with the system by using permanent application program Example: Query made by the student, namely number of books borrowed in library database.

System Analysts

System analysts determine the requirements of end user, and develop specification for canned transaction that meets this requirement.

Canned Transaction

Ready made programs through which naïve end users interact with the database is called canned transaction.

1.14.6 Data Dictionary

A data dictionary, also known as a “system catalog,” is a centralized store of information about the database. It contains information about the tables, the fields the tables contain, data types, primary keys, indexes, the joins which have been established between those tables, referential integrity, cascades update, cascade delete, etc. This information stored in the data dictionary is called the “Metadata.” Thus a data dictionary can be considered as a file that stores Metadata. Data dictionary is a tool for recording and processing information about the data that an organization uses. The data dictionary is a central catalog for Metadata. The data dictionary can be integrated within the DBMS or separate. Data dictionary may be referenced during system design, programming, and by actively-executing programs. One of the major functions of a true data dictionary is to enforce the constraints placed upon the database by the designer, such as referential integrity and cascade delete.

Metadata

The information (data) about the data in a database is called Metadata. The Metadata are available for query and manipulation, just as other data in the database.

1.14.7 Functional Components of Database System Structure

The functional components of database system structure are:

1. Storage manager.
2. Query processor.

Storage Manager

Storage manager is responsible for storing, retrieving, and updating data in the database. Storage manager components are:

1. Authorization and integrity manager.
2. Transaction manager.
3. File manager.
4. Buffer manager.

Transaction Management

- A transaction is a collection of operations that performs a single logical function in a database application.
- Transaction-management component ensures that the database remains in a consistent state despite system failures and transaction failure.
- Concurrency control manager controls the interaction among the concurrent transactions, to ensure the consistency of the database.

Authorization and Integrity Manager

Checks the integrity constraints and authority of users to access data.

Transaction Manager

It ensures that the database remains in a consistent state despite system failures. The transaction manager manages the execution of database manipulation requests. The transaction manager function is to ensure that concurrent access to data does not result in conflict.

File Manager

File manager manages the allocation of space on disk storage. Files are used to store collections of similar data. A file management system manages independent files, helping to enter and retrieve information records. File manager establishes and maintains the list of structure and indexes defined in the internal schema. The file manager can:

- Create a file
- Delete a file
- Update the record in the file
- Retrieve a record from a file

Buffer

The area into which a block from the file is read is termed a buffer. The management of buffers has the objective of maximizing the performance or the utilization of the secondary storage systems, while at the same time keeping the demand on CPU resources tolerably low. The use of two or more buffers for a file allows the transfer of data to be overlapped with the processing of data.

Buffer Manager

Buffer manager is responsible for fetching data from disk storage into main memory. Programs call on the buffer manager when they need a block from disk. The requesting program is given the address of the block in main memory, if it is already present in the buffer. If the block is not in the buffer, the buffer manager allocates space in the buffer for the block, replacing some other block, if required, to make space for new block. Once space is allocated in the buffer, the buffer manager reads in the block from the disk to the buffer, and passes the address of the block in main memory to the requester.

Indices

Indices provide fast access to data items that hold particular values. An index is a list of numerical values which gives the order of the records when they are sorted on a particular field or column of the table.

1.15 Database Architecture

Database architecture essentially describes the location of all the pieces of information that make up the database application. The database architecture can be broadly classified into two-, three-, and multitier architecture.

1.15.1 Two-Tier Architecture

The two-tier architecture is a client–server architecture in which the client contains the presentation code and the SQL statements for data access. The database server processes the SQL statements and sends query results back to the client. The two-tier architecture is shown in Fig. 1.9. Two-tier client/server provides a basic separation of tasks. The client, or first tier, is primarily responsible for the *presentation* of data to the user and the “server,” or second tier, is primarily responsible for supplying *data services* to the client.

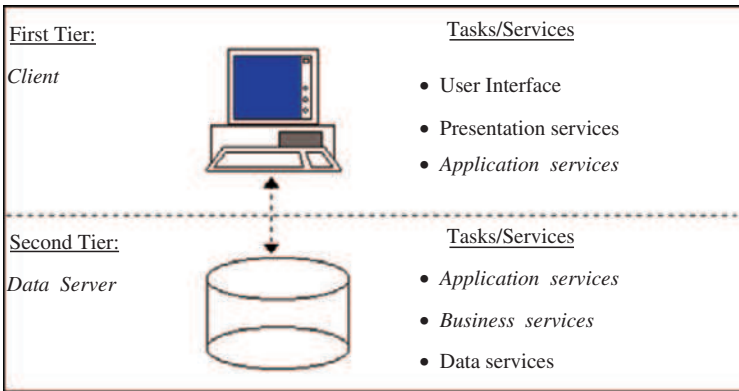


Fig. 1.9. Two-tier client–server architecture

Presentation Services

“Presentation services” refers to the portion of the application which presents data to the user. In addition, it also provides for the mechanisms in which the user will interact with the data. More simply put, presentation logic defines and interacts with the user interface. The presentation of the data should generally not contain any validation rules.

Business Services/objects

“Business services” are a category of application services. Business services encapsulate an organizations business processes and requirements. These rules are derived from the steps necessary to carry out day-today business in an organization. These rules can be validation rules, used to be sure that the incoming information is of a valid type and format, or they can be process rules, which ensure that the proper business process is followed in order to complete an operation.

Application Services

“Application services” provide other functions necessary for the application.

Data Services

“Data services” provide access to data independent of their location. The data can come from legacy mainframe, SQL RDBMS, or proprietary data access systems. Once again, the data services provide a standard interface for accessing data.

Advantages of Two-tier Architecture

The two-tier architecture is a good approach for systems with stable requirements and a moderate number of clients. The two-tier architecture is the simplest to implement, due to the number of good commercial development environments.

Drawbacks of Two-tier Architecture

Software maintenance can be difficult because PC clients contain a mixture of presentation, validation, and business logic code. To make a significant change in the business logic, code must be modified on many PC clients. Moreover the performance of two-tier architecture can be poor when a large number of clients submit requests because the database server may be overwhelmed with managing messages. With a large number of simultaneous clients, three-tier architecture may be necessary.

1.15.2 Three-tier Architecture

A “Multitier,” often referred to as “three-tier” or “ N -tier,” architecture provides greater application scalability, lower maintenance, and increased reuse of components. Three-tier architecture offers a technology neutral method of building client/server applications with vendors who employ standard interfaces which provide services for each logical “tier.” The three-tier architecture is shown in Fig. 1.10. From this figure, it is clear that in order to improve the performance a second-tier is included between the client and the server.

Through standard tiered interfaces, services are made available to the application. A single application can employ many different services which may reside on dissimilar platforms or are developed and maintained with different tools. This approach allows a developer to leverage investments in existing systems while creating new application which can utilize existing resources.

Although the three-tier architecture addresses performance degradations of the two-tier architecture, it does not address division-of-processing concerns. The PC clients and the database server still contain the same division of code although the tasks of the database server are reduced. Multiple-tier architectures provide more flexibility on division of processing.

1.15.3 Multitier Architecture

A multi-tier, three-tier, or N -tier implementation employs a three-tier logical architecture superimposed on a distributed physical model. Application Servers can access other application servers in order to supply services to the client application as well as to other Application Servers. The multiple-tier architecture is the most general client–server architecture. It can be most difficult to implement because of its generality. However, a good design and

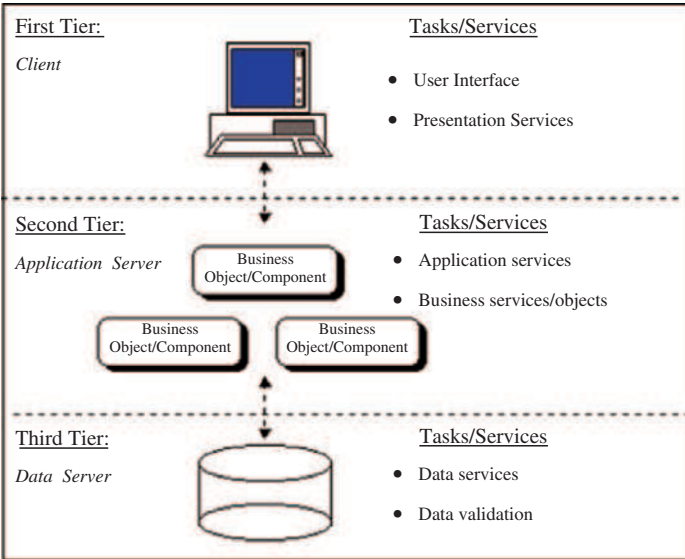


Fig. 1.10. Three-tier client-server architecture

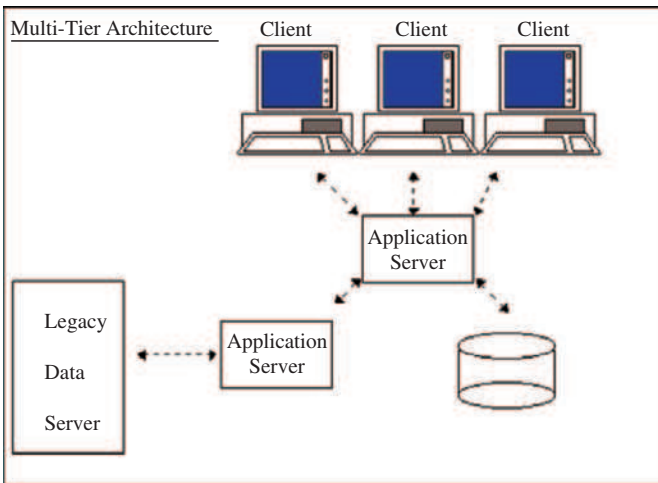


Fig. 1.11. Multiple-tier architecture

implementation of multiple-tier architecture can provide the most benefits in terms of scalability, interoperability, and flexibility.

For example, in the diagram shown in Fig.1.11, the client application looks to *Application Server #1* to supply data from a mainframe-based application. *Application Server #1* has no direct access to the mainframe application, but it does know, through the development of application services, that

Application Server #2 provides a service to access the data from the main-frame application which satisfies the client request. Application Server #1 then invokes the appropriate service on Application Server #2 and receives the requested data which is then passed on to the client.

Application Servers can take many forms. An Application Server may be anything from custom application services, Transaction Processing Monitors, Database Middleware, Message Queue to a CORBA/COM based solution.

1.16 Situations where DBMS is not Necessary

It is also necessary to specify situations where it is not necessary to use a DBMS. If traditional file processing system is working well, and if it takes more money and time to design a database, it is better not to go for the DBMS. Moreover if only one person maintains the data and that person is not skilled in designing a database as well as not comfortable in using the DBMS then it is not advisable to go for DBMS.

DBMS is undesirable under following situations:

- DBMS is undesirable if the application is simple, well-defined, and not expected to change.
- Runtime overheads are not feasible because of real-time requirements.
- Multiple accesses to data are not required.

Compared with file systems, databases have some disadvantages:

1. High cost of DBMS which includes:
 - Higher hardware costs
 - Higher programming costs
 - High conversion costs
2. Slower processing of some applications
3. Increased vulnerability
4. More difficult recovery

1.17 DBMS Vendors and their Products

Some of the popular DBMS vendors and their corresponding products are given Table 1.1.

Summary

The main objective of database management system is to store and manipulate the data in an efficient manner. A database is an organized collection of related data. All the data will not give useful information. Only processed data gives useful information, which helps an organization to take important

Table 1.1. DBMS vendors and their products

vendor	product
IBM	-DB2/MVS
	-DB2/UDB
	-DB2/400
	-Informix Dynamic Server (IDS)
Microsoft	-Access
	-SQL Server
	-DesktopEdition(MSDE)
Open Source	-MySQL
	-PostgreSQL
Oracle	-Oracle DBMS
	-RDB
Sybase	-Adaptive Server Enterprise (ASE)
	-Adaptive Server Anywhere (ASA)
	-Watcom

decisions. Before DBMS, computer file processing systems were used to store, manipulate, and retrieve large files of data. Computer file processing systems have limitations such as data duplications, limited data sharing, and no program data independence. In order to overcome these limitations database approach was developed. The main advantages of DBMS approach are program-data independence, improved data sharing, and minimal data redundancy. In this chapter we have seen the evolution of DBMS and broad introduction to DBMS. The responsibilities of Database administrator, ANSI/SPARK, two-tier, three-tier architecture were analyzed in this chapter.

Review Questions

1.1. What are the drawbacks of file processing system?

The drawbacks of file processing system are:

- Duplication of data, which leads to wastage of storage space and data inconsistency.
- Separation and isolation of data, because of which data cannot be used together.
- No program data independence.

1.2. What is meant by Metadata?

Metadata are data about data but not the actual data.

1.3. Define the term data dictionary?

Data dictionary is a file that contains Metadata.

1.4. What are the responsibilities of database administrator?**1.5.** Mention three situations where it is not desirable to use DBMS?

The situations where it is not desirable to use DBMS are:

- The database and applications are not expected to change.
- Data are not accessed by multiple users.

1.6. What is meant by data independence?

Data independence renders application programs (e.g., SQL scripts) immune to changes in the logical and physical organization of data in the system.

Logical organization refers to changes in the Schema. Example adding a column or tuples does not stop queries from working.

Physical organization refers to changes in indices, file organizations, etc.

1.7. What is meant by Physical and Logical data independence?

In logical data independence, the conceptual schema can be changed without changing the external schema. In physical data independence, the internal schema can be changed without changing the conceptual schema.

1.8. What are some disadvantages of using a DBMS over flat file system?

- DBMS initially costs more than flat file system
- DBMS requires skilled staff

1.9. What are the steps to design a good database?

- First find out the requirements of the user
- Design a view for each important application
- Integrate the views giving the conceptual schema, which is the union of all views
- Map to the data model provided by the DBMS (usually relational)
- Design external views
- Choose physical structures (indexes, etc.)

1.10. What is Database? Give an example.

A Database is a collection of related data. Here, the term “data” means that known facts that can be record. Examples of database are library information system, bus, railway, and airline reservation system, etc.

1.11. Define – DBMS.

DBMS is a collection of programs that enables users to create and maintain a database.

1.12. Mention various types of databases?

The different types of databases are:

- Multimedia database
- Spatial database (Geographical Information System Database)
- Real-time or Active Database
- Data Warehouse or On-line Analytical Processing Database

1.13. Mention the advantages of using DBMS?

The advantages of using DBMS are:

- Controlling Redundancy
- Enforcing Integrity Constraints so as to maintain the consistency of the database
- Providing Backup and recovery facilities
- Restricting unauthorized access
- Providing multiple user interfaces
- Providing persistent storage of program objects and datastructures

1.14. What is “Snapshot” or “Database State”?

The data in the database at a particular moment is known as “Database State” or “Snapshot” of the Database.

1.15. Define Data Model.

It is a collection of concepts that can be used to describe the structure of a database.

The datamodel provides necessary means to achieve the abstraction i.e., hiding the details of data storage.

1.16. Mention the various categories of Data Model.

The various categories of datamodel are:

- High Level or Conceptual Data Model (Example: ER model)
- Low Level or Physical Data Model
- Representational or Implementational Data Model
- Relational Data Model
- Network and Hierarchal Data Model
- Record-based Data Model
- Object-based Data Model

1.17. Define the concept of “database schema.” Describe the types of schemas that exist in a database complying with the three levels ANSI/SPARC architecture.

Database schema is nothing but description of the database. The types of schemas that exist in a database complying with three levels of ANSI/SPARC architecture are:

- External schema
- Conceptual schema
- Internal schema

Entity–Relationship Model

Learning Objectives. This chapter presents a top-down approach to data modeling. This chapter deals with ER and Enhanced ER (EER) model and conversion of ER model to relational model. After completing this chapter the reader should be familiar with the following concepts:

- Entity, Attribute, and Relationship.
- Entity classification – Strong entity, Weak entity, and Associative entity.
- Attribute classification – Single value, Multivalued, Derived, and Null attribute.
- Relationship – Unary, binary, and ternary relationship.
- Enhanced ER model – Generalization, Specialization.
- Mapping ER model to relation model or table.
- Connection traps.

2.1 Introduction

Peter Chen first proposed modeling databases using a graphical technique that humans can relate to easily. Humans can easily perceive entities and their characteristics in the real world and represent any relationship with one another. The objective of modeling graphically is even more profound than simply representing these entities and relationship. The database designer can use tools to model these entities and their relationships and then generate database vendor-specific schema automatically. Entity–Relationship (ER) model gives the conceptual model of the world to be represented in the database. ER Model is based on a perception of a real world that consists of collection of basic objects called entities and relationships among these objects. The main motivation for defining the ER model is to provide a high level model for conceptual database design, which acts as an intermediate stage prior to mapping the enterprise being modeled onto a conceptual level. The ER model achieves a high degree of data independence which means that the database designer do not have to worry about the physical structure of the database. A database schema in ER model can be pictorially represented by Entity–Relationship diagram.

2.2 The Building Blocks of an Entity–Relationship Diagram

ER diagram is a graphical modeling tool to standardize ER modeling. The modeling can be carried out with the help of pictorial representation of entities, attributes, and relationships. The basic building blocks of Entity–Relationship diagram are Entity, Attribute and Relationship.

2.2.1 Entity

An entity is an object that exists and is distinguishable from other objects. In other words, the entity can be uniquely identified.

The examples of entities are:

- A particular person, for example Dr. A.P.J. Abdul Kalam is an entity.
- A particular department, for example Electronics and Communication Engineering Department.
- A particular place, for example Coimbatore city can be an entity.

2.2.2 Entity Type

An entity type or entity set is a collection of similar entities. Some examples of entity types are:

- All students in PSG, say STUDENT.
- All courses in PSG, say COURSE.
- All departments in PSG, say DEPARTMENT.

An entity may belong to more than one entity type. For example, a staff working in a particular department can pursue higher education as part-time. Hence the same person is a LECTURER at one instance and STUDENT at another instance.

2.2.3 Relationship

A relationship is an association of entities where the association includes one entity from each participating entity type whereas relationship type is a meaningful association between entity types.

The examples of relationship types are:

- Teaches is the relationship type between LECTURER and STUDENT.
- Buying is the relationship between VENDOR and CUSTOMER.
- Treatment is the relationship between DOCTOR and PATIENT.

2.2.4 Attributes

Attributes are properties of entity types. In other words, entities are described in a database by a set of attributes.

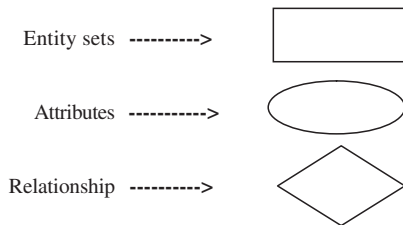
The following are example of attributes:

- Brand, cost, and weight are the attributes of CELLPHONE.
- Roll number, name, and grade are the attributes of STUDENT.
- Data bus width, address bus width, and clock speed are the attributes of MICROPROCESSOR.

2.2.5 ER Diagram

The ER diagram is used to represent database schema. In ER diagram:

- A rectangle represents an entity set.
- An ellipse represents an attribute.
- A diamond represents a relationship.
- Lines represent linking of attributes to entity sets and of entity sets to relationship sets.



Example of ER diagram

Let us consider a simple ER diagram as shown in Fig. 2.1.

In the ER diagram the two entities are STUDENT and CLASS. Two simple attributes which are associated with the STUDENT are Roll number and the name. The attributes associated with the entity CLASS are Subject Name and Hall Number. The relationship between the two entities STUDENT and CLASS is Attends.

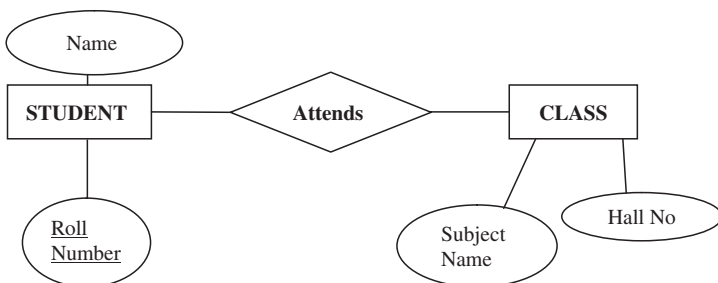
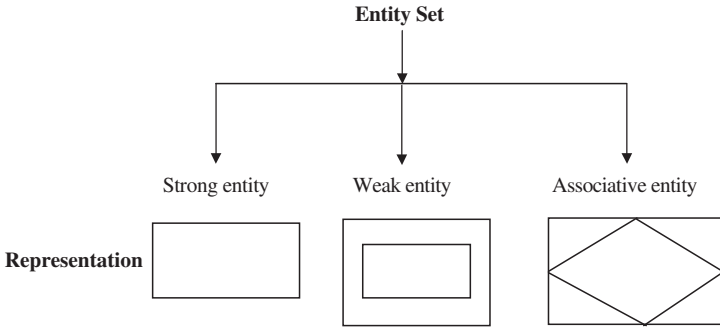


Fig. 2.1. ER diagram

2.3 Classification of Entity Sets

Entity sets can be broadly classified into:

1. Strong entity.
2. Weak entity.
3. Associative entity.

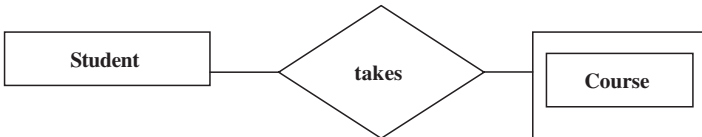


2.3.1 Strong Entity

Strong entity is one whose existence does not depend on other entity.

Example

Consider the example, student takes course. Here student is a strong entity.



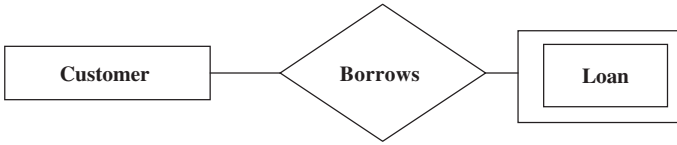
In this example, course is considered as weak entity because, if there are no students to take a particular course, then that course cannot be offered. The COURSE entity depends on the STUDENT entity.

2.3.2 Weak Entity

Weak entity is one whose existence depends on other entity. In many cases, weak entity does not have primary key.

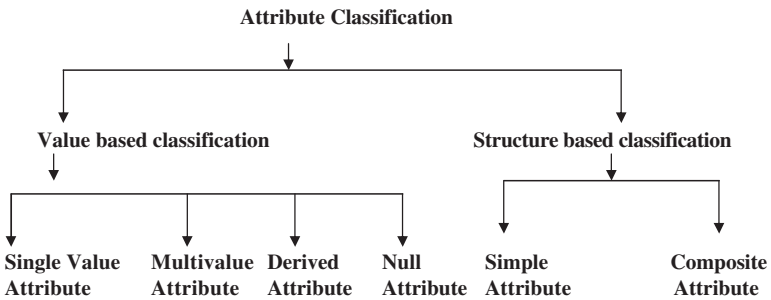
Example

Consider the example, customer borrows loan. Here loan is a weak entity. For every loan, there should be at least one customer. Here the entity loan depends on the entity customer hence loan is a weak entity.



2.4 Attribute Classification

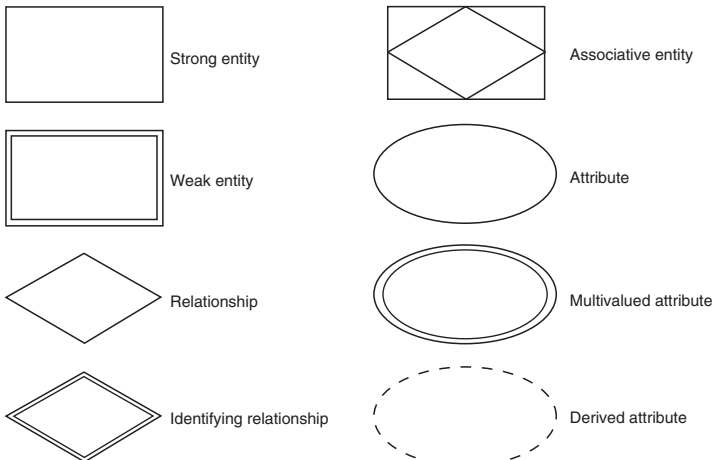
Attribute is used to describe the properties of the entity. This attribute can be broadly classified based on value and structure. Based on value the attribute can be classified into single value, multivalued, derived, and null value attribute. Based on structure, the attribute can be classified as simple and composite attribute.



2.4.1 Symbols Used in ER Diagram

The elements in ER diagram are Entity, Attribute, and Relationship. The different types of entities like strong, weak, and associative entity, different types of attributes like multivalued and derived attributes and identifying relationship and their corresponding symbols are shown later.

Basic symbols




Single Value Attribute

Single value attribute means, there is only one value associated with that attribute.

Example

The examples of single value attribute are age of a person, Roll number of the student, Registration number of a car, etc.

Representation of Single Value Attribute in ER Diagram 

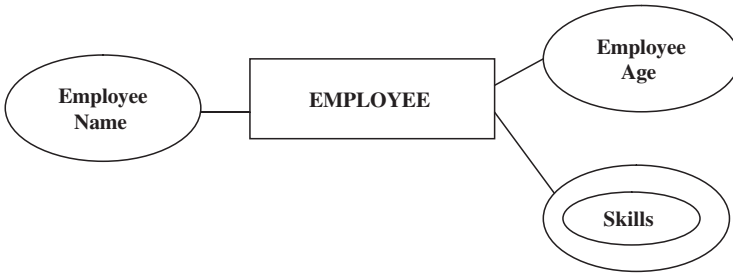
Multivalued Attribute

In the case of multivalued attribute, more than one value will be associated with that attribute.

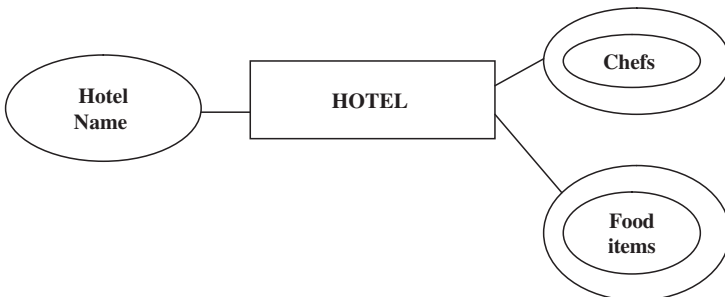
Representation of Multivalued Attribute in ER Diagram 

Examples of Multivalued Attribute

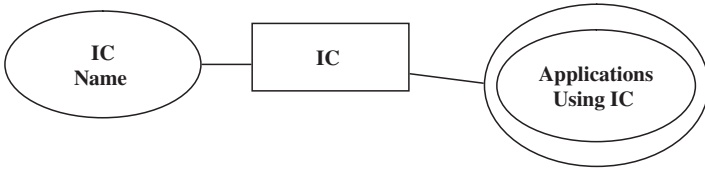
1. Consider an entity EMPLOYEE. An Employee can have many skills; hence skills associated to an employee are a multivalued attribute.



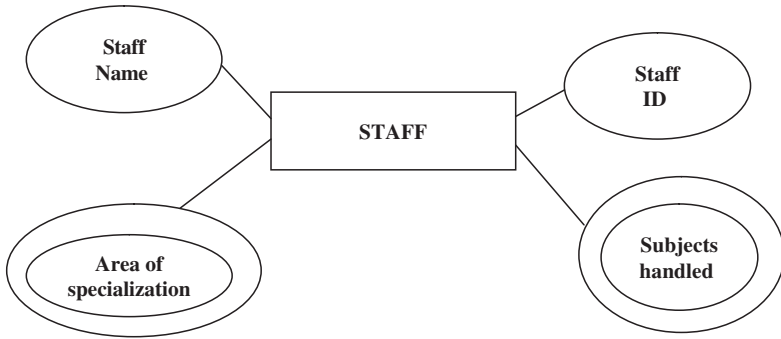
2. Number of chefs in a hotel is an example of multivalued attribute. Moreover, a hotel will have variety of food items. Hence food items associated with the entity HOTEL is an example of multivalued attribute.



- Application associated with an IC (Integrated Circuit). An IC can be used for several applications. Here IC stands for Integrated Circuit.



- Subjects handled by a staff. A staff can handle more than one subject in a particular semester; hence it is an example of multivalued attribute.



Moreover a staff can be an expert in more than one area, hence area of specialization is considered as multivalued attribute.

Derived Attribute

The value of the derived attribute can be derived from the values of other related attributes or entities.

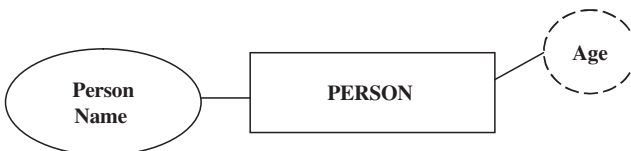
In ER diagram, the derived attribute is represented by dotted ellipse.

Representation of Derived Attribute in ER Diagram

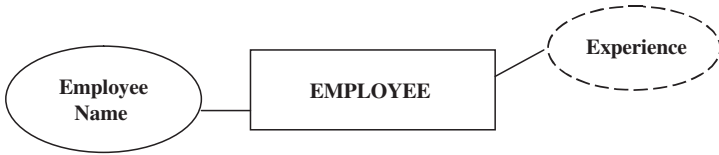


Example of Derived Attribute

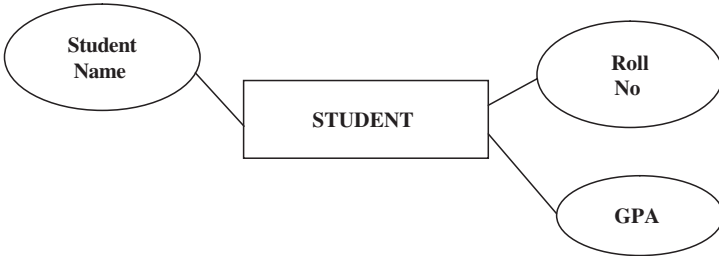
- Age of a person can be derived from the date of birth of the person. In this example, age is the derived attribute.



- Experience of an employee in an organization can be derived from date of joining of the employee.

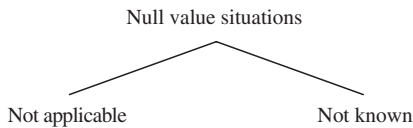


- CGPA of a student can be derived from GPA (Grade Point Average).



Null Value Attribute

In some cases, a particular entity may not have any applicable value for an attribute. For such situation, a special value called null value is created.



Example

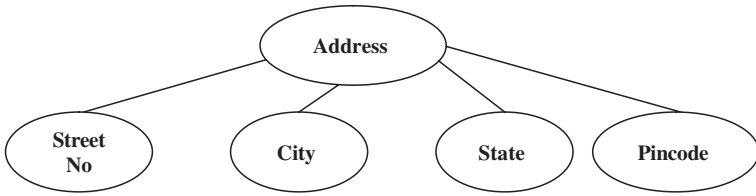
In application forms, there is one column called phone no. if a person do not have phone then a null value is entered in that column.

Composite Attribute

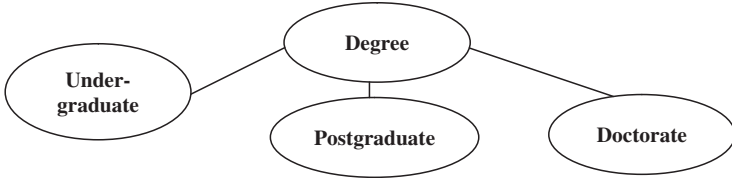
Composite attribute is one which can be further subdivided into simple attributes.

Example

Consider the attribute “address” which can be further subdivided into Street name, City, and State.



As another example of composite attribute consider the degrees earned by a particular scholar, which can range from undergraduate, postgraduate, doctorate degree, etc. Hence degree can be considered as composite attribute.

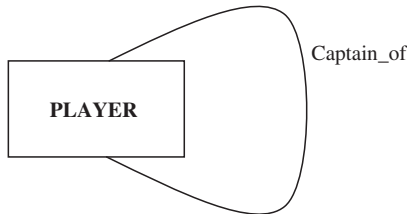


2.5 Relationship Degree

Relationship degree refers to the number of associated entities. The relationship degree can be broadly classified into unary, binary, and ternary relationship.

2.5.1 Unary Relationship

The unary relationship is otherwise known as recursive relationship. In the unary relationship the number of associated entity is one. An entity related to itself is known as recursive relationship.

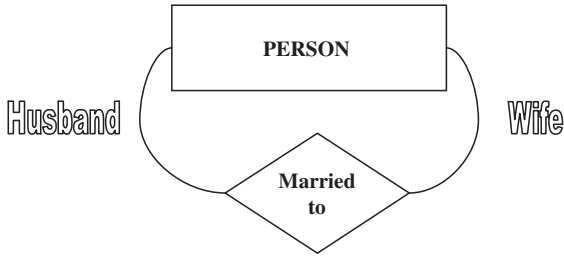


Roles and Recursive Relation

When an entity sets appear in more than one relationship, it is useful to add labels to connecting lines. These labels are called as roles.

Example

In this example, Husband and wife are referred as roles.



2.5.2 Binary Relationship

In a binary relationship, two entities are involved. Consider the example; each staff will be assigned to a particular department. Here the two entities are STAFF and DEPARTMENT.

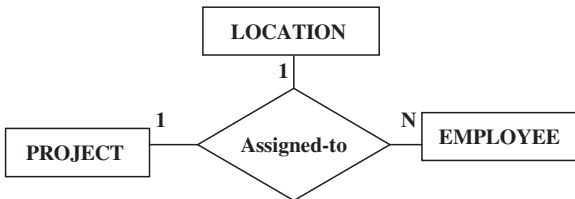


2.5.3 Ternary Relationship

In a ternary relationship, three entities are simultaneously involved. Ternary relationships are required when binary relationships are not sufficient to accurately describe the semantics of an association among three entities.

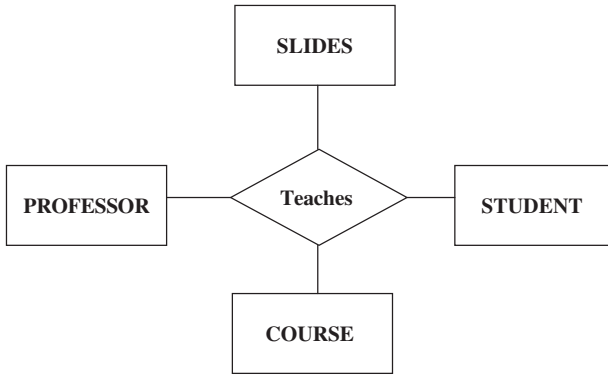
Example

Consider the example of employee assigned a project. Here we are considering three entities EMPLOYEE, PROJECT, and LOCATION. The relationship is “assigned-to.” Many employees will be assigned to one project hence it is an example of one-to-many relationship.



2.5.4 Quaternary Relationships

Quaternary relationships involve four entities. The example of quaternary relationship is “A professor teaches a course to students using slides.” Here the four entities are PROFESSOR, SLIDES, COURSE, and STUDENT. The relationships between the entities are “Teaches.”



2.6 Relationship Classification

Relationship is an association among one or more entities. This relationship can be broadly classified into one-to-one relation, one-to-many relation, many-to-many relation and recursive relation.

2.6.1 One-to-Many Relationship Type

The relationship that associates one entity to more than one entity is called one-to-many relationship. Example of one-to-many relationship is Country having states. For one country there can be more than one state hence it is an example of one-to-many relationship. Another example of one-to-many relationship is parent-child relationship. For one parent there can be more than one child. Hence it is an example of one-to-many relationship.

2.6.2 One-to-One Relationship Type

One-to-one relationship is a special case of one-to-many relationship. True one-to-one relationship is rare. The relationship between the President and the country is an example of one-to-one relationship. For a particular country there will be only one President. In general, a country will not have more than one President hence the relationship between the country and the President is an example of one-to-one relationship. Another example of one-to-one relationship is House to Location. A house is obviously in only one location.

2.6.3 Many-to-Many Relationship Type

The relationship between EMPLOYEE entity and PROJECT entity is an example of many-to-many relationship. Many employees will be working in many projects hence the relationship between employee and project is many-to-many relationship.

Table 2.1. Relationship types

Relationship type	Representation	Example
One-to-one		
One-to-many		
Many-to-many		
Many-to-one		

2.6.4 Many-to-One Relationship Type

The relationship between EMPLOYEE and DEPARTMENT is an example of many-to-one relationship. There may be many EMPLOYEES working in one DEPARTMENT. Hence relationship between EMPLOYEE and DEPARTMENT is many-to-one relationship. The four relationship types are summarized and shown in Table 2.1.

2.7 Reducing ER Diagram to Tables

To implement the database, it is necessary to use the relational model. There is a simple way of mapping from ER model to the relational model. There is almost one-to-one correspondence between ER constructs and the relational ones.

2.7.1 Mapping Algorithm

The mapping algorithm gives the procedure to map ER diagram to tables. The rules in mapping algorithm are given as:

- For each strong entity type say E, create a new table. The columns of the table are the attribute of the entity type E.
- For each weak entity W that is associated with only one 1–1 identifying owner relationship, identify the table T of the owner entity type. Include as columns of T, all the simple attributes and simple components of the composite attributes of W.
- For each weak entity W that is associated with a 1–N or M–N identifying relationship, or participates in more than one relationship, create a new table T and include as its columns, all the simple attributes and simple components of the composite attributes of W. Also form its primary key by including as a foreign key in R, the primary key of its owner entity.

- For each binary 1–1 relationship type R, identify the tables S and T of the participating entity types. Choose S, preferably the one with total participation. Include as foreign key in S, the primary key of T. Include as columns of S, all the simple attributes and simple components of the composite attributes of R.
- For each binary 1–N relationship type R, identify the table S, which is at N side and T of the participating entities. Include as a foreign key in S, the primary key of T. Also include as columns of S, all the simple attributes and simple components of composite attributes of R.
- For each M–N relationship type R, create a new table T and include as columns of T, all the simple attributes and simple components of composite attributes of R. Include as foreign keys, the primary keys of the participating entity types. Specify as the primary key of T, the list of foreign keys.
- For each multivalued attribute, create a new table T and include as columns of T, the simple attribute or simple components of the attribute A. Include as foreign key, the primary key of the entity or relationship type that has A. Specify as the primary key of T, the foreign key and the columns corresponding to A.

Regular Entity

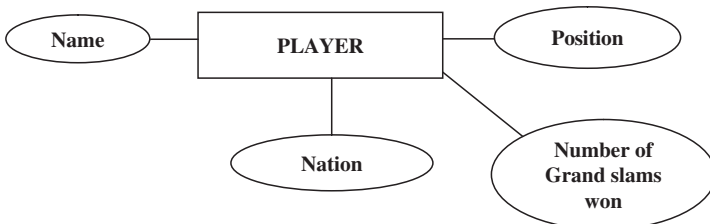
Regular entities are entities that have an independent existence and generally represent real-world objects such as persons and products. Regular entities are represented by rectangles with a single line.

2.7.2 Mapping Regular Entities

- Each regular entity type in an ER diagram is transformed into a relation. The name given to the relation is generally the same as the entity type.
- Each simple attribute of the entity type becomes an attribute of the relation.
- The identifier of the entity type becomes the primary key of the corresponding relation.

Example 1

Mapping regular entity type tennis player



This diagram is converted into corresponding table as

Player Name	Nation	Position	Number of Grand slams won
Roger Federer	Switzerland	1	5
Roddick	USA	2	4

Here,

- **Entity name = Name of the relation or table.**

In our example, the entity name is PLAYER which is the name of the table

- **Attributes of ER diagram = Column name of the table.**

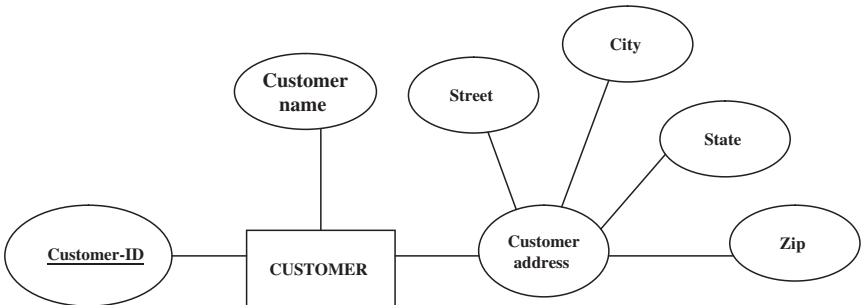
In our example the Name, Nation, Position, and Number of Grand slams won which forms the column of the table.

2.7.3 Converting Composite Attribute in an ER Diagram to Tables

When a regular entity type has a composite attribute, only the simple component attributes of the composite attribute are included in the relation.

Example

In this example the composite attribute is the Customer address, which consists of Street, City, State, and Zip.



CUSTOMER

<u>Customer-ID</u>	Customer name	Street	City	State	Zip
--------------------	---------------	--------	------	-------	-----

When the regular entity type contains a multivalued attribute, two new relations are created.

The first relation contains all of the attributes of the entity type except the multivalued attribute.

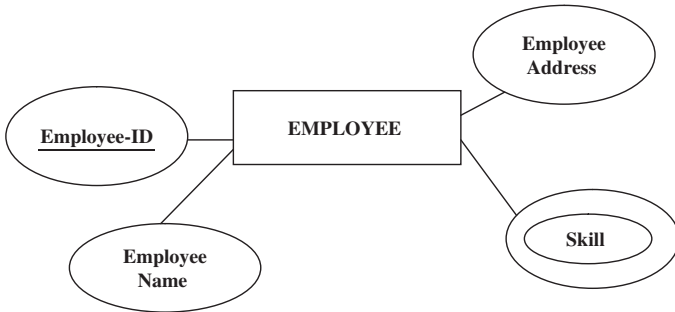
The second relation contains two attributes that form the primary key of the second relation. The first of these attributes is the primary key from the first relation, which becomes a foreign key in the second relation. The second is the multivalued attribute.

2.7.4 Mapping Multivalued Attributes in ER Diagram to Tables

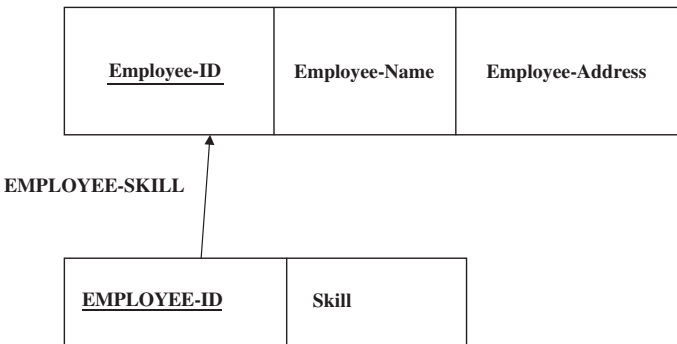
A multivalued attribute is having more than one value. One way to map a multivalued attribute is to create two tables.

Example

In this example, the skill associated with the EMPLOYEE is a multivalued attribute, since an EMPLOYEE can have more than one skill as fitter, electrician, turner, etc.



EMPLOYEE

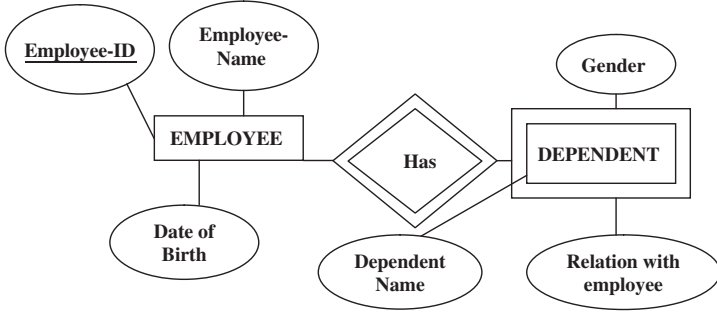


2.7.5 Converting “Weak Entities” in ER Diagram to Tables

Weak entity type does not have an independent existence and it exists only through an identifying relationship with another entity type called the owner.

For each weak entity type, create a new relation and include all of the simple attributes as attributes of the relation. Then include the primary key of the identifying relation as a foreign key attribute to this new relation.

The primary key of the new relation is the combination of the primary key of the identifying and the partial identifier of the weak entity type. In this example DEPENDENT is weak entity.



The corresponding table is given by

EMPLOYEE

<u>Employee-ID</u>	Employee-Name	Date of Birth
--------------------	---------------	---------------

DEPENDENT

Dependent-Name	Gender	<u>Employee-ID</u>	Relation with Employee
----------------	--------	--------------------	------------------------

An arrow points from the Employee-ID attribute in the DEPENDENT table to the Employee-ID attribute in the EMPLOYEE table.

2.7.6 Converting Binary Relationship to Table

A relationship which involves two entities can be termed as binary relationship. This binary relationship can be one-to-one, one-to-many, many-to-one, and many-to-many.

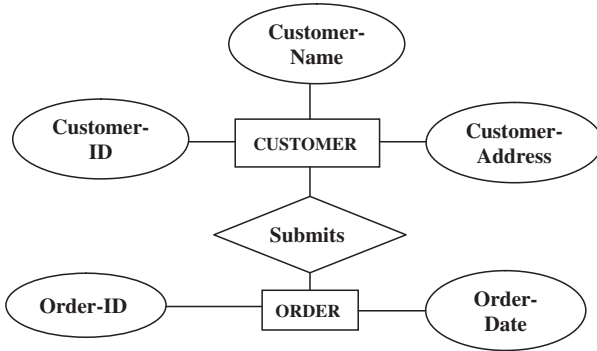
Mapping one-to-Many Relationship

For each 1-M relationship, first create a relation for each of the two entity type's participation in the relationship.

Example

One customer can give many orders. Hence the relationship between the two entities CUSTOMER and ORDER is one-to-many relationship. In one-to-many relationship, include the primary key attribute of the entity on the

one-side of the relationship as a foreign key in the relation that is on the many side of the relationship.



Here we have two entities CUSTOMER and ORDER. The relationship between CUSTOMER and ORDER is one-to-many. For two entities CUSTOMER and ORDER, two tables namely CUSTOMER and ORDER are created as shown later. The primary key CUSTOMER_ID in the CUSTOMER relation becomes the foreign key in the ORDER relation.

CUSTOMER

<u>Customer-ID</u>	Customer-Name	Customer-Address
--------------------	---------------	------------------

ORDER

<u>Order-ID</u>	Order-Date	Customer-ID
-----------------	------------	-------------

Binary one-to-one relationship can be viewed as a special case of one-to-many relationships.

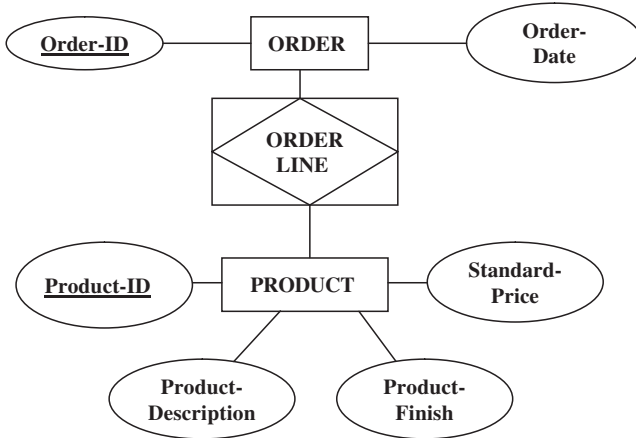
The process of mapping one-to-one relationship requires two steps. First, two relations are created, one for each of the participating entity types. Second, the primary key of one of the relations is included as a foreign key in the other relation.

2.7.7 Mapping Associative Entity to Tables

Many-to-many relationship can be modeled as an associative entity in the ER diagram.

Example 1. (Without Identifier)

Here the associative entity is ORDERLINE, which is without an identifier. That is the associative entity ORDERLINE is without any key attribute.



The first step is to create three relations, one for each of the two participating entity types and the third for the associative entity. The relation formed from the associative entity is associative relation.

ORDER

<u>Order-ID</u>	Order-Date
-----------------	------------

ORDER LINE

<u>Product-ID</u>	<u>Order-Date</u>	Quantity
-------------------	-------------------	----------

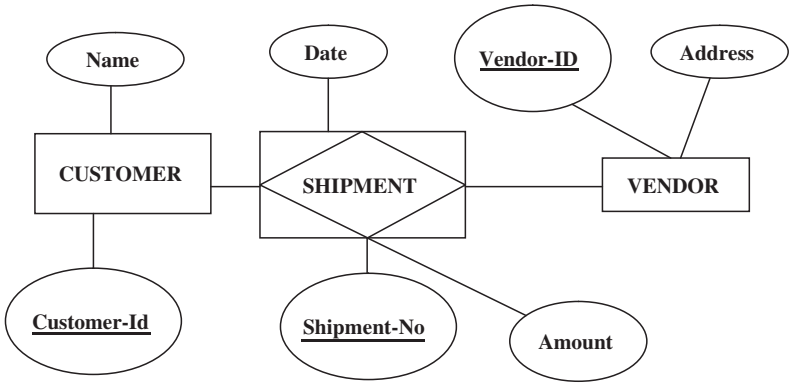
PRODUCT

Product-ID	Product-Description	Product-Finish	Standard-Price
------------	---------------------	----------------	----------------

Example 2. (With Identifier)

Sometimes data models will assign an identifier (surrogate identifier) to the associative entity type on the ER diagram. There are two reasons to motivate this approach:

1. The associative entity type has a natural identifier that is familiar to end user.
2. The default identifier may not uniquely identify instances of the associative entity.



- (a) Shipment-No is a natural identifier to end user.
 (b) The default identifier consisting of the combination of Customer-ID and Vendor-ID does not uniquely identify the instances of SHIPMENT.

CUSTOMER

<u>Customer-ID</u>	Name	Other Attributes
--------------------	------	------------------

SHIPMENT

<u>Shipment-No</u>	Customer-ID	Vendor-ID	Date	Amount
--------------------	-------------	-----------	------	--------

VENDOR

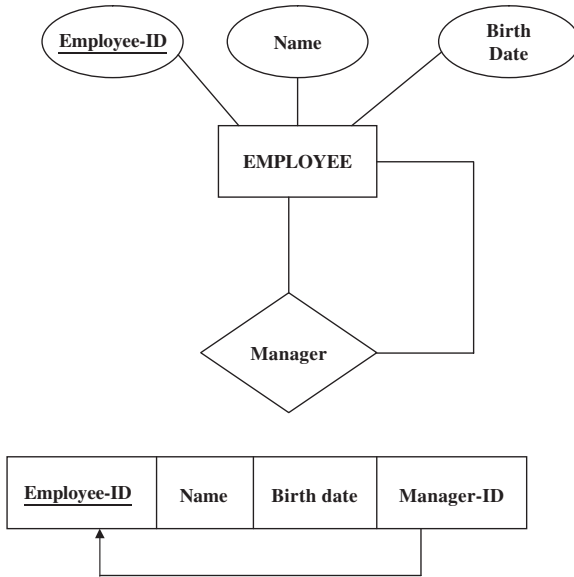
<u>Vendor-ID</u>	Address	Other Attributes
------------------	---------	------------------

2.7.8 Converting Unary Relationship to Tables

Unary relationships are also called recursive relationships. The two most important cases of unary relationship are one-to-many and many-to-many.

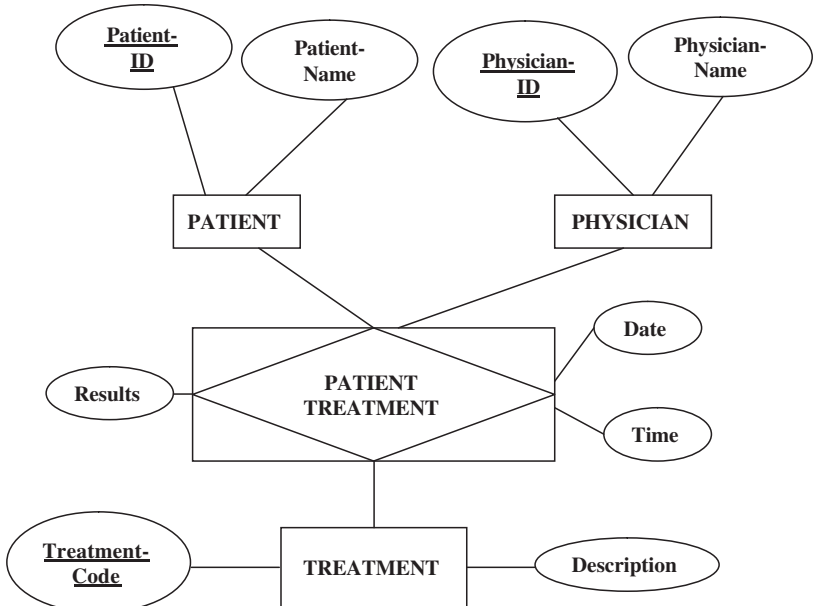
One-to-many Unary Relationship

Each employee has exactly one manager. A given employee may manage zero to many employees. The foreign key in the relation is named Manager-ID. This attribute has the same domain as the primary key Employee-ID.



2.7.9 Converting Ternary Relationship to Tables

A ternary relationship is a relationship among three entity types. The three entities given in this example are PATIENT, PHYSICIAN, and TREATMENT. The PATIENT-TREATMENT is an associative entity.



The primary key attributes – Patient ID, Physician ID, and Treatment Code – become foreign keys in PATIENT TREATMENT. These attributes are components of the primary key of PATIENT TREATMENT.

PATIENT TREATMENT

<u>Patient-ID</u>	Patient-Name
-------------------	--------------

PHYSICIAN

<u>Physician-ID</u>	Physician-Name
---------------------	----------------

PATIENT TREATMENT

<u>Patient-ID</u>	<u>Physician-ID</u>	<u>Treatment-Code</u>	<u>Date</u>	<u>Time</u>	Results
-------------------	---------------------	-----------------------	-------------	-------------	---------

TREATMENT

<u>Treatment-Code</u>	Description
-----------------------	-------------

2.8 Enhanced Entity–Relationship Model (EER Model)

The basic concepts of ER modeling are not powerful enough for some complex applications. Hence some additional semantic modeling concepts are required, which are being provided by Enhanced ER model. The Enhanced ER model is the extension of the original ER model with new modeling constructs. The new modeling constructs introduced in the EER model are supertype (superclass)/subtype (subclass) relationships. The supertype allows us to model general entity type whereas the subtype allows us to model specialized entity types.

Enhanced ER model = ER model + hierarchical relationships.

EER modeling is especially useful when the domain being modeled is object-oriented in nature and the use of inheritance reduces the complexity of the design. The extended ER model extends the ER model to allow various types of abstraction to be included and to express constraints more clearly.

2.8.1 Supertype or Superclass

Supertype or superclass is a generic entity type that has a relationship with one or more subtypes. For example PLAYER is a generic entity type which has

a relationship with one or more subtypes like CRICKET PLAYER, FOOTBALL PLAYER, HOCKEY PLAYER, TENNIS PLAYER, etc.

2.8.2 Subtype or Subclass

A subtype or subclass is a subgrouping of the entities in an entity type that is meaningful to the organization. A subclass entity type is a specialized type of superclass entity type. A subclass entity type represents a subset or subgrouping of superclass entity type’s instances. Subtypes inherit the attributes and relationships associated with their supertype.

Consider the entity type ENGINE, which has two subtypes PETROL ENGINE and DIESEL ENGINE.

Consider the entity type STUDENT, which has two subtypes UNDERGRADUATE and POSTGRADUATE.

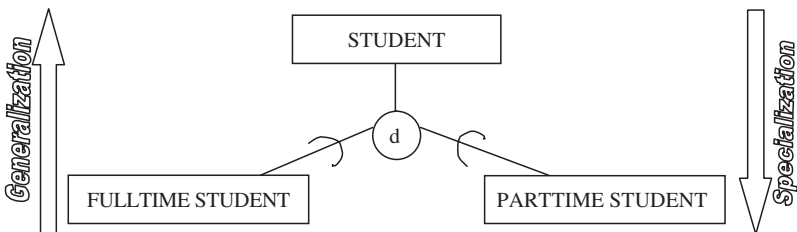
2.9 Generalization and Specialization

Generalization and specialization are two words for the same concept, viewed from two opposite directions. Generalization is the bottom-up process of defining a generalized entity type from a set of more specialized entity types. Specialization is the top-down process of defining one or more subtypes of a supertype.

Generalization is the process of minimizing the differences between entities by identifying common features. It can also be defined as the process of defining a generalized entity type from a set of entity types.

Specialization is a process of identifying subsets of an entity set (the superset) that share some distinguishing characteristics. In specialization the superclass is defined first and the subclasses are defined next. Specialization is the process of viewing an object as a more refined, specialized object. Specialization emphasizes the differences between objects.

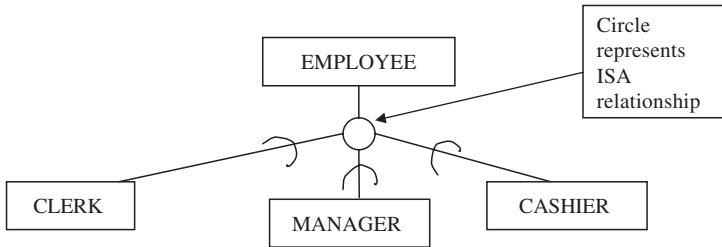
For example consider the entity type STUDENT, which can be further classified into FULLTIME STUDENT and PARTTIME STUDENT. The classification of STUDENT into FULLTIME STUDENT and PARTTIME STUDENT is called Specialization.



2.10 ISA Relationship and Attribute Inheritance

IS_A relationship supports attribute inheritance and relationship participation. In the EER diagram, the subclass relationship is represented by ISA relationship. Attribute inheritance is the property by which subclass entities inherit values for all attributes of the superclass.

Consider the example of EMPLOYEE entity set in a bank. The EMPLOYEE in a bank can be CLERK, MANAGER, CASHIER, ACCOUNTANT, etc. It is to be observed that the CLERK, MANAGER, CASHIER, ACCOUNTANT inherit some of the attributes of the EMPLOYEE.



In this example the superclass is EMPLOYEE and the subclasses are CLERK, MANAGER, and CASHIER. The subclasses inherit the attributes of the superclass. Since each member of the subclass is an ISA member of the superclass, the circle below the EMPLOYEE entity set represents ISA relationship.

2.11 Multiple Inheritance

A subclass with more than one superclass is called a shared subclass. A subclass inherits attributes not only of its direct superclass, but also of all its predecessor superclass, that is it has multiple inheritance from its superclasses. In multiple inheritance a subclass can be subclass of more than one superclass.

Example of Multiple Inheritance

Consider a person in an educational institution. The person can be employee, alumnus, and student. The employee entity can be staff or faculty. The student can be a graduate student or a postgraduate student. The postgraduate student can be a teaching assistant. If the postgraduate student is a teaching assistant, then he/she inherits the characteristics of the faculty as well as student class. That is the teaching assistant subclass is a subclass of more than one superclass (faculty, student). This phenomenon is called multiple inheritance and is shown in the Fig. 2.2.

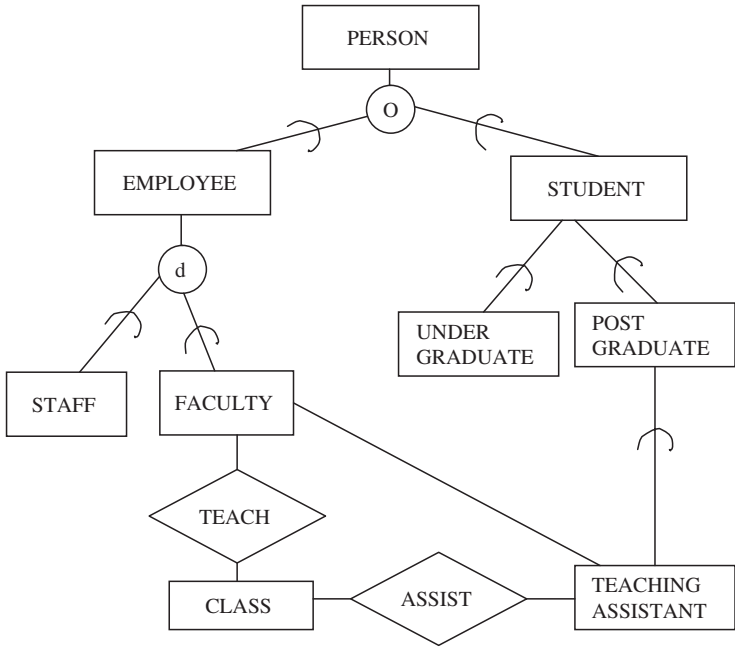


Fig. 2.2. Multiple inheritance

2.12 Constraints on Specialization and Generalization

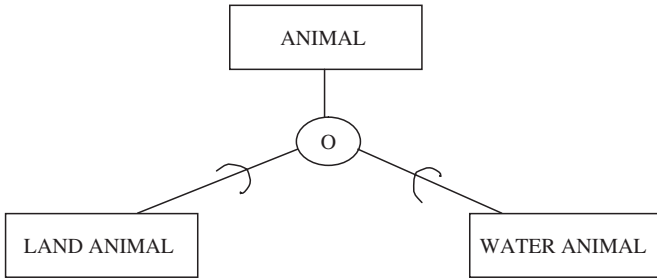
The constraints on specialization and generalization can be broadly classified into disjointness and completeness. The disjointness constraint allows us to specify whether an instance of a supertype may simultaneously be a member of two or more subtypes. In disjointness we have two categories (1) Overlap and (2) Disjoint. In completeness we have two categories (1) Total and (2) Partial. The completeness constraint addresses the question whether an instance of a supertype must also be a member of at least one subtype.

2.12.1 Overlap Constraint

Overlap refers to the fact that the same entity instance may be a member of more than one subclass of the specialization.

Example of Overlap Constraint

Consider the example of ANIMAL entity, which can be further subdivided into LAND ANIMAL and WATER ANIMAL. Consider the example of Frog and Crocodile which can live in both land and water hence the division of ANIMAL into LAND and WATER animals is an example of overlap constraint.

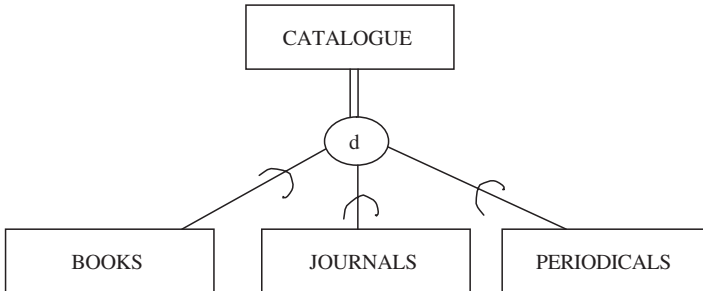


2.12.2 Disjoint Constraint

Disjoint refers to the fact that the same entity instance may be a member of only one subclass of the specialization.

Example of Disjointness Constraint

Consider the example of CATALOGUE. The CATALOGUE is a superclass, which can be further subdivided into BOOKS, JOURNALS, and PERIODICALS. This falls under disjointness because a BOOK entity can be neither JOURNAL nor PERIODICAL.

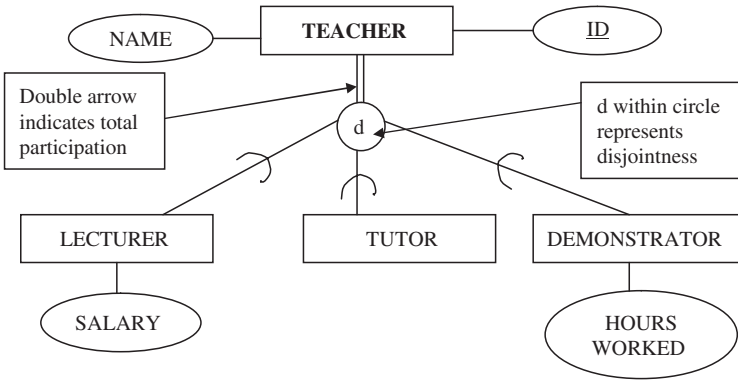


2.12.3 Total Specialization

Total completeness refers to the fact that every entity instance in the superclass must be a member of some subclass in the specialization. With total specialization, an instance of the supertype must be a member of at least one subtype.

Example of Total Specialization

Consider the example of TEACHER; the teacher is a general term, which can be further specialized into LECTURER, TUTOR, and DEMONSTRATOR. Here every member in the superclass participates as a member of a subclass, hence it is an example of total participation.

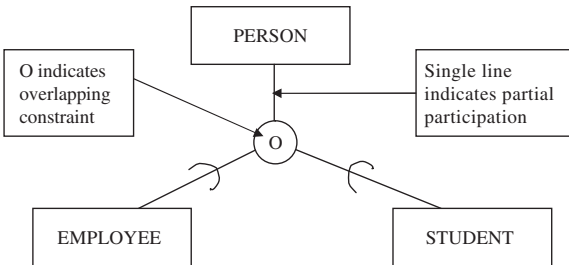


2.12.4 Partial Specialization

Partial completeness refers to the fact that an entity instance in the superclass need not be a member of any subclass in the specialization. With partial specialization, an instance of a supertype may or may not be a member of any subtype.

Example of Partial Specialization

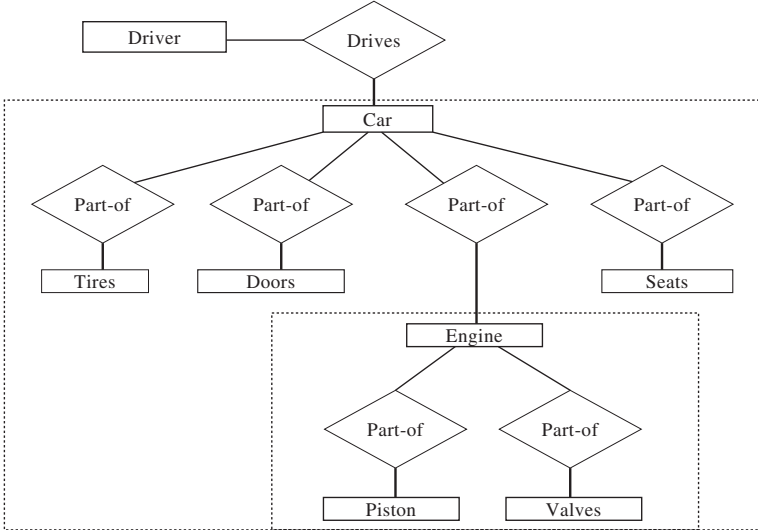
Consider the **PERSON** specialization into **EMPLOYEE** and **STUDENT**. This is an example of partial specialization because there can be a person who is unemployed and does not study.



2.13 Aggregation and Composition

Relationships among relationships are not supported by the ER model. Groups of entities and relationships can be abstracted into higher level entities using aggregation. Aggregation represents a “HAS-A” or “IS-PART-OF” relationship between entity types. One entity type is the whole, the other is the part. Aggregation allows us to indicate that a relationship set participates in another relationship set.

Consider the example of a driver driving a car. The car has various components like tires, doors, engine, seat, etc., which varies from one car to another. Relationship drives is insufficient to model the complexity of this system. *Part-of* relationships allow abstraction into higher level entities. In this example engine, tires, doors, and seats are aggregated into car.



Composition is a stronger form of aggregation where the part cannot exist without its containing whole entity type and the part can only be part of one entity type.

Consider the example of DEPARTMENT has PROJECT. Each project is associated with a particular DEPARTMENT. There cannot be a PROJECT without DEPARTMENT. Hence DEPARTMENT has PROJECT is an example of composition.

2.14 Entity Clusters

EER diagrams are difficult to read when there are many entities and relationships. One possible solution is to group entities and relationships into entity clusters. Entity cluster is a set of one or more entity types and associated relationships grouped into a single abstract entity type. Entity cluster behaves like an entity type; hence entity clusters and entity types can be further grouped to form a higher level entity cluster. Entity clustering is a hierarchical decomposition of a macrolevel view of the data model into finer and finer views, eventually resulting in the full detailed data model.

To understand entity cluster, consider the example of Hospital Management. In hospital, the DOCTORS treat the PATIENT. The DOCTORS are paid by the MANAGEMENT which builds buildings. The DOCTORS can

be either general physician or specialist like those with MS or MD. The patient can be either inpatient or outpatient. It is to be noted that only outpatient will be allotted bed. If we have to represent the earlier ideas, it can be done using EER diagram as shown in Fig. 2.3. The EER diagram is found to be complex; the same idea is represented using Entity Clusters as shown in Fig. 2.4. Here the DOCTOR specialization is clustered into DOCTORS entity and the PATIENT specialization is clustered into simply PATIENT. At the first glance, it may look like reduction of EER model to ER model, but it is not so. Here the entities as well as relationships are clustered into simply entity set.

2.15 Connection Traps

Connection trap is the misinterpretation of the meaning of certain relationships. This connection traps can be broadly classified into fan and chasm trap. Any conceptual model will contain potential connection traps. An error in the interpretation of the meaning of the relationship may cause the database to be incapable of storing certain information. Both the fan and chasm trap arise when the relationships appear to exist between entity types, but the links between occurrences may be ambiguous or not exist. Related groups of entities could become clusters.

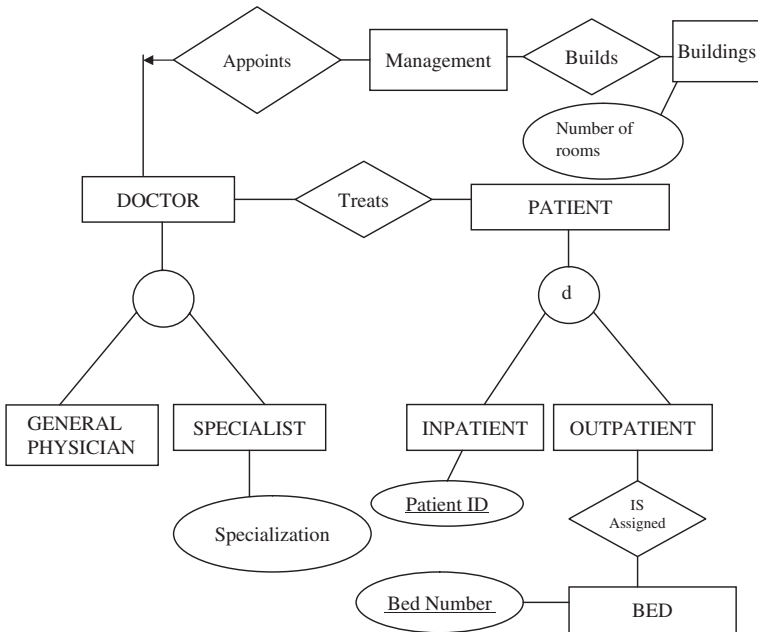


Fig. 2.3. EER diagram of Hospital Management

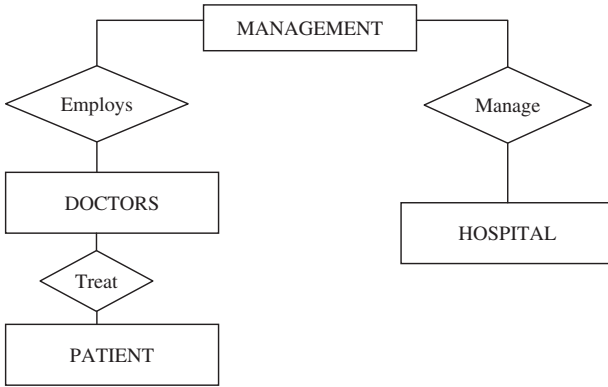


Fig. 2.4. Entity Cluster

2.15.1 Fan Trap

Fan trap occurs when the model represents a relationship between entity types but the pathway between certain entity occurrences is ambiguous. Fan trap occurs when 1–M relationships fan out from a single entity. In order to understand the concept of Fan trap, consider the following example

Contractor works in a team.....Statement (1)
 Team develops projects.....Statement (2)

Statement (1) represents M–1 relationship. Statement (2) represents 1–M relationship. But the information about which contractors are involved in developing which projects is not clear.

Consider another example of Fan trap.

Department is on Site.....Statement (1)
 Site employs Staff.....Statement (2)

Statement (1) represents M–1 relationship, because many departments may be in a single site. Statement (2) represents 1–M relationships. However which staff works in a particular department is ambiguous. The fan trap is resolved by reconstructing the original ER model to represent the correct association.



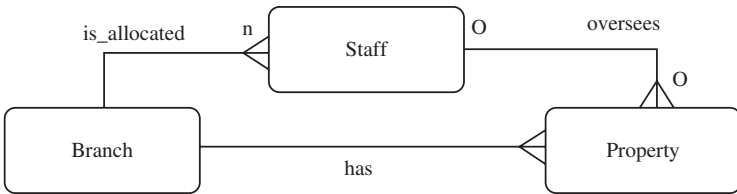
2.15.2 Chasm Trap

A chasm trap occurs when a model suggests the existence of a relationship between entity types, but the pathway does not exist between certain entity

occurrences. It occurs where there is a relationship with partial participation, which forms part of the pathway between entities that are related. Consider the relationship shown later.



A single branch may be allocated to many staff who oversees the management of properties for rent. It should be noted that not all staff oversee property and not all property is managed by a member of staff. Hence there exist a partial participation of Staff and Property in the relation “oversees,” which means that some properties cannot be associated with a branch office through a member of staff. Hence the model has to be modified as shown later.



2.16 Advantages of ER Modeling

An ER model is derived from business specifications. ER models separate the information required by a business from the activities performed within a business. Although business can change their activities, the type of information tends to remain constant. Therefore, the data structures also tend to be constant. The advantages of ER modeling are summarized later:

1. The ER modeling provides an easily understood pictorial map for the database design.
2. It is possible to represent the real world problems in a better manner in ER modeling.
3. The conversion of ER model to relational model is straightforward.
4. The enhanced ER model provides more flexibility in modeling real world problems.
5. The symbols used to represent entity and relationships between entities are simple and easy to follow.

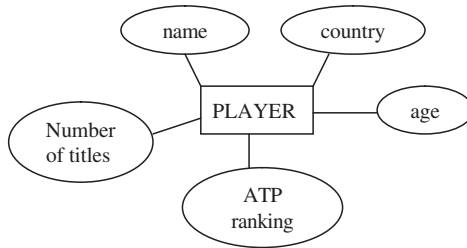
Summary

This chapter has described the fundamentals of ER modeling of data. An ER model is a logical representation of data. The ER model was introduced

by Peter Chen in 1976. An ER model is usually expressed in the form of ER diagram. The basic constructs of ER model are entity types, relationships, and attributes. This chapter also described the types of entities like strong and weak entity, types of relationships like one-to-one, one-to-many, and many-to-many relationship. Attributes can also be classified as single valued, multivalued and derived attribute. In this chapter different types of entities, attributes, and relationship were explained with simple examples.

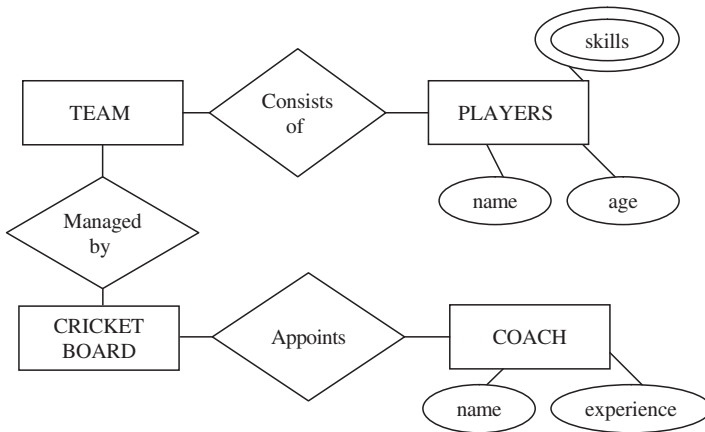
Review Questions

2.1. Construct an ER diagram of tennis player.



2.2. Construct an ER diagram of Indian cricket team.

One way of constructing ER diagram for Indian cricket team is shown later.



Here skills refers to player’s skill which may be batting, bowling, and fielding. All-rounders can have many skills.

2.3. What is Weak entity type?

Entity types that do not have key attribute of their own are called Weak entity type.

2.4. Define entity with example?

An entity is an object with a physical existence.

Examples of entity is a person, a car, an organization, a house, etc.

2.5. Define Entity type, Entity set?

An entity type defines a collection of entities that have same attribute

Entity Set

Entity set is the collection of a particular entity type that are grouped into an “Entity Set.”

2.6. Should a real world object be modeled as an entity or as an attribute?

Object should be an entity if a number of attributes could be associated with it for proper identification and description, either now or later. Object should be an attribute, if it has an atomic nature. For example, Color should be an attribute, unless we identify Color either as a process (e.g., painting) where a number of attributes codes are to be recorded (e.g., type, shade, gray-scale, manufacturer, or as an object with properties (e.g., car-color with details).

2.7. When composite attribute usage is preferred than set of attributes?

Composite attribute is chosen when a meaningful name can be assigned to the set of attributes, e.g., data, address. Otherwise a set of simple attributes should be chosen.

2.8. Distinguish between strong and weak entity?

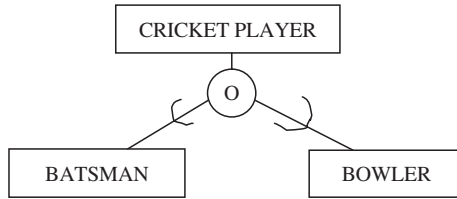
Strong entity	Weak entity
Exists independently of other entities	Dependent on a strong entity, cannot exist on its own
Strong entity has its own unique identifier	Does not have a unique identifier
Represented by a single line rectangle in ER diagram	Represented with a double-line rectangle in ER diagram

2.9. What is inheritance in generalization hierarchies?

Inheritance is a data modeling feature that supports sharing of attributes between a supertype and a subtype. Subtype inherits attributes from their supertype.

2.10. Give an example of supertype/subtype relationship where the overlap rule applies?

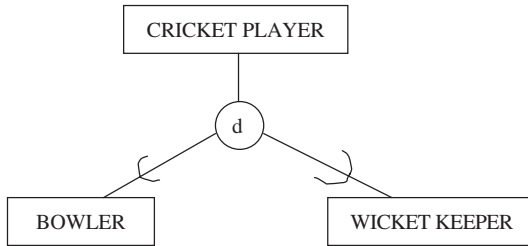
Overlap refers to the fact that the same entity instance may be a member of more than one subclass of the specialization. Consider the example of CRICKET PLAYER. Here CRICKET PLAYER is the supertype. The subtype can be BOWLER, BATSMAN.



Same player can be both batsman and bowler. Hence overlap rule holds good in this example.

2.11. Give an example of supertype/subtype relationship where the disjoint rule applies?

Let us consider the example of CRICKET PLAYER again. Here the super type is CRICKET PLAYER. The subtypes are BOWLER and WICKETKEEPER. We know that the same cricket player cannot be both bowler and wicket keeper hence disjoint rule applies for this example.



II. Match the following

- | | | |
|-----------------|-------|--|
| (1) Relation | _____ | (a) Rows |
| (2) Tuples | _____ | (b) Number of Rows of a Relation |
| (3) Cardinality | _____ | (c) Number of Columns of a Relation |
| (4) Degree | _____ | (d) Columns or Range of values a column may have |
| (5) Domain | _____ | (e) Table |

Answer

- (1) → (e)
 (2) → (a)
 (3) → (b)
 (4) → (c)
 (5) → (d)

Relational Model

Learning Objectives. This chapter is dedicated to relational model which is in use since late 1970s. Various operations in relational algebra and relational calculus are given in this chapter. After completing this chapter the reader should be familiar with the following concepts:

- Evolution and importance of relational model
- Terms in relational model like tuple, domain, cardinality, and degree of a relation
- Operations in relational algebra and relational calculus
- Relational algebra vs relational calculus
- QBE and various operations in QBE

3.1 Introduction

E.F. Codd (Edgar Frank Codd) of IBM had written an article “A relational model for large shared data banks” in June 1970 in the Association of Computer Machinery (ACM) Journal, Communications of the ACM. His work triggered people to work in relational model. One of the most significant implementations of the relational model was “System R,” which was developed by IBM during the late 1970s. System R was intended as a “proof of concept” to show that relational database systems could really build and work efficiently. It gave rise to major developments such as a structured query language called SQL which has since become an ISO standard and de facto standard relational language. Various commercial relational DBMS products were developed during the 1980s such as DB2, SQL/DS, and Oracle. In relational data model the data are stored in the form of tables.

3.2 CODD’S Rules

In 1985, Codd published a list of rules that became a standard way of evaluating a relational system. After publishing the original article Codd stated that there are no systems that will satisfy every rule. Nevertheless the rules represent relational ideal and remain a goal for relational database designers.

Note: The rules are numbered from 1 to 12 whereas the statements preceded by the *bullet mark* are interpretations of the Codd's rule:

1. *The Information Rule.* All information in a relational database is represented explicitly at the logical level and in exactly one way-by values in tables:
 - Data should be presented to the user in the tabular form.
2. *Guaranteed Access Rule.* Each and every datum (atomic value) in a relational database is guaranteed to be logically accessible by resorting to a combination of table name, primary key value, and column name:
 - Every data element should be unambiguously accessible.
3. *Systematic Treatment of Null Values.* Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type.
4. *Dynamic On-line Catalog Based on the Relational Model.* The database description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data:
 - The database description should be accessible to the users.
5. *Comprehensive Data Sublanguage Rule.* A relational system may support several languages and various modes of terminal use (for example the fill-in-the-blanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and whose ability to support all the following is comprehensive: data definition, view definition, data manipulation (interactive and by program), integrity constraints, and transaction boundaries:
 - A database supports a clearly defined language to define the database, view the definition, manipulate the data, and restrict some data values to maintain integrity.
6. *View Updating Rule.* All views that are theoretically updatable are also updatable by the system:
 - Data should be able to be changed through any view available to the user.
7. *High-level Insert, Update, and Delete.* The capacity of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update, and deletion of data:
 - All records in a file must be able to be added, deleted, or updated with singular commands
8. *Physical Data Independence.* Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods:

- Changes in how data are stored or retrieved should not affect how a user accesses the data.
9. *Logical Data Independence.* Application programs and terminal activities remain logically unimpaired whenever information-preserving changes of any kind that theoretically permit unimpairment are made to the base tables:
 - A user's view of data should be unaffected by its actual form in files.
 10. *Integrity Independence.* Integrity constraints specific to a particular relational database must be definable in a relational data sublanguage and storable in the catalog, not in the application programs.
 - Constraints on user input should exist to maintain data integrity.
 11. *Distribution Independence.* A relational DBMS has distribution independence. Distribution independence implies that users should not have to be aware of whether a database is distributed.
 - A database design should allow for distribution of data over several computer sites.
 12. *Nonsubversion Rule.* If a relational system has a low-level (single-record-at-a-time) language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher level relational language (multiple-records-at-a-time):
 - Data fields that affect the organization of the database cannot be changed.

There is one more rule called Rule Zero which states that “For any system that is claimed to be a relational database management system, that system must be able to manage data entirely through capabilities.”

3.3 Relational Data Model

The relational model uses a collection of tables to represent both data and the relationships among those data. Tables are logical structures maintained by the database manager. The relational model is a combination of three components, such as Structural, Integrity, and Manipulative parts.

3.3.1 Structural Part

The structural part defines the database as a collection of relations.

3.3.2 Integrity Part

The database integrity is maintained in the relational model using primary and foreign keys.

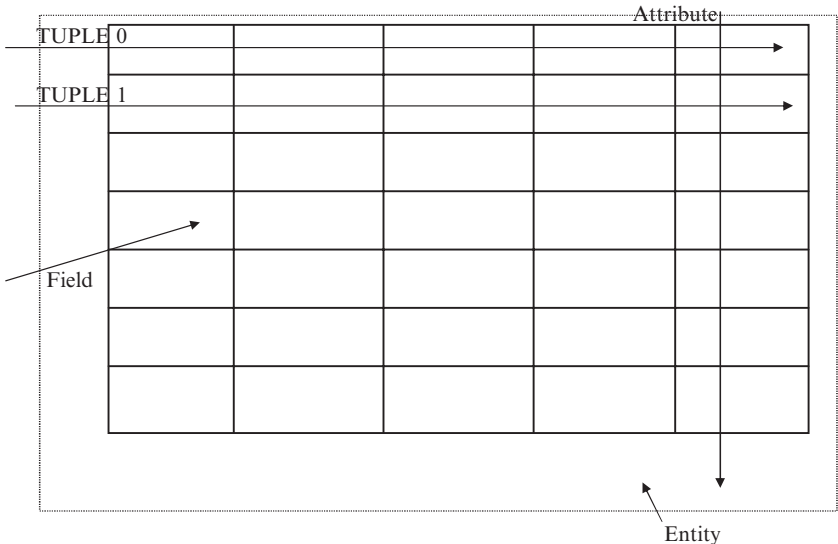
3.3.3 Manipulative Part

The relational algebra and relational calculus are the tools used to manipulate data in the database. Thus relational model has a strong mathematical background. The key features of relational data model are as follows:

- Each row in the table is called tuple.
- Each column in the table is called attribute.
- The intersection of row with the column will have data value.
- In relational model rows can be in any order.
- In relational model attributes can be in any order.
- By definition, all rows in a relation are distinct. No two rows can be exactly the same.
- Relations must have a key. Keys can be a set of attributes.
- For each column of a table there is a set of possible values called its domain. The domain contains all possible values that can appear under that column.
- Domain is the set of valid values for an attribute.
- Degree of the relation is the number of attributes (columns) in the relation.
- Cardinality of the relation is the number of tuples (rows) in the relation.

The terms commonly used by user, model, and programmers are given later.

<u>User</u>	<u>Model</u>	<u>Programmer</u>
Row	Tuple	Record
Column	Attribute	Field
Table	Relation	File



3.3.4 Table and Relation

The general doubt that will rise when one reads the relational model is the difference between table and relation. For a table to be relation, the following rules holds good:

- The intersection row with the column should contain single value (atomic value).
- All entries in a column are of same type.
- Each column has a unique name (column order not significant).
- No two rows are identical (row order not significant).

Example of Relational Model

Representation of Movie data in tabular form is shown later.

MOVIE			
Movie Name	Director	Actor	Actress
Titanic	James Cameron	Leonardo DiCapiro	Kate Winslet
Autograph	Cheran	Cheran	Gopika
Roja	Maniratnam	AravindSwamy	Madubala

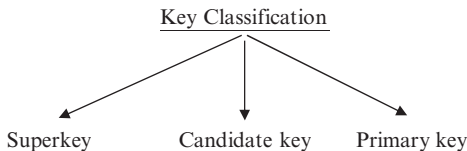
In the earlier relation:

The degree of the relation (i.e., is the number of column in the relation) = 4.

The cardinality of the relation (i.e., the number of rows in the relation) = 3.

3.4 Concept of Key

Key is an attribute or group of attributes, which is used to identify a row in a relation. Key can be broadly classified into (1) Superkey (2) Candidate key, and (3) Primary key



3.4.1 Superkey

A superkey is a subset of attributes of an entity-set that uniquely identifies the entities. Superkeys represent a constraint that prevents two entities from ever having the same value for those attributes.

3.4.2 Candidate Key

Candidate key is a minimal superkey. A candidate key for a relation schema is a minimal set of attributes whose values uniquely identify tuples in the corresponding relation.

Primary Key

The primary key is a designated candidate key. It is to be noted that the primary key should not be null.

Example

Consider the employee relation, which is characterized by the attributes, employee ID, employee name, employee age, employee experience, employee salary, etc. In this employee relation:

Superkeys can be employee ID, employee name, employee age, employee experience, etc.

Candidate keys can be employee ID, employee name, employee age.

Primary key is employee ID.

Note: If we declare a particular attribute as the primary key, then that attribute value cannot be NULL. Also it has to be distinct.

3.4.3 Foreign Key

Foreign key is set of fields or attributes in one relation that is used to “refer” to a tuple in another relation.

3.5 Relational Integrity

Data integrity constraints refer to the accuracy and correctness of data in the database. Data integrity provides a mechanism to maintain data consistency for operations like INSERT, UPDATE, and DELETE. The different types of data integrity constraints are Entity, NULL, Domain, and Referential integrity.

3.5.1 Entity Integrity

Entity integrity implies that a primary key cannot accept null value. The primary key of the relation uniquely identifies a row in a relation. Entity integrity means that in order to represent an entity in the database it is necessary to have a complete identification of the entity’s key attributes.

Consider the entity `PLAYER`; the attributes of the entity `PLAYER` are Name, Age, Nation, and Rank. In this example, let us consider `PLAYER`'s name as the primary key even though two players can have same name. We cannot insert any data in the relation `PLAYER` without entering the name of the player. This implies that primary key cannot be null.

3.5.2 Null Integrity

Null implies that the data value is not known temporarily. Consider the relation `PERSON`. The attributes of the relation `PERSON` are name, age, and salary. The age of the person cannot be `NULL`.

3.5.3 Domain Integrity Constraint

Domains are used in the relational model to define the characteristics of the columns of a table. Domain refers to the set of all possible values that attribute can take. The domain specifies its own name, data type, and logical size. The logical size represents the size as perceived by the user, not how it is implemented internally. For example, for an integer, the logical size represents the number of digits used to display the integer, not the number of bytes used to store it. The domain integrity constraints are used to specify the valid values that a column defined over the domain can take. We can define the valid values by listing them as a set of values (such as an enumerated data type in a strongly typed programming language), a range of values, or an expression that accepts the valid values. Strictly speaking, only values from the same domain should ever be compared or be integrated through a union operator. The domain integrity constraint specifies that each attribute must have values derived from a valid range.

Example 1

The age of the person cannot have any letter from the alphabet. The age should be a numerical value.

Example 2

Consider the relation `APPLICANT`. Here `APPLICANT` refers to the person who is applying for job. The sex of the applicant should be either male (M) or female (F). Any entry other than M or F violates the domain constraint.

3.5.4 Referential Integrity

In the relational data model, associations between tables are defined through the use of foreign keys. The referential integrity rule states that a database

must not contain any unmatched foreign key values. It is to be noted that referential integrity rule does not imply a foreign key cannot be null. There can be situations where a relationship does not exist for a particular instance, in which case the foreign key is null. A referential integrity is a rule that states that either each foreign key value must match a primary key value in another relation or the foreign key value must be null.

3.6 Relational Algebra

The relational algebra is a theoretical language with operations that work on one or more relations to define another relation without changing the original relation. Thus, both the operands and the results are relations; hence the output from one operation can become the input to another operation. This allows expressions to be nested in the relational algebra. This property is called closure. Relational algebra is an abstract language, which means that the queries formulated in relational algebra are not intended to be executed on a computer. Relational algebra consists of group of relational operators that can be used to manipulate relations to obtain a desired result. Knowledge about relational algebra allows us to understand query execution and optimization in relational database management system.

3.6.1 Role of Relational Algebra in DBMS

Knowledge about relational algebra allows us to understand query execution and optimization in relational database management system. The role of relational algebra in DBMS is shown in Fig. 3.1. From the figure it is evident that when a SQL query has to be converted into an executable code, first it has to be parsed to a valid relational algebraic expression, then there should be a proper query execution plan to speed up the data retrieval. The query execution plan is given by query optimizer.

3.7 Relational Algebra Operations

Operations in relational algebra can be broadly classified into set operation and database operations.

3.7.1 Unary and Binary Operations

Unary operation involves one operand, whereas binary operation involves two operands. The selection and projection are unary operations. Union, difference, Cartesian product, and Join operations are binary operations:

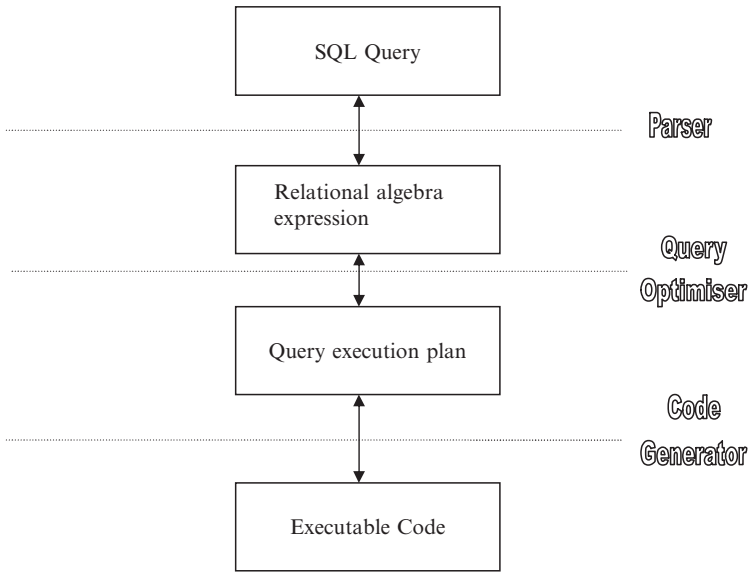
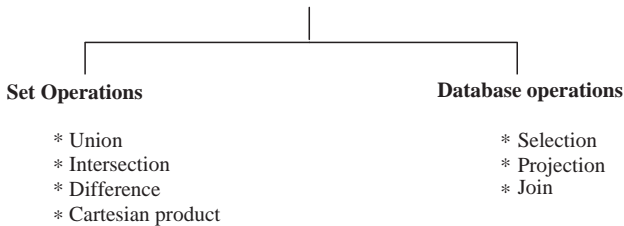


Fig. 3.1. Relational algebra in DBMS

- Unary operation operate on one relation
- Binary operation operate on more than one relation

Relational algebra operations



Three main database operations are SELECTION, PROJECTION, and JOIN.

Selection Operation

The selection operation works on a single relation R and defines a relation that contains only those tuples of R that satisfy the specified condition (Predicate). Selection operation can be considered as row wise filtering. This is pictorially represented in Fig. 3.2

Syntax of Selection Operation

The syntax of selection operation is: $\sigma_{\text{Predicate}}(R)$. Here R refers to relation and predicate refers to condition.



Fig. 3.2. Pictorial representation of SELECTION operation

Illustration of Selection Operation

To illustrate the SELECTION operation consider the STUDENT relation with the attributes Roll number, Name, and GPA (Grade Point Average).

Example

Consider the relation STUDENT shown later:

STUDENT		
Student Roll. No	Name	GPA
001	Aravind	7.2
002	Anand	7.5
003	Balu	8.2
004	Chitra	8.0
005	Deepa	8.5
006	Govind	7.2
007	Hari	6.5

Query 1: List the Roll. No, Name, and GPA of those students who are having GPA of above 8.0

Query expressed in relational algebra as $\sigma_{\text{GPA} > 8}(\text{Student})$.

The result of the earlier query is:

Student Roll. No	Name	GPA
003	Balu	8.2
005	Deepa	8.5

Query 2: Give the details of first four students in the class.

Relational algebra expression is $\sigma_{\text{Roll. No} \leq (\text{student})}$.

Table as a result of query 2 is

Student Roll. No	Name	GPA
001	Aravind	7.2
002	Anand	7.5
003	Balu	8.2
004	Chitra	8.0

Projection Operation

The projection operation works on a single relation R and defines a relation that contains a vertical subject of R, extracting the values of specified attributes and elimination duplicates. The projection operation can be considered as column wise filtering. The projection operation is pictorially represented in Fig. 3.3.

Syntax of Projection Operation

The syntax of projection operation is given by: $\prod_{a_1, a_2, \dots, a_n} (R)$.

Where a_1, a_2, \dots, a_n are attributes and R stands for relation.

STAFF				
Staff No	Name	Gender	Date of birth	Salary
SL21	Raghavan	M	1-5-76	15,000
SL22	Raghu	M	1-5-77	12,000
SL55	Babu	M	1-6-76	12,500
SL66	Kingsly	M	1-8-78	10,000



Fig. 3.3. Pictorial representation of Projection operation

Illustration of Projection Operation

To illustrate projection operation consider the relation STAFF, with the attributes Staff number, Name, Gender, Date of birth, and Salary.

Query 1: Produce the list of salaries for all staff showing only the Name and salary detail. Relational algebra expression: $\Pi_{\text{Name, salary}}(\text{staff})$

Output for the Query 1

Name	Salary
Raghavan	15,000
Raghu	12,000
Babu	12,500
Kingsly	10,000

Query 2: Give the name and Date of birth of the all the staff in the STAFF relation.

Relational algebra expression for query 2: $\Pi_{\text{Name, date of birth}}(\text{staff})$

Name	Date of birth
Raghavan	1-5-76
Raghu	1-5-77
Babu	1-6-76
Kingsly	1-8-78

3.7.2 Rename operation (ρ)

The rename operator returns an existing relation under a new name. $\rho_A(B)$ is the relation B with its name changed to A. The results of operation in the relational algebra do not have names. It is often useful to name such results for use in further expressions later on. The rename operator can be used to name the result of relational algebra operation.

Example of Rename Operation

Consider the relation BATSMAN with the attributes name, nation, and BA.

Name	Nation	BA
Sachin Tendulkar	India	45.5
Brian Lara	West Indies	43.5
Inzamamulhaq	Pakistan	42.5

The attributes of the relation BATSMAN can be renamed as name, nation and batting average as name, nation, batting average (BATSMAN) so that the relation BATSMAN after rename operation as shown later.

BATSMAN		
Name	Nation	Batting average
Sachin Tendulkar	India	45.5
Brian Lara	West Indies	43.5
Inzamamulhaq	Pakistan	42.5

From the earlier operation it is clear that rename operation changes the schema of the database and it does not change the instance of the database.

Union Compatibility

In order to perform the Union, Intersection, and the Difference operations on two relations, the two relations should be union compatible. Two relations are union compatible if they have same number of attributes and belong to the same domain. Mathematically UNION COMPATIBILITY it is given as:

Let $R(A_1, A_2, \dots, A_n)$ and $S(B_1, B_2, \dots, B_n)$ be the two relations. The relation R has the attributes A_1, A_2, \dots, A_n and the relation S has the attributes B_1, B_2, \dots, B_n . The two relations R and S are union compatible if $\text{dom}(A_i) = \text{dom}(B_i)$ for $i = 1$ to n .

3.7.3 Union Operation

The union of two relations R and S defines a relation that contains all the tuples of R or S or both R and S, duplicate tuples being eliminated.

Relational Algebra Expression

The union of two relations R and S are denoted by $R \cup S$. $R \cup S$ is pictorially represented in the Fig. 3.4.

Illustration of UNION Operation

To illustrate the UNION operation consider the two relations Customer 1 and Customer 2 with the attributes Name and city.

Customer 1		Customer 2	
Name	City	Name	City
Anand	Coimbatore	Gopu	Tirunelveli
Aravind	Chennai	Balu	Kumbakonam
Gopu	Tirunelveli	Rahu	Chidambaram
Helan	Palayankottai	Helan	Palayamkottai

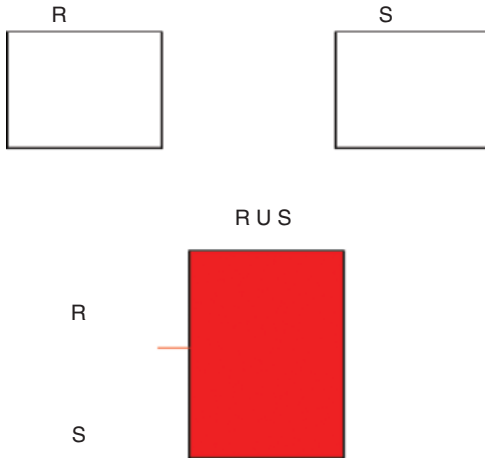


Fig. 3.4. Union of two relations R and S

Example

Query Determine Customer 1 \cup Customer 2

Result of Customer 1 \cup Customer 2

Customer 1 \cup Customer 2	
Name	City
Anand	Coimbatore
Aravind	Chennai
Balu	Kumbakonam
Gopu	Tirunelveli
Rahu	Chidambaram
Helan	Palayamkottai

3.7.4 Intersection Operation

The intersection operation defines a relation consisting of the set of all tuples that are in both R and S.

Relational Algebra Expression

The intersection of two relations R and S is denoted by $R \cap S$.

Illustration of Intersection Operation

The intersection between the two relations R and S is pictorially shown in Fig. 3.5.

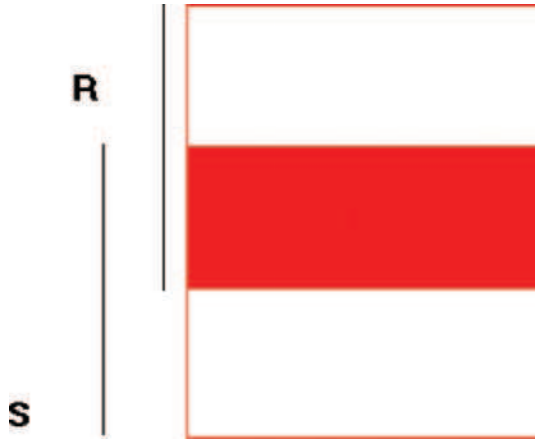


Fig. 3.5. Intersection of two relations R and S

Example

Find the intersection of Customer 1 with Customer 2 in the following table.

Customer 1 \cap Customer 2	
Name	City
Gopu	Tirunelveli
Helan	Palayamkottai

3.7.5 Difference Operation

The set difference operation defines a relation consisting of the tuples that are in relation R but not in S.

Relational Algebra Expression

The difference between two relations R and S is denoted by $\mathbf{R-S}$.

Illustration of Difference Operation

The difference between two relations R and S is pictorially shown in Fig. 3.6.

Example

Compute $\mathbf{R-S}$ for the relation shown in the following table.



Fig. 3.6. Difference between two relations R and S

Customer 1 – Customer 2	
Name	City
Anand	Coimbatore
Aravind	Chennai

3.7.6 Division Operation

The division of the relation R by the relation S is denoted by $R \div S$, where $R \div S$ is given by:

$$R \div S = \Pi_{R--S(r)} - \Pi_{R--S}((\Pi_{R--S(r)} \times s) - r)$$

To illustrate division operations consider two relations STUDENT and MARK. The STUDENT relation has the attributes Student Name and the mark in particular subject say mathematics. The MARK relation consists of only one column mark and only one row.

Student		Mark
Name	Mark	Mark
Arul	97	100
Banu	100	
Christi	98	
Dinesh	100	
Krishna	95	
Ravi	95	
Lakshmi	98	

Case (1)

If we divide the STUDENT relation by the MARK relation, the resultant relation is shown as:

Case (2)

Now modify the relation MARK that is change the mark to be 98. So that the entry in the MARK relation is modified as 98.

<u>Answer</u>	
<u>Name</u>	
Banu	
Dinesh	

<u>Student</u>		<u>Mark</u>
<u>Name</u>	<u>Mark</u>	<u>Mark</u>
Arul	97	98
Banu	100	
Christi	98	
Dinesh	100	
Krishna	95	
Ravi	95	
Lakshmi	98	

If we divide the relation STUDENT by MARK relation then the resultant relation is given by ANSWER

<u>Answer</u>
<u>Name</u>
Christi
Lakshmi

Case (3)

Now the MARK relation is modified in such a way that the entry in the MARK relation is 99. If we divide the STUDENT relation with the MARK relation, the result is NULL. Because there is no student in the STUDENT relation with the mark 99.

<u>Student</u>		<u>Mark</u>
<u>Name</u>	<u>Mark</u>	<u>Mark</u>
Arul	97	99
Banu	100	
Christi	98	
Dinesh	100	
Krishna	95	
Ravi	95	
Lakshmi	98	

The division of the STUDENT relation with the MARK relation is given by the ANSWER relation.

The division operation extracts records and fields from one table on the basis of data in the second table.

<u>Answer</u>
<u>Name</u>
<u>NULL</u>

3.7.7 Cartesian Product Operation

The Cartesian product operation defines a relation that is the concatenation of every tuples of relation R with every tuples of relation S. The result of Cartesian product contains all attributes from both relations R and S.

Relational Algebra Symbol for Cartesian Product:

The Cartesian product between the two relations R and S is denoted by $R \times S$.

Note: If there are n_1 tuples in relation R and n_2 tuples in S, then the number of tuples in $R \times S$ is $n_1 * n_2$.

Example

If there are 5 tuples in relation “R” and 2 tuples in relation “S” then the number of tuples in $R \times S$ is $5 * 2 = 10$.

Illustration of Cartesian Product

To illustrate Cartesian product operation, consider two relations R and S as given later:

R	S
a	1
b	2
	3

Determine $R \times S$:

R	S
a	1
a	2
a	3
b	1
b	2
b	3

Note:

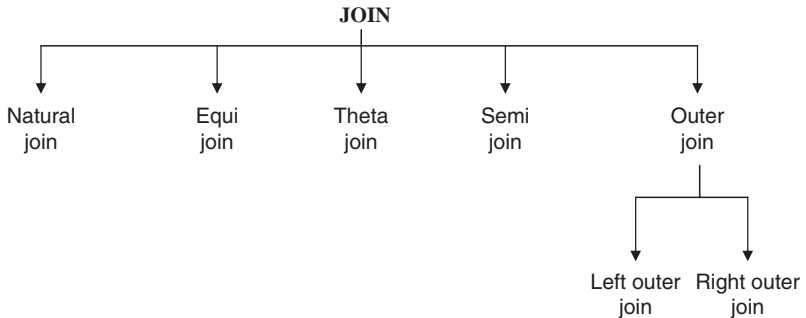
No. of tuples in $R \times S = 2 * 3 = 6$

No. of attributes in $R \times S = 2$

3.7.8 Join Operations

Join operation combines two relations to form a new relation. The tables should be joined based on a common column. The common column should be compatible in terms of domain.

Types of Join Operation



Natural Join

The natural join performs an equi join of the two relations R and S over all common attributes. One occurrence of each common attribute is eliminated from the result. In other words a natural join will remove duplicate attribute. In most systems a natural join will require that the attributes have the same name to identify the attributes to be used in the join. This may require a renaming mechanism. Even if the attributes do not have same name, we can perform the natural join provided that the attributes should be of same domain.

Input: Two relations (tables) R and S

Notation: $R \bowtie S$

Purpose: Relate rows from second table and

- Enforce equality on all column attributes
- Eliminate one copy of common attribute

* Short hand for $\prod_{L}(R \times S)$:

- L is the union of all attributes from R and S with duplicate removed
- P equates all attributes common to R and S

Example of Natural Join Operation

Consider two relations EMPLOYEE and DEPARTMENT. Let the common attribute to the two relations be DEPTNUMBER. The two relations are shown later:

It is worth to note that Natural join operation is associative. (i.e.,) If R, S, and T are three relations then

$$R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$$

Employee				Department	
Employee ID	Designation	Dept Number		Dept name	Dept Number
C100	Lecturer	E1	\bowtie	Electrical	E1
C101	Assistant Professor	E2		Computer	C1
C102	Professor	C1			

Employee \bowtie Department			
Employee ID	Designation	Dept Number	Dept name
C100	Lecturer	E1	Electrical
C102	Professor	C1	Computer

Equi Join

A special case of condition joins where the condition C contains only equality.

Example of Equi Join

Given the two relations STAFF and DEPT, produce a list of staff and the departments they work in.

STAFF			DEPT	
Staff No	Job	Dept	Dept	Name
1	salesman	100	100	marketing
2	draftsman	101	101	civil

Answer for the earlier query is equi-join of STAFF and DEPT:

STAFF EQUI JOIN DEPARTMENT				
Staff No	Job	dept	dept	Name
1	salesman	100	100	marketing
2	draftsman	101	101	civil

Theta Join

A conditional join in which we impose condition other than equality condition. If equality condition is imposed then theta join become equi join. The symbol θ stands for the comparison operator which could be $>$, $<$, $>=$, $<=$.

Expression of Theta Join

$$\sigma_{\theta}(R \times S)$$

Illustration of Theta Join

To illustrate theta join consider two relations FRIENDS and OTHERS with the attributes Name and age.




FRIENDS		OTHERS	
Name	Age	Alias	Size
Joe	4	Bob	8
Sam	9	Gim	10
Sue	10		

Result of theta join

Name	Age	Alias	Size
Joe	4	Bob	8
Sam	9	Gim	10
Sue	10		

Outer Join

In outer join, matched pairs are retained unmatched values in other tables are left null.

- 1. Full outer join 
- 2. Left outer join 
- 3. Right outer join 

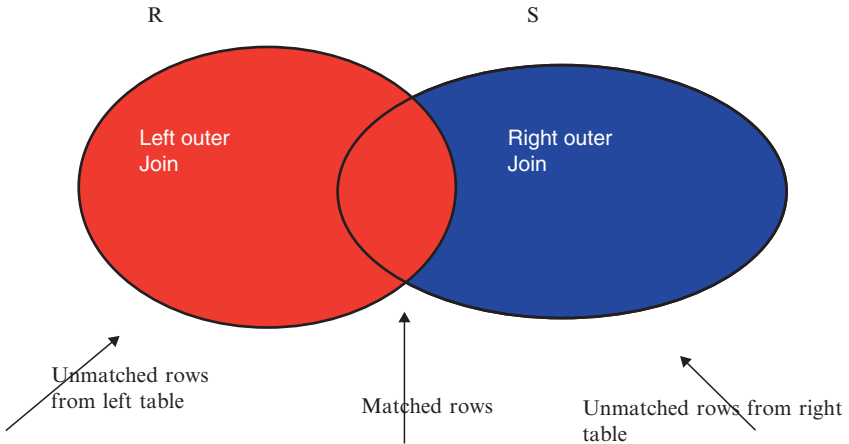


Fig. 3.7. Representation of left and right outer join

Types of Outer Join

The pictorial representation of the left and the right outer join of two relations R and S are shown in Fig. 3.7:

1. *Left Outer Join.* Left outer joins is a join in which tuples from R that do not have matching values in the common column of S are also included in the result relation.
2. *Right Outer Join.* Right outer join is a join in which tuples from S that do not have matching values in the common column of R are also included in the result relation.
3. *Full Outer Join.* Full outer join is a join in which tuples from R that do not have matching values in the common columns of S still appear and tuples in S that do not have matching values in the common columns of R still appear in the resulting relation.

Example of Full Outer Left Outer and Right Outer Join

Consider two relations PEOPLE and MENU determine the full outer, left outer, and right outer join.

Table 3.1. Left outer join of PEOPLE and MENU relation
$$\text{PEOPLE} \bowtie \text{PEOPLE.Food} = \text{MENU.Food} \text{ MENU}$$

Name	Age	People.Food	Menu.Food	Day
Raja	21	Idly	Idly	Tuesday
Ravi	22	Dosa	Dosa	Wednesday
Rani	20	Pizza	NULL	NULL
Devi	21	Pongal	Pongal	Monday

Table 3.2. Right outer join of PEOPLE and MENU relation
$$\text{PEOPLE} \bowtie \text{PEOPLE.Food} = \text{Menu.Food} \text{ MENU}$$

Name	Age	People.Food	Menu.Food	Day
Devi	21	Pongal	Pongal	Monday
Raja	21	Idly	Idly	Tuesday
Ravi	22	Dosa	Dosa	Wednesday
NULL	NULL	NULL	Fried rice	Thursday
NULL	NULL	NULL	Parotta	Friday

PEOPLE		
Name	Age	Food
Raja	21	Idly
Ravi	22	Dosa
Rani	20	Pizza
Devi	21	Pongal

MENU	
Food	Day
Pongal	Monday
Idly	Tuesday
Dosa	Wednesday
Fried rice	Thursday
Parotta	Friday

1. The left outer join of PEOPLE and MENU on Food is represented as

$\text{PEOPLE} \bowtie \text{PEOPLE.Food} = \text{MENU.Food} \text{ MENU}$. The result of the left outer join is shown in Table 3.1.

From this table, it is to be noted that all the tuples from the left table (in our case it is PEOPLE relation) appears in the result. If there is any unmatched value then a NULL value is returned.

2. The right outer join of PEOPLE and MENU on Food is represented in

the relational algebra as $\text{PEOPLE} \bowtie \text{PEOPLE.Food} = \text{Menu.Food} \text{ MENU}$. The result of the right outer join is shown in Table 3.2.

Table 3.3. Full outer join of PEOPLE and MENU relation

Name	Age	People.Food	Menu.Food	Day
Raja	21	Idly	Idly	Tuesday
Ravi	22	Dosa	Dosa	Wednesday
Rani	20	Pizza	NULL	NULL
Devi	21	Pongal	Pongal	Monday
NULL	NULL	NULL	Fried rice	Thursday
NULL	NULL	NULL	Parotta	Friday

From this table, it is clear that all tuples from the right-hand side relation (in our case the right hand relation is MENU) appears in the result.

3. The full outer join of PEOPLE and MENU on Food is represented in the relational algebra as $\text{PEOPLE} \bowtie \text{PEOPLE.Food} = \text{MENU.Food}$ MENU. The result of the full outer join is shown in Table 3.3.

From this table, it is clear that tuples from both the PEOPLE and the MENU relation appears in the result.

Semi-Join

The semi-join of a relation R, defined over the set of attributes A, by relation S, defined over the set of attributes B, is the subset of the tuples of R that participate in the join of R with S. The advantage of semi-join is that it decreases the number of tuples that need to be handled to form the join. In centralized database system, this is important because it usually results in a decreased number of secondary storage accesses by making better use of the memory. It is even more important in distributed databases, since it usually reduces the amount of data that needs to be transmitted between sites in order to evaluate a query.

Expression for Semi-Join

$$R \ltimes_F S = \prod_A (R \ltimes_F S) \quad \text{where } F \text{ is the predicate.}$$

Example of Semi-Join

In order to understand semi-join consider two relations EMPLOYEE and PAY

EMPLOYEE			PAY	
Employee Number	Employee Name	Designation	Designation	Salary
E1	Rajan	Programmer	Programmer	25,000
E2	Krishnan	System Analyst	Consultant	70,000
E3	Devi	Database Administrator		
E4	Vidhya	Consultant		

The semi-join of EMPLOYEE with the PAY is denoted by:

$EMPLOYEE \bowtie_{EMPLOYEE.DESIGNATION=PAY.DESIGNATION} PAY$. The result of this semi-join is given later:

Employee Number	Employee Name	Designation
E1	Rajan	Programmer
E4	Vidhya	Consultant

From the result of the semi-join it is clear that a semi-join is half of a join: the rows of one table that match with at least one row of another table. Only the rows of the first table appear in the result.

3.8 Advantages of Relational Algebra

The relational algebra has solid mathematical background. The mathematical background of relational algebra is the basis of many interesting developments and theorems. If we have two expressions for the same operation and if the expressions are proved to be equivalent, then a query optimizer can automatically substitute the more efficient form. Moreover, the relational algebra is a high level language which talks in terms of properties of sets of tuples and not in terms of for-loops.

3.9 Limitations of Relational Algebra

The relational algebra cannot do arithmetic. For example, if we want to know the price of 10l of petrol, by assuming a 10% increase in the price of the petrol, which cannot be done using relational algebra.

The relational algebra cannot sort or print results in various formats. For example we want to arrange the product name in the increasing order of their price. It cannot be done using relational algebra.

Relational algebra cannot perform aggregates. For example we want to know how many staff are working in a particular department. This query cannot be performed using relational algebra.

The relational algebra cannot modify the database. For example we want to increase the salary of all employees by 10%. This cannot be done using relational algebra.

The relational algebra cannot compute “transitive closure.” In order to understand the term transitive closure consider the relation RELATIONSHIP, which describes the relationship between persons.

Consider the query, Find all direct and indirect relatives of Gopal? It is not possible to express such kind of query in relational algebra. Here transitive means, if the person A is related to the person B and if the person B is related to the person C means indirectly the person A is related to the person C. But relational algebra cannot express the transitive closure.

RELATIONSHIP		
Person1	Person2	Relationship
Gopal	Nandini	Father
Siva	Raja	Brother
Gopal	Neena	Husband
Deepa	Lakshmi	Sister

3.10 Relational Calculus

The purpose of relational calculus is to provide a formal basis for defining declarative query languages appropriate for relational databases. Relational Calculus comes in two flavors (1) Tuple Relational Calculus (TRC) and (2) Domain Relational Calculus (DRC). The basic difference between relational algebra and relational calculus is that the former gives the procedure of how to evaluate the query whereas the latter gives only the query without giving the procedure of how to evaluate the query:

- The variable in tuple relational calculus formulae range over tuples.
- The variable in domain relational calculus formulae range over individual values in the domains of the attributes of the relations.
- Relational calculus is nonoperational, and users define queries in terms of what they want, not in terms of how to compute it. (Declarativeness.)

Relational Calculus and Relational Algebra:

The major difference between relational calculus and relational algebra is summarized later:

- A relational calculus query specifies *what* information is retrieved
- A relational algebra query specifies *how* information is retrieved

3.10.1 Tuple Relational Calculus

Tuple relational calculus is a logical language with variables ranging over tuples. The general form of tuple relational calculus is given by:

$$\{\langle \text{tuple variable list} \rangle \mid \langle \text{conditions} \rangle\}$$

$$\{t \mid \text{COND}(t)\}$$

Here t is the tuple variable, which stands for tuples of relation. $\text{COND}(t)$ is a formula that describes t . The meaning of the earlier expression is to return all tuples T that satisfy the condition COND :

- $T/R(T)$ means return all tuples T such that T is a tuple in relation R .
- For example, $\{T.\text{name}/\text{FACULTY}(T)\}$ means return all the names of faculty in the relation FACULTY .
- $\{T.\text{name}/\text{FACULTY}(T) \text{ AND } T.\text{deptid} = \text{'EEE'}\}$ means return the value of the name of the faculty who are working in EEE department.

Quantifiers

Quantifiers are words that refer to quantities such as “some” or “all” and tell for how many elements a given predicate is true. A predicate is a sentence that contains a finite number of variables and becomes a statement when specific values are substituted for the variables. Quantifiers can be broadly classified into two types (1) Universal Quantifier and (2) Existential Quantifier.

Existential Quantifier

symbol: \exists

$$\exists T \in \text{Cond}(R)$$

It will succeed if the condition succeeds for at least one tuple in T .

- $(\exists t)(C)$ – Existential operator – True if there exists a tuple t such that the condition(s) C are true.
- Example of existential quantifier is $\exists(m)$ such that $m^2 = m$. (i.e., $m = 1$).

Universal Quantifier

symbol: \forall

- $(\forall t)(C)$ – Universal operator – True if C is true for every tuple t .
- Example of universal quantifier is $\forall(2), \sin^2(2) + \cos^2(2) = 1$.
The example refers to the fact that for all values of $2 \sin^2(2) + \cos^2(2) = 1$.

Free Variable

Any variable that is not bound by a quantifier is said to be free.

Bound Variable

Any variable which is bounded by universal or existential quantifier is called bound variable.

Example of *selection* operation in TRC:

1. To find details of all staff earning more than Rs. 10,000:

$$\{S \mid \text{Staff}(S) \wedge S.\text{salary} > 10000\}$$

Example of *projection* operation in TRC:

2. To find a particular attribute, such as salary, write:

$$\{S.\text{salary} \mid \text{Staff}(S) \wedge S.\text{salary} > 10000\}$$

Quantifier Example

Client(ID, fName, lName, Age)

Matches(Client1, Client2, Type)

- List the first and last names of clients that appear as client1 in a match of any type.

RAlg: $p(\text{fName}, \text{lName})(\text{Client } (\text{ID} = \text{Client1}) \text{ Matches})$

RCalc: $\{c.\text{fName}, c.\text{lName} \mid \text{CLIENT}(c) \text{ AND } (\exists m)(\text{MATCHES}(m) \text{ AND } c.\text{ID} = m.\text{Client1})\}$

Joins in Relational Calculus

Consider the two relations Client and Matches as

Client(ID, fName, lName, Age)

Matches(Client1, Client2, Type)

- **List all information about clients and the corresponding matches that appear as client1 in a match of any type.**

The earlier query can be expressed both in Relational Algebra and Tuple relational Calculus as:

- **RAlg:** Client (ID = Client1) Matches

- **RCalc:**

$$\{c, m \mid \text{CLIENT}(c) \text{ AND MATCHES}(m) \text{ AND } c.\text{ID} = m.\text{Client1}\}$$

3.10.2 Set Operators in Relational Calculus

The set operations like Union, Intersection, difference, and Cartesian Product can be expressed in Tuple Relational Calculus as:

Union

- $R1(A,B,C) \cup R2(A, B, C)$
- $\{r \mid R1(r) \text{ OR } R2(r)\}$

Intersection

- $R1(A,B,C) \cap R2(A, B, C)$
- $\{r \mid R1(r) \text{ AND } R2(r)\}$

Cartesian Product

- $R(A, B, C) \times S(D, E, F)$
- $\{r, s \mid R(r) \text{ AND } S(s)\}$ // same as join without the select condition

Subtraction

- $R1(A,B,C) - R2(A, B, C)$
- $\{r \mid R1(r) \text{ AND NOT } R2(r)\}$

Queries and Tuple Relational Calculus Expressions

Some of the queries and the corresponding relational calculus and their explanations are given later. Here we have given set of queries like SET 1, SET 2, and SET 3.

- Query set 1 deals with Railway Reservation Management
- Query set 2 deals with Library Database Management
- Query set 3 deals with Hostel Database Management

Query Set1: Query set 1 deals with railway reservation system.

Query 1: Find all the train details for the trains where starting place is “Chennai.”

Relational calculus expression: $\{t \mid t \in \text{train_details} \wedge \text{start place} = \text{“Chennai”}\}$

Explanation: Set of all tuples “t” that belong to the relation “train details” and also the starting place is “Chennai” is found by the query.

Query 2: Find all train names whose destination is “Salem.”

Relational calculus expression

$\{t \mid \exists s \in \text{train_details} (t [\text{train_no}] = s [\text{train_no}] \wedge s [\text{destination}] = \text{“Salem”})\}$

Explanation: There exist a tuple “t” in the relation “r” such that the predicate is true.

The set of all tuples “t” such that, there exists a tuple “s” in relation train details for which the values of “t” and “s” for the train_no attribute are equal and the value of “s” for the destination is “Salem.”

Query 3: Find the names of all passengers who have canceled the ticket and whose age is above 40.

Relational calculus expression $\{t \mid \exists s \in \text{cancel} (t [\text{train_no}] = s [\text{train_no}] \wedge \exists u \in \text{passen_details} (u [\text{name}] = s [\text{name}] \wedge u[\text{age}] > 40))\}$

Explanation: Set of all passenger names tuples for which the age is above 40 and the ticket is canceled. The tuple variable “s” ensures that the passenger canceled the ticket. The tuple “u” is restricted to having the same passenger name as “s.”

Query 4: List the train numbers of all trains which has no cancelation and only reservation.

Relational Calculus Expression

Relational calculus expression $\{t \mid \exists s \in \text{reserve} (t [\text{train_no}] = s [\text{train_no}]) \wedge \neg \exists u \in \text{cancel} (t [\text{train_no}] = u[\text{train_no}])\}$

Explanation: Set of all tuples “t” such that there exists a tuple “s” that belongs to reserve such that the train_no attribute is equal for “t” and “s” and there exists a tuple “u” that belongs to cancel where the values of “t” and “u” for the train_no attribute is the same.

Query 5: List all female passengers name who are traveling by the train “Blue Mountain.”

Relational Calculus Expression

Relational calculus expression $\{t \mid \exists s \in \text{passen_details} (t [\text{p_name}] = s [\text{p_name}] \wedge s[\text{sex}] = \text{“female”} \wedge s[\text{train_name}] = \text{“Blue mountain”})\}$.

Explanation: Set of all tuples “t” such that there exists a tuple “s” that belongs to passen_details for which the values of “t” and “s” for the p_name attribute is same and the sex attribute = “female” and train_name attribute = “Blue mountain.”

Query Set 2: Query set 2 deals with frequent queries in library database management.

Query 1: Find the acc.no/- for each book whose price >1000.

Relational Calculus Expression

Relational calculus expression $\{t \mid \exists s \in \text{book} (t[\text{acc_no/-}] = s[\text{acc_no/-}] \wedge s[\text{price}] > 1000)\}$

Explanation: The set of all tuples “t” such that there exists a tuple “s” in relation book for which the values “t” and “s” for the acc_no/- attribute are equal and the value of the s for the price attribute is greater than 1000.

Query 2: Find the name of all the students who have borrowed a book and price of those book is greater than 1000.

Relational Calculus Expression

$$\{t \mid \exists s \in \text{books_borrowed}(t[\text{std_name}] = s[\text{std_name}] \wedge \exists u \in \text{book}(u[\text{acc_no}/-] = s[\text{acc_no}/-] \wedge u[\text{price}] > 1000))\}$$

Explanation: The set of all tuples “t” such that there exists a tuple “s” in relation books.borrowed for which the values “t” and “s” for the student name attribute are equal and “u” tuple variable on book relation for which “u” and “s” for the acc_no/- attribute are equal and the value of “u” for the price attribute is greater than 1000.

Query 3: Find the name of the students who borrowed book, have book in his account or both.

Relational Calculus Expression

$$\{t \mid \exists s \in \text{books_borrowed}(t[\text{stud_name}] = s[\text{stud_name}]) \vee \exists u \in \text{books_remaining}(t[\text{stud_name}] = su[\text{stud_name}])\}$$

Explanation: The set of all tuples “t” such that there exists a tuple “s” in relation books_ borrowed for which the values “t” and “s” for the student name attribute are equal and “u” tuple variable on books_remaining relation for which “u” and “s” for the stud_name attribute are equal.

Query 4: Find only those students’ names who are having both the books in their account as well as the books borrowed from their account.

Relational Calculus Expression

$$\{t \mid \exists s \in \text{books_borrowed}(t[\text{std_name}] = s[\text{std_name}]) \wedge \exists u \in \text{books_remaining}(t[\text{std_name}] = s[\text{std_name}])\}$$

Explanation: The set of all tuples “t” such that there exists a tuple “s” such that in relation books.borrowed for which the values “t” and “s” for the student name attribute are equal and “u” tuple variable on books_remaining relation for which “u” and “s” for the student name attribute are equal.

Query 5: Query that uses implication symbol $p \Rightarrow q$ find all students belongs to EEE department who borrowed the books.

Relational Calculus Expression

$$\{t \mid \exists r \in \text{books_borrowed} (r[\text{std_name}] = t[\text{std_name}] \wedge (\forall u \in \text{department} (u[\text{dept_name}] = \text{"EEE"})))\} \Rightarrow \{t \mid \exists r \in \text{books_borrowed} (r[\text{std_name}] = t[\text{std_name}] \wedge \exists w \in \text{student} (w[\text{roll_no}/-] = r[\text{roll_no}/-] \wedge w[\text{dept_name}] = u[\text{dept_name}]))\}$$

Explanation: The set of all tuples “t” such that there exists a tuple “s” such that in relation books_borrowed for which the values “t” and “s” for the student name attribute are equal and “u” tuple variable on department relation must be equal to “EEE.” And this must be equal to the set of all tuple “t” such that there exists a tuple “r” in relation books_borrowed for which the values “r” and “t” for the student name attribute are equal and “w” the variable on relation student for which “w” and “r” are equal for the roll_no/-attribute and “w” and “u” are equal for the dept_name.

Query Set 3: Query set 3 deals with hostel management.

Query 1: Find all the students id who are staying in hostel.

Tuple Relational Calculus Expression

$$\{t \mid \exists s \in \text{student_detail} (t[\text{roll no}] = s[\text{rollno}])\}$$

Explanation: Here t is the set of tuples in the relation student_detail such that there exists a tuple s which consists of students ID who are staying in the hostel.

Query 2: Find all the details of the student who are belonging to EEE branch.

Tuple Relational Calculus Expression

$$\{t \mid t \in \text{student_detail} \wedge t[\text{course name}] = \text{"EEE"}\}$$

Explanation: Here t is the set of tuples in the relation student_detail such that it consists of all the details of the student who are belonging to the “EEE” branch.

Query 3: Find all the third semester BE-EEE students.

Tuple Relational Calculus Expression

$$\{t \mid t \in \text{student_detail} \wedge t[\text{coursename}] = \text{"EEE"} \wedge t[\text{semester}] = 3\}$$

Explanation: Here t is the set of tuples in the relation student_detail such that it consists of all the details of the student who belongs to the third semester BE-EEE branch.

Query 4: Find all the lecturers name belonging to the EEE department.

Tuple Relational Calculus Expression

$\{t \mid \exists s \in \text{staff_detail} (t[\text{staffname}] = s[\text{staffname}])\}$

Explanation: Here t is the set of tuples in the relation `staff_detail` and there exists a tuple s which consists of lecturers name who belongs to the “EEE” department.

Query 5: Find all the staff who are having leisure period at third hour on Monday.

Tuple Relational Calculus Expression

$\{t \mid \exists s \in \text{staff_detail} (t[\text{staffname}] = s[\text{staffname}] \wedge \exists u \in \text{lecturersched-} \\ \text{ule_monday} (s[\text{staffid}] = u[\text{staffid}] \wedge u[\text{third hour}] = \text{“EEE”}))\}$

Explanation: Here t is the set of tuples in the relation `staff_detail` and there exists a tuple s which consists of staff name who are all having leisure period at third hour on Monday for every week.

Safety of Expression

It is possible to write tuple calculus expressions that generate infinite relations. For example $\{t/\sim t \in R\}$ results in an infinite relation if the domain of any attribute of relation R is infinite. To guard against the problem, we restrict the set of allowable expressions to safe expressions. An expression $\{t/P(t)\}$ in the tuple relational calculus is safe if every component of t appears in one of the relations, tuples, or constants that appear in P (Here P refers to Predicate or condition).

Limitations of TRC

TRC cannot express queries involving:

- Aggregations.
- Groupings.
- Orderings.

3.11 Domain Relational Calculus (DRC)

Domain relational calculus is a nonprocedural query language equivalent in power to tuple relational calculus. In domain relational calculus each query is an expression of the form:

$$\{ \langle X_1, X_2, \dots, X_n \rangle / P(X_1, X_2, \dots, X_n) \} \text{ where}$$

- X_1, X_2, \dots, X_n represent domain variables
- P represents a formula similar to that of the predicate calculus.

Domain variable: A domain variable is a variable whose value is drawn from the domain of an attribute.

3.11.1 Queries in Domain Relational Calculus:

Consider the ER diagram:



<u>STUDENT</u>	<u>CLASS</u>	<u>TAKES</u>
<u>ID Name Address</u>	<u>CID CNAME location</u>	<u>ID CID GRADE</u>
123 Anbu		
456 Anu		

Query 1:

Get the details of all students?

This query can be expressed in DRC as

$$\{ \langle I, n, a \rangle / \langle I, n, a \rangle \in \text{STUDENT} \}$$

Query 2: (Selection operation)

Find the details of the student whose roll no (or) ID is 123?

$$\{ \langle 123, n, a \rangle / \langle 123, n, a \rangle \in \text{STUDENT} \}$$

(OR)

$$\{ \langle I, n, a \rangle / \langle I, n, a \rangle \in \text{STUDENT} \wedge I = 123 \}$$

(Here I, n, a are referred to as domain variables)

Query 3: (Projection)

Find the name of the student whose roll no. is 456?

$$\{ \langle I \rangle / \langle I, n, a \rangle \in \text{STUDENT} \wedge I = 456 \}$$

3.11.2 Queries and Domain Relational Calculus Expressions

Some of the queries and the corresponding relational calculus and their explanations are given later. Here we have given set of queries like SET 1, SET 2, and SET 3:

- Query set 1 deals with Railway Reservation Management
- Query set 2 deals with Library Database Management
- Query set 3 deals with Department Database Management

Query Set 1: Query set 1 deals with railway reservation system.

Query 1: List the details of the passengers traveling by the train “Intercity express.”

Domain Relational Calculus Expression

$$\{ \langle \text{name, age, sex, train_no, "blue mountain"} \rangle \mid \langle \text{name, age, sex, train_no, train_name} \rangle \in \text{passen_details} \}$$

Explanation: The attributes of the `passen_details` are listed where the `train_name` attribute = "Intercity express."

Query 2: Select names of passengers whose sex = "female" and age > 20.

Domain Relational Calculus Expression

$$\{ \langle \text{p_name} \rangle \mid \exists \text{p_age, p_sex, p_trainno. } (\langle \text{p_name, p_age, p_sex, p_trainno} \rangle \in \text{passen_details} \wedge \text{p_sex} = \text{"female"} \wedge \text{p_age} > 20) \}$$

Explanation: Lists the names of passengers from the relation `passenger_details` where there are two constraints which are sex = female and age > 20.

Query 3: Find all the names of passengers who have "Salem" as start place and find their train names.

Domain Relational Calculus Expression

$$\{ \langle \text{p_name, train_name} \rangle \mid \exists \text{p_name} > \text{p_name, p_age, p_trainno, } (\langle \text{p_name, p_age, p_sex, p_train_no, p_trainname} \rangle \in \text{passen_details} \\ \wedge \exists \text{t_start, t_dest, t_route, t_no } (\langle \text{t_name, t_no, t_start, t_dest, t_route} \rangle \in \text{train_details} \wedge \text{t_start} = \text{"salem"})) \}$$

Explanation: Two relations – `passen_details` and `train_details` are involved in this query. The train names and the passenger names whose start place = Salem is displayed.

Query 4: Find all train names which has reservation and no cancellation.

Domain Relational Calculus Expression

$$\{ \langle \text{t_name} \rangle \mid \exists \text{t_name, p_name, p_source, p_dest } (\langle \text{t_name, t_no, p_name, p_source, p_dest} \rangle \\ \in \text{reserve} \wedge \exists \text{ticket_no, t_no, s_no, p_name } (\langle \text{t_name, t_no, tick_no, p_name, s_no} \rangle \in \text{cancel})) \}$$

Explanation: The `reserve` and `cancel` relations are involved here. The train names which satisfies both the conditions are displayed.

Query 5: Find names of all trains whose destination is "CHENNAI" and source is "COIMBATORE."

Domain Relational Calculus Expression

$$\{ \langle \text{t_name} \rangle \mid \exists \text{t_no, t_start, t_dest, t_route } (\langle \text{t_name, t_no, t_start, t_dest, t_route} \rangle \in \text{train_details} \wedge \text{t_source} = \text{"coimbatore"} \wedge \text{t_dest} = \text{"chennai"}) \}$$

Explanation: The name of the trains that start from Coimbatore and reach Chennai are listed from the relations train_details.

Query Set 2:

Query set 2 deals with Library Management.

Query 1: Find the student name, roll_no. for those belongs to “EEE” department.

Domain Relational Calculus Expression

$$\{ \langle \text{std_name}, \text{std_roll_no} \rangle \mid \text{dept_name}(\langle \text{std_name}, \text{roll_no}, \text{depart_name} \rangle \in \text{student} \wedge \text{depart_name} = \text{“EEE”}) \}$$

Explanation: Student relation is involved in this. Std_name, roll_no are the attribute belongs to the student relation whose department name is “EEE.”

Query 2: Find the acc_no, books_cal_no, and author name for the books of price >120.

Domain Relational Calculus Expression

$$\{ \langle \text{acc_no}, \text{book_call_no}, \text{author_name} \rangle \mid \exists \text{book_name}, \text{price}(\langle \text{book_name}, \text{acc_no}, \text{call_no}, \text{author_name}, \text{price} \rangle \in \text{books} \wedge \text{price} > 120) \}$$

Explanation: Books relation is involved here. In this expression acc_no, book_call_no, and author name are selected for the book for which the price is greater than 120.

Query 3: Find the roll_no of all the students who have borrowed book from library and find the no/- of books they borrowed an that books belongs to “EEE” department.

Domain Relational Calculus Expression

$$\{ \langle \text{roll_no}/- \rangle \mid \exists \text{std_name}, \text{book_acc_no}(\langle \text{std_name}, \text{roll_no}, \text{book_acc_no}, \text{number of books borrowed} \rangle \in \text{books_borrowed} \wedge \exists \text{name}, \text{dept_name}(\langle \text{name}, \text{roll_no}, \text{dept_name} \rangle \in \text{student} \wedge \text{dept_name} = \text{“EEE”})) \}$$

Explanation: Here two relations are involved (1) books_borrowed and (2) student. The roll_no/- of the students who borrowed “EEE” department book involves both the earlier relations. Roll_no/- are selected from the both the relation of the student who borrowed book from library which belongs to “EEE” department.

Query 4: Find the std_name and their depart_name who have borrowed a book which is less than 2 in number.

Domain Relational Calculus Expression

$$\{ \langle \text{dept_name}, \text{name} \rangle \mid \exists \text{roll_no}/-, \text{book_acc_no}/-, \text{no_of_books_borrowed} \langle \text{roll_no}/-, \text{book_acc_no}/-, \text{no}/- \text{ of books.borrowed, std_name} \rangle \in \text{books.borrowed} \wedge \text{no}/- \text{ of books borrowed} < 2 \wedge \exists \text{roll_no}/-(\text{roll_no}/-, \text{name}, \text{dept_name}) \in \text{student} \}$$

Explanation: Here two relations are involved (1) books_borrowed and (2) student. For student name the relation involved is books_borrowed and for depart_name the relation involved is student and the constraint is no/- of books_borrowed is less than two.

Query 5: Find the name of all the students who have borrowed, having books in his account or both in the department EEE.

Domain Relational Calculus Expression

$$\{ \langle \text{name} \rangle \mid \exists \text{roll_no}/-, \text{book_acc_no}/-, \text{no_of_books_borrowed} \langle \text{name}, \text{roll_no}/-, \text{book_acc_no}/-, \text{no}/- \text{ of books.borrowed} \rangle \in \text{books.borrowed} \wedge \exists \text{roll_no}/-, \text{depart_name} \langle \text{name}, \text{roll_no}/-, \text{dept_name} \rangle \in \text{student} \wedge \text{dept_name} = \text{"eee"} \} \vee \exists \text{roll_no}/-, \text{no}/- \text{ of books.remaining} \langle \text{name}, \text{roll_no}/-, \text{no}/- \text{ of books.remaining} \rangle \in \text{books.remaining} \wedge \exists \text{roll_no}/-, \text{dept_name} \langle \text{name}, \text{roll_no}/-, \text{dept_name} \rangle \in \text{student} \wedge \text{dept_name} = \text{"EEE"} \}$$

Explanation: Here three relations are involved (1) books_remaining, (2) books_borrowed, and (3) student. Name is an attribute belonging to books_borrowed and books_remaining relations, dept_name belongs to student relation. The student borrowed books or having books in his account or both which belongs to “EEE” department is selected.

Query Set 3: Query set 2 deals with Department Database Management system.

Query 1: Find all the student name belongs to fifth sem ECE branch.

Domain Relational Calculus Expression

$$\{ \langle \text{stud_name} \rangle \mid \exists \langle \text{r,cn,s,h,dob,pn,b} \rangle \in \text{student_detail} \wedge \text{s} = \text{"V"} \wedge \text{b} = \text{"ECE"} \}$$

Explanation: Students name domain is formed from relation V semester “ECE” branch.

Domain variables used:

r - roll no.; cn - course name; s - semester; h - hosteller
dob - date of birth; pn - phone no.; b - branch name

Query 2: Find all the details of students belonging to CSE branch.

Domain Relational Calculus Expression

$$\{ \langle sn, r, cn, s, h, dob, pn, b \rangle \mid \langle sn, r, cn, s, h, dob, pn, b \rangle \in \text{student_detail} \wedge b = \text{"CSE"} \}$$

Explanation: All domain variables included from student-detail table which consists of all details about students belonging to the CSE branch.

Query 3: Find all the students id whose date of birth is above 1985.

Domain Relational Calculus Expression

$$\{ \langle r \rangle \mid \exists sn, cn, s, h, dob, pn, b (\langle r, sn, cn, b, s, h, dob, pn \rangle \in \text{student_detail} \mid \wedge \text{dob} > \text{"1985"}) \}$$

Explanation: Domain variable r (roll no) is included from student_detail relation, which consists of students ID whose date of birth is above 1985.

Query 4: Find all the lecturers id belonging to production dept.

Domain Relational Calculus Expression

$$\{ \langle sid \rangle \mid \exists sn, dob, desg, y, foi, e, d \mid \langle sid, sn, dob, desg, y, foi, e, d \rangle \in \text{staff_detail} \wedge d = \text{"prod"} \}$$

Explanation: Domain variables from staff_detail:

sid - staff_ID; dob - date of birth; sn - staff name; desg - designation
y - year since serving; foi - field of interest; e - email id; d - department

The sid (staff id) from staff detail belonging to production department.

Query 5: Find all the lecturers' names who are having fifth period as leisure period on Friday.

Domain Relational Calculus Expression

$$\{ \langle sn \rangle \mid \exists sed, dob, desg, y, foi, e, d \mid \langle sn, sid, dob, desg, y, foi, e, d \rangle \in \text{staff_detail} \wedge \exists \langle sid, i, ii, iii, iv, v, vi, vii \rangle (\langle sid, sn, i, ii, iii, iv, v, vi, vii \rangle \in \text{rev_schedul_friday} \wedge v = \text{"free"}) \}$$

Explanation: Staff name domain variable from staff detail relation with fifth period as leisure which is checked using lecture schedule relation on Friday. Thus, in this, we have used two relations: staff detail and lecture schedule for Friday.

3.12 QBE

QBE stands for **Q**uery **B**y **E**xample. QBE uses a terminal display with attribute names as table headings for queries. This looks a little strange in textbooks, but people like it when they have worked with it for a while on a

terminal screen. It is very easy to list the entire schema, simply by scrolling information on the screen. QBE was developed originally by IBM in the 1970s to help users in their retrieval of data from a database. QBE represents a visual approach for accessing data in a database through the use of query templates. QBE can be considered as GUI (Graphical User Interface) based on domain calculus. QBE allows users to key in their input requests by filling in empty tables on the screen, and the system will also display its response in tabular form. QBE is user-friendly because the users are not required to formulate sentences for query requests with rigid query-language syntax. In QBE the request is entered in the form of tables whose skeletons are initially constructed by QBE.

Some of the QBE query template examples:

Example 1. Projection operation

In this template **P.** implies “Print.” The meaning is: Print the PLAYER_ADDRESS who belong to the country INDIA. To make a projection only put **P.** in any column of the projection. QBE will enforce uniqueness of projections automatically.

PLAYER_ADDRESS	NAME	CITY	COUNTRY
P.			INDIA

Example 2. Selection operation

To make a selection, put quantifiers in the columns of the attributes in the question. To print a whole record, put **P.** in the column with the name of the record.

PLAYER_ADDRESS	NAME	CITY	COUNTRY
P.			INDIA

The meaning is to print the PLAYER_ADDRESS who belong to the country INDIA.

Example 3. AND condition

To understand the AND condition consider the following template.

PLAYER_ADDRESS	NAME	CITY	COUNTRY
P.		CHENNAI	INDIA

The meaning of the earlier template is: Print the PLAYER_ADDRESS who live in INDIA and belong to the city CHENNAI.

Example 4. OR condition

To understand the OR condition consider the following template:

PLAYER_ADDRESS	NAME	CITY	COUNTRY
	P.	CHENNAI	INDIA
	P.	DELHI	INDIA

The meaning of the earlier template is “Print the name of the Player who belongs to the country INDIA and city either CHENNAI or DELHI”.

Example 5. Query involving more than one table

Let us consider a query which involves data from more than one table. Let us consider two tables PLAYER_ADDRESS and PLAYER_RANK. Here we have two tables PLAYER_ADDRESS and PLAYER_RANK, the template meaning is: Print the name of the player who belong to the country INDIA and rank less than 50. The clue for understanding the query is the fact the variable_NAME is the same in all rows of the display.

PLAYER_ADDRESS	NAME	CITY	COUNTRY
	P_NAME		INDIA

PLAYER_RANK	NAME	RANK	COUNTRY
	P_NAME	<50	INDIA

Example 6. Comparison operation

Consider the EMPLOYEE table with the columns EMPLOYEE_ID, EMPLOYEE_NAME, SALARY, and MANAGER_ID. If one wants to know the name of the employees who make more money than their managers, it can be shown in QBE as:

EMPLOYEE_ID	EMPLOYEE_NAME	SALARY	MANAGER_ID
_X	P_N	X_Y <_Y	–X

Example 7. Ordering of records

The records can be arranged either in the ascending order or in the descending order using the operator **AO.** and **DO.**, respectively.

- **AO.** implies arrange the records in ascending order.
- **DO.** implies arrange the records in descending order.

- **AO.ALL.** implies arrange the records in ascending order by preserving duplicate records.
- **DO.ALL.** implies arrange the records in descending order by preserving duplicate records.

Both **AO.** and **DO.** operators automatically eliminates duplicate responses. However, if one wishes to have all duplicate records displayed, an **ALL.** Operator must be added.

Consider the relation VEGETABLE which has three attributes VEGETABLENAME, QUANTITY, and PRICE.

VEGETABLE			
VEGETABLENAME	QUANTITY(in Kg)	PRICE(in Rs)	
Brinjal	1	13	
Potato	1	17	
Ladies Finger	1	12	
Carrot	1	16	
Tomato	1	14	

The QBE template to print the VEGETABLE in the increasing order of price is given later:

VEGETABLE	VEGETABLENAME	QUANTITY	PRICE
P.AO.			

The QBE template to print the VEGETABLE in the decreasing order of price is given later:

VEGETABLE	VEGETABLENAME	QUANTITY	PRICE
P.DO.			

Example 8. Retrieval using Negation

The symbol used for negation is +. For example print the quantity and price of the VEGETABLE that do not belong to Brinjal is given by:

VEGETABLE	VEGETABLENAME	QUANTITY	PRICE
+	Brinjal	P.	P.

Condition Box:

The condition box is used to store logical conditions that are not easily expressed in the table skeleton. A condition box can be obtained by pressing a special function key.

Example 9. Retrieval using condition box:

For example, if we want to print the quantity and price of the VEGETABLE, which is either Ladies Finger or Carrot, the condition box is used.

VEGETABLE	VEGETABLENAME	QUANTITY	PRICE
	VN	P.	P.

CONDITIONS

VN = Ladies Finger OR Carrot

Example 10. QBE Built-In Functions

QBE provides **MIN**, **MAX**, **CNT**, **SUM**, and **AVG** built-in functions:

- **MIN.ALL** implies the computation of minimum value of an attribute.
- **MAX.ALL** implies the computation of maximum value of an attribute.
- **CNT.ALL** implies COUNT the number of tuples in the relation.
- **SUM.ALL** implies the computation of sum of an attribute.
- **AVG.ALL** implies the computation of average value of an attribute.

Note: UNQ. which stands for unique operator is used to eliminate duplicates. For example, **CNT.UNQ.ALL** computes the number of tuples in the relation by eliminating duplicate values.

Example 10.1. MIN and MAX command

The QBE template to get the minimum and maximum vegetable price is given later:

VEGETABLE	VEGETABLE	QUANTITY	PRICE
	NAME		
			P.MIN.ALL.CX
			P.MAX.ALL.CY

Example 10.2. AVG command

The QBE template to get the average price of the vegetable is given later.

VEGETABLE	VEGETABLENAME	QUANTITY	PRICE
			P.AVG.CX

Example 10.3. CNT command

The QBE template to count the number of unique vegetables in the VEGETABLE relation is shown later.

VEGETABLE	VEGETABLENAME	QUANTITY	PRICE
P.CNT.UNQ.ALL			

Example 11. Update operation

The QBE template to increase the price of all vegetables by 10% is given as:

Here U. implies Update. The price UX of the vegetable is increased by 10% which is denoted by $1.1 * \underline{UX}$

VEGETABLE	VEGETABLENAME	QUANTITY	PRICE
U.			<u>UX</u> 1.1 * <u>UX</u>

Example 12. Record deletion

The QBE template to delete the record of all vegetables is shown later:

VEGETABLE	VEGETABLENAME	QUANTITY	PRICE
D.			

Here D. implies deletion of the entire relation.

Single Record Deletion

The QBE form to delete the record of the vegetable “Brinjal” is shown later:

VEGETABLE	VEGETABLENAME	QUANTITY	PRICE
D.	Brinjal		

Summary

In relational model, the data are stored in the form of tables or relations. Each table or relation has a unique name. Tables consist of a given number of columns or attributes. Every column of a table must have a name and no two columns of the same table may have identical names. The rows of the table are called tuples. The total number of columns or attributes that comprises a table is known as the *degree* of the table. The chapter has introduced the basic terminology used in relational model. Specific importance is given to E.F. Codd’s rule.

This chapter also introduced different integrity rules. Relational algebra concepts, different operators like SELECTION, PROJECTION, UNION, INTERSECTION, and JOIN operators were discussed with suitable examples. Relational calculus and its two branches, tuple relational calculus and domain relational calculus, were discussed in this chapter.

Finally, graphical user interface QBE, its relative advantage, different operations in QBE, concept of *condition box* in QBE, and aggregate functions in QBE were explained with suitable examples.

Review Questions

3.1. What is the degree and cardinality of the “Tennis Player” relation shown later:

Position	Player	Points	Nation
1	Federer	1117	Switzerland
2	Roddick, A.	671	USA
3	Hewitt, L.	638	Australia
4	Safin, M.	497	Russia
5	Moya, C.	484	Spain

Hint: Degree of the relation = Number of columns in the relation.
Cardinality of the relation = Number of rows in the relation.

3.2. A relation has a degree of 5 and cardinality of 7. How many attributes and tuples does the relation have?

3.3. A relation R has a degree of 3 and cardinality of 2 and the relation S has a degree of 2 and cardinality of 3, then what will be the degree and cardinality of the Cartesian product of R and S?

Ans: Cardinality = 6, Degree = 5.

3.4. What is the key of the following EMPLOYEE table?

EMPLOYEE				
EMPLOYEE NUMBER	EMPLOYEE NAME	DEPARTMENT	AGE	DESIGNATION
C100	Dr. Vijayarangan	Mechanical	51	Principal
C202	Dr. S. Jayaraman	ECE	50	Head
C203	Dr. Murugesh	EEE	50	Head
C204	Dr. Sivanandam	ComputerScience	53	Head
C208	Dr. Selvan	IT	51	Head

Ans: In the earlier table, EMPLOYEE NUMBER is the primary key. Because keys are used to enforce that no two rows are identical.

3.5. Define the operators in the core relational algebra?

3.6. Explain the following concepts in relational databases:

- Entity integrity constraint
- Foreign key and how it can specify a referential integrity constraint between two relations
- Semantic integrity constraint

3.7. Mention the pros and cons of relational data model?

Pros of relational data model:

1. The relational data model is a well formed and data independent model which is easy to use for applications which fit well into the model.
2. The data used by most business applications fits this model, and that business applications were the first large customers of database system explains the popularity of the model.

Cons of relational data model:

1. The simplicity of the model restricts the amount of semantics, which can be expressed directly by the database.
2. Different groups of information, or tables, must be joined in many cases to retrieve data.

3.8. Bring out the reasons, why relational model became more popular?

1. Relational model was based on strong mathematical background.
2. Relational model used the power of mathematical abstraction. Operations do not require user to know storage structures used.
3. Strong mathematical theory provides tool for improving design.
4. Basic structure of the relation is simple, easy to understand and implement.

3.9. A union, intersection or difference can only be performed between two relations if they are type compatible. What is meant by type compatibility? Give an example of two type compatible and two nontype compatible relations?

Two relations are type compatible if they have same set of attributes. Example of two type compatible relations is:

```
Men {<name:varchar>, <dob:date>, <address:varchar>}
Women {<name:varchar>, <dob:date>, <address:varchar>}
Example of two relations which are nontype compatible is:
Husband {<name:varchar>, <dob:date>, <salary: number>}
Wife {<name:varchar>, <dob:date>, <address:varchar>}
```

3.10. What are the advantages of QBE?

QBE can be considered as GUI (Graphical User Interface) based on domain calculus. QBE allows users to key in their input requests by filling in empty tables on the screen, and the system will also display its response in tabular form. QBE is user-friendly because the users are not required to formulate sentences for query requests with rigid query-language syntax.

3.11. What do you understand by domain integrity constraint?

The domain integrity constraints are used to specify the valid values that a column defined over the domain can take. We can define the valid values by listing them as a set of values (such as an enumerated data type in a strongly typed programming language), a range of values, or an expression that accepts the valid values.

3.12. What do you understand by “safety of expressions”?

It is possible to write tuple calculus expressions that generate infinite relations. For example $\{t/\sim t \in R\}$ results in an infinite relation if the domain of any attribute of relation R is infinite. To guard against the problem, we restrict the set of allowable expressions to safe expressions.

3.13. What are “quantifiers”? How will you classify them?

Quantifiers are words that refer to quantities such as “some” or “all” and tell for how many elements a given predicate is true. A predicate is a sentence that contains a finite number of variables and becomes a statement when specific values are substituted for the variables. Quantifiers can be broadly classified into two types (1) Universal Quantifier and (2) Existential Quantifier.

Structured Query Language

Learning Objectives. This chapter focuses on how to access the data within a DBMS. An introduction to SQL, an international standard language for manipulating relational database is given in this chapter. After completing this chapter the reader should be familiar with the following concepts in SQL.

- Evolution and benefits of SQL
- Datatypes in SQL
- SQL commands to create a table, inserting records into the table, and extracting information from the table
- Aggregate functions, GROUP BY clause
- Implementation of constraints in SQL using CHECK, PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE commands
- Concepts of sub query, view, and trigger

4.1 Introduction

SQL stands for “Structured Query Language.” The Structured Query Language is a relational database language. By itself, SQL does not make a DBMS. SQL is a medium which is used to communicate to the DBMS. SQL commands consist of English-like statements which are used to query, insert, update, and delete data. English-like statements mean that SQL commands resemble English language sentences in their construction and use and therefore are easy to learn and understand.

SQL is referred to as nonprocedural database language. Here nonprocedural means that, when we want to retrieve data from the database it is enough to tell SQL what data to be retrieved, rather than how to retrieve it. The DBMS will take care of locating the information in the database.

Commercial database management systems allow SQL to be used in two distinct ways. First, SQL commands can be typed at the command line directly. The DBMS interprets and processes the SQL commands immediately, and the results are displayed. This method of SQL processing is called interactive SQL. The second method is called programmatic SQL. Here, SQL

statements are embedded in a host language such as COBOL, FORTRAN, C, etc. SQL needs a host language because SQL is not a really complete computer programming language as such because it has no statements or constructs that allow branch or loop. The host language provides the necessary looping and branching structures and the interface with the user, while SQL provides the statements to communicate with the DBMS.

Some of the features of SQL are:

- SQL is a language used to interact with the database.
- SQL is a data access language.
- SQL is based on relational tuple calculus.
- SQL is a standard relational database management language.
- The first commercial DBMS that supported SQL was Oracle in 1979.
- SQL is a “nonprocedural” or “declarative” language.

4.2 History of SQL Standard

The origin of the SQL language date back to a research project conducted by IBM at their research laboratories in San Jose, California in the early 1970s. The aim of the project was to develop an experimental RDBMS which would eventually lead to a marketable product. At that time, there was a lot of interest in the relational model for databases at the academic level, in conferences and seminars. IBM, which already had a large share of the commercial database market with hierarchical and network model DBMSs, realized that the relational model would dominate the future database products. The project at IBM’s San Jose labs was started in 1974 and was named System R. A language called SEQUEL (Structured English QUery Language) was chosen as the relational database language for System R. A version of SEQUEL was developed at the IBM San Jose research facilities and tested with college students.

In November 1976, specifications for SEQUEL2 were published. In 1980 minor revisions were made to SEQUEL, and it was renamed “SQL.” SEQUEL was renamed to SQL because the name SEQUEL had already been used for hardware product. In order to avoid confusion and legal problems SEQUEL was renamed to SQL. In the first phase of the System R project, researchers concentrated on developing a basic version of the RDBMS. The main aim at this stage was to verify that the theories of the relational model could be translated into a working, commercially viable product. This first phase was successfully completed by the end of 1975, and resulted in a single-user DBMS based on the relational model. The System R project was completed in 1979. The theoretical work of the System R project resulted in the development and release of IBM’s first commercial relational database management system in 1981. The product was called SQL/DS (Structured Query Language/Data Store) and ran under the DOS/VSE operating system environment. Two years later, IBM announced a version of SQL/DS for VM/CMS operating system.

In 1983, IBM released a second SQL-based RDBMS called DB2, which ran under the MVS operating system. DB2 quickly gained widespread popularity and even today, versions of DB2 form the basis of many database systems found in large corporate data-centers. During the development of System R and SQL/DS, other companies were also at work creating their own relational database management systems. Some of them, Oracle being an example, even implemented SQL as the relational database language for their DBMSs concurrently with IBM. Later on, SQL language was standardized by ANSI and ISO. The ANSI SQL standards were first published in 1986 and updated in 1989, 1992, and 1999.

4.2.1 Benefits of Standardized Relational Language

The main advantages of standardized language are given below.

1. Reduced training cost
2. Enhanced productivity
3. Application portability
Application portability means applications can be moved from machine to machine when each machine uses SQL.
4. Application longevity
A standard language tends to remain so for a long time, hence there will be little pressure to rewrite old applications.
5. Reduced dependence on a single vendor

SQL language development is given in a nutshell below:

1. In 1970 E.F. Codd of IBM released a paper “A relational model of data for large shared data banks.” IBM started the project System R to demonstrate the feasibility of implementing the relational model in a database management system. The language used in system R project was SEQUEL. SEQUEL was renamed SQL during the project, which took place from 1974 to 1979.
2. The first commercial RDBMS from IBM was SQL/DS. It was available in 1981.
3. Oracle from relational software (now Oracle corporation) was on the market before SQL/DS, i.e., 1979.
4. Other products included INGRES from relational Technology Sybase from Sybase, Inc. (1986), DG/SQL from Data General Corporation (1984).

4.3 Commands in SQL

SQL commands can be classified in to three types:

1. Data Definition Language commands (DDL)
2. Data Manipulation Language commands (DML)
3. Data Control Language commands (DCL)

DDL

DDL commands are used to define a database, including creating, altering, and dropping tables and establishing constraints.

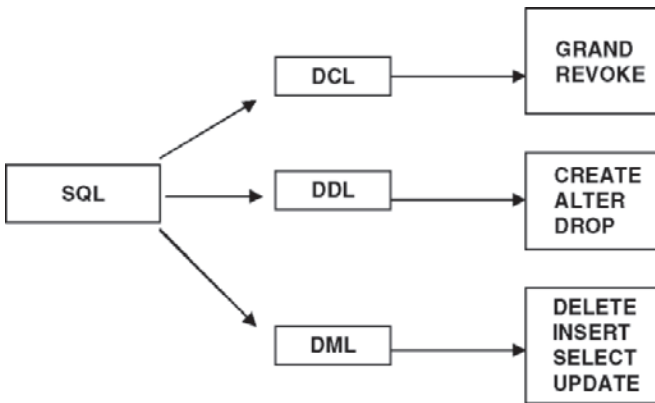
DML

DML commands are used to maintain and query a database, including updating, inserting, modifying, and querying data.

DCL

DCL commands are used to control a database including administering privileges and saving of data. DCL commands are used to determine whether a user is allowed to carry out a particular operation or not. The ANSI standard groups these commands as being part of the DDL.

The classification of commands in SQL is shown below.



4.4 Datatypes in SQL

In relational model the data are stored in the form of tables. A table is composed of rows and columns. When we create a table we must specify a datatype for each of its columns. These datatypes define the domain of values that each column can take. Oracle provides a number of built-in datatypes as well as several categories for user-defined types that can be used as datatypes. Some of the built-in datatypes are string datatype to store characters, number datatype to store numerical value, and date and time datatype to store when the event happened (history, date of birth, etc.).

STRING

In string we have CHAR and VARCHAR datatypes. Character datatype store data which are words and free-form text, in the database character set.

CHAR Datatype

The CHAR datatype specifies a fixed-length character string. The syntax of CHAR datatype declaration is:

CHAR (n) – Fixed length character data, “n” characters long.

Here “n” specifies the character length. If we insert a value that is shorter than the column length, then Oracle blank-pads the value to column length. If we try to insert a value that is too long for the column then Oracle returns error message.

VARCHAR2 Datatype

The VARCHAR2 datatype specifies a variable-length character string. The syntax of VARCHAR2 datatype declaration is:

VARCHAR2 (n) – Variable length character of “n” length.

Here “n” specifies the character length.

VARCHAR vs. VARCHAR2

The VARCHAR datatype behaves like VARCHAR2 datatype in the current version of Oracle.

In order to justify the above statement, let us create a table CHAMPION, which refers to Wimbledon Champions. The attributes of the table CHAMPION are *Name*, *Nation*, *Year* (the year in which the sportsman has won the title). For our example, let us use the datatype VARCHAR for the attribute Name and VARCHAR2 for the datatype Nation. The SQL command to create CHAMPION is shown in Fig. 4.1.

Now let us try to see the description of the table. The description of the table is shown in Fig. 4.2.

From Fig. 4.2, it is clear that both name and nation are stored as VARCHAR2(12). This means that VARCHAR datatype in the Oracle 8i version behaves the same as VARCHAR2.

NUMBER Datatype

The NUMBER datatype stores zero, positive, and negative fixed and floating point numbers.

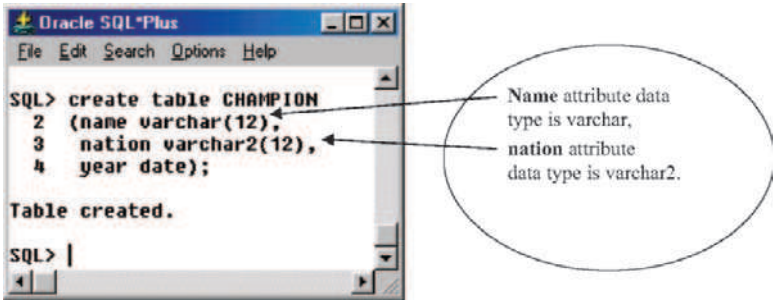


Fig. 4.1. CHAR and VARCHAR2 datatype



Fig. 4.2. Table description

The syntax to store fixed-point number is `NUMBER (p, q)` where “p” is the total number of digits and “q” is the number of digits to the right of decimal point.

The syntax to specify an integer is `NUMBER (p)`.

DATE Datatype

The `DATE` datatype is used to store the date and time information. For each `DATE` value, Oracle stores the century, year, month, date, hour, minute, and second information. The ANSI date literal contains no time portion, and must be specified in `YYYY-MM-DD` format where Y stands for Year, M for month, and D for date.

TIME STAMP Datatype

The `TIME STAMP` datatype is used to store both date and time. It stores the year, month, and day of the `DATE` datatype, and also hour, minute, and second values.

LOB Datatype

Multimedia data like sound, picture, and video need more storage space. The LOB datatypes such as `BLOB`, `CLOB`, and `BFILE` allows us to store large block of data.

BLOB Datatype

The BLOB datatype stores unstructured binary data in the database. BLOBs can store up to 4 GB of binary data.

CLOB Datatype

The CLOB datatype can store up to 4 GB of character data in the database.

BFILE Datatype

The BFILE datatype stores unstructured binary data in operating system files outside the database. A BFILE can store up to 4 GB of data.

4.5 Data Definition Language (DDL)

The Data Definition Language is

- Used to define schemas, relations, and other database structures
- Also used to update these structures as the database evolves

Examples of Structure Created by DDL

The different structures that are created by DDL are Tables, Views, Sequences, Triggers, Indexes, etc.

1. Tables

The main features of table are:

- It is a relation that is used to store records of related data. It is a logical structure maintained by the database manager.
- It is made up of columns and rows.
- At the intersection of every column and row there is a specific data item called a value.
- A base table is created with the CREATE TABLE statement and is used to hold persistent user data.

2. Views

The basic concepts of VIEW are:

- It is a stored SQL query used as a “Virtual table.”
- It provides an alternative way of looking at the data in one or more tables.
- It is a named specification of a result table. The specification is a SELECT statement that is executed whenever the view is referenced in an SQL statement. Consider a view to have columns and rows just like a base table. For retrieval, all views can be used just like base tables.

- When the column of a view is directly derived from the column of a base table, that column inherits any constraints that apply to the column of the base table. For example, if a view includes a foreign key of its base table, INSERT and UPDATE operations using that view are subject to the same referential constraints as the base table. Also, if the base table of a view is a parent table, DELETE and UPDATE operations using that view are subject to the same rule as DELETE and UPDATE operations on the base table.

3. Sequences

- A sequence is an integer that varies by a given constant value. Typically used for unique ID assignment

4. Triggers

- Trigger automatically executes certain commands when given conditions are met.

5. Indexes

- Indexes are basically used for performance tuning. Indexes play a crucial role in fast data retrieval.

Create Table Command

- The CREATE TABLE command is used to implement the schemas of individual relations.

Steps in Table Creation

1. Identify datatypes for attributes
2. Identify columns that can and cannot be null
3. Identify columns that must be unique
4. Identify primary key–foreign key mates
5. Determine default values
6. Identify constraints on columns (domain specifications)
7. Create the table

Syntax

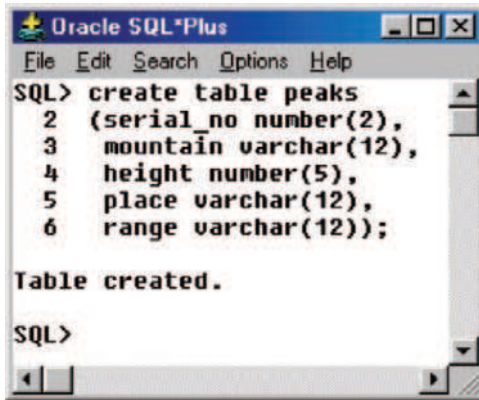
```
CREATE TABLE table name
(column-name1      data-type-1      [constraint],
column-name2      data-type-2      [constraint],
column-nameN      data-type-N      [constraint]
);
```

Example Table

See Table 4.1.

Table 4.1. Peaks of the world

Serial number	Peak	Mountain range	Place	Height
1	Everest	Himalayas	Nepal	8,848
2	Godwin Austin	Karakoram	India	8,611
3	Kanchenjunga	Himalayas	Nepal	8,579



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> create table peaks
2  (serial_no number(2),
3  mountain varchar(12),
4  height number(5),
5  place varchar(12),
6  range varchar(12));

Table created.

SQL>

```

Fig. 4.3. Table creation example

Syntax to Create the Table

The general syntax to create the table is given below. Here the key words are shown in bold and capital letters.

```

CREATE TABLE table name
(column name1    data type    (size),
column name2    data type    (size),
column name N   data type    (size));

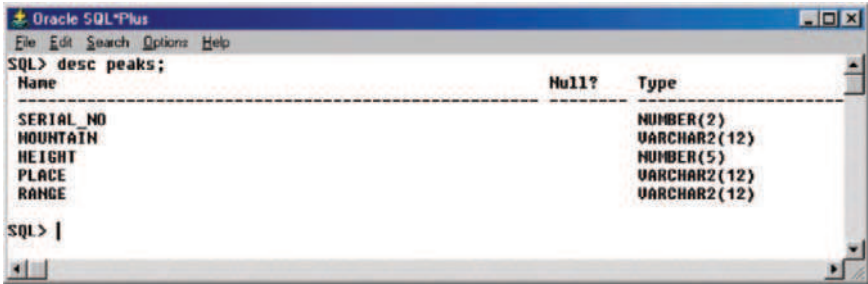
```

Example

The SQL command to define Table 4.1 is shown in Fig. 4.3. In this example the name of the table is *peaks*. The table has five columns which are serial number, name of the mountain (peak), height, place where the mountain is situated, range of the mountain.

To see the description of the table

To see the description of the table we have created we have the command **DESC**. Here **DESC** stands for description of the table. The syntax of **DESC** command is:

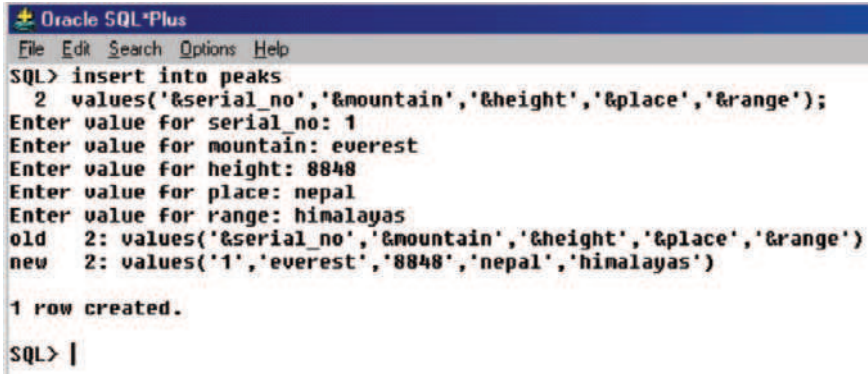


```

Oracle SQL*Plus
File Edit Search Options Help
SQL> desc peaks;
Name                                     Null?    Type
-----
SERIAL_NO                               NUMBER(2)
MOUNTAIN                                VARCHAR2(12)
HEIGHT                                  NUMBER(5)
PLACE                                    VARCHAR2(12)
RANGE                                    VARCHAR2(12)
SQL> |

```

Fig. 4.4. Table description



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> insert into peaks
  2 values('&serial_no','&mountain','&height','&place','&range');
Enter value for serial no: 1
Enter value for mountain: everest
Enter value for height: 8848
Enter value for place: nepal
Enter value for range: himalayas
old 2: values('&serial_no','&mountain','&height','&place','&range')
new 2: values('1','everest','8848','nepal','himalayas')

1 row created.
SQL> |

```

Fig. 4.5. Inserting values into the table

Syntax: DESC table name;

The DESC command returns the attributes (columns) of the table, the datatype associated with the column, and also any constraint (if any) imposed on the column. Figure 4.4 shows the description of the table PEAKS.

To insert values into the table

Syntax: Insert into <tablename> values ('&columnname1', '&columnname2', &col3,...);

(e.g.) The SQL syntax and the corresponding output are shown in Fig. 4.5. Now to insert the next set of values, use the slash as shown in Fig. 4.6.

To view the entire table

The SQL syntax to see all the columns of the table is:

SELECT * FROM table name;

Here the asterisk symbol indicates the selection of all the columns of the table.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> /
Enter value for serial_no: 2
Enter value for mountain: godwinaustin
Enter value for height: 8611
Enter value for place: india
Enter value for range: karakoram
old 2: values('&serial_no','&mountain','&height','&place','&range')
new 2: values('2','godwinaustin','8611','india','karakoram')

1 row created.

SQL> |

```

Fig. 4.6. Inserting successive values into the table

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select * from peaks;

SERIAL_NO MOUNTAIN          HEIGHT PLACE          RANGE
-----
1 everest             8848 nepal          himalayas
2 godwinaustin       8611 india          karakoram
3 kanchenjunga      8579 nepal          himalayas

SQL> |

```

Fig. 4.7. SELECTION of all columns of the table

Example

The SQL command to see all the columns of the table PEAKS and the corresponding output are shown in Fig. 4.7.

```
SQL> select * from peaks;
```

4.6 Selection Operation

Selection operation can be considered as row wise filtering. We can select specific row(s) using condition.

Syntax of SELECTION Operation

```
SELECT * FROM table name
WHERE condition;
```

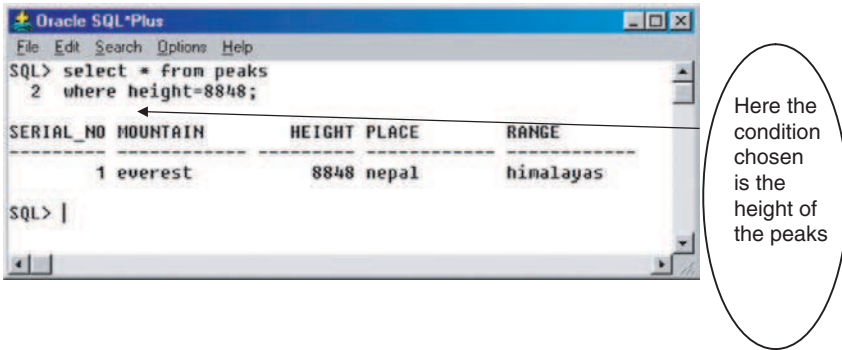


Fig. 4.8. SELECTION operation

Example of SELECTION operation

In the example Table 4.1, there are three rows. Let us filter two rows so that only one row will appear in the result. Here the condition used to filter the rows is the “height” of the PEAKS. The SQL command to implement SELECTION operation and the corresponding output are shown in Fig. 4.8.

From Fig. 4.8 it is clear that even though there are three rows in the Table 4.1, it is reduced to one using the condition the height of the peaks. This operation which filters the rows of the relation is called SELECTION.

4.7 Projection Operation

The projection operation performs column wise filtering. Specific columns are selected in projection operation.

Syntax of PROJECTION Operation

SELECT column name1, column name2, Column name N **FROM** table name;

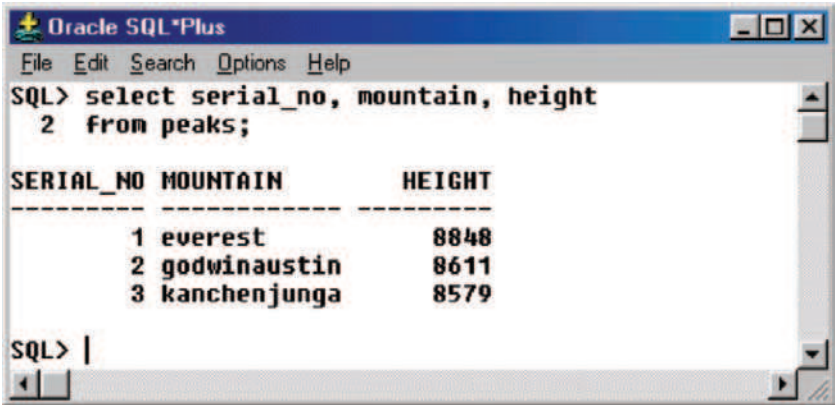
If all the columns of the table are selected, then it cannot be considered as PROJECTION.

The SQL command to perform PROJECTION operation on the relation PEAKS and the corresponding results are shown in Fig. 4.9.

From Fig. 4.9, it is clear that only three columns are selected in the result, even though there are five columns in the Table 4.1.

SELECTION and PROJECTION Operation

We can perform both selection and projection operation in a relation. If we combine selection and projection operation means naturally we are restricting the number of rows and the columns of the relation.



```

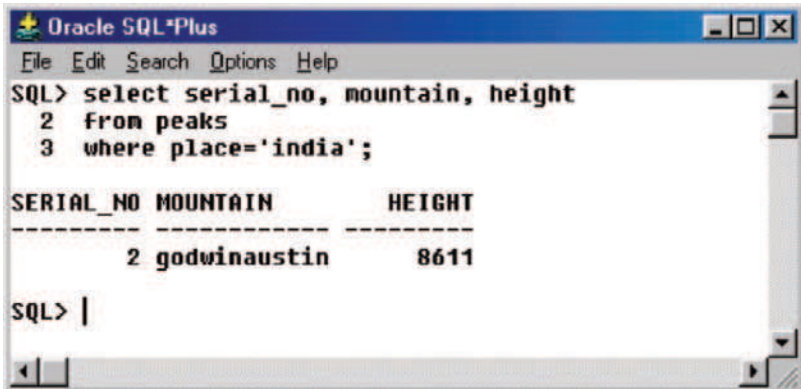
Oracle SQL*Plus
File Edit Search Options Help
SQL> select serial_no, mountain, height
2 from peaks;

SERIAL_NO MOUNTAIN          HEIGHT
-----
1 everest                8848
2 godwinaustin           8611
3 kanchenjunga           8579

SQL> |

```

Fig. 4.9. PROJECTION operation



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select serial_no, mountain, height
2 from peaks
3 where place='india';

SERIAL_NO MOUNTAIN          HEIGHT
-----
2 godwinaustin           8611

SQL> |

```

Fig. 4.10. SELECTION and PROJECTION operation

Syntax for Selection and Projection

```

SELECT column name1, column name 2. .... column name N
FROM table name
WHERE condition;

```

The selection and projection operation applied to the peaks relation is shown in Fig. 4.10.

From Fig. 4.10, we can observe that the selection operation is based on the “place” of the peaks. As a result only one row is obtained as the result. Because of projection operation only three columns are obtained in the result as shown in Fig. 4.10.

4.8 Aggregate Functions

SQL provides seven built-in functions to facilitate query processing. The seven built-in functions are COUNT, MAX, MIN, SUM, AVG, STDDEV, and VARIANCE. The uses of the built-in functions are shown in Table 4.2.

4.8.1 COUNT Function

The built-in function returns the number of rows of the table. There are variations of COUNT function. First let us consider COUNT (*) function. In order to understand the COUNT (*) function consider the relation PERSON_SKILL as shown in Table 4.3, the relation PERSON has only two columns, name of the person and skills associated with the person. It is to be noted that some persons may have more than one skill and some persons may not have any skills.

From Table 4.3, we can observe that the table PERSON_SKILL has six rows and two columns and the person Ashok has more than one skill and Sam has no skill hence a NULL is inserted against Sam.

(A) COUNT (*) Function

The syntax of Count (*) function is:

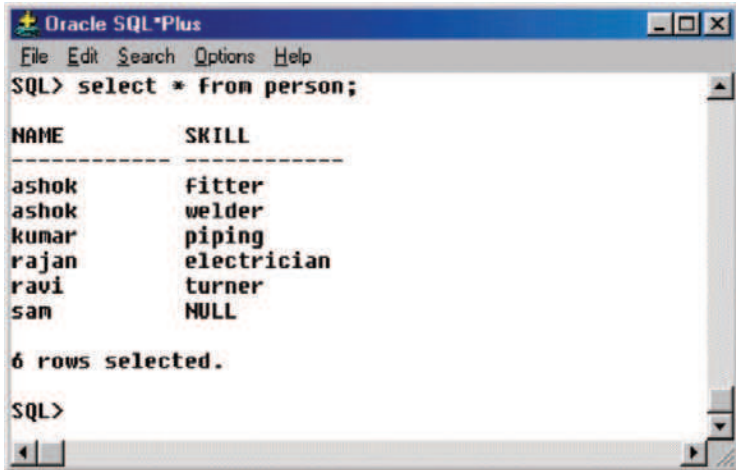
```
SELECT COUNT (*)
FROM table name;
```

Table 4.2. Built-in functions

Serial number	Built-in function	Use
1	COUNT	to count the number of rows of the relation
2	MAX	to find the maximum value of the attribute (column)
3	MIN	to find the minimum value of the attribute
4	SUM	to find the sum of values of the attribute provided the datatype of the attribute is number
5	AVG	to find the average of n values, ignoring null values
6	STDDEV	standard deviation of n values ignoring null values
7	VARIANCE	variance of n values ignoring null values

Table 4.3. PERSON_SKILL

Name	Skill
Ashok	fitter
Ashok	welder
Kumar	pipng
Rajan	electrician
Ravi	turner
Sam	NULL

**Fig. 4.11.** PERSON table

Now let us try to view the table PERSON, and the contents of the table PERSON as shown in Fig. 4.11. From this figure, it is clear that the number of rows of the table is six.

Now let us use the COUNT (*) function to view the number of rows of the relation PERSON. The SQL command and the corresponding output are shown in Fig. 4.12.

From Fig. 4.12, we can observe that the number of rows returned is six, which means that the COUNT(*) function takes into account the NULL values.

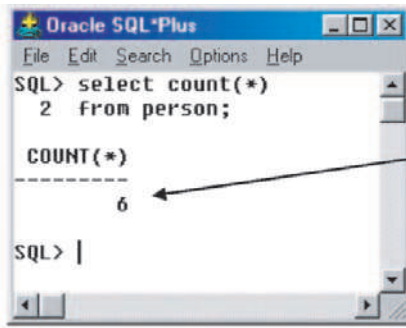
(B) COUNT (attribute name) Function

A slight variation of COUNT (*) function is COUNT (attribute name) function. The syntax of this function is given by:

```

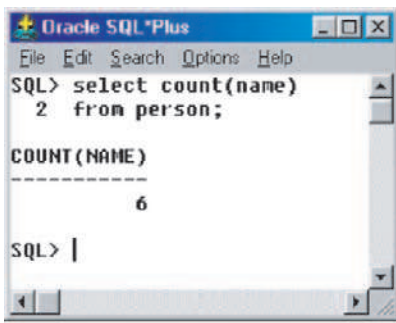
SELECT COUNT (attribute name)
FROM table name;

```



See the number of rows returned is six. This means NULL values are taken into account.

Fig. 4.12. COUNT (*) Function



COUNT (attribute name) command also returns the number of rows of the relation without taking into consideration the NULL values.

Fig. 4.13. SELECT (attribute name) command

The application of COUNT (attribute name) to the PERSON table and the corresponding output are shown in Fig. 4.13.

From Fig. 4.13, it is clear that count (attribute name) command will take NULL values into account as a result the number of rows selected is six.

(C) COUNT (DISTINCT attribute name)

The COUNT (DISTINCT attribute name) command returns the number of rows of the relation, by eliminating duplicate values. The syntax of COUNT (DISTINCT attribute name) is:

```

SELECT COUNT (DISTINCT attribute name)
FROM table name;

```

The usage of COUNT (DISTINCT attribute name) in the table PERSON and the corresponding output is shown in Fig. 4.14.

It is worthwhile to note that the DISTINCT command will not take into consideration the NULL value. In order to prove this, let us select the attribute be skill rather than the attribute name. The result of choosing the attribute as skill is show in Fig. 4.15.


```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select count(distinct name)
      2  from person;

COUNT(DISTINCTNAME)
-----
                    5

SQL> |

```

DISTINCT key word eliminates the duplicate value as a result the number of rows returned is five.

Fig. 4.14. COUNT (DISTINCT attribute name)

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select count(distinct skill)
      2  from person;

COUNT(DISTINCTSKILL)
-----
                    6

SQL>

```

The result of COUNT (DISTINCT skill) statement returns the number of rows to be six. This shows that DISTINCT command will not take into consideration the NULL value.

Fig. 4.15. COUNT command

4.8.2 MAX, MIN, and AVG Aggregate Function

In order to understand MAX, MIN, and AVG aggregate function consider the relation CONSUMER PRODUCTS. The relation CONSUMER PRODUCTS has two attributes, the name of the product and the price associated with the product as shown in Table 4.4.

(A) MAX Command

The MAX command stands for maximum value. The MAX command returns the maximum value of an attribute. The syntax of MAX command is:

```

SELECT MAX (attribute name)
FROM table name;

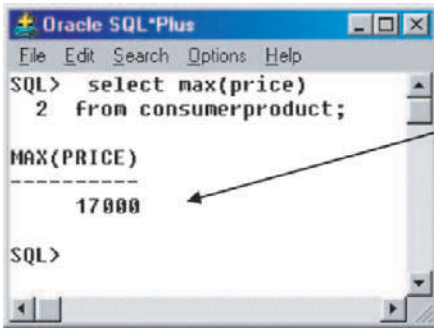
```

Let us apply the MAX command to Table 4.4 to get the maximum price of the product, the SQL command and the corresponding output are shown in Fig. 4.16.

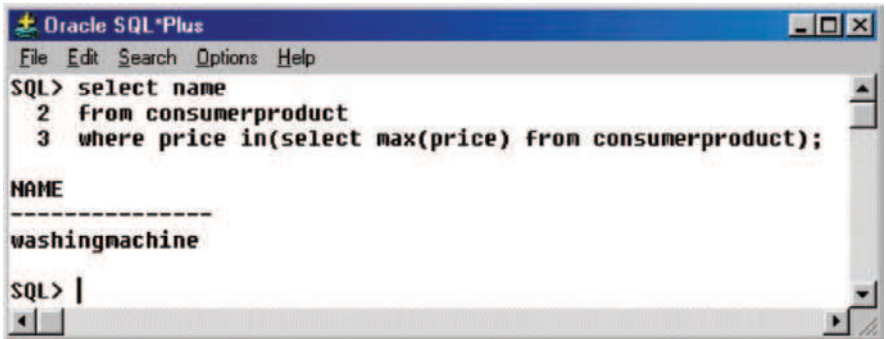
Let us try to find the name of the product which has maximum price by using PROJECTION operation and the IN operator as shown in Fig. 4.17.

Table 4.4. Consumer product

Name	Price (in Rs.)
TV	15,000
refrigerator	10,000
washing machine	17,000
mixie	3,500



The MAX command returns the maximum price of the product which is 17000 (Refer table 4.4)

Fig. 4.16. MAX command**Fig. 4.17.** Maximum price product name

(B) MIN Command

The MIN command is used to return the minimum value of an attribute. The syntax of MIN command is same as MAX command.

Syntax of MIN Command is

```

SELECT MIN (attribute name)
FROM table name;

```

The use of MIN command and the corresponding result are shown in Fig. 4.18.

From Table 4.4 the minimum price of the product is 3,500 which are returned as the result.

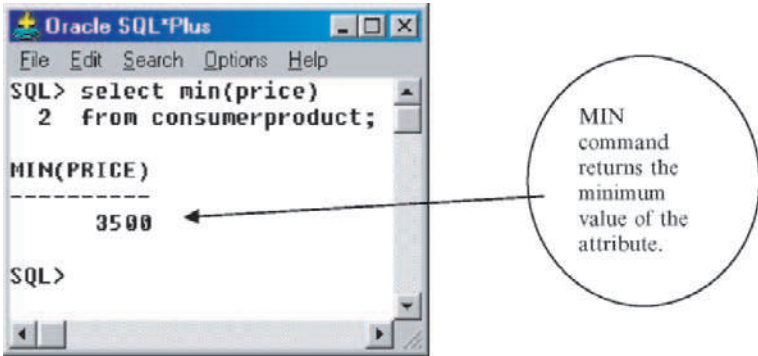


Fig. 4.18. MIN command applied to Table 4.4

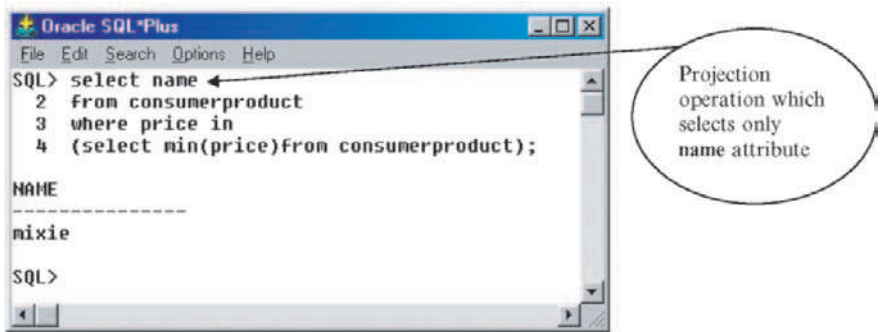


Fig. 4.19. Minimum price product name

To know the name of the product which has minimum price, we can use IN operator as shown in Fig. 4.19.

From Fig. 4.19, it is clear that we can use IN operator along with PROJECTION operation to get the name of the product with minimum price.

(C) AVG Command

The AVG command is used to get the average value of an attribute. The syntax of AVG command is:

```

SELECT AVG (attribute name)
FROM table name;

```

Let us apply AVG command to the Table 4.4, to get the average price of the product. The result of applying AVG command is shown in Fig. 4.20. The average price of the product is $(15,000 + 10,000 + 17,000 + 3,500)/4$ which is 11,375 as shown in Fig. 4.20.

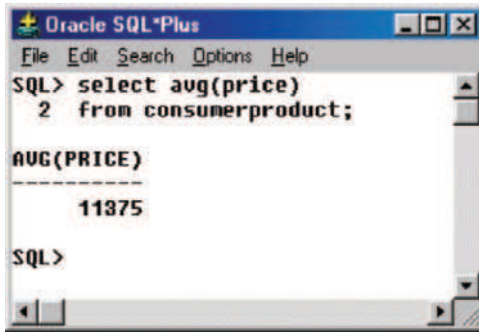


Fig. 4.20. AVG command

(D) STDDEV Function

The STDDEV function is used to compute the standard deviation of the attribute values. The syntax of the standard deviation function is:

```
SELECT STDDEV (attribute name)
FROM table name;
```

The STDDEV function applied to the relation CONSUMERPRODUCT (Table 4.4) is shown in Fig. 4.21.

(E) VARIANCE Function

The variance function is used to get the variance of the attribute values. The syntax of VARIANCE function is:

```
VARIANCE (attribute name)
FROM table name;
```

Let us apply the VARIANCE to the consumer product table; the result is shown in Fig. 4.22. We know that the variance is the square of the standard deviation. We have obtained the standard deviation from Fig. 4.21 as 6019.0669; the square of this value is approximately 36229167 which is obtained in Fig. 4.22.

(F) GROUP BY Function

The GROUP BY clause is used to group rows to compute group-statistics. It is to be noted that when the GROUP BY clause is present, then the SELECT clause may include only the columns that appear in the GROUP BY clause and aggregate functions.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select stddev(price)
      2 from consumerproduct;

STDDEV(PRICE)
-----
      6019.0669

SQL> |

```

Fig. 4.21. STDDEV function

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select variance(price)
      2 from consumerproduct;

VARIANCE(PRICE)
-----
      36229167

SQL>

```

Fig. 4.22. Variance function

In order to understand the GROUP BY Function let us consider the table PLACEMENT as shown in Table 4.5 which refers to the number students placed in different companies. The table PLACEMENT consists of three attributes (columns) which are company name, department name which refers to the curriculum stream and strength which refers to the number of students placed.

Now we want to know the total number of students placed in each branch. For this we can use the GROUP BY command. The syntax of GROUP BY command is:

```

SELECT attribute name, aggregate function
FROM table name
GROUP BY attribute name;

```

Table 4.5. Placement

Company name	Department	Strength
TCS	CSE	54
TCS	ECE	40
TCS	EEE	32
GE	CSE	5
GE	ECE	8
GE	EEE	20
L&T	CSE	12
L&T	ECE	20
L&T	EEE	18
IBM	CSE	24
IBM	ECE	20
IBM	EEE	12

It is to be noted that the attribute name after SELECT command should match with the attribute name after GROUP BY command. The GROUP BY command which is used to find the total number of students placed in each branch is shown in Fig. 4.23.

(G) HAVING Command

The HAVING command is used to select the group. In other words HAVING restricts the groups according to a specified condition. The syntax of HAVING command is:

```
SELECT attribute name, aggregate function
FROM table name
GROUP BY attribute name
HAVING condition;
```

Let us use the HAVING command as shown in Fig. 4.24 to find the details of the department in which more than 90 students got placement.

From Fig. 4.24, we are able to get the details of the department where more than 90 students were placed.

(H) SORTING of Results

The SQL command ORDER BY is used to sort the result in ascending or descending order.

The table used to understand ORDER BY command is BESTCRICKETER. The table BESTCRICKETER as shown in Table 4.6 gives the details of best batsman of the world. The attributes of the BESTCRICKETER are the name of the batsman, the country they belong to, and the number of centuries they scored.

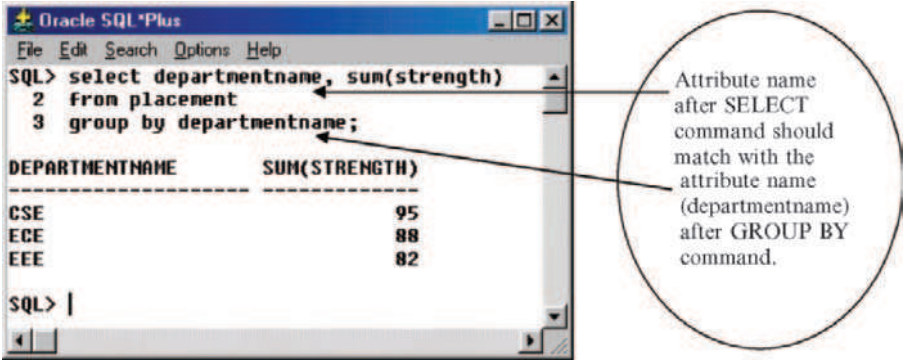


Fig. 4.23. GROUP BY command

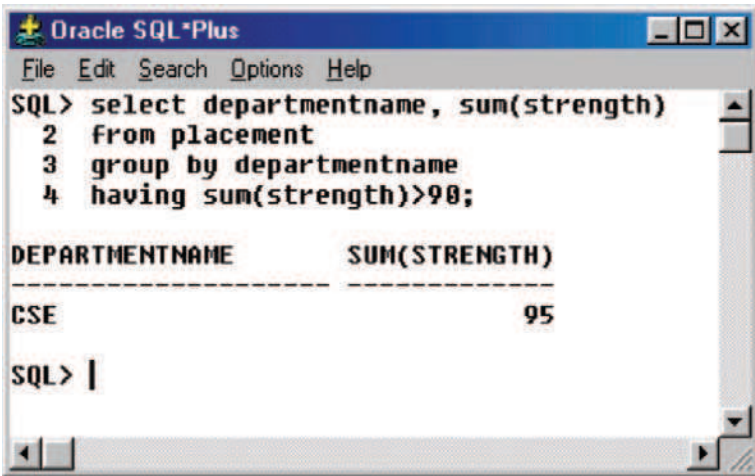


Fig. 4.24. GROUP BY and HAVING command

Table 4.6. BESTCRICKETER

Name	Country	Centuries
Gavaskar	India	34
Sobers	Westindies	26
Chappel	Australia	24
Bradman	Australia	29
Border	Australia	27
Gooch	England	20

Case 1: The syntax of ORDER BY command to arrange the result in ascending order is:

```
SELECT *
FROM table name
ORDER BY attribute name ASC;
```

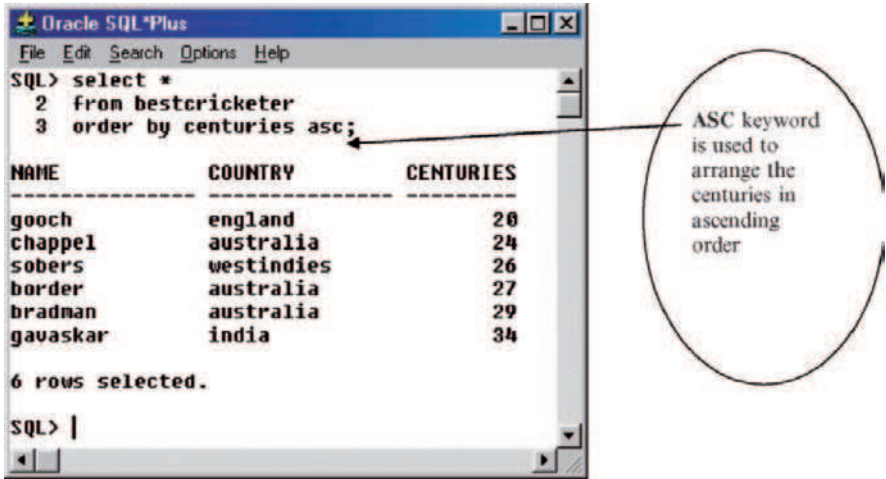


Fig. 4.25. Sorting in ascending order

Here ASC stands for ascending order.

Let us apply the command to the Table 4.6, the result of using ORDER BY command and the corresponding results are shown in Fig. 4.25.

Case 2: The syntax to arrange the result in descending order is:

```
SELECT *
FROM table name
ORDER BY attribute name DESC.
```

Here DESC stands for descending order.

Let us apply this DESC keyword to arrange the centuries in descending order. The SQL command and the corresponding output are shown in Fig. 4.26.

Case 3: If we do not specify as ASC or DESC after ORDER BY key word, by default, the results will be arranged in ascending order.

From Fig. 4.27, it is evident that if nothing is specified as ASC or DESC then by default, the results will be displayed in ascending order.

(I) Range Queries Using Between

The SQL has built-in command BETWEEN which is used to perform range queries.

Let us try to find the details of the batsman who has scored centuries greater than 20 and less than 30. The SQL command to accomplish this task and the corresponding output are shown in Fig. 4.28.

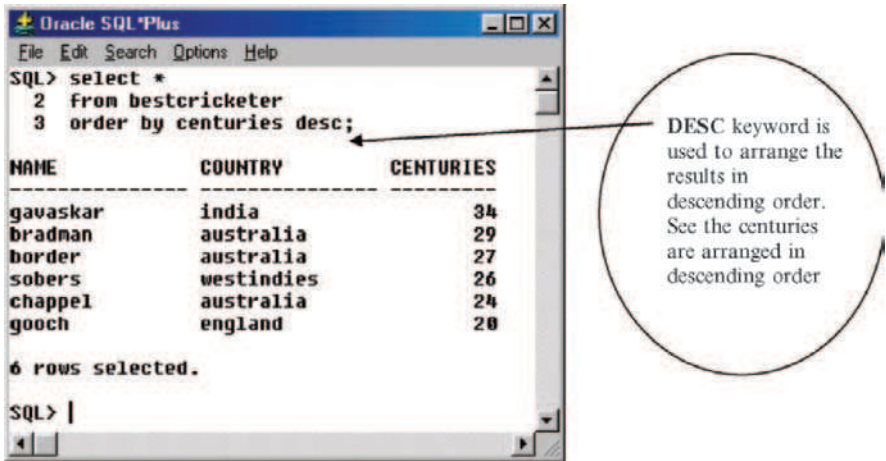


Fig. 4.26. Sorting in descending order

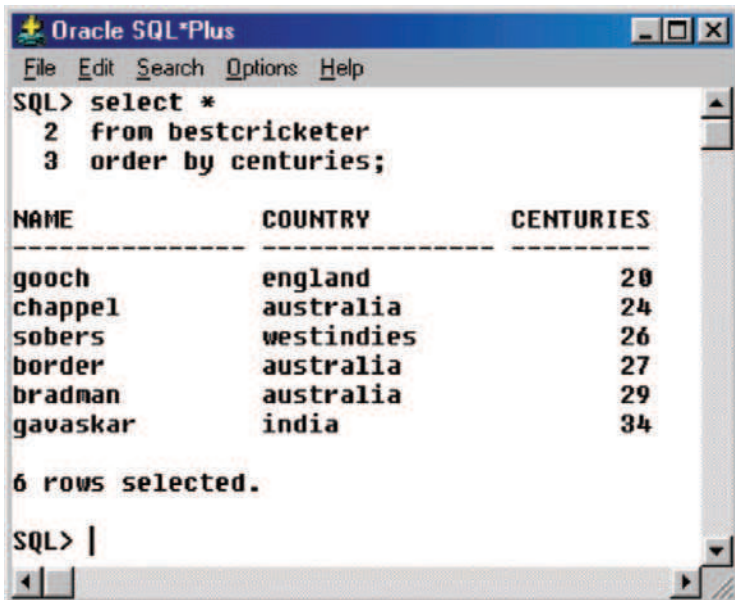


Fig. 4.27. Ascending order

4.9 Data Manipulation Language

The data manipulation language is used to add, update, and delete data in the database. The SQL command INSERT is used to add data into the database, the SQL command UPDATE is used to modify the data in the database, and the SQL command DELETE is used to delete data in the database. Here the term database refers to the table.

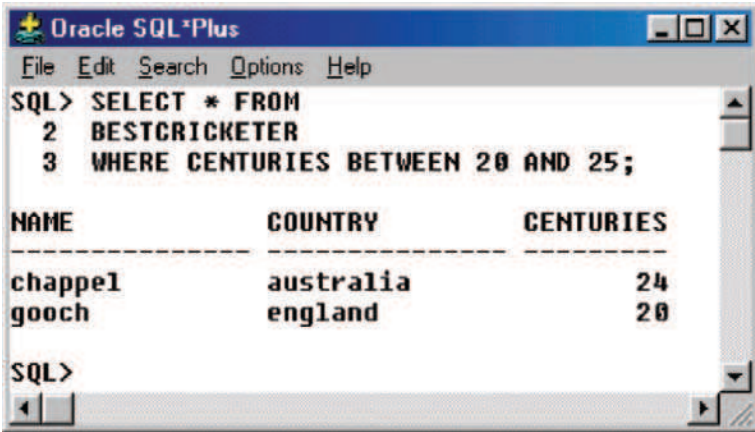


Fig. 4.28. Range query using BETWEEN command

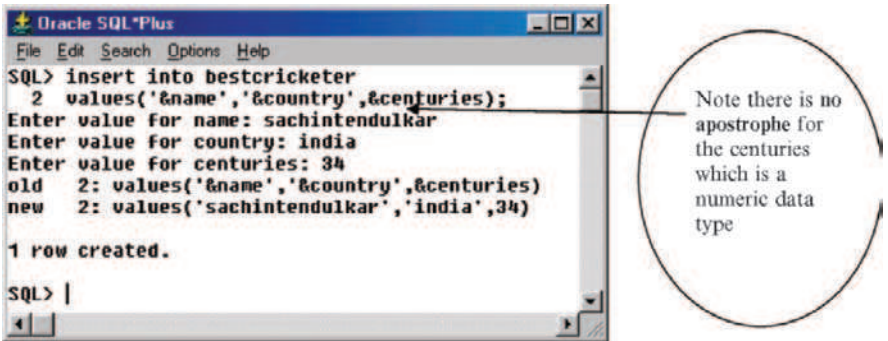


Fig. 4.29. Inserting a new row to the table

4.9.1 Adding a New Row to the Table

The INSERT command is to add new row to the table. The syntax of INSERT command is:

```

INSERT INTO table name
VALUES ('&column1-name', '&column2-name'... &columnN-name);
    
```

It is to be noted that apostrophe is not required for numeric datatype.

Let us try to insert a new row to the Table 4.6 (which has already six rows) to include the little master Sachin Tendulkar. The SQL command and the corresponding output are shown in Fig. 4.29.

To verify whether the new row has been added to the Table 4.6 which had six rows before inserting the new row, let us issue SELECT command as shown in Fig. 4.30.

From Fig. 4.30, it is clear that little master Sachin Tendulkar record being added to the best cricketer table so that the total number of rows is seven.

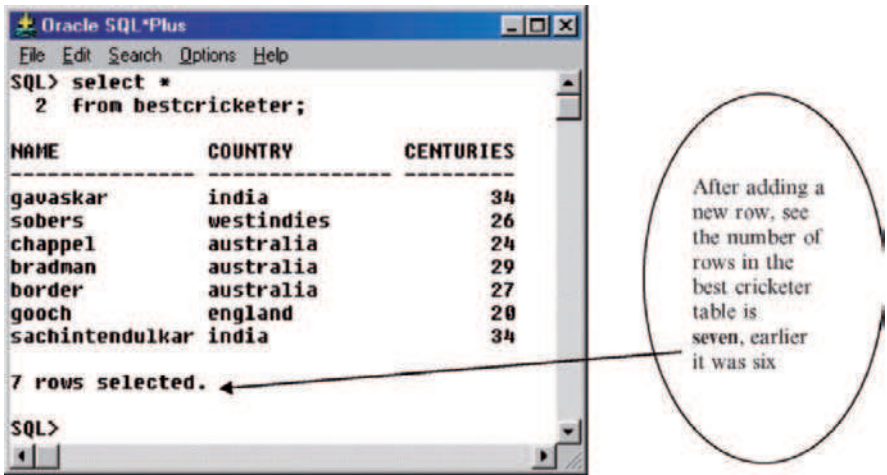


Fig. 4.30. Modified table

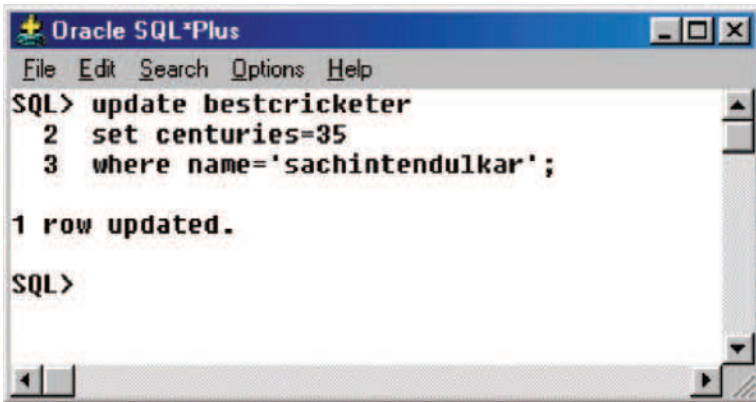


Fig. 4.31. Table updation using UPDATE command

4.9.2 Updating the Data in the Table

The data in the table can be updated by using UPDATE command. The syntax of the UPDATE command is:

```
UPDATE table name
SET attribute value=new value
WHERE condition;
```

Let us apply this UPDATE command to the table BESTCRICKETER. The motive is to modify the number of centuries hit by Sachin Tendulkar to 35. The corresponding SQL command and the output are shown in Fig. 4.31.

The screenshot shows the Oracle SQL*Plus interface. The command entered is `SQL> select * from bestcricketer;`. The output is a table with 7 rows. The row for Sachin Tendulkar has 35 centuries, which is circled in red in the original image. A callout box points to this value with the text: "The number of centuries scored by Sachin Tendulkar has been updated to 35 which were 34 earlier."

NAME	COUNTRY	CENTURIES
gavaskar	india	34
sobers	westindies	26
chappel	australia	24
bradman	australia	29
border	australia	27
gooch	england	20
sachintendulkar	india	35

7 rows selected.

Fig. 4.32. Updated table BESTCRICKETER

To see whether the table has been updated or not use `SELECT` statement to view the content of the table `BESTCRICKETER`. The updated table is shown in Fig. 4.32.

4.9.3 Deleting Row from the Table

The `DELETE` command in SQL is used to delete row(s) from the table. The syntax of `DELETE` command is

```
DELETE FROM table name
WHERE condition;
```

Let us delete the record of a particular player (say Gooch) from the table `BESTCRICKETER`. The SQL command to delete a particular row and the corresponding output are shown in Fig. 4.33.

To verify whether the player Gooch record has been deleted, let us use `SELECT` command to view the content of the table as shown in Fig. 4.34. From this figure it is evident that the player Gooch record has been successfully deleted.

4.10 Table Modification Commands

We can use `ALTER` command to alter the structure of the table, that is we can add a new column to the table. It is also possible to delete the column from the table using `DROP COLUMN` command.

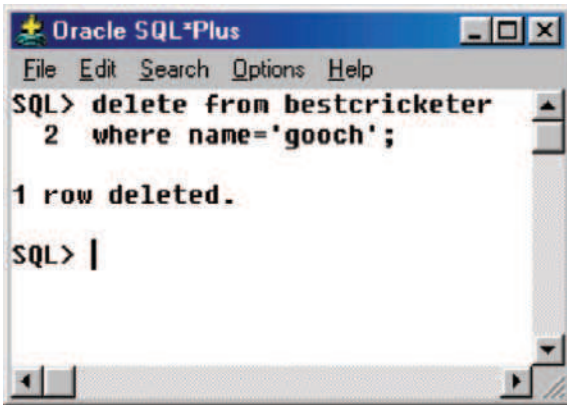
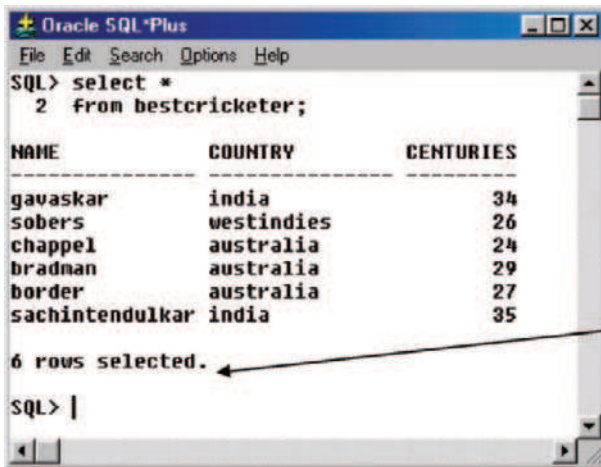


Fig. 4.33. Deletion of row from table



Because of deletion of the player gooch record the number of rows selected is six

Fig. 4.34. Modified table

4.10.1 Adding a Column to the Table

We can add a column to the table by using ADD command. The syntax to add a new column to the table is:

```

ALTER TABLE table name
ADD column name datatype;
  
```

Example to Add a New Column

Let us consider the Table 4.6 BESTCRICKETER, which has three columns which are **name** of the player, **country** the player belong to, and the **centuries** which refer to the number of centuries scored by the player. Now try

to add one more column to the table **BESTCRICKETER**. The new column to be added is **age** which refers to player age. The SQL command to add the new column age and the corresponding output are shown in Fig. 4.35.

To see the description of the table after adding the new column **age** to the table **bestcricketer**, let us use **DESC** command as shown in Fig. 4.36.

From Fig. 4.36 we can observe that a new column **age** of datatype number has been added to the table **bestcricketer**.

After successfully inserting the column **age**, we will be interested to know the content of the table to see any value is assigned to the column age. Figure 4.37 shows the content of the table after adding a new column.

From Fig. 4.37, it is clear that the table already contains rows when the column **age** is added, then the new column **age** is initially null for all the rows.

To Insert Values into the New Column

Data can be inserted to the newly added column (in our example it is *age*) by using **UPDATE** command.

For example, we want to insert the age of Sachin Tendulkar to be 33. This is done using **UPDATE** command as shown in Fig. 4.38.

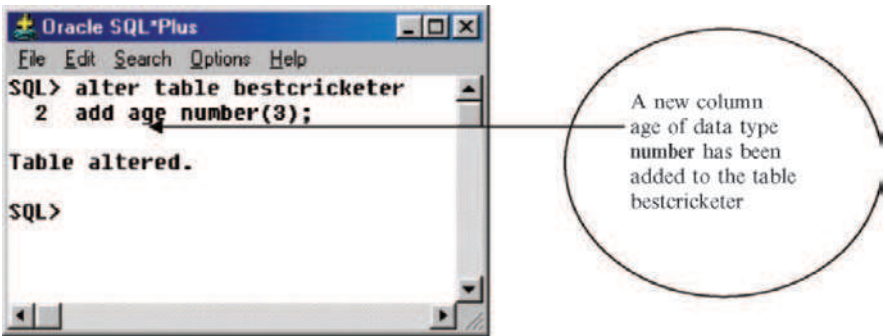


Fig. 4.35. Adding a column to the table



Fig. 4.36. Table descriptions after the addition of new column

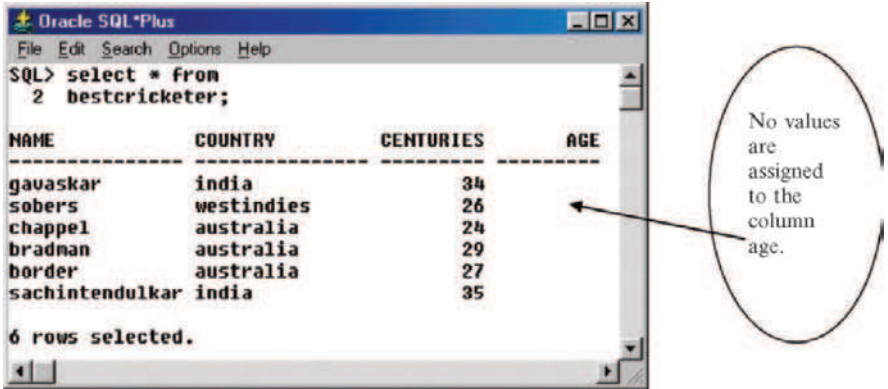


Fig. 4.37. Content of the table after the insertion of new column

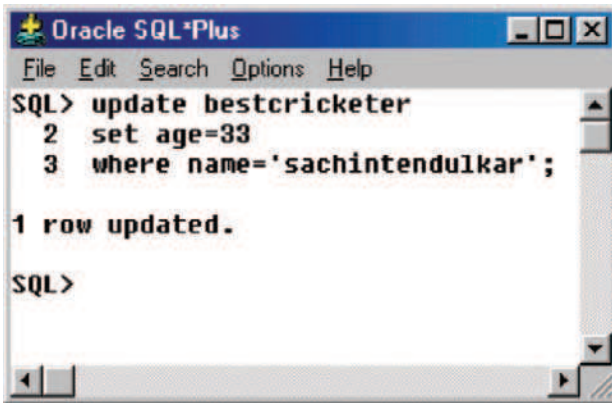


Fig. 4.38. Insertion of data to the new column age

To verify whether the age of sachintendulkar has been added as 33, see Fig. 4.39.

4.10.2 Modifying the Column of the Table

We can modify the width of the datatype of the column by using ALTER and MODIFY command. The syntax to change the datatype of the column is:

```

ALTER table name
MODIFY column-name datatype;

```

Example to Modify the Width of the Datatype of the Column

For example, we want to modify the width of the datatype **age** which is three as shown in Fig. 4.36 to four. The SQL command and the corresponding output are shown in Fig. 4.40.

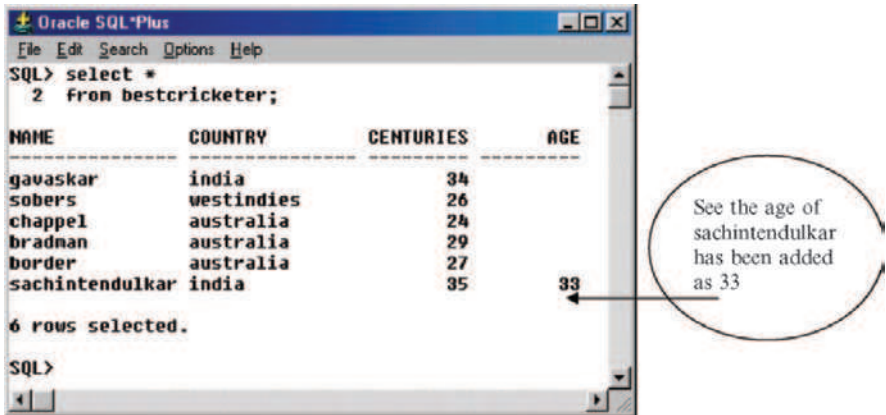


Fig. 4.39. Modified table

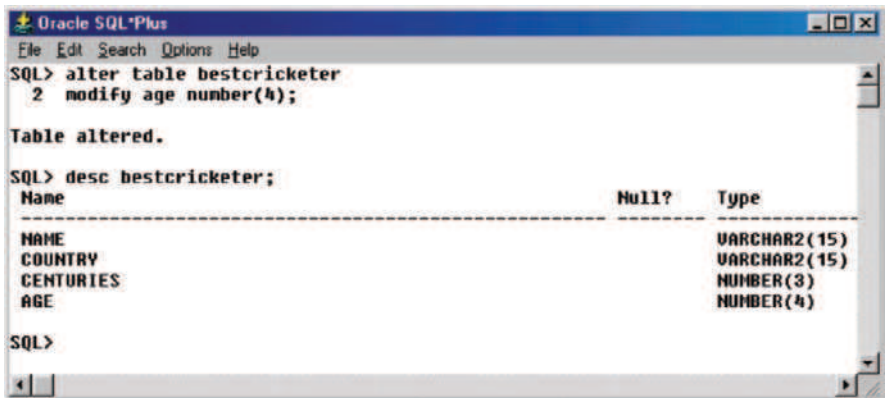


Fig. 4.40. Modified width of the datatype

From Fig. 4.40 we can observe that the width of the datatype **age** modified as four which was three earlier as shown in Fig. 4.36.

4.10.3 Deleting the Column of the Table

The DROP COLUMN command can be used along with the ALTER table command to delete the column of the table. The syntax to delete the column from the table is:

```

ALTER table name
DROP COLUMN column name;

```


Example

Let us try to delete the column **age** from the **BESTCRICKETER** by using **DROP COLUMN** command. The syntax to drop the column and the corresponding output are shown in Fig. 4.41.

After dropping the column **age**, the description of the table will be as shown in Fig. 4.42.

From Fig. 4.42, it is evident that the column **age** is not included in the table description.

The content of the table after dropping the column **age** is shown in Fig. 4.43.

4.11 Table Truncation

The **TRUNCATE TABLE** command removes all the rows from the table. The truncate table also releases the storage space used by the table. The syntax of **TRUNCATE** command is:

```
TRUNCATE TABLE table name;
```

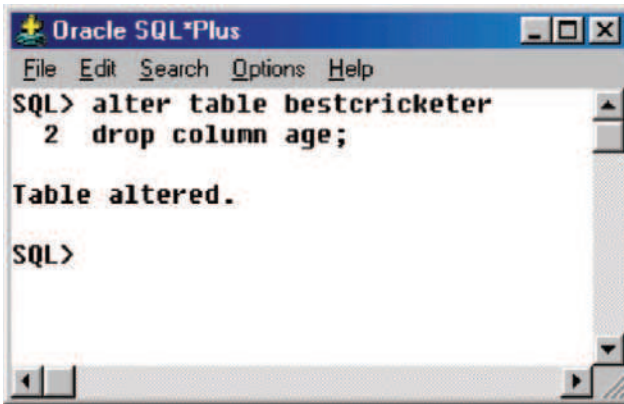


Fig. 4.41. Dropping a column from the table



Fig. 4.42. Table descriptions after dropping the column **age**

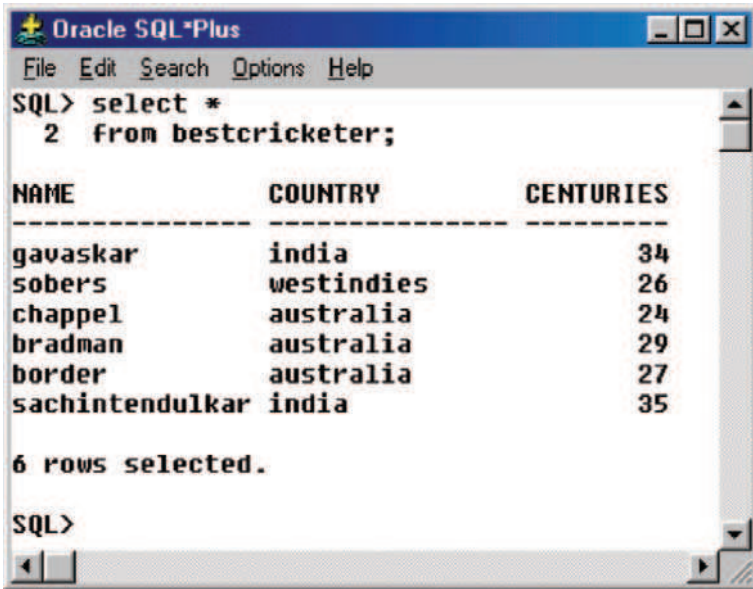


Fig. 4.43. Table content after dropping the column age

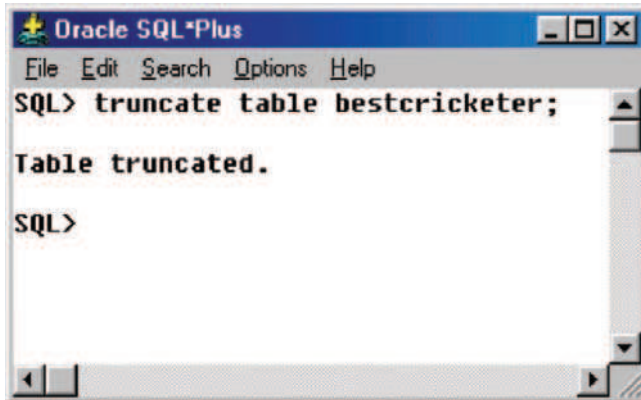


Fig. 4.44. Table truncation

Example

Let us try to delete all the rows of the table `bestcricketer` by issuing `TRUNCATE TABLE` command. The SQL command and the corresponding output are shown in Fig. 4.44.

After table truncation, if we try to select the rows, what will be the output? To answer this question, let us try to see the content of the table by using `SELECT` command as shown in Fig. 4.45.

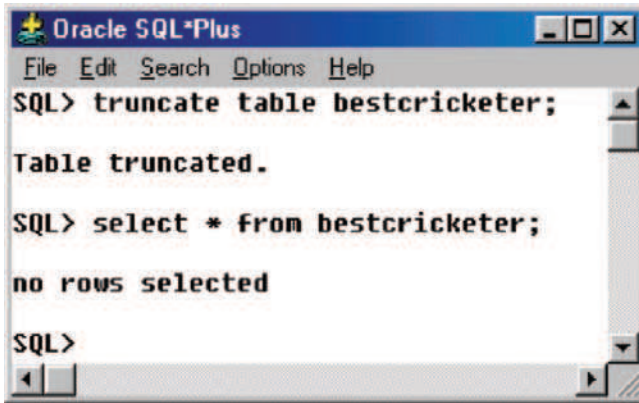


Fig. 4.45. Table content after truncation



Fig. 4.46. Table descriptions after table truncation

From Fig. 4.45, it is clear that all the rows are deleted by issuing TRUNCATE TABLE command. After the TRUNCATE TABLE command if we try to see the description of the table by issuing DESC command as shown in Fig. 4.46.

From Fig. 4.46, it is clear that the TRUNCATE TABLE command deletes the content (all rows) of the table but not the table definition.

Note Another way to delete all the rows of the table is to use DELETE command. The syntax is:

```
DELETE FROM table name;
```

4.11.1 Dropping a Table

The definition of the table as well as the contents of the table is deleted by issuing DROP TABLE command. The syntax of DROP TABLE command is:

```
DROP TABLE table name;
```

Example

Let us issue the DROP TABLE command to the table BESTCRICKETER as shown in Fig. 4.47.

After issuing the DROP TABLE command if we try to see the description of the table, we will get the result as shown in Fig. 4.48.

From Fig. 4.48 it is clear that DROP TABLE command deletes both the content and the descriptions of the table.

4.12 Imposition of Constraints

Constraints are basically used to impose rules on the table, whenever a row is inserted, updated, or deleted from the table. Constraints prevent the deletion of a table if there are dependencies. The different types of constraints that

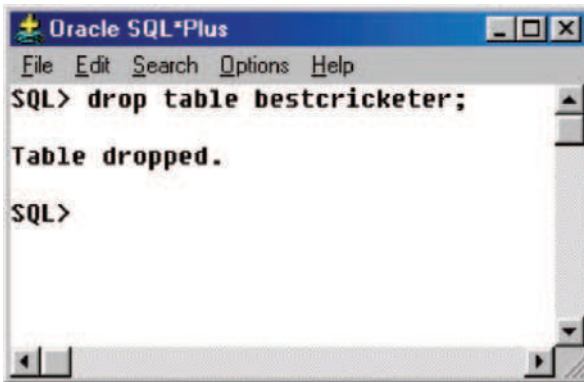


Fig. 4.47. Dropping a table

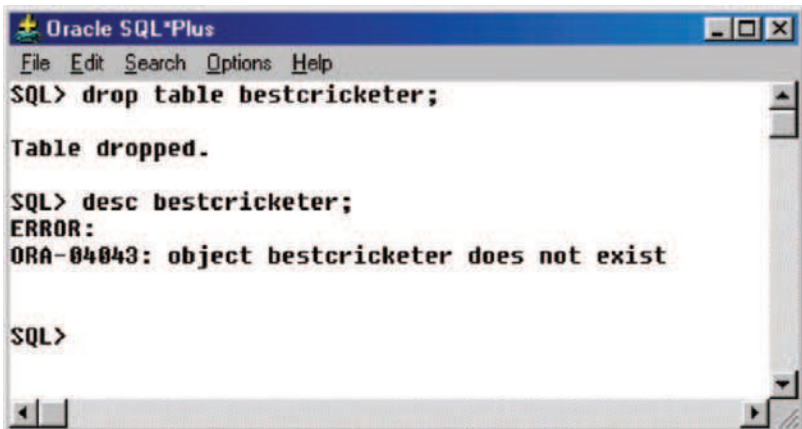


Fig. 4.48. Table descriptions after dropping the table

can be imposed on the table are NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, and CHECK.

Whenever an attribute is declared as NOT NULL then it specifies that the attribute cannot contain a NULL value.

The UNIQUE constraint specifies that whenever an attribute or set of attributes are specified as UNIQUE, then the values of the attribute should be unique for all the rows of the table. For example, consider the Roll number of the student in the class, every student should have UNIQUE roll number.

PRIMARY KEY constraint is used to identify each row of the table uniquely.

FOREIGN KEY constraint specifies that the value of an attribute in one table depends on the value of the same attribute in another table.

CHECK constraint defines a condition that each row must satisfy. Also there is no limit to the number of CHECK constraints that can be imposed on a column.

4.12.1 NOT NULL Constraint

If one is very much particular that the column is not supposed to take NULL value then we can impose NOT NULL constraint on that column. The syntax of NOT NULL constraint is:

```
CREATE TABLE table name
(column name1,      data-type of the column1,      NOT NULL
column name2,      data-type of the column2,
column nameN,      data-type of the columnN);
```

The above syntax indicates that column1 is declared as NOT NULL.

Example

Consider the relation PERSON, which has the attributes **name** of the person, **salary** of the person, **phone number** of the person. Let us try to declare the column **name** as **NOT NULL**. This implies that every person should have a name. The syntax to declare the column **name** as NOT NULL is shown in Fig. 4.49.

From Fig. 4.49, it is clear that the attribute **name** is declared as NOT NULL. Now let us try to insert NOT NULL values and NULL value to the attribute **name**.

*Case 1: Inserting a NOT NULL value to the attribute **name**.*

From Fig. 4.50, it is clear that when we try to insert a NOT NULL name into the name attribute, the name is included in the relation **PERSON1**.

*Case 2: A NULL value to the attribute **name**.*

From Fig. 4.51, it is clear that when we try to insert a NULL value into the **PERSON1** relation, we get the error message as shown in Fig. 4.51 since the attribute **name** is declared as **NOT NULL**.

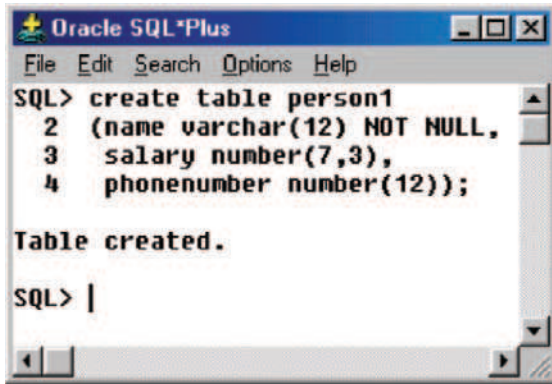


Fig. 4.49. NOT NULL constraint

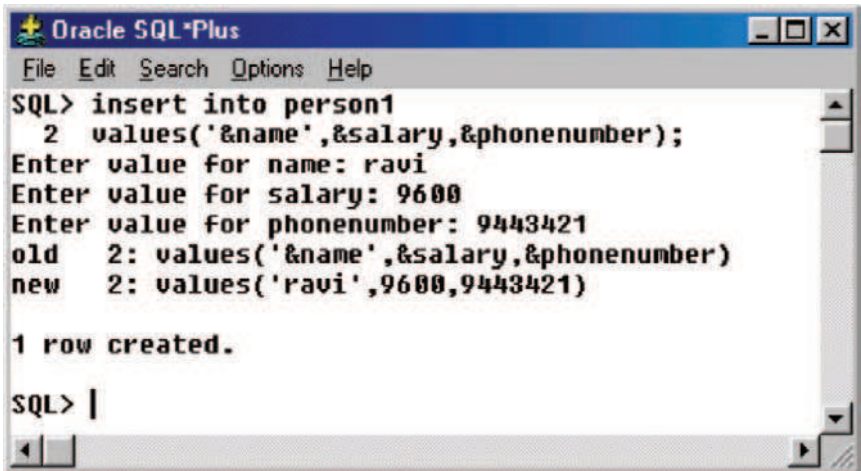


Fig. 4.50. A NOT NULL value to the attribute name

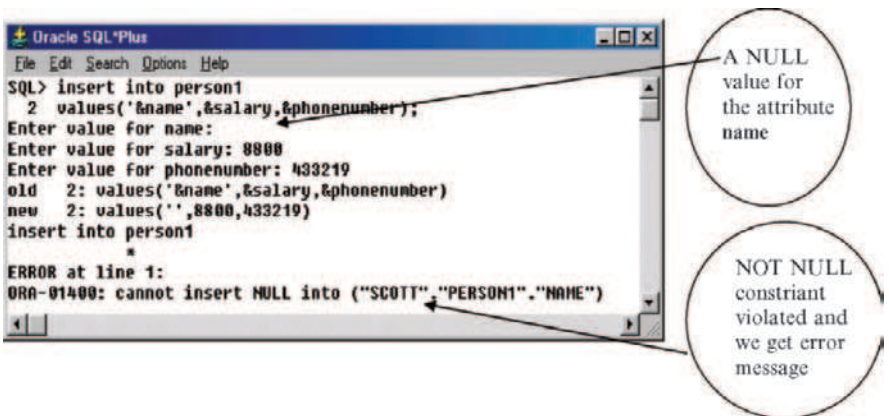


Fig. 4.51. NOT NULL constraint violated

4.12.2 UNIQUE Constraint

The UNIQUE constraint imposes that every value in a column or set of columns be unique. It means that no two rows of a table can have duplicate values in a specified column or set of columns.

Example

In order to understand unique constraint, let us create the table CELLPHONE, which has three attributes. The three attributes are **model** of the cellphone, **make** which refers to manufacturer, and the **price**.

The relation CELLPHONE is created as shown in Fig. 4.52 with unique constraint on **model**. When a **unique** constraint is imposed on the attribute **model**, then no two models should have same number.

The values are inserted into the table CELLPHONE. The resulting tables after inserting the values are shown in Fig. 4.53.

From Fig. 4.53, we can observe that the table CELLPHONE has three rows.

Case 1: Now let us try to insert a row in the relation CELLPHONE by violating the UNIQUE constraint, i.e., we are trying to insert a row with model number 1100 which already exists. The insertion and the corresponding result are shown in Fig. 4.54. From this figure, we can observe that there is an error message “unique constraint (SCOTT.SYS_C00820) violated.” The reason for getting this error message is we tried to enter the model (1100) which exists already in the CELLPHONE relation as shown in Fig. 4.53.

Case 2: Insertion of NULL Value to the Model Attribute. Let us try to insert a null value to the attribute model. The SQL command to insert a null value to the attribute model and the corresponding result are shown in Fig. 4.55.

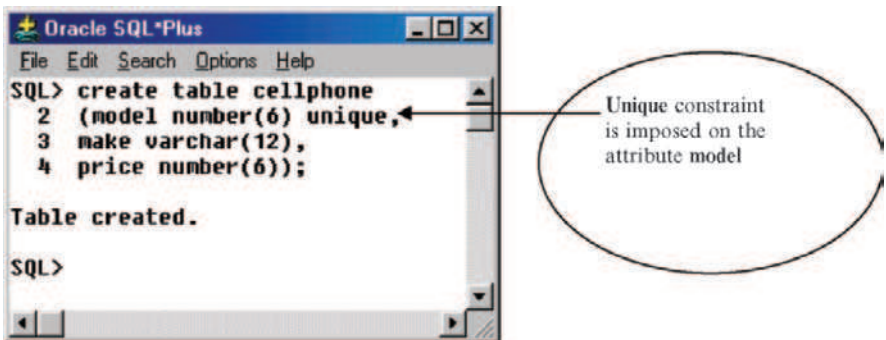


Fig. 4.52. Unique constraint on a column

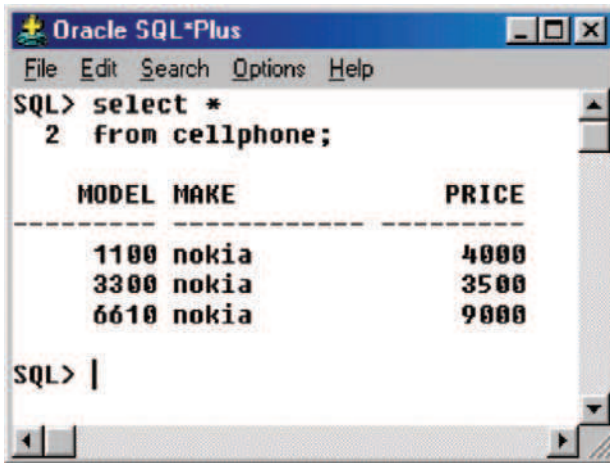


Fig. 4.53. Values inserted into the table CELLPHONE

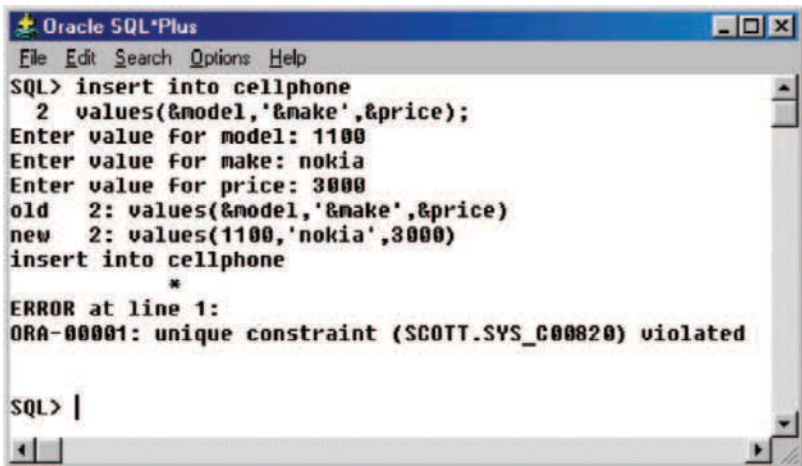


Fig. 4.54. Violation of UNIQUE constraint

Difference Between NOT NULL and UNIQUE Constraint

The unique constraint accepts NULL value as shown in Fig. 4.55, whereas the NOT NULL constraint will not accept NULL values.

Note NOT NULL constraint accepts duplicate values, whereas UNIQUE constraint will not accept null values. Moreover when a UNIQUE constraint is imposed on an attribute means that attribute can accept NULL values. Whereas NOT NULL constraint will not accept NULL values.

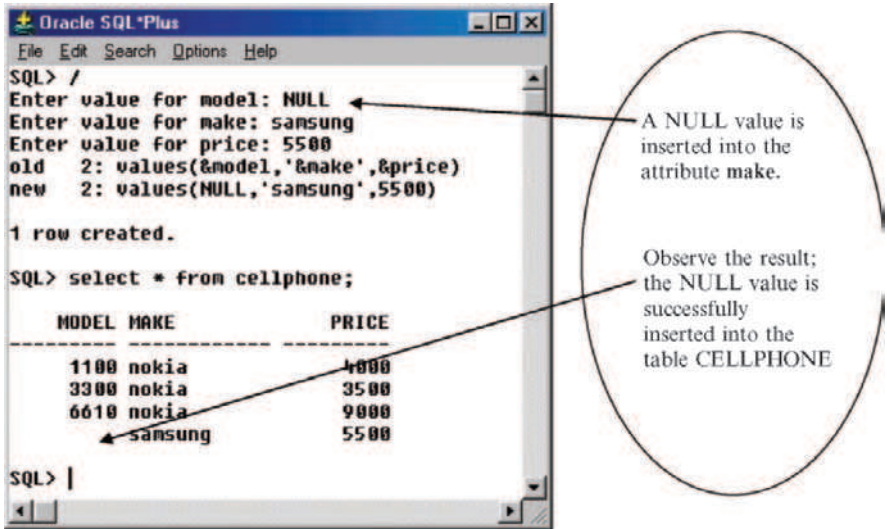


Fig. 4.55. Insertion of NULL value into CELLPHONE

4.12.3 Primary Key Constraint

When an attribute or set of attributes is declared as the primary key, then the attribute will not accept NULL value moreover it will not accept duplicate values. It is to be noted that “only one primary key can be defined for each table.”

Example

Consider the relation **EMPLOYEE** with the attributes **ID** which refers to Employee identity, **NAME** of the employee, and **SALARY** of the employee. Each employee will have unique ID hence ID is declared as the primary key as shown in Fig. 4.56.

From Fig. 4.56, it is clear that the attribute employee ID is declared as the primary key.

Case 1: Insertion of NULL Value to the Primary Key Attribute.

It is to be noted that the primary key will not take any NULL value. This is called entity integrity. Now let us try to insert a NULL value to the employee ID in the SQL syntax, and the corresponding output is shown in Fig. 4.57. From Fig. 4.57, it is evident that an attribute or set of attributes declared as primary key will not accept NULL values.

Case 2: Insertion of Duplicate Values into an Attribute Declared as Primary Key.

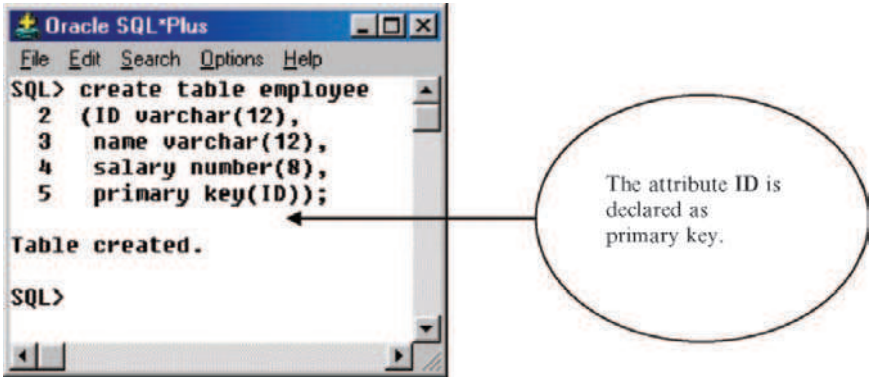


Fig. 4.56. Attribute declared as primary key

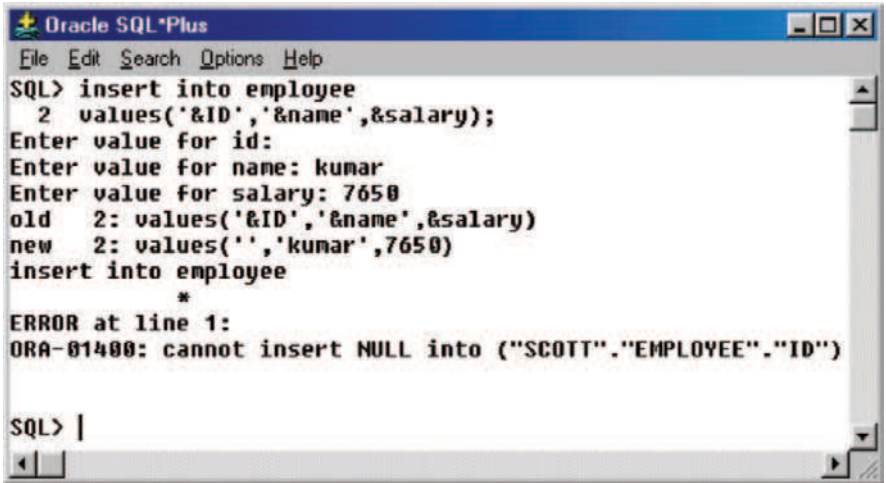


Fig. 4.57. Inserting NULL value into primary key attribute

When an attribute is declared as primary key, all the values of the attribute should be UNIQUE. The primary key attribute will not accept duplicate values.

Let us try to insert duplicate values to the attribute employee ID which is declared as primary key. The SQL command and the corresponding output are shown in Fig. 4.58.

We got an error message in Fig. 4.54, because we have tried to insert the employee ID e101 twice. From this we can understand that when an attribute is declared as primary key, the values of the attribute should be UNIQUE.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> insert into employee
  2 values('&ID','&name',&salary);
Enter value for id: e101
Enter value for name: rajan
Enter value for salary: 1200
old  2: values('&ID','&name',&salary)
new  2: values('e101','rajan',1200)

1 row created.

SQL> /
Enter value for id: e101
Enter value for name: ravi
Enter value for salary: 1500
old  2: values('&ID','&name',&salary)
new  2: values('e101','ravi',1500)
insert into employee
      *
ERROR at line 1:
ORA-00001: unique constraint (SCOTT.SYS_C00025) violated

SQL> |

```

Fig. 4.58. Insertion of duplicate values to an attribute declared as primary key

Difference Between UNIQUE and NOTNULL Constraint

The difference between UNIQUE and NOTNULL constraint is given in the tabular form as

NOTNULL constraint	UNIQUE constraint
an attribute declared as NOTNULL will not accept NULL values	an attribute declared as UNIQUE can accept NULL values
an attribute declared as NOTNULL will accept duplicate values	an attribute declared as UNIQUE will not accept duplicate values

Difference Between UNIQUE and PRIMARY KEY Constraint

The difference between UNIQUE and PRIMARY KEY is given in tabular form as

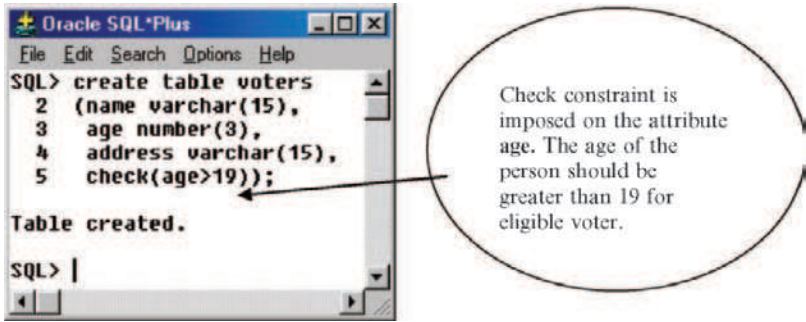


Fig. 4.59. Check constraint on an attribute

PRIMARY KEY constraint	UNIQUE constraint
an attribute declared as primary key will not accept NULL values	an attribute declared as UNIQUE will accept NULL values
only one PRIMARY KEY can be defined for each table	more than one UNIQUE constraint can be defined for each table

4.12.4 CHECK Constraint

CHECK constraint is added to the declaration of the attribute. The CHECK constraint may use the name of the attribute or any other relation or attribute name may in a subquery. Attribute value check is checked only when the value of the attribute is inserted or updated.

Syntax of CHECK Constraint

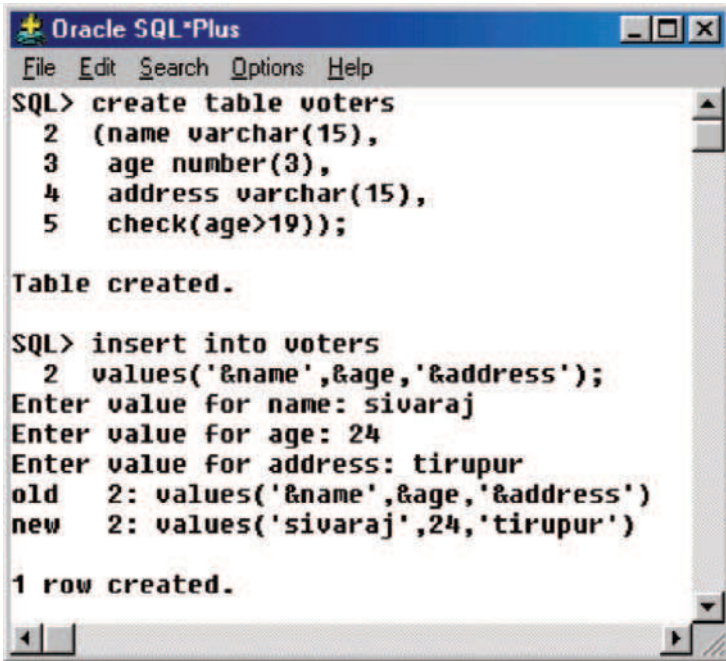
In order to understand check constraint, consider the relation **VOTERS**. In India, only those who have completed the age of 19 are eligible to vote. Let us impose this constraint on age in our relation **VOTERS**. The **VOTERS** relation has the attributes **name**, which refers to the name of the voter, **age** of the voter, **address** of the voter.

The creation of the table **VOTERS** with **CHECK** constraint imposed on **age** is shown in Fig. 4.59.

From Fig. 4.59, we can observe that CHECK constraint is imposed on the attribute **age**.

Case 1: Insertion of Data Without Violating the Constraint.

Let us try to insert the values into the table **VOTERS** without violating the constraint, that is the age of the voter is greater than 19. The SQL syntax and the corresponding output are shown in Fig. 4.60. From this figure, it is evident that the data are successfully inserted into the table **VOTERS** because the age of the voter is greater than 19.



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> create table voters
  2  (name varchar(15),
  3  age number(3),
  4  address varchar(15),
  5  check(age>19));

Table created.

SQL> insert into voters
  2  values('&name',&age,'&address');
Enter value for name: sivaraj
Enter value for age: 24
Enter value for address: tirupur
old  2: values('&name',&age,'&address')
new  2: values('sivaraj',24,'tirupur')

1 row created.

```

Fig. 4.60. Data insertion without violating the constraint

Case 2: Insertion of Data into the Table VOTERS by Violating the CHECK Constraint.

Now let us try to insert data into the table VOTERS by violating the CHECK constraint, that is inserting the record of the voter with age less than 19. The SQL command to insert the data and the corresponding output are shown in Fig. 4.61.

From Fig. 4.61, we can observe that we try to insert a value which violates the CHECK constraint, we get error message.

Case 3: CHECK Constraint During Updation of Record.

The content of the VOTER table is given in Fig. 4.62.

For simplicity, there is only one record in the VOTERS table. Now let us try to update the record by changing the age of the voter to less than 19, as shown in Fig. 4.63.

From Fig. 4.63, we can observe that it is not possible to update the record by violating the CHECK constraint.

4.12.5 Referential Integrity Constraint

According to referential integrity constraint, when a foreign key in one relation **references** primary key in another relation, the foreign key value must

```

Oracle SQL*Plus
File Edit Search Options Help
SQL>
SQL> insert into voters
  2 values('&name',&age,'&address');
Enter value for name: madhavan
Enter value for age: 18
Enter value for address: trichy
old 2: values('&name',&age,'&address')
new 2: values('madhavan',18,'trichy')
insert into voters
*
ERROR at line 1:
ORA-02290: check constraint (SCOTT.SYS_C00021) violated

SQL> |

```

Fig. 4.61. Data insertion by violating the CHECK constraint

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select *
  2 from voters;

NAME                AGE ADDRESS
-----
sivaraj              24  tirupur

SQL>

```

Fig. 4.62. The content of VOTERS table

match with the primary key value. In other words, the referential integrity says “pointed to” information must exist.

Example

In order to understand referential constraint, consider two relation DEPARTMENT and EMPLOYEE. Here the DEPARTMENT relation forms the parent table. The meaning is the DEPARTMENT table contains the primary key. The relation EMPLOYEE forms the child table. The meaning is the relation EMPLOYEE has foreign key which references to primary key in DEPARTMENT table. Figure 4.64 shows parent–child relationship.

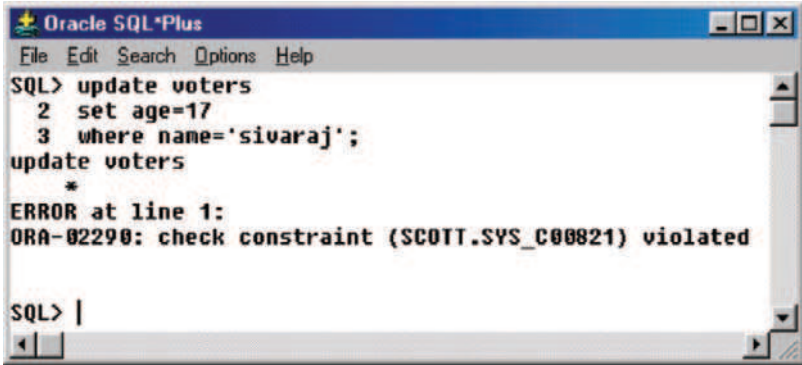


Fig. 4.63. Updation of record voters by violating CHECK constraint

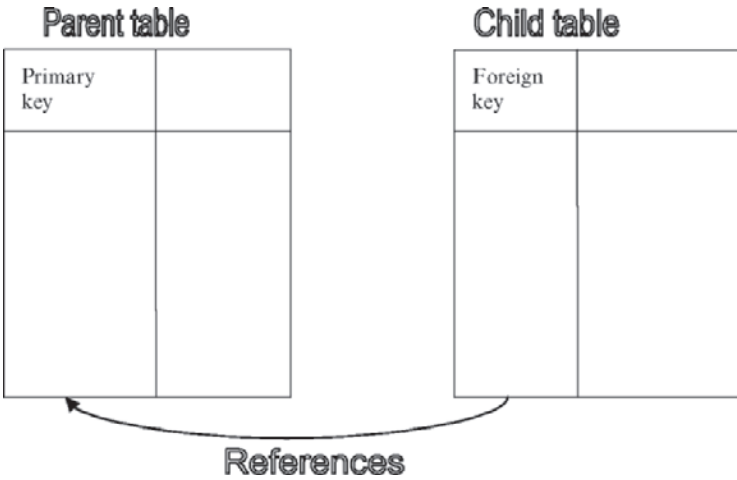


Fig. 4.64. Primary key and foreign key relationship

In our example, the relation DEPARTMENT is the parent table which holds the parent table, and the relation EMPLOYEE forms the child table which has foreign key which references primary key in DEPARTMENT table. It is to be noted that the parent table should be created first, then the child table.

DEPARTMENT			EMPLOYEE		
DeptID	Dname	Location	EID	DID	Ename
D100	electrical	B	E201	D100	Raman
D101	civil	A	E202	D101	Ravi
D102	computer	C	E203	D101	Krishnan

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> create table department
2 (deptid varchar(12),
3  deptname varchar(12),
4  deptlocation varchar(12),
5  primary key(deptid));

Table created.

SQL> |

```

Table department is created with department ID, deptid as primary key.

Fig. 4.65. DEPARTMENT table

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> create table employee
2 (eid varchar(12),
3  did varchar(12),
4  ename varchar(12),
5  foreign key(did) references department(deptid));

Table created.

SQL> |

```

did in employee table references deptid in department table

Fig. 4.66. EMPLOYEE table

The SQL syntax to create the two relations DEPARTMENT and EMPLOYEE with primary key and foreign key constraints is shown in Fig. 4.65 and Fig. 4.66, respectively.

Case 1: Now let us try to insert a value into DepartmentID of the **employee** table which is not in **department** table. The department relation has only three department IDs D100, D101, D102. Now we are trying to insert D103 in the **DID** (which stands for department ID) of employee table. The SQL command and the corresponding output are shown in Fig. 4.67.

From Fig. 4.67, it is evident that the values are not able to insert into the **employee** table. The reason for not able to insert value into the **employee** table is: we have tried to insert the **DID** (department id) into the **employee** table (child table) which is not matching with **DeptID** (department id) of the **department** table (parent table). In other words the foreign key value in the child table does not match with the primary key value in the parent relation.

The referential integrity rule says that the foreign key value should match with the primary key value.

Case 2: NULL Value into Foreign Key Attribute.

Now let us try to insert a null value into the foreign key attribute. The SQL command and the corresponding output are shown in Fig. 4.68.


```

Oracle SQL*Plus
File Edit Search Options Help
SQL> insert into employee
  2 values('&eid','&did','&ename');
Enter value for eid: E204
Enter value for did: D105
Enter value for ename: kumar
old 2: values('&eid','&did','&ename')
new 2: values('E204','D105','kumar')
insert into employee
*
ERROR at line 1:
ORA-02291: integrity constraint (SCOTT.SYS_C00827) violated - parent key not found

SQL> |

```

Fig. 4.67. Violation of referential integrity

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> insert into employee
  2 values('&eid','&did','&ename');
Enter value for eid: E205
Enter value for did: NULL
Enter value for ename: kannan
old 2: values('&eid','&did','&ename')
new 2: values('E205','NULL','kannan')
insert into employee
*
ERROR at line 1:
ORA-02291: integrity constraint (SCOTT.SYS_C00827) violated - parent key not found

SQL>

```

Fig. 4.68. NULL value to the foreign key attribute

From Fig.4.68, it is evident that NULL value cannot be inserted into foreign key attribute unless it matches with the primary key attribute.

4.12.6 ON DELETE CASCADE

When the clause ON DELETE CASCADE is included in the child table, and if a row is deleted from the parent table then the corresponding referenced value in the child table will also be deleted.

Example

Let us consider the DEPARTMENT (parent table) and EMPLOYEE (child table) relation. The employee relation is modified as shown in Fig. 4.69. From this figure, it is clear that we have included the clause ON DELETE CASCADE in the child table.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> drop table employee;

Table dropped.

SQL> create table employee
  2  (eid varchar(12),
  3  did varchar(12),
  4  ename varchar(12),
  5  foreign key(did) references department(deptid)
  6  ON DELETE CASCADE);

Table created.

SQL> |

```

Fig. 4.69. Modified EMPLOYEE relation

The content of the table DEPARTMENT and EMPLOYEE are shown below.

DEPARTMENT			EMPLOYEE		
DeptID	Dname	Location	EID	DID	Ename
D100	electrical	B	E201	D100	Raman
D101	civil	A	E202	D101	Ravi
D102	computer	C	E203	D101	Krishnan

Now let us try to delete the department “Civil” in the DEPARTMENT table. If we delete the row “civil” in the DEPARTMENT table, what will be the impact in the EMPLOYEE table?

First the content of employee table is shown in Fig. 4.70. The number of tuples in the EMPLOYEE relation is three.

Now we are going to delete the department “civil” in the table DEPARTMENT. The SQL command and the corresponding output are shown in Fig. 4.71.

Now let us see the impact of deleting the record “civil” in the child table which is EMPLOYEE in our case. The modified table EMPLOYEE is shown in Fig. 4.72.

By carefully analyzing the Figs. 4.71 and 4.72, we can observe that the record “civil” in the child table (employee) being deleted.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select *
      2 from employee;

EID          DID          ENAME
-----
E201        D100         raman
E202        D101         ravi
E203        D101         krishnan

SQL> |

```

Fig. 4.70. EMPLOYEE table (child table) before deletion of record in parent table

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> delete from department
      2 where deptname='civil';

1 row deleted.

SQL> select *
      2 from department;

DEPTID      DEPTNAME      DEPTLOCATION
-----
D100        electrical     B
D102        computer       C

SQL>

```

Fig. 4.71. DEPARTMENT table without “civil” department

If ON DELETE CASCADE clause is included in the child table means whatever record deleted in the parent table will be deleted in the child table.

4.12.7 ON DELETE SET NULL

If ON DELETE SET NULL clause is include in the child table means, whenever a row in the parent table is deleted, then the corresponding referenced value in the child table will be set null.

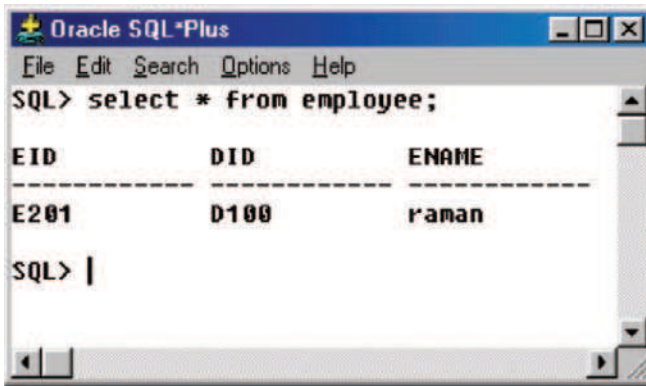


Fig. 4.72. Modified EMPLOYEE table

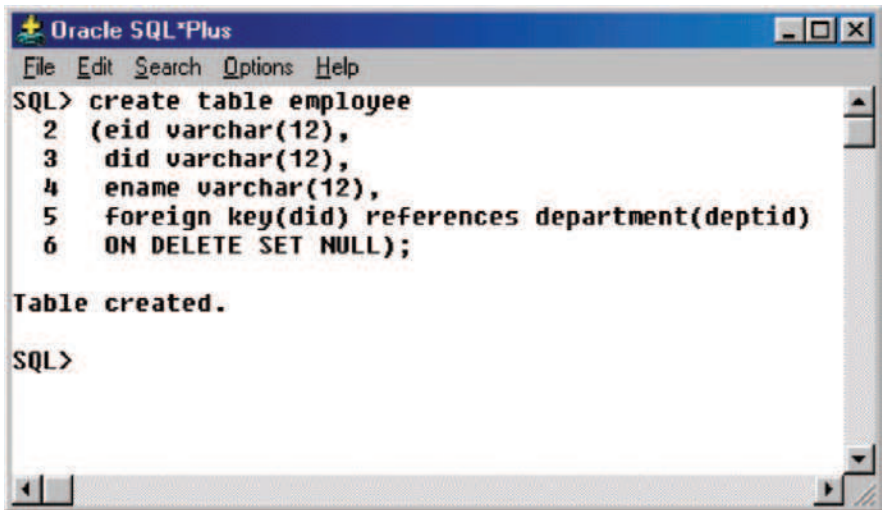


Fig. 4.73. Modified employee table definition

Example

Let us consider the parent table as DEPARTMENT and the child table as EMPLOYEE as before. The child table is created with ON DELETE SET NULL as shown in Fig. 4.73.

The EMPLOYEE table before modification is shown below.

EID	DID	Ename
E201	D100	Raman
E202	D101	Ravi
E203	D101	Krishnan

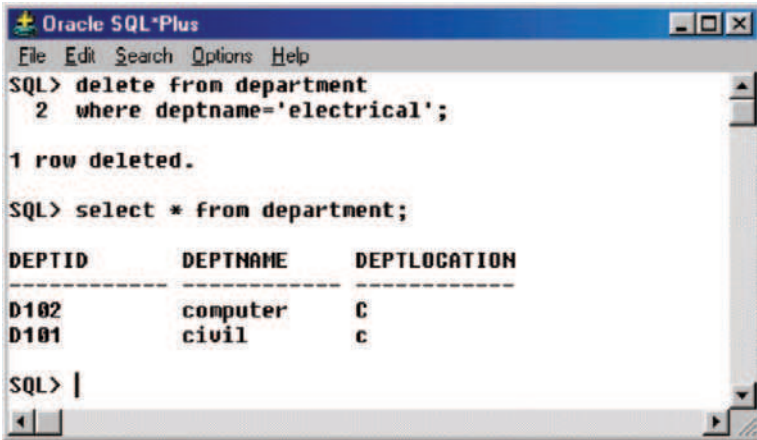


Fig. 4.74. Modified table DEPARTMENT

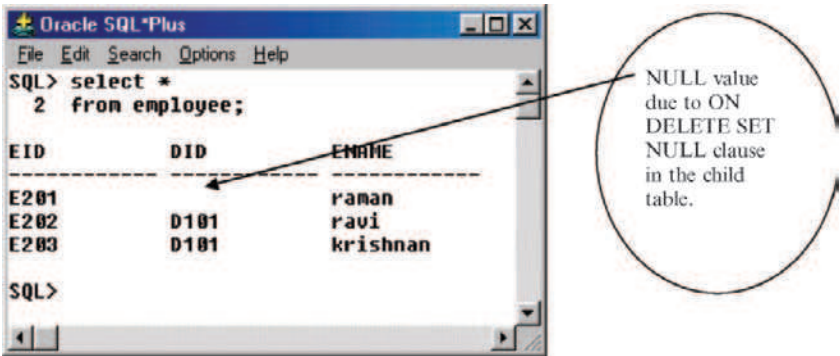


Fig. 4.75. Modified child table (EMPLOYEE)

Now modify the table DEPARTMENT by deleting the “electrical” department record. The SQL command to delete the record “electrical” and the corresponding output are shown in Fig. 4.74.

The impact of deleting the record “electrical” in parent table DEPARTMENT on the child table EMPLOYEE is shown in Fig. 4.75.

From Fig. 4.75, we can observe that a NULL value is there corresponding to the ID of the “electrical” department. This is due to inclusion of the clause ON DELETE NULL in the child table (EMPLOYEE).

4.13 Join Operation

Join operation is used to retrieve data from more than one table. Before proceeding to JOIN operation let us discuss first the Cartesian product. Cartesian product with suitable selection and projection operation forms different types of join.

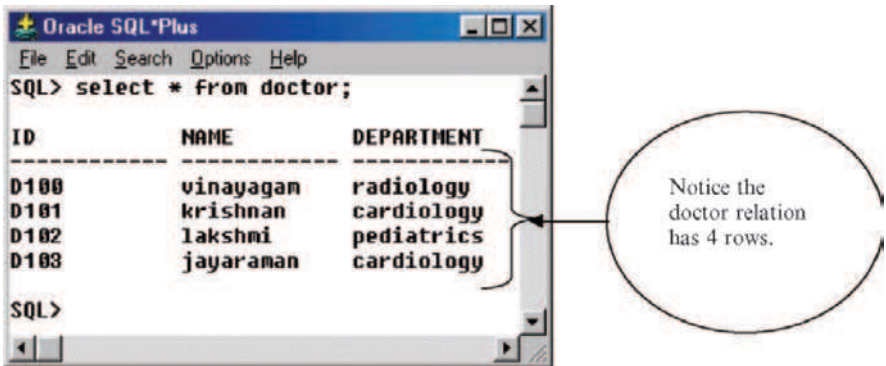
Cartesian Product

If we have two tables A and B, then Cartesian product combines all rows in the table A with all rows in the table B. If n_1 is the number of rows in the table A and n_2 is the number of rows in the table B. Then the Cartesian product between A and B will have $n_1 \times n_2$ rows.

Example

In order to understand Cartesian product, let us consider two relations **doctor** and **nurse**. The relation **doctor** has the attribute **ID** which refers to identity of the doctor, **name** and **department**. Similarly, the relation **nurse** has three attributes **NID**, which refers to nurse identity, **name** and **department**. The doctor relation is shown in Fig. 4.76.

Similarly the nurse relation is shown in Fig. 4.77.



Oracle SQL*Plus

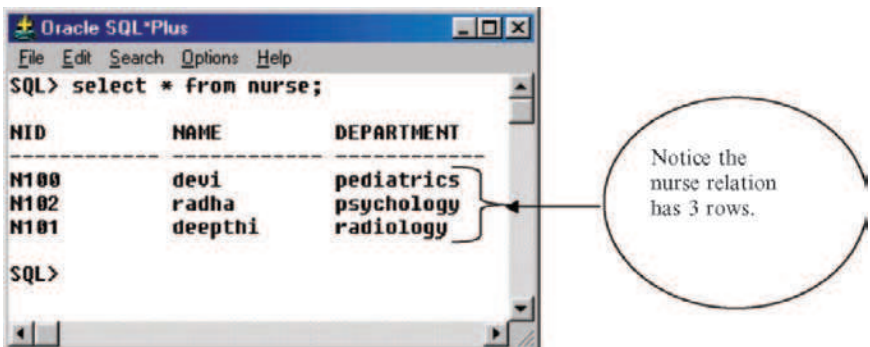
```
File Edit Search Options Help
SQL> select * from doctor;
```

ID	NAME	DEPARTMENT
D100	vinayagan	radiology
D101	krishnan	cardiology
D102	lakshmi	pediatrics
D103	jayaraman	cardiology

SQL>

Notice the doctor relation has 4 rows.

Fig. 4.76. DOCTOR relation



Oracle SQL*Plus

```
File Edit Search Options Help
SQL> select * from nurse;
```

NID	NAME	DEPARTMENT
N100	devi	pediatrics
N102	radha	psychology
N101	deepthi	radiology

SQL>

Notice the nurse relation has 3 rows.

Fig. 4.77. NURSE relation

From Figs. 4.76 and 4.77 we can observe that the number of rows in doctor and nurse relation is 4. Now let us try to find the Cartesian product between the two relations doctor and nurse. The Cartesian product should return $4 \times 3 = 12$ rows. The SQL command to perform Cartesian product between the two relations doctor and nurse and the corresponding output are shown in Fig. 4.78. From this figure, it is evident that the Cartesian product between two relations has 12 tuples (rows).

4.13.1 Equijoin

In equijoin, the join condition is based on equality between values in the common columns. Moreover the common columns appear redundantly in the result. Equijoins are also called as simple joins or inner joins. The equijoin between the two relations doctor and nurse (The relations doctor and nurse are shown in Figs. 4.76 and 4.77, respectively) is shown in Fig. 4.79.

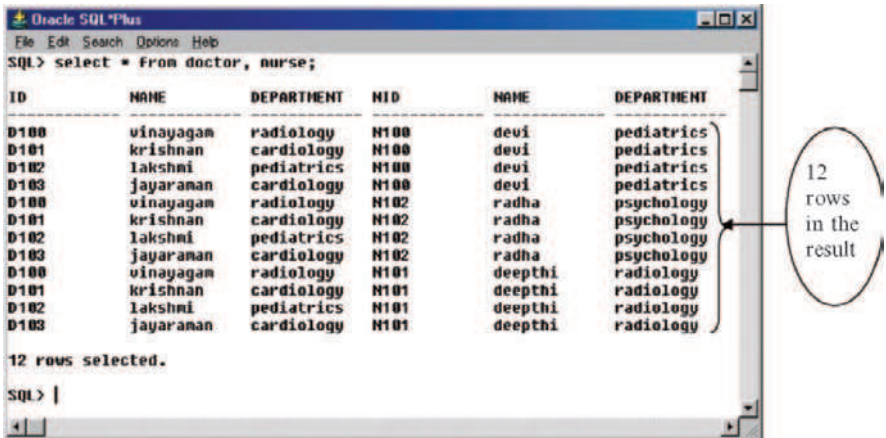


Fig. 4.78. Cartesian product between the relations doctor and nurse

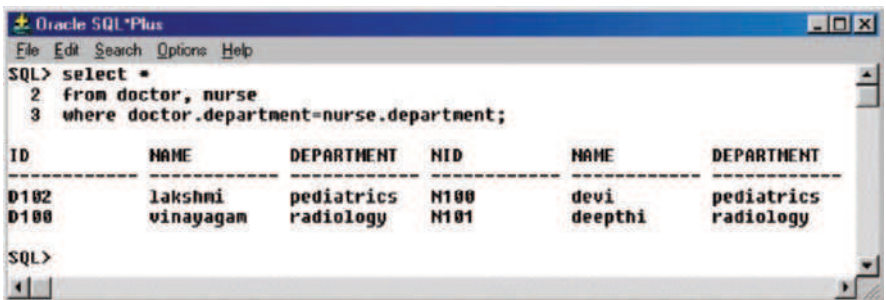


Fig. 4.79. Equijoin between doctor and nurse relation

From Fig. 4.79, it is evident that the join condition is equality condition on the attribute **department**. We can also observe that the common columns appear redundantly in the result.

4.14 Set Operations

The UNION, INTERSECTION, and the MINUS (Difference) operations are considered as SET operations. Out of these three set operations, UNION, INTERSECTION operations are commutative, whereas MINUS (Difference) operation is not commutative. All the three operations are binary operations. The relations that we are going to consider for UNION, DIFFERENCE, and MINUS operations are IBM_DESKTOP and DELL_DESKTOP as shown in Figs. 4.80 and 4.81, respectively.

4.14.1 UNION Operation

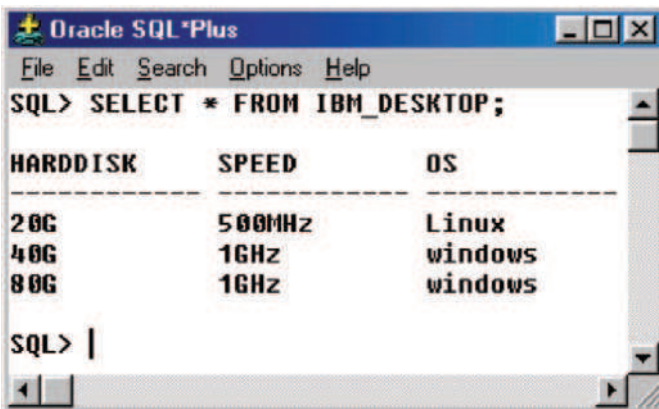
If we have two relations R and S then the set UNION operation contains tuples that either occurs in R or S or both.

Case 1: UNION command.

The union of two relations IBM_DESKTOP, DELL_DESKTOP is given in Fig. 4.80. From Fig. 4.81, it is clear that the UNION command eliminates duplicate values.

Case 2: UNION ALL command.

The UNION command removes duplicate values. In order to get the duplicate values, we can use UNION ALL command. The use of UNION ALL command and the corresponding results are shown in Fig. 4.83.



The screenshot shows the Oracle SQL*Plus interface. The command prompt shows the execution of the query: `SQL> SELECT * FROM IBM_DESKTOP;`. The output is a table with three columns: **HARDDISK**, **SPEED**, and **OS**. The data rows are:

HARDDISK	SPEED	OS
20G	500MHz	Linux
40G	1GHz	windows
80G	1GHz	windows

The prompt `SQL> |` is visible at the bottom of the window.

Fig. 4.80. IBM_DESKTOP

Oracle SQL*Plus window showing the execution of the query: `SQL> SELECT * FROM DELL_DESKTOP;`

HARDDISK	SPEED	OS
20G	500MHz	Linux
40G	1.2GHz	windows

SQL>

Fig. 4.81. DELL_DESKTOP

Oracle SQL*Plus window showing the execution of the query: `SQL> select *
2 from IBM_DESKTOP
3 UNION
4 select *
5 from DELL_DESKTOP;`

HARDDISK	SPEED	OS
20G	500MHz	Linux
40G	1.2GHz	windows
40G	1GHz	windows
80G	1GHz	windows

SQL>

Fig. 4.82. UNION command

By carefully looking into the Figs.4.82 and 4.83, the number of tuples in the Fig.4.82 is four; whereas the number of tuples in Fig.4.83 is five. The difference in two results is due to the fact that UNION command rejects duplicate values, whereas UNION ALL command includes duplicate values.

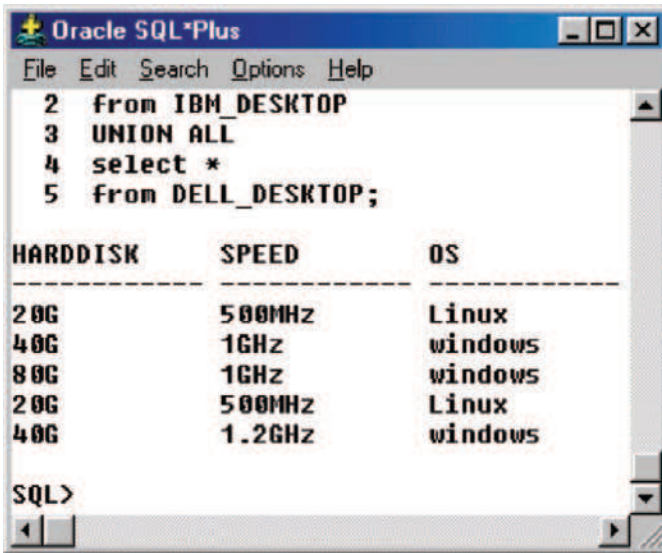


Fig. 4.83. UNION ALL command

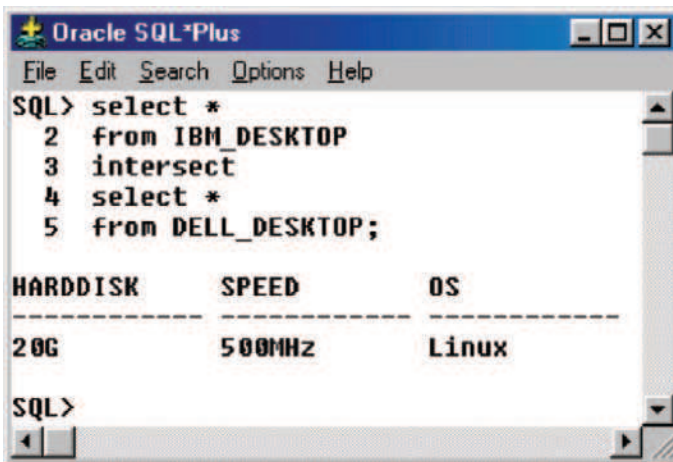


Fig. 4.84. INTERSECTION operation

4.14.2 INTERSECTION Operation

The intersection operation returns the tuples that are common to the two relations. The intersection of the two relations IBM_DESKTOP and DELL_DESKTOP is shown in Fig. 4.84.

4.14.3 MINUS Operation

If R and S are two union compatible relations then $R-S$ returns the tuples that are present in R but not in S. $S-R$ returns the tuples that are present in S but not in R. It is to be noted that MINUS operation is not commutative. That is $R-S \neq S-R$.

Case 1: IBM_DESKTOP-DELL_DESKTOP.

Let us first determine $IBM_DESKTOP-DELL_DESKTOP$. The SQL command and the corresponding output are shown in Fig. 4.85.

From Fig. 4.85, we can observe that the result contains the tuples that are present in $IBM_DESKTOP$ and not in $DELL_DESKTOP$.

Case 2: DELL_DESKTOP-IBM_DESKTOP.

Let us try to compute $DELL_DESKTOP-IBM_DESKTOP$. The SQL command and the corresponding output are shown in Fig. 4.86. From Fig. 4.86, it is clear that the result contains tuple that are present in $DELL_DESKTOP$ but not in $IBM_DESKTOP$.

Note From Figs. 4.85 and 4.86 it is clear that MINUS operation is not commutative.

4.15 View

View is a pseudotable or virtual table. View is called as “pseudotable” because view does not actually store data. View just displays the data. The data are derived from one or more base tables. View table can be used like any other table for querying. View can be considered as a window to the database. The view can also be considered as customized presentation of data from one or more tables. It is to be noted that all views are not updatable.

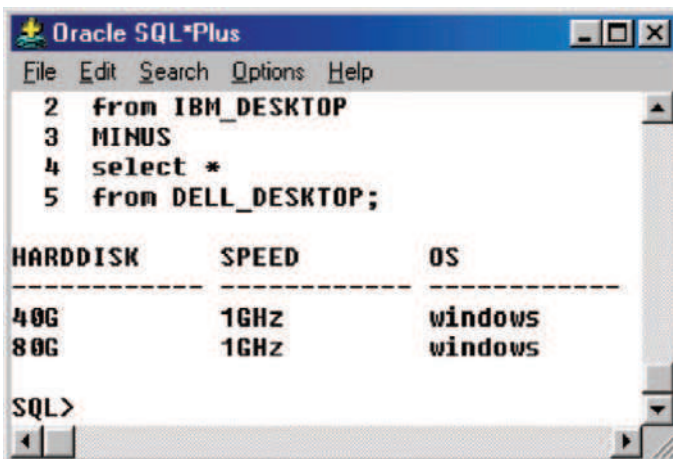


Fig. 4.85. $IBM_DESKTOP-DELL_DESKTOP$

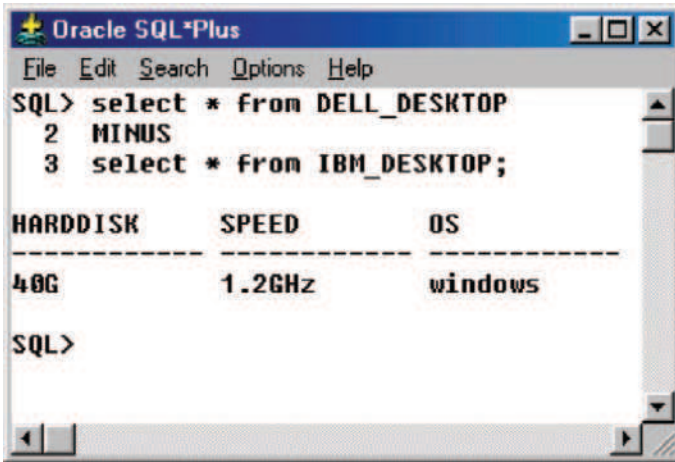


Fig. 4.86. DELL_DESKTOP-IBM_DESKTOP

The Syntax of **VIEW** is given as

```
CREATE VIEW view name
AS SELECT attribute list
FROM table(s)
WHERE condition(s)
```

Case 1: VIEW from a Single Table.

Consider the base table **RECORD** which gives the record of the student such as his/her Roll Number, Age, GPA (Grade Point Average), and institution which refers to the institution where he/she has got the degree (Fig. 4.87). The base table **RECORD** is shown below.

RECORD				
S.I. No	Name	Age	GPA	Institution
1	Anbalagan	22	9.2	PSG
2	Balu	22	9.4	PSG
3	Dinesh	22	8.4	CIT
4	Karthik	21	8.5	REC
5	Kumar	22	8.7	MIT
6	Kishore	22	8.8	MIT
7	Rajan	22	9.1	PSG
8	Lavanya	21	9.1	CIT

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select * from RECORD;

  SINO NAME          AGE    GPA INSTITUTION
-----
    1 Anbalagan      22    9.2    PSG
    2 Balu           22    9.4    PSG
    3 Dinesh         22    8.4    CIT
    4 Karthick       21    8.5    REC
    5 Kumar          22    8.7    MIT
    6 Kishore        22    8.8    MIT
    7 Rajan          22    9.1    PSG
    8 Lavanya        21    9.1    CIT

8 rows selected.

SQL> |

```

Fig. 4.87. Base table RECORD

Now we want to create a view by name PLACED, which gives the list of students placed in a particular organization (say IBM). The attribute associated with the view PLACED are Name, Age, and Institution. The view PLACED is shown below.

PLACED		
Name	Age	Institution
Anbalagan	22	PSG
Balu	22	PSG
Rajan	22	PSG
Lavanya	21	CIT

From the table PLACED, it is obvious that only those students with GPA greater than nine are placed. The SQL command to create the view PLACED from the base table RECORD and the output are shown in Fig. 4.88. From Fig. 4.88, it is clear that the view PLACED has only three columns Name, Age, and Institution.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> create view PLACED
  2 as select Name, Age, Institution
  3 from RECORD
  4 where gpa>9.0;

View created.

SQL> select * from placed;

NAME                AGE INSTITUTION
-----
Anbalagan           22 PSG
Balu                 22 PSG
Rajan                22 PSG
Lavanya             21 CIT

SQL> |

```

Fig. 4.88. View PLACED from base table RECORD

4.15.1 Nonupdatable View

Case 1: A view created using DISTINCT clause is usually nonupdatable.

Example

To prove that the view created using DISTINCT clause is nonupdatable, consider the base relation SAMPLE, which has two attributes Name and Age. Let us create a view UPSAMPLE from the base relation SAMPLE using DISTINCT clause. The base relation SAMPLE and the view UPSAMPLE is shown below:

SAMPLE		
Roll No	Name	Age
1	Anand	20
2	Anandi	19
3	Banu	20
4	Chandran	20
5	Ravi	21
6	Chandran	21
7	Anand	20

UPSAMPLE	
Name	Age
Anand	20
Anandi	19
Banu	20
Chandran	20
Chandran	21
Ravi	21

The SQL command to create the view UPSAMPLE from the base relation SAMPLE using DISTINCT clause is shown in Fig. 4.89.

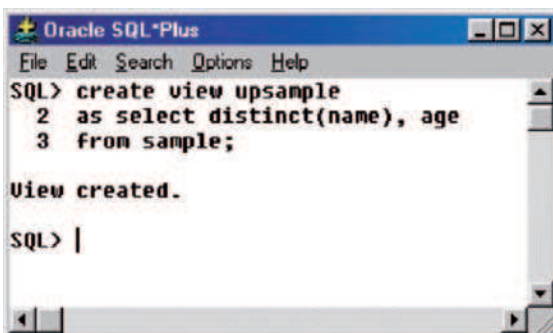
The created view UPSAMPLE is shown in Fig. 4.90. Now let us try to update the view UPSAMPLE, the SQL command to update the view and the corresponding output are shown in Fig. 4.91.

From Fig. 4.91, it is clear that the view defined by DISTINCT clause is nonupdatable.

Case 2: It is not possible to update the view if it contains group function or if it contains group by clause.

Example

In order to prove that the view is nonupdatable if it contains group function or group by clause, let us consider the base relation BOOKS. The attributes of the relation BOOKS are author, title, price. The content of the base relation BOOKS is shown in Fig. 4.92. Now let us define the view COUNTS, which gives the number of books written by the author. The SQL syntax to create the view is shown in Fig. 4.93. The contents of the view COUNTS are shown in Fig. 4.94.



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> create view upsample
  2  as select distinct(name), age
  3  from sample;

View created.

SQL> |

```

VIEW
upsample is
created using
distinct clause.

Fig. 4.89. VIEW creation using DISTINCT

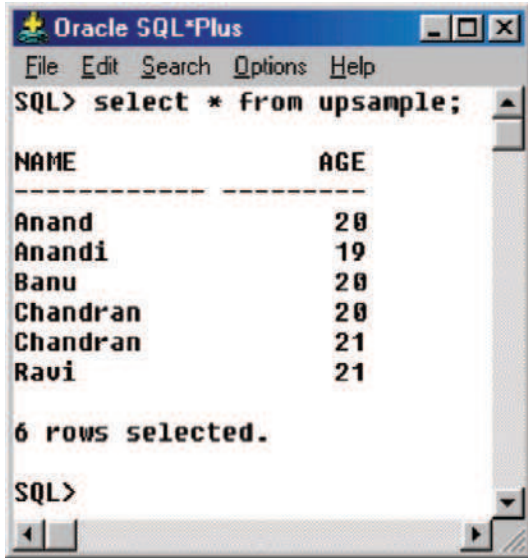


Fig. 4.90. Contents of the view UPSAMPLE

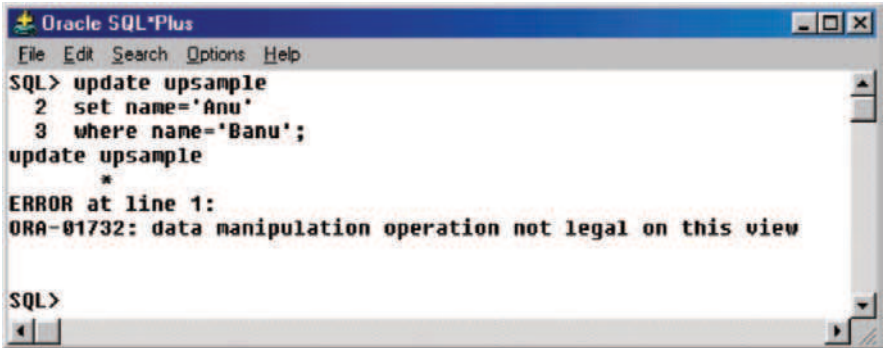


Fig. 4.91. Result of update operation in nonupdatable view

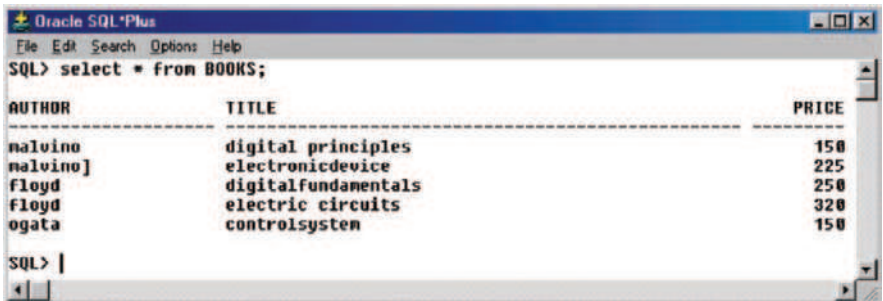
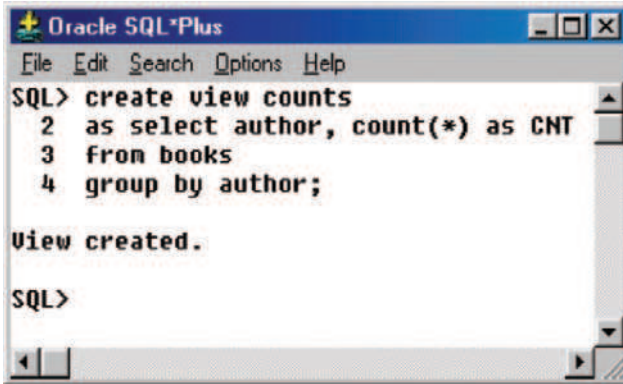


Fig. 4.92. The base relation BOOKS



```

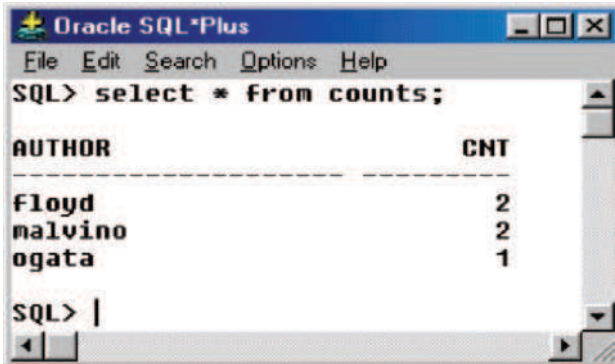
Oracle SQL*Plus
File Edit Search Options Help
SQL> create view counts
  2 as select author, count(*) as CNT
  3 from books
  4 group by author;

View created.

SQL>

```

Fig. 4.93. View COUNTS from BOOKS



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select * from counts;

AUTHOR                                CNT
-----                                -
Floyd                                  2
malvino                                2
ogata                                   1

SQL> |

```

Fig. 4.94. Contents of COUNTS

Try1

First let us try to delete a row from the view COUNTS. The SQL command to delete a row from the view COUNTS and the corresponding output are shown in Fig. 4.95. From Fig. 4.95, it is clear that it is not possible to delete a row from the view if it is created using group function or group by clause.

Try2

Now let us try to update the view COUNTS by modifying the name malvino to malvinoleech. The SQL command to modify the name in the view COUNTS and the corresponding output are shown in Fig. 4.96.

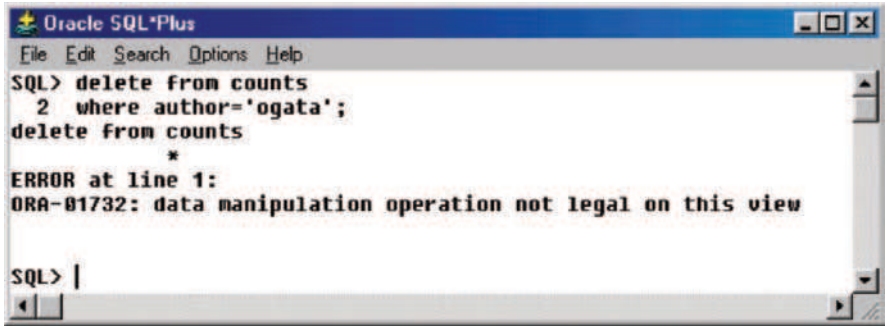
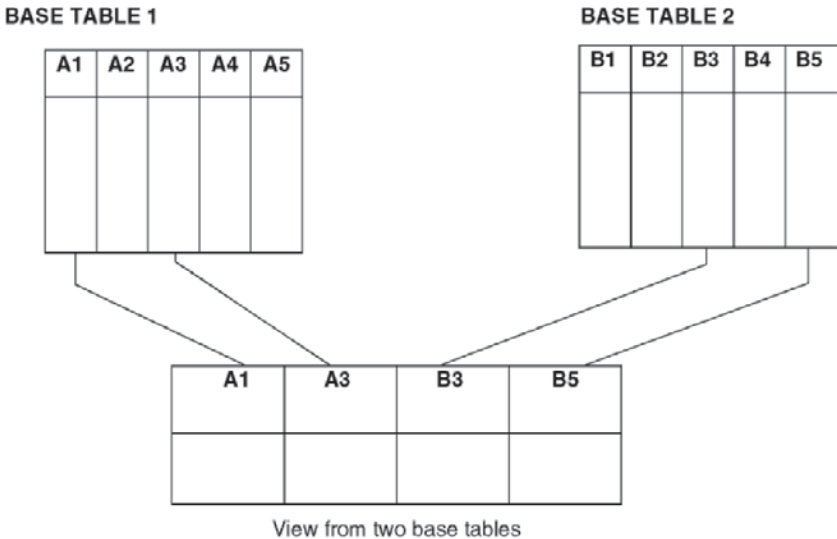


Fig. 4.95. Deletion of row in the view COUNTS

From Fig. 4.96, it is clear it is not possible to update the view if it contains group function or group by clause.

4.15.2 Views from Multiple Tables

Views from multiple tables are termed as complex views, whereas views from single table are termed as simple views. View from multiple tables is illustrated as follows:



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> update counts
  2 set author='malvinoleech'
  3 where author='malvino';
update counts
      *
ERROR at line 1:
ORA-01732: data manipulation operation not legal on this view

SQL>

```

Fig. 4.96. View updation

Example

Let us try to create view from two tables. Here one table is COURSE and the other table is STAFF. The attribute of the COURSE table are courseID, course name, LectID (which refers to Lecturer Identity number). The attributes of STAFF table are name, LectID, and position.

STAFF		
Name	LectID	Position
Rajan	E121	lecturer
Sridevi	E122	lecturer
Jayaraman	E123	professor
Navaneethan	E124	professor

COURSE		
CourseID	Course Name	LectID
C200	RDBMS	E121
C201	GraphTheory	E122
C202	DSP	E123
C203	OS(Operating System)	E124

The view COURSE_STAFF is created by selecting course name from course and Name from staff as shown in Fig. 4.97.

The SQL command to create the view COURSE_STAFF from COURSE and STAFF is shown in Fig. 4.98. From Fig. 4.98 it is evident that the view COURSE_STAFF is created from two tables COURSE and STAFF.

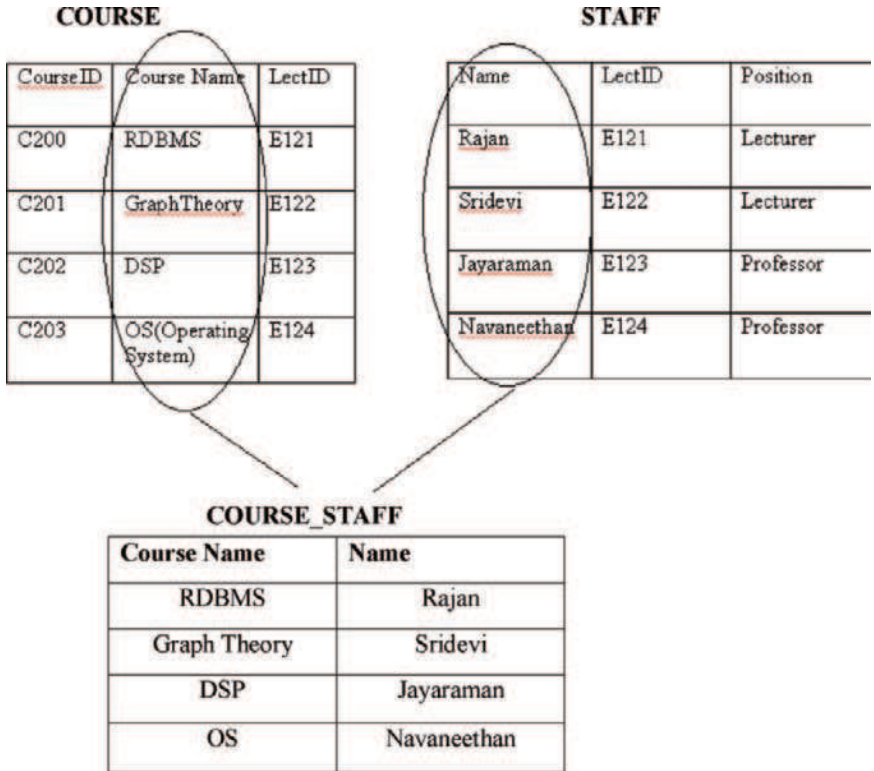


Fig. 4.97. View COURSE_STAFF from COURSE and STAFF

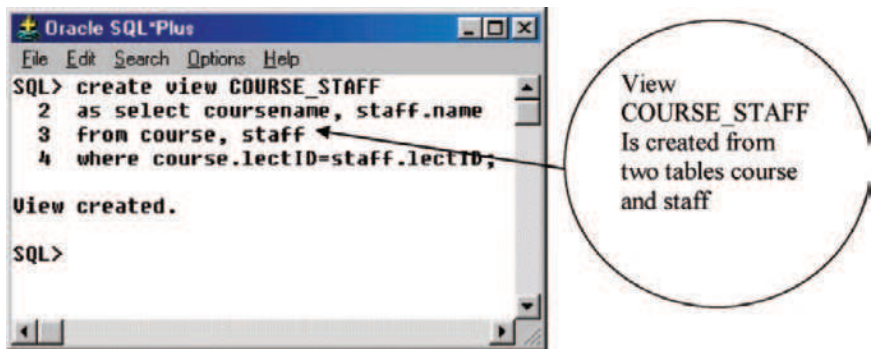


Fig. 4.98. VIEW from two tables

Let us try to see the contents of the view COURSE_STAFF by using SELECT command as shown in Fig. 4.99.

Note The view COURSE_STAFF is created from two tables, hence it can be considered as complex views. Complex views are in general not updatable. Let

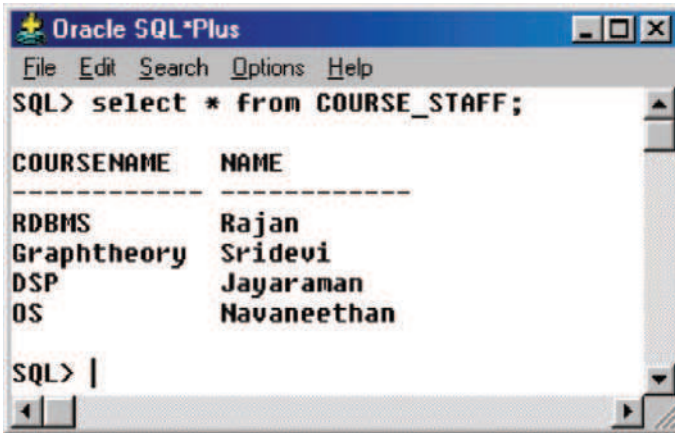


Fig. 4.99. Contents of the view COURSE_STAFF

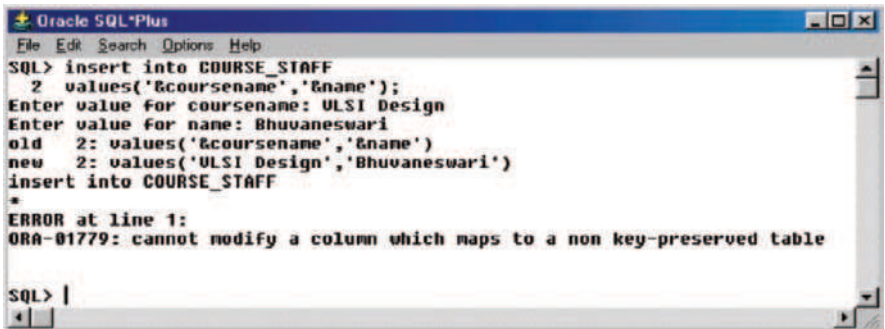


Fig. 4.100. View nonupdatable

us check whether the view COURSE.STAFF is updatable or not by trying to insert tuples into the view COURSE.STAFF as shown in Fig. 4.100.

From Fig. 4.100, it is clear that it is not possible to insert tuples into complex view (COURSE_STAFF). Now let us try to update the view COURSE.STAFF by modifying the name Rajan as Siva as shown in Fig. 4.101.

From Fig. 4.101, it is clear that the complex view (view created from more than one table) is usually nonupdatable.

4.15.3 View From View

It is possible to create view from another view. This is diagrammatically shown in Fig. 4.102. From Fig. 4.102, it is clear that the view2 is created from view1 and not from the base table. View1, View2 can be queried similar to the base table.

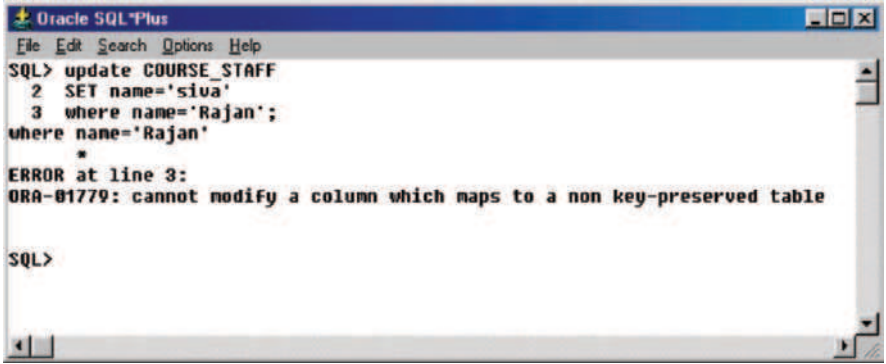


Fig. 4.101. Nonupdatable view COURSE_STAFF

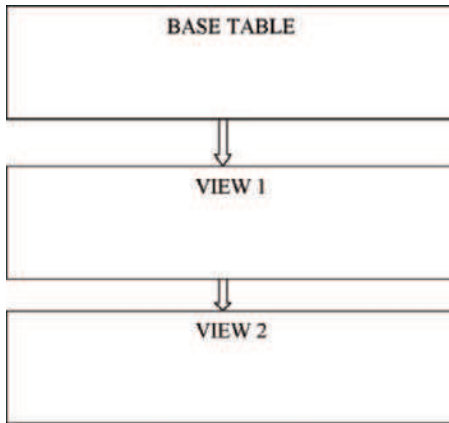


Fig. 4.102. View from a view

Example

Let us consider base table STAFF as shown in Fig. 4.103, the view ITSTAFF is created from the base table STAFF (Fig. 4.104). Then the view YOUNGITSTAFF is created from the view ITSTAFF (Fig. 4.105). The view ITSTAFF is shown in Fig. 4.106 and the view YOUNGITSTAFF is shown in Fig. 4.107.

Figure 4.104 shows the SQL command to create the view ITSTAFF from the base table STAFF. The view ITSTAFF contains only the details of the staff who belong to the IT department as shown in Fig. 4.104.

The contents of the view YOUNGITSTAFF is shown in Fig. 4.107. We can observe that the view YOUNGITSTAFF contains only the details of IT staff whose age is less than 30.

Doubt 1: Whether the view YOUNGSTAFF which is created from another view ITSTAFF can be queried like the base table?

Oracle SQL*Plus

```
File Edit Search Options Help
SQL> select * from STAFF;
```

EMPID	EMPNAME	DEPTNAME	SALARY	AGE
C202	ramakrishnan	electronics	24000	44
C201	Bhaskar	electrical	12500	24
C203	Mathew	electronics	23000	43
C204	Natrajan	IT	18500	38
C205	Krishnan	IT	17000	36
C206	Usha	electronics	20000	40
C207	Radha	IT	16000	24
C208	Jayakumar	IT	17000	26

```
8 rows selected.
SQL>
```

Fig. 4.103. Base table STAFF

Oracle SQL*Plus

```
File Edit Search Options Help
SQL> create view ITSTAFF
 2 as select *
 3 from staff
 4 where deptname='IT';

View created.

SQL>
```

Fig. 4.104. View ITSTAFF from base table STAFF

Oracle SQL*Plus

```
File Edit Search Options Help
SQL> create view YOUNGITSTAFF
 2 as select *
 3 from ITSTAFF
 4 where age<30;

View created.

SQL> |
```

Fig. 4.105. View YOUNGITSTAFF from the view ITSTAFF

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select * from ITSTAFF;

```

EMPID	EMPNAME	DEPTNAME	SALARY	AGE
C204	Natrajan	IT	18500	38
C205	Krishnan	IT	17000	36
C207	Radha	IT	16000	24
C208	Jayakumar	IT	17000	26

```

SQL>

```

Fig. 4.106. Contents of the view ITSTAFF

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select * from YOUNGITSTAFF;

```

EMPID	EMPNAME	DEPTNAME	SALARY	AGE
C207	Radha	IT	16000	24
C208	Jayakumar	IT	17000	26

```

SQL> |

```

Fig. 4.107. Contents of the view YOUNGITSTAFF

Answer : Yes. The view YOUNGITSTAFF, which is created from another view ITSTAFF can be queried like the base table.

Example

Let us consider the query: What is the pay offered to the YOUNGITSTAFF Radha? The SQL command to answer the query is shown in Fig. 4.108.

From Fig. 4.108, it is clear that the view YOUNGITSTAFF which is created from another view ITSTAFF can be queried similar to the base table STAFF

Doubt 2: If it is possible to make any change in the view ITSTAFF which was created from the base table STAFF, will it reflect in the base table STAFF.

Answer : Yes, if it is possible to make any change in the view which was derived from the base table then the change will be reflected in the base table.

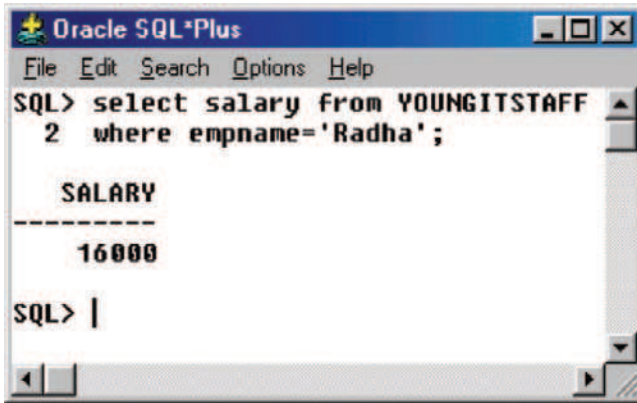


Fig. 4.108. Query on YOUNGITSTAFF

The screenshot shows the Oracle SQL*Plus interface displaying the contents of the STAFF table. The command prompt shows the following SQL query and its result:

```
SQL> select * from staff;
```

EMPID	EMPNAME	DEPTNAME	SALARY	AGE
C202	ramakrishnan	electronics	24000	44
C201	Bhaskar	electrical	12500	24
C203	Mathew	electronics	23000	43
C204	Natrajan	IT	18500	38
C205	Krishnan	IT	17000	36
C206	Usha	electronics	20000	40
C207	Radha	IT	16000	24
C208	Jayakumar	IT	17000	26

8 rows selected.

The prompt returns to SQL>

Fig. 4.109. Contents of the base table before any updation in the view ITSTAFF

Example

Let us modify the view ITSTAFF by including one row. Before modification the contents of the base table STAFF is shown in Fig. 4.109.

From Fig. 4.109, we can observe that there are eight rows in the base table STAFF.

Now let us update the view ITSTAFF by including one row in the view ITSTAFF. The SQL command to insert the row in the view ITSTAFF is shown in Fig. 4.110.

Contents of the ITSTAFF after inserting a row are shown in Fig. 4.111. From Fig. 4.111, we can observe that the new row being included in the ITSTAFF view.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> insert into ITSTAFF
  2 values('&empid','&empname','&deptname',&salary,&age);
Enter value for empid: C209
Enter value for empname: Akila
Enter value for deptname: IT
Enter value for salary: 13500
Enter value for age: 31
old 2: values('&empid','&empname','&deptname',&salary,&age)
new 2: values('C209','Akila','IT',13500,31)

1 row created.

SQL> |

```

Fig. 4.110. Insertion of a row into the view ITSTAFF

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select * from ITSTAFF;

EMPID      EMPNAME      DEPTNAME      SALARY      AGE
-----
C204      Natrajan      IT              18500        38
C205      Krishnan      IT              17000        36
C207      Radha         IT              16000        24
C208      Jayakumar     IT              17000        26
C209      Akila         IT              13500        31

SQL>

```

Fig. 4.111. Content of the view ITSTAFF after inserting a row

Now let us see the content of the base table STAFF to find whether the change made in the view ITSTAFF is reflected in the base table STAFF. The content of the base table STAFF is shown in Fig. 4.112.

Comparing Fig. 4.109 with Fig. 4.112 it is clear that one new row being included in the base table STAFF. This means that the change in the view will be reflected in the base table.

Doubt 3: If the view ITSTAFF is dropped, then is it possible to get the content of the view YOUNGITSTAFF which is derived from ITSTAFF?

Answer : For the view YOUNGITSTAFF, the contents are from another view ITSTAFF. Hence if ITSTAFF is dropped means it is not possible to get the contents of the view YOUNGITSTAFF.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select * from staff;

EMPID      EMPNAME      DEPTNAME      SALARY      AGE
-----
C202      ramakrishnan electronics    24000       44
C201      Bhaskar      electrical    12500       24
C203      Mathew       electronics    23000       43
C204      Natrajan     IT            18500       38
C205      Krishnan     IT            17000       36
C206      Usha         electronics    20000       40
C207      Radha        IT            16000       24
C208      Jayakumar    IT            17000       26
C209      Akila        IT            13500       31

9 rows selected.

SQL> |

```

Fig. 4.112. Content of the base table STAFF after modification in the view ITSTAFF

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> drop view ITSTAFF;

View dropped.

SQL> |

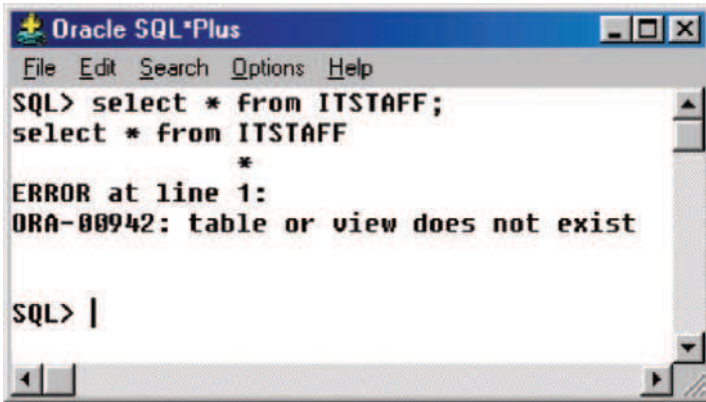
```

Fig. 4.113. Dropping the view

Example

Let us drop the view ITSTAFF as shown in Fig. 4.113. Figure 4.114 ensures that the view ITSTAFF is successfully dropped.

Now let us try to see the content of the view YOUNGITSTAFF which is derived from the view ITSTAFF. The SQL command to retrieve the contents of the view YOUNGITSTAFF is shown in Fig. 4.115.



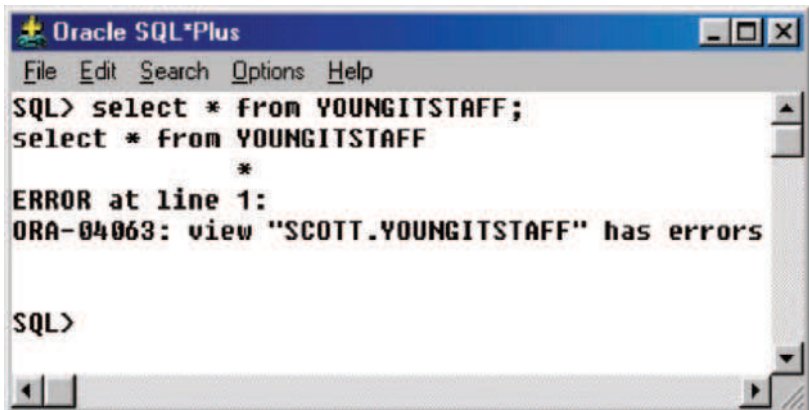
```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select * from ITSTAFF;
select * from ITSTAFF
      *
ERROR at line 1:
ORA-00942: table or view does not exist

SQL> |

```

Fig. 4.114. Contents after dropping the view



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select * from YOUNGITSTAFF;
select * from YOUNGITSTAFF
      *
ERROR at line 1:
ORA-04063: view "SCOTT.YOUNGITSTAFF" has errors

SQL>

```

Fig. 4.115. Contents of YOUNGITSTAFF after dropping the view ITSTAFF

From Fig. 4.115, it is clear that once the view ITSTAFF is dropped then it is not possible to retrieve the contents of the view YOUNGITSTAFF which is derived from the view ITSTAFF.

4.15.4 VIEW with CHECK Constraint

It is possible to create view with CHECK constraint. If we create a view with CHECK constraint, then it is not possible to update the view if the CHECK constraint is violated.

Example of View with CHECK Constraint

Let us consider the base relation CITIZEN which has the attributes name, age, and address. Now let us create the view VOTERS from the base relation

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select * from citizen;

NAME                AGE ADDRESS
-----
Anand                23 45, Main road, Trichy.
Anbu                 25 55, Kallurinagar, Coimbatore
Babu                 22 52, Peelamedu, Coimbatore
Chitra               45 32, Anbunagar, Trichy

SQL>

```

Fig. 4.116. Contents of base table CITIZEN

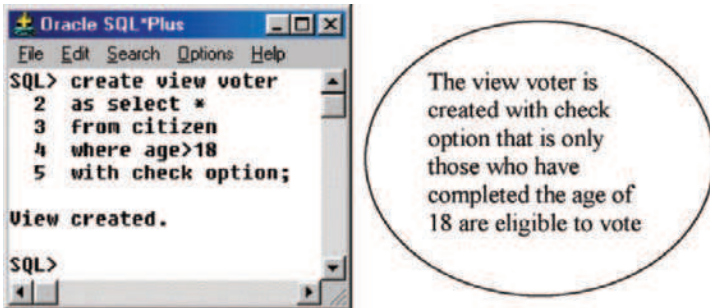


Fig. 4.117. View with check option

CITIZEN. We know that, the citizen of India becomes eligible voter if he/she attains the age of 18. The base relation CITIZEN is shown in Fig. 4.116. The view VOTER from base relation CITIZEN is shown in Fig. 4.117.

Case 1: Let us try to insert value into the view voter who is eligible to vote, that is the age of the voter is greater than 18. The SQL command and the corresponding output are shown in Fig. 4.118. From Fig. 4.118, it is clear that the value is successfully inserted into the view VOTER.

Case 2: Let us try to insert a row into the view VOTER by violating the check constraint (age of the voter is less than 18). The SQL command and the corresponding output are shown in Fig. 4.119.

4.15.5 Views with Read-only Option

A view can be created with read only option. Such views cannot be modified using INSERT, DELETE, and UPDATE commands.

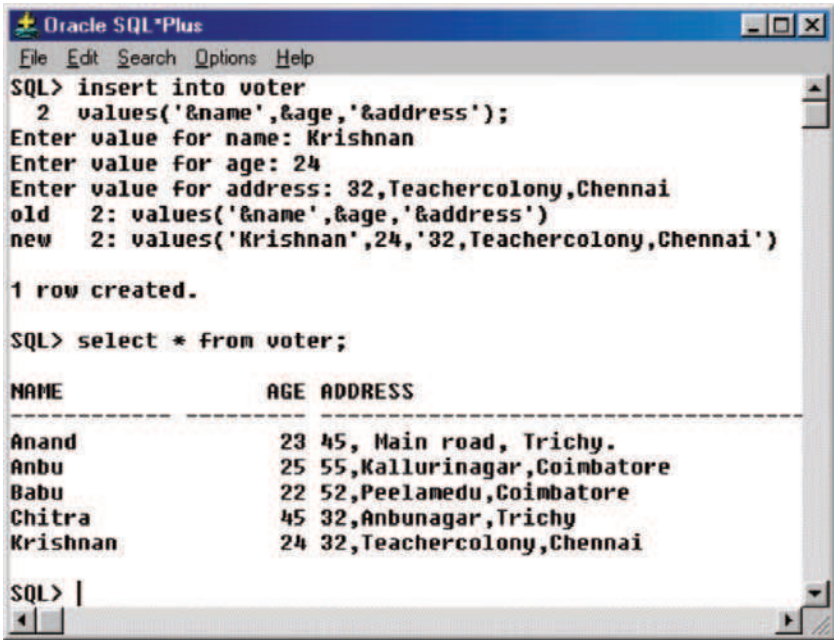


Fig. 4.118. Inserting valuable record into the view VOTER

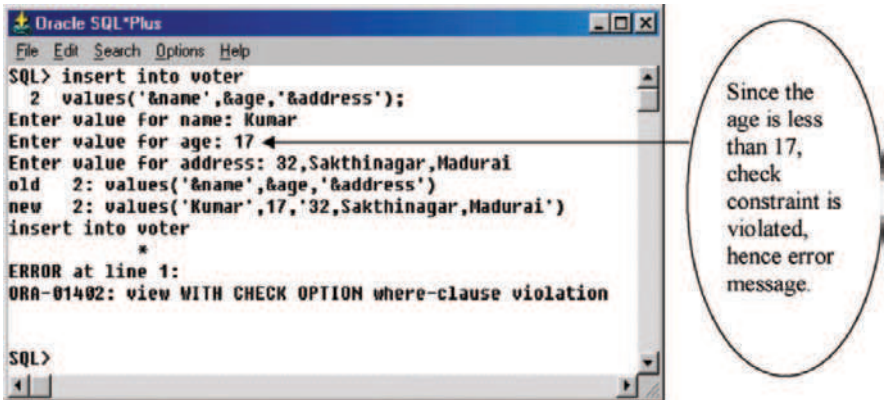
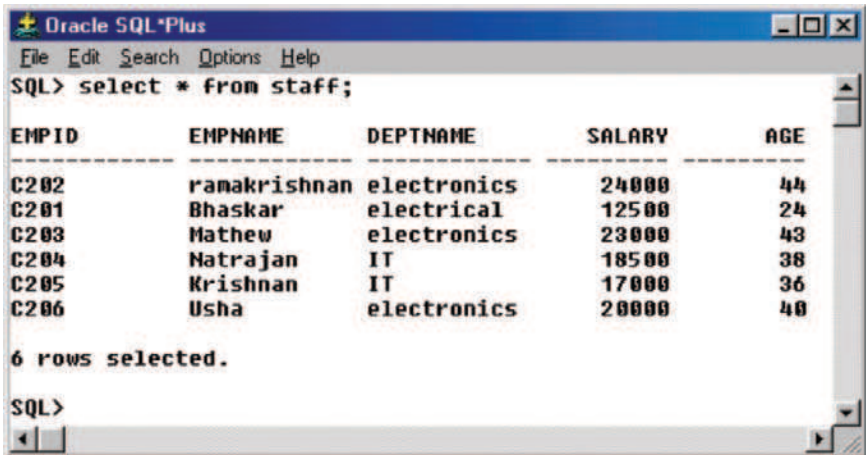


Fig. 4.119. Violation of check constraint

Example

Consider the base table STAFF as shown in Fig. 4.120. Let us create the view electronicsstaff from the base table staff with readonly option as shown in Fig. 4.121.



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select * from staff;

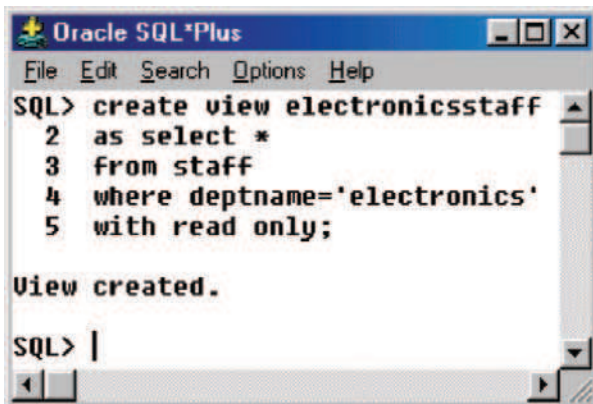
EMPID      ENPNAME      DEPTNAME      SALARY      AGE
-----
C202      ramakrishnan electronics      24000      44
C201      Bhaskar      electrical      12500      24
C203      Mathew      electronics      23000      43
C204      Natrajan      IT      18500      38
C205      Krishnan      IT      17000      36
C206      Usha      electronics      20000      40

6 rows selected.

SQL>

```

Fig. 4.120. Base table STAFF



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> create view electronicsstaff
2 as select *
3 from staff
4 where deptname='electronics'
5 with read only;

View created.

SQL> |

```

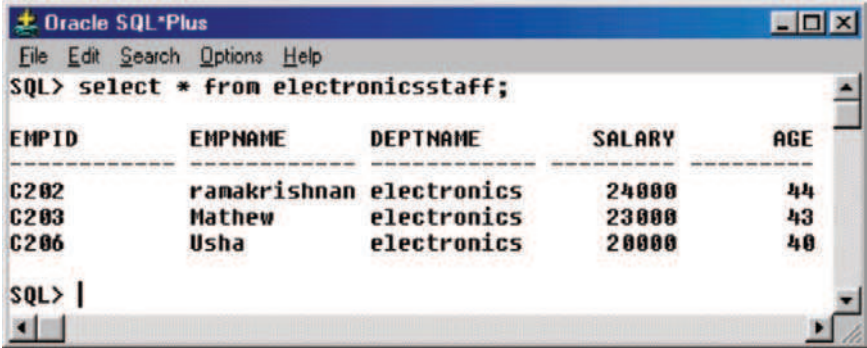
Fig. 4.121. View with read only option

From Fig. 4.121 it is clear that the view `electronicsstaff` is created with read only option. Now we have to check whether the view `electronicsstaff` is updatable, that is whether it is possible to `INSERT`, `DELETE` and `UPDATE` values in the view `electronicsstaff`. The content of the view `electronicsstaff` is shown in Fig. 4.122.

Case 1: INSERTING Values into the Read-Only View.

Let us try to insert values into the view “`electronicsstaff`.” The SQL command and the corresponding output are shown in Fig. 4.123.

From Fig. 4.123, it is clear that it is not possible to insert values into a read-only view.



```

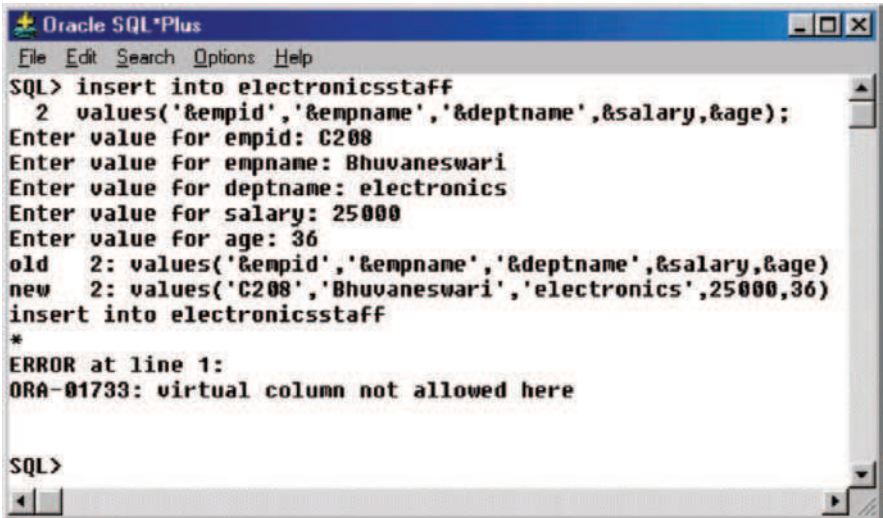
Oracle SQL*Plus
File Edit Search Options Help
SQL> select * from electronicsstaff;

EMPID      EMPNAME      DEPTNAME      SALARY      AGE
-----
C202      ramakrishnan electronics      24000      44
C203      Mathew       electronics      23000      43
C206      Usha         electronics      20000      40

SQL> |

```

Fig. 4.122. Contents of the read only view electronicsstaff



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> insert into electronicsstaff
  2 values('&empid','&empname','&deptname',&salary,&age);
Enter value for empid: C208
Enter value for empname: Bhuvaneshwari
Enter value for deptname: electronics
Enter value for salary: 25000
Enter value for age: 36
old  2: values('&empid','&empname','&deptname',&salary,&age)
new  2: values('C208','Bhuvaneshwari','electronics',25000,36)
insert into electronicsstaff
*
ERROR at line 1:
ORA-01733: virtual column not allowed here

SQL>

```

Fig. 4.123. Insertion of values into read-only view

Case 2: Deleting value from a read-only view.

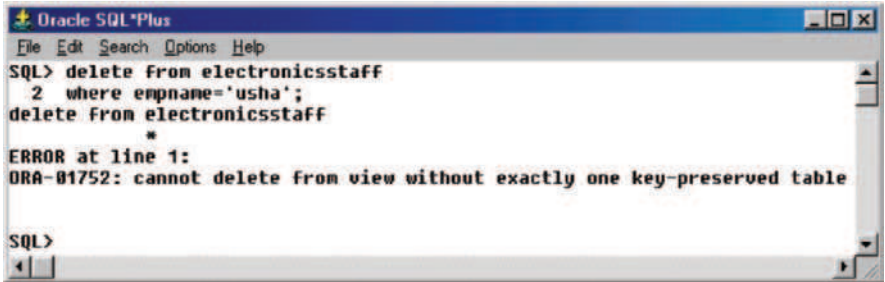
Let us try to delete a value (in our case deleting the record of the electronicsstaff “usha”) from the read-only view “electronicsstaff,” the SQL command and the corresponding output are shown in Fig. 4.124.

From Fig. 4.124, it is evident that it is not possible to delete value from the read-only view.

Case 3: Updating the record of read-only view.

Let us try to update the record of the read-only view “electronicsstaff” by modifying the age of “usha” to 30. The SQL command to modify the age of the staff “usha” and the corresponding output are shown in Fig. 4.125.

From Fig. 4.125, it is clear that it is not possible to update the view since it is read-only.



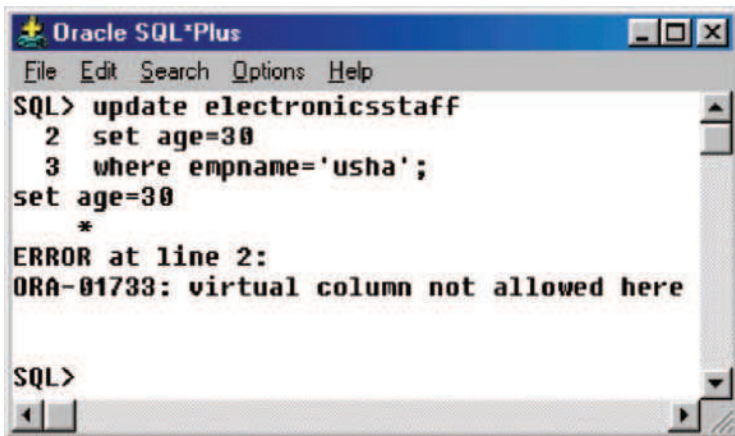
```

Oracle SQL*Plus
File Edit Search Options Help
SQL> delete from electronicsstaff
  2  where empname='usha';
delete from electronicsstaff
  *
ERROR at line 1:
ORA-01752: cannot delete from view without exactly one key-preserved table

SQL>

```

Fig. 4.124. Deleting a tuple from read-only view



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> update electronicsstaff
  2  set age=30
  3  where empname='usha';
set age=30
  *
ERROR at line 2:
ORA-01733: virtual column not allowed here

SQL>

```

Fig. 4.125. Updating the record in read-only view

4.15.6 Materialized Views

A materialized view is a physical copy of the base table with the results moved to another schema object. Materialized views are also called snapshots, because they are a kind of photograph of the base table.

Advantage of VIEW

The main advantages of view are improved security, less complexity, better convenience, and customization.

1. *Improved security.* We can restrict the user to access on the data that are appropriate for the user. Hence views provide improved security.
2. *Less complexity.* A view can simplify queries, by getting data from several tables into a single table thus transforming multitable queries into a single table queries.

3. *Convenience.* A database may contain much information. All the information will not be useful to the users. The users are provided with only the part of the database that is relevant to them rather than the entire database; hence views provide great convenience to the users.
4. *Customization.* Views provide a method to customize the appearance of the database so that the users need not see full complexity of database. View creates the illusion of a simpler database customized to the needs of a particular category of users.

Drawback of VIEW

1. If the base table is modified by adding one or more columns then the columns added will not be available in the view unless it is recreated.
2. When a view is created from the base table, it is to be noted that all the views are not updatable. Views created from multiple tables are in general not updatable when there is a group function, a GROUP BY clause, or restriction operators.

4.16 Subquery

Subquery is query within a query. A SELECT statement can be nested inside another query to form a subquery. The query which contains the subquery is called outer query.

Scalar subquery

A scalar subquery returns single row, single column result.

Example of Scalar Subquery

Scalar subquery returns single row single column result. To understand scalar subquery, consider two relations STUDENT and COURSE. The attributes of the STUDENT relation are SID, SNAME, AGE, and GPA. The attributes of COURSE relation are CID (Course ID), CNAME (Course ID), SID (Student ID), and INSTRUCTOR (Name of the Instructor). The two relations are shown below.

STUDENT			
SID	SNAME	AGE	GPA
E100	Anbu	21	9.6
E101	Aravind	21	9.2
E102	Balu	21	9.4
E103	Chitra	22	8.8
E104	Sowmya	21	9.8

COURSE			
CID	CNAME	SID	INSTRUCTOR
C100	RDBMS	E100	Rajan
C101	OS	E102	Sumathi
C102	DSP	E101	Jayaraman
C103	DSP	E104	Jayaraman

Query 1: Find the name of the student who has opted for the course RDBMS?

Solution. From the STUDENT and COURSE table, it is clear that only one student has opted for RDBMS (just for example). We can get the name of the student using scalar subquery. The SQL command and the corresponding output are shown in Fig. 4.126.

Query 2: Find the Names of the Student who have Opted for DSP Course

Solution. From the STUDENT and COURSE table, we can observe that more than one student has opted for DSP course. Here we cannot use scalar subquery because scalar subquery gives single row and single column result. But our result has more than one row. First let us try to get by scalar subquery. The SQL command and the corresponding output are shown in Fig. 4.127.

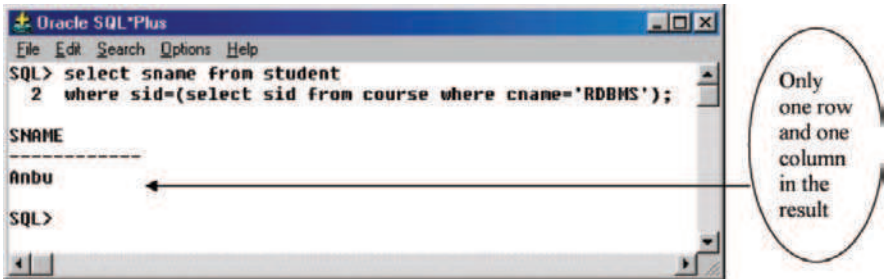


Fig. 4.126. Scalar subquery

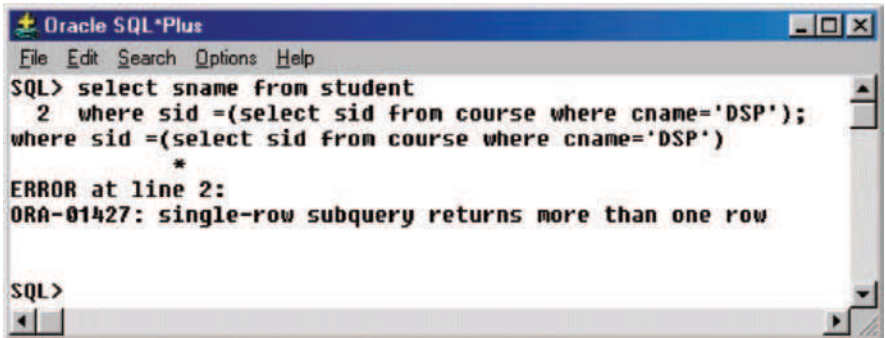


Fig. 4.127. Wrong use of scalar subquery

From Fig. 4.127, it is clear that scalar subquery cannot be used to retrieve multiple rows or multiple column result.

The solution to get the name of the student who has opted for DSP course is to use IN operator. The IN operator is true if value exists in the result of subquery. The SQL command using IN operator and the corresponding output are shown in Fig. 4.128.

4.16.1 Correlated Subquery

In the case of correlated subquery, the processing of subquery requires data from the outer query.

EXISTS Operator in Correlated Subquery

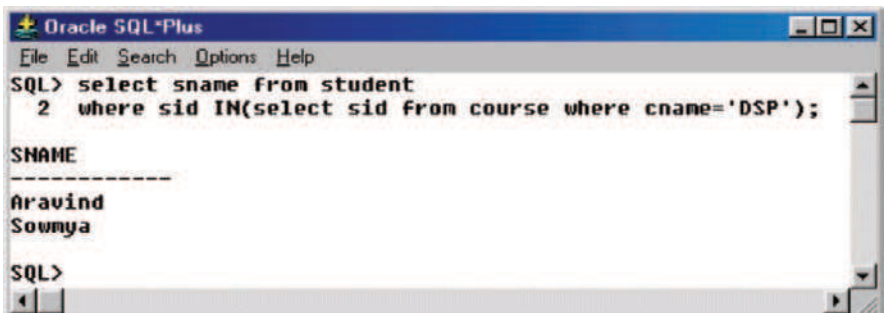
The **EXISTS** operator is used in correlated subquery to find whether a value retrieved by the outer query exists in the results set of the values retrieved by the inner query or subquery.

Example of EXISTS Command

Let us consider two tables ORDER1 and PRODUCT. The attributes (columns) of the table ORDER1 are orderID, quantity, productID. The attributes of the table PRODUCT are productID, productname, and price. The contents of the two table ORDER1 and PRODUCT are shown in Figs. 4.129 and 4.130.

The orderID which gives the order for the car “Maruti Esteem” can be found using the SQL command EXISTS. The SQL command and the corresponding output are shown in Fig. 4.131.

From Fig. 4.131, we can observe that the data for the inner query require the data from the outer query.



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select sname from student
  2  where sid IN(select sid from course where cname='DSP');

SNAME
-----
Aravind
Soumya

SQL>

```

Fig. 4.128. Subquery to return multiple row result

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select * from order1;
ORDERID          QUANTITY  PRODUCTID
-----
C121              3         P122
C122              2         P132
C123              5         P142
SQL>

```

Fig. 4.129. Table order1

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select * from product;
PRODUCTID  PRODUCTNAME  PRODUCTPRICE
-----
P122       Marutiesteem  200000
P132       Santro       300000
P142       Fordicon     400000
SQL>

```

Fig. 4.130. Table product

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select orderID from order1
2  where exists
3  (select *
4  from product
5  where order1.productID=product.productID
6  AND product.productname='Marutiesteem');
ORDERID
-----
C121
SQL>

```

Fig. 4.131. Data retrieval using EXISTS command

Example of NOT EXISTS Operator

In order to understand NOT EXISTS clause, let us consider two relations EMPLOYEE and DEPENDENT. Here DEPENDENT refers to those who are dependent on EMPLOYEE. The attributes of EMPLOYEE relation are eid (employee ID), ename (employee name). The attributes of the DEPENDENT relation are name (which refers to dependent name) and eid (employee ID).

The screenshot shows the Oracle SQL*Plus interface with the command `SQL> select * from employee;` entered. The output is a table with two columns: EID and ENAME. The data rows are as follows:

EID	ENAME
C101	Rangan
C102	Mohan
C103	Kunar
C104	Jayaraman
C105	Selvan

Fig. 4.132. EMPLOYEE table

The screenshot shows the Oracle SQL*Plus interface with the command `SQL> select * from dependent;` entered. The output is a table with two columns: NAME and EID. The data rows are as follows:

NAME	EID
Radha	C105
Subha	C103
Chitra	C101

Fig. 4.133. DEPENDENT table

The contents of the table EMPLOYEE and DEPENDENT are shown in Figs. 4.132 and 4.133.

Query: Find the name of the employee who is not having any dependent?

Solution. The SQL command to get the name of the employee who is not having any dependent and the corresponding output are shown in Fig. 4.134.

The NOT EXISTS clause is used to retrieve the name of the employee who is not having dependent.

Comparison Operator ALL

The comparison operators that are used in multiple row subqueries are IN, ANY, ALL. In this section let us discuss the use of ALL comparison operator. The ALL comparison operator compare value to every value returned by the subquery.

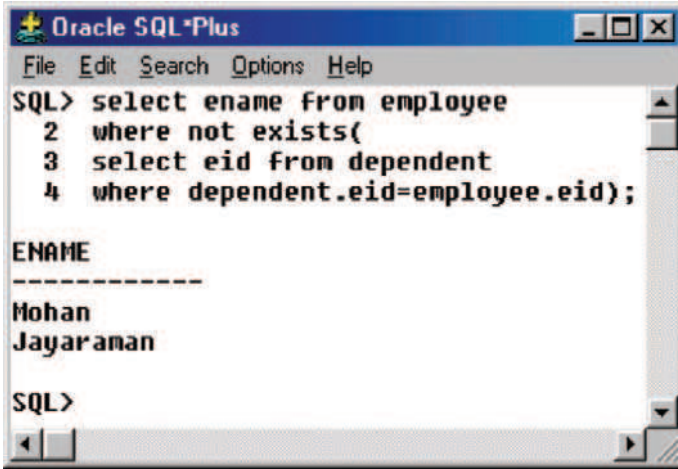


Fig. 4.134. NOT EXISTS command

Example

In order to understand the ALL comparison operator, let us consider the relation STAFF. The attributes of the staff relation are shown in table STAFF.

STAFF				
EMPID	EMPNAME	DEPTNAME	AGE	SALARY
C201	Bhaskar	Electrical	24	12,500
C202	Ramakrishnan	Electronics	44	24,000
C203	Mathew	Electronics	43	23,000
C204	Natrajan	IT	38	18,500
C205	Krishnan	IT	36	17,000
C206	Usha	Electronics	40	20,000

Query: Find the name of the employee in Electronics Department who is getting the maximum salary?

Solution: The SQL command ALL can be used to find the name of the STAFF in the Electronics who is getting maximum salary. The SQL command and the corresponding output are shown in Fig. 4.135.

Here the ALL comparison operator is used to retrieve the name of the staff from a particular department who is getting maximum salary.

Comparison Operator ANY

The ANY operator compares a value to each value returned by a subquery. Here <ANY means less than maximum

>ANY means more than the minimum

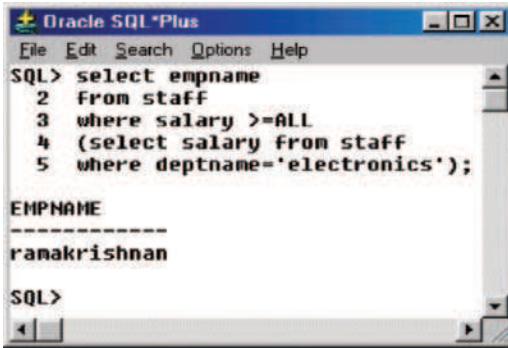


Fig. 4.135. Use of ALL comparison operator

Case 1: <ANY. Let us use the operator <ANY to retrieve the names of the staff who are getting salary less than the staff who is getting the maximum salary (in our case it is “ramakrishnan”).

The SQL command to retrieve the names of the staff who are getting salary less than the staff who is getting the maximum salary is shown in Fig. 4.136.

The SQL command <ANY is used to retrieve the name of the staff who are getting the salary less than the staff who is getting the maximum salary. In our case staff “ramakrishnan” of electronics is getting the maximum salary (refer STAFF table). Our query should return the name of the staff who are getting salary less than the staff “ramakrishnan.” From Fig. 4.136, it is evident the name returned by the query are the staff who are getting salary less than the staff “ramakrishnan.”

Case 2: >ANY Clause. The operator >ANY returns values that are greater than the minimum value.

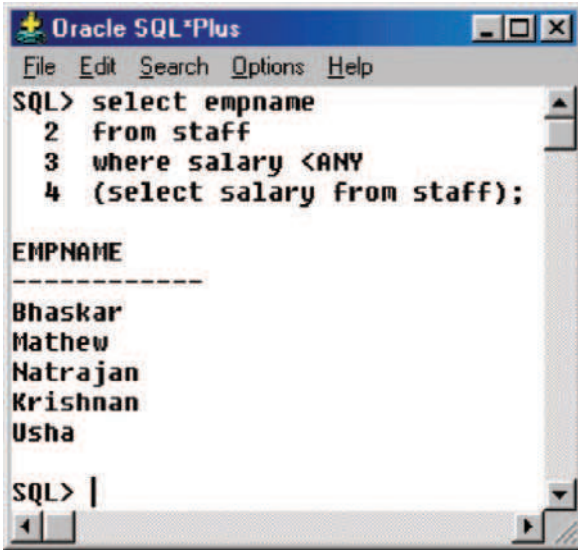
Example

Query: Retrieve the name of the staff who are getting salary greater than the staff who is getting the least salary?

Solution: The SQL operator >ANY can be used to get answer for the query mentioned above. The SQL command and the corresponding output are shown in Fig. 4.137.

From the table STAFF it is clear that the staff who is getting the least salary is “Bhaskar.” We have to get the names of the staff who are getting salary greater than “Bhaskar.”

From Fig. 4.137, it is clear that the operator >ANY has returned the names of the staff who are getting salary greater than “Bhaskar.”



```

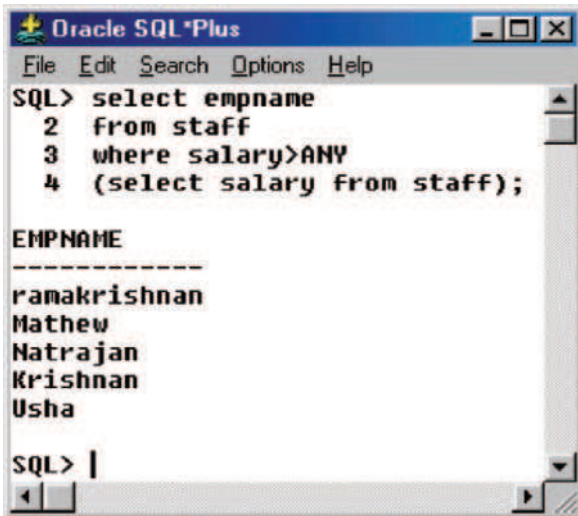
Oracle SQL*Plus
File Edit Search Options Help
SQL> select empname
  2  from staff
  3  where salary <ANY
  4  (select salary from staff);

EMPNAME
-----
Bhaskar
Mathew
Natrajan
Krishnan
Usha

SQL> |

```

Fig. 4.136. Use of <ANY clause



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select empname
  2  from staff
  3  where salary >ANY
  4  (select salary from staff);

EMPNAME
-----
ramakrishnan
Mathew
Natrajan
Krishnan
Usha

SQL> |

```

Fig. 4.137. Use of >ANY clause

Dual Table

The dual table contains one row and one column. The datatype associated with the dual table is varchar2(1). In order to know about dual table, we can issue DESC command as shown in Fig. 4.138.



Fig. 4.138. Description of dual table

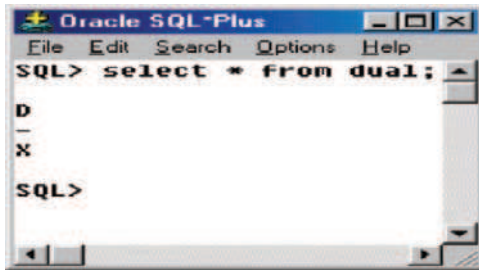


Fig. 4.139. Selection from Dual

From Fig. 4.138, it is clear that the name of the column is DUMMY. If we want to know how many rows that a DUAL table can return, we can issue SELECT command as shown in Fig. 4.139. From Fig. 4.139, it is clear that the dual table can return a row. Dual table can be used to compute a constant expression.

Determining System Date from Dual

It is possible to determine system date from the dual table. The SQL command and the corresponding output are shown in Fig. 4.140.

We have evaluated the system date from the dual table. It is also possible to evaluate constant expression using the dual table.

Evaluation of Constant Expression Using DUAL

It is possible to evaluate constant expressions using DUAL table. Some of the examples of evaluation of constant expressions are shown in Fig. 4.141.

From Fig. 4.141 it is clear that DUAL table can be used to evaluate constant expressions which will give single row output. For our example we have taken simple mathematical operations like addition, multiplication, division, and subtraction.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select sysdate from dual;
SYSDATE
-----
10-JUL-05
SQL> |

```

Fig. 4.140. System date from dual

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select 15+15 from dual;
15+15
-----
30
SQL> select 15*15 from dual;
15*15
-----
225
SQL> select 15/15 from dual;
15/15
-----
1
SQL> select 15-15 from dual;
15-15
-----
0
SQL> |

```

Fig. 4.141. Evaluation of constant expressions

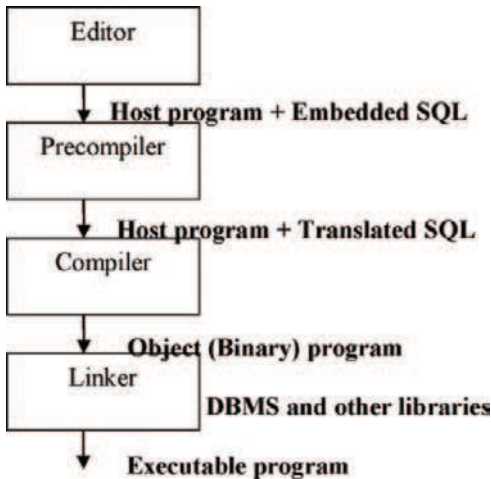
4.17 Embedded SQL

SQL can be used in conjunction with a general purpose programming language such as PASCAL, C, C++, etc. The programming language is called the host language. Embedded SQL statements are SQL statements written within application programming languages such as C and Java. The embedded SQL statement is distinguished from programming language statements by prefixing it with a special character or command so that a preprocessor

can extract the SQL statements. These statements are preprocessed by an SQL precompiler before the application program is compiled. There are two types of embedded SQL, Static SQL, and Dynamic SQL. Embedded SQL provides the 3GL (Third Generation Language) with a way to manipulate a database. Embedded SQL supports highly customized applications. It also supports background applications running without user intervention.

SQL Precompiler

A precompiler is used to translate SQL statements embedded in a host language into DBMS library calls, which can be implemented in the host language. The function of the precompiler is shown below:



Sharing Variables

Variables to be shared between the embedded SQL code and the host language have to be specified in the program.

```
EXEC SQL begin declare section;
    Varchar userid [10], password [10], cname [15];
    Int cno;
```

```
EXEC SQL end declare section;
```

We also should declare a link to the DBMS so that database status information can be accessed.

```
EXEC SQL include sqlca;
```

This allows access to a structure `sqlca`, of which the most common element `sqlca.sqlcode` has the value 0 (operation OK), >0 (no data found), and <0 (an error).

Connecting to the DBMS

Before operations can be performed on the database, a valid connection has to be established. A model is shown below:

```
EXEC SQL connect :userid identified by :password;
```

- In all SQL statements, variables with the “:” prefix refer to shared host variables, as opposed to database variables.
- This assumes that userid and password have been properly declared and initialized.

When the program is finished using the DBMS, it should disconnect using:

```
EXEC SQL commit release;
```

Queries Producing a Single Row

A single piece of data (or row) can be queried from the database so that the result is accessible from the host program.

```
EXEC SQL SELECT custname
```

```
INTO :cname
```

```
FROM customers
```

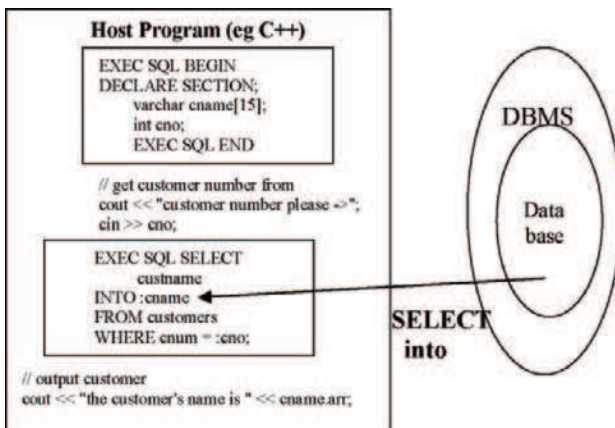
```
WHERE cno = :cno;
```

Thus the custname with the unique identifier :cno is stored in :cname.

However, a selection query may generate many rows, and a way is needed for the host program to access results one row at a time.

SELECT with a Single Result

The syntax to select with a single result is shown below:



Static SQL

The source form of a static SQL statement is embedded within an application program written in a host language such as COBOL. The statement is prepared before the program is executed and the operational form of the statement persists beyond the execution of the program.

A source program containing static SQL statements must be processed by an SQL precompiler before it is compiled. The precompiler turns the SQL statements into host language comments, and generates host language statements to invoke the database manager. The syntax of the SQL statements is checked during the precompile process.

The preparation of an SQL application program includes precompilation, the binding of its static SQL statements to the target database, and compilation of the modified source program.

Dynamic SQL

Programs containing embedded dynamic SQL statements must be precompiled like those containing static SQL, but unlike static SQL, the dynamic SQL statements are constructed and prepared at run time. The SQL statement text is prepared and executed using either the `PREPARE` and `EXECUTE` statements, or the `EXECUTE IMMEDIATE` statement. The statement can also be executed with cursor operations if it is a `SELECT` statement.

Summary

This chapter has introduced the most popular relational database language SQL (Structured Query Language). SQL has become the de facto standard language for interacting with all major database programs. The three main divisions in SQL are DDL, DML, and DCL. The data definition language (DDL) commands of SQL are used to define a database which includes creation of tables, indexes, and views. The data manipulation commands (DML) are used to load, update, and query the database through the use of the `SELECT` command. Data control language (DCL) is used to establish user access to the database.

This chapter has focused on how to create the table, how to insert data into the table. Examples are shown to understand the table creation and manipulation process. The subset of `SELECT` command described in this chapter allows the reader to formulate problems involving the project, restrict, join, union, intersection, and difference operators of relational algebra.

Review Questions

4.1. Prove the statement “When the column of a view is directly derived from a column of a base table, that column inherits any constraints that apply to the column of the base table” by using suitable example.

To prove this statement, let us create a base table by name t1. The base table t1 has two columns name and age. Now a constraint is imposed on the age, that is age should be greater than 18. The syntax to create the base table t1 with the constraint on the age is shown below:

Step 1: Base table creation with the name t1 and constraint on age (Fig. 4.142).

```
SQL> create table t1
2 (name varchar(12),
3 age number(3),
4 check(age>18));
Table created.
```

Step 2: Create a view by name t2 from the base table t1. The SQL command to create the view t2 is shown in Fig. 4.143.

Step 3: Now try to insert values into view t2 by not violating the constraint and then by violating the constraint (Fig. 4.144). Then try to insert values into the view t2 by violating the check constraint.

Note: Since the age is greater than 18 the values are inserted into view t2. Now insert value into t2 by violating the constraint (by inserting the age less than or equal to 18).

If we are violating the constraints on the column of the base table we are getting an error message.

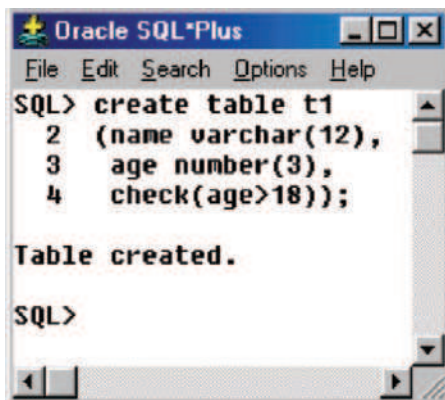


Fig. 4.142. Creation of table t1

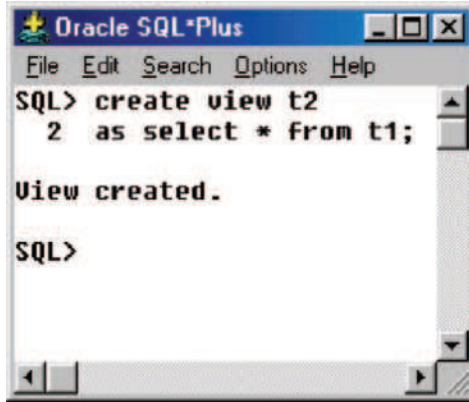


Fig. 4.143. Creation of view t2

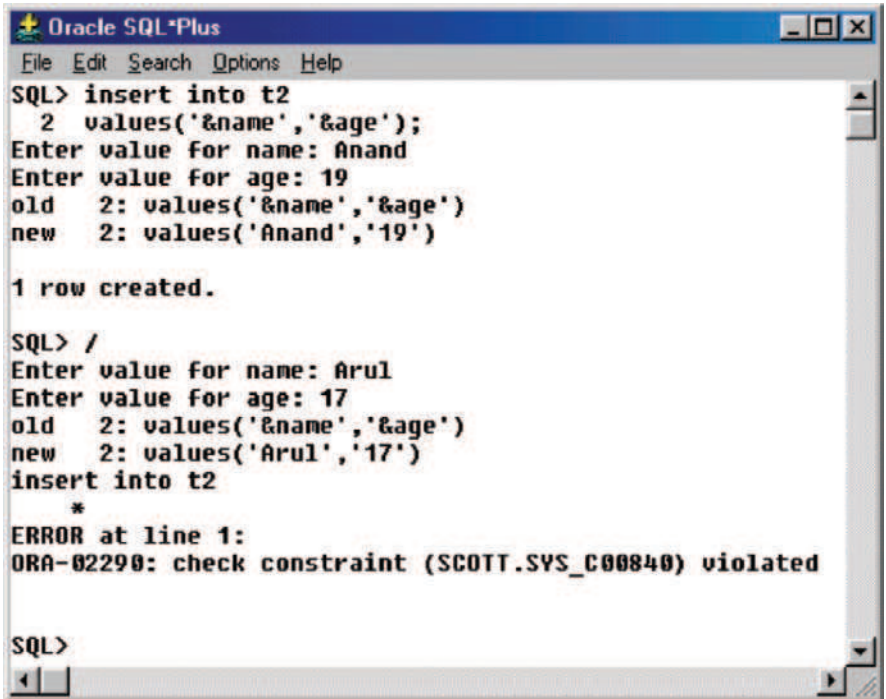


Fig. 4.144. Insertion of values into t2 without violating and violating constraint

4.2. What is the difference between the two SQL commands DROP TABLE and TRUNCATE TABLE?

Drop table command deletes the definition as well as the contents of the table, whereas truncate table command deletes only the contents of the table but not the definition of the table.

Example

We have a table by name t1. The contents of the table are seen by issuing the select command as shown in Fig. 4.145.

Step 1: Now issue the truncate table command. The syntax is:
TRUNCATE TABLE table name; as shown in Fig. 4.146.

Step 2: After issuing the truncate table command try to see the contents of the table. You will get the message as no rows selected as shown in Fig. 4.147.

Step 3: Now we have the table t2. See the contents of the table by issuing select command as shown in Fig. 4.148.

Step 4: Now use the drop command, to drop the table t2 as shown in Fig. 4.149.

Step 5: Now see the effect of the drop command by using the select command as shown in Fig. 4.150.

Note: If we issue the drop command, the definition as well as the contents of the table is deleted and we get the error message as shown in Fig. 4.150.

4.3. Is it possible to create a table from another table. If so give an example

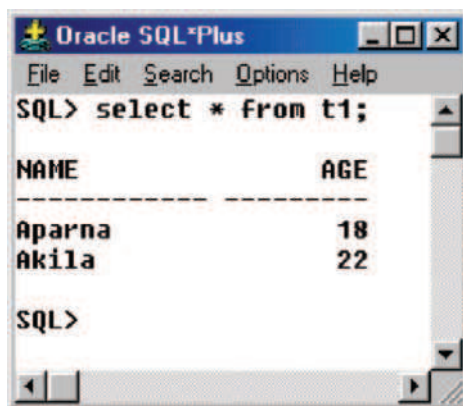


Fig. 4.145. Content of the table t1

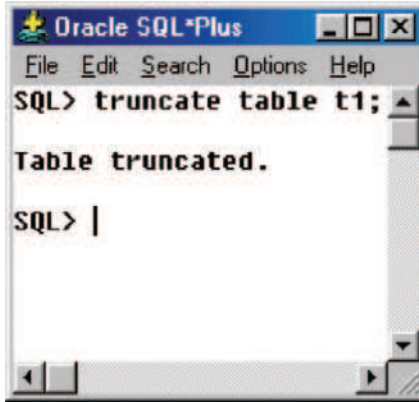


Fig. 4.146. Truncation of table t1

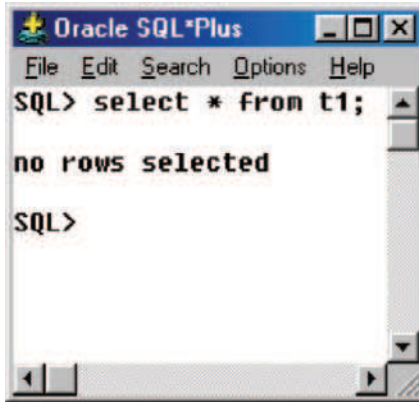


Fig. 4.147. Selection after truncation

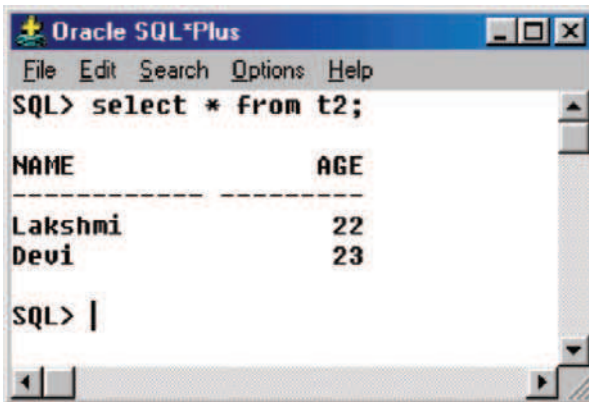


Fig. 4.148. Contents of the table t2

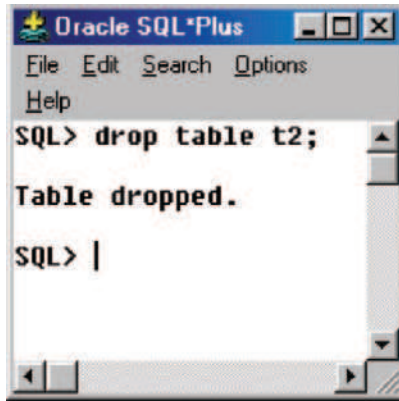


Fig. 4.149. Dropping the table t2

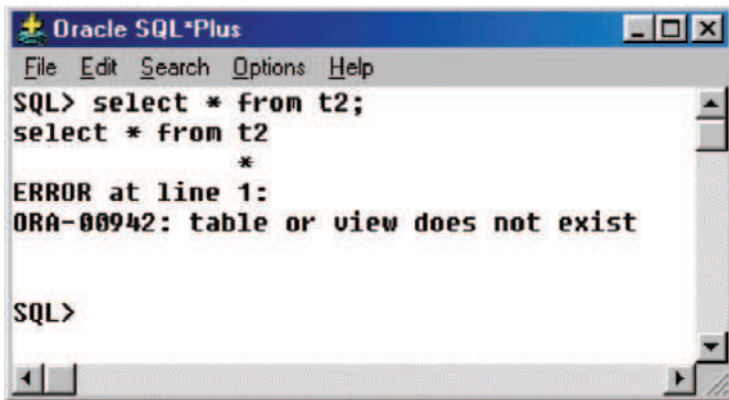


Fig. 4.150. Selection after dropping the table

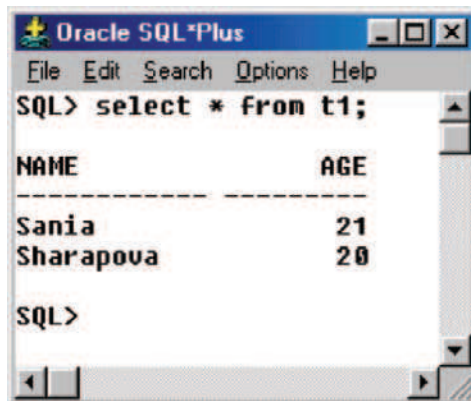


Fig. 4.151. Contents of table t1

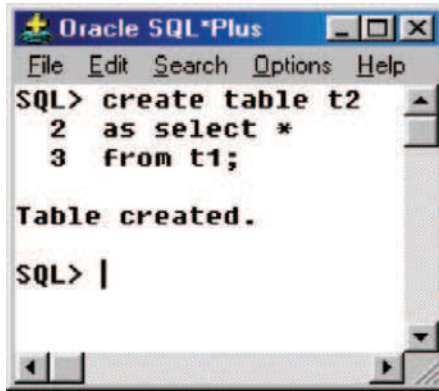


Fig. 4.152. Table t2 from table t1

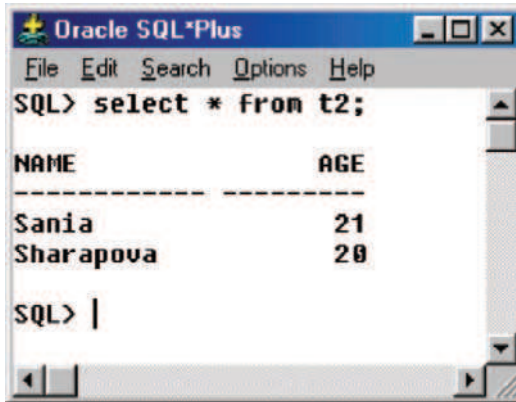


Fig. 4.153. Contents of table t2

Yes, it is possible to create table from another table using SQL. Consider table t1 as shown in Fig. 4.151. We can create another table t2 from the table t1. The SQL command to create the table t2 from the table t1 is shown in the Fig. 4.152.

Now let us try to view the content of the table t2. The content of the table t2 is shown in Fig. 4.153.

From Fig. 4.153, it is clear that the contents of the table t2 matches with the table t1 (refer Fig. 4.151). Hence it is possible to create table from another table.

4.4. What is the difference between COUNT, COUNT DISTINCT, and COUNT (*) in SQL?

The command COUNT counts the number of rows in a table by ignoring all null values. The command COUNT (*) counts the number of rows in a

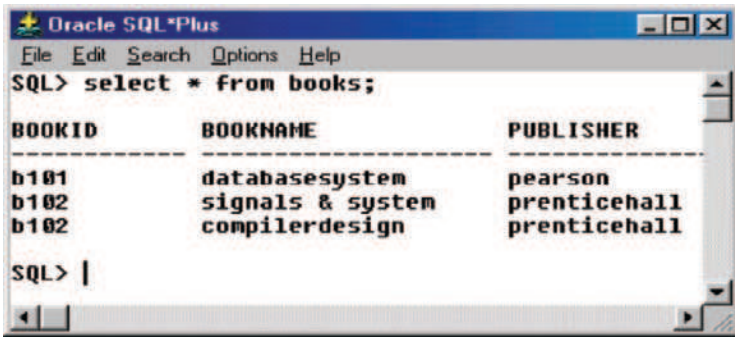


Fig. 4.154. Contents of the table BOOKS

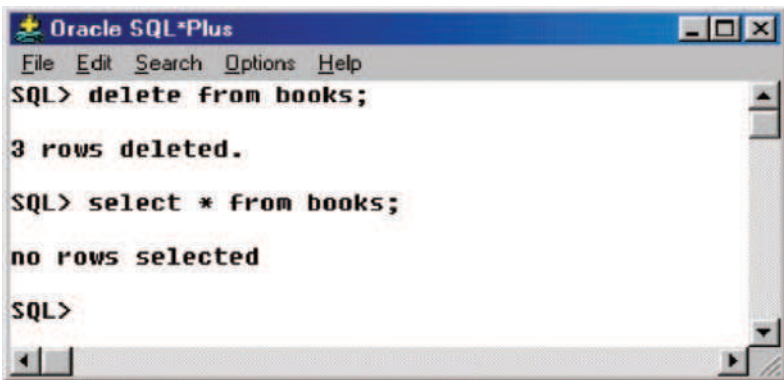


Fig. 4.155. Contents of the table BOOKS deleted using DELETE command

table by including the rows that contains null values. COUNT DISTINCT counts the number of rows in the table by ignoring duplicate values.

4.5. If we want to delete all the rows in the table, it can be done in two ways (1) Issue the command DELETE FROM table name (2) TRUNCATE TABLE table name. What is the difference between these two commands?

We have a table by name BOOKS. The content of the table BOOKS are shown in the Fig. 4.154.

Step 1: The contents of the table BOOKS are deleted by using DELETE command as shown in Fig. 4.155.

Step 2: The table BOOKS is again populated with the data and the command TRUNCATE is used to delete the contents of the table which is shown in Fig. 4.156.

The advantage offered by the TRUNCATE command is the speed. When Oracle executes this command, it does not evaluate the existing records within a table; it basically chops them off. In addition to speed, the TRUNCATE

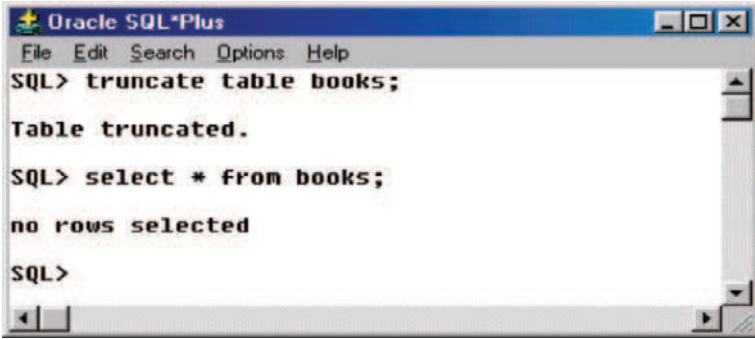


Fig. 4.156. Contents of the table BOOKS deleted using TRUNCATE command

command provides the added benefit of automatically freeing up the table space that the truncated records previously occupied.

When the table contents are deleted by using DELETE command, it forces Oracle to read every row before deleting it. This can be extremely time consuming.

4.6. What are subqueries? How will you classify them?

Subquery is query within a query. A SELECT statement can be nested inside another query to form a subquery. The query which contains the subquery is called outer query. It can be classified as (a) scalar subquery and (b) correlated subquery, and (c) uncorrelated subquery.

PL/SQL

Learning Objectives. This chapter focuses on the shortcomings of SQL and how it is overcome by PL/SQL. An introduction to PL/SQL is given in this chapter. After completing this chapter the reader should be familiar with the following concepts in PL/SQL.

- Structure of PL/SQL
- PL/SQL language elements
- Control structure in PL/SQL
- Steps to create PL/SQL program
- Concept of CURSOR
- Basic concepts related to Procedure, Functions
- Basic concept of Trigger

5.1 Introduction

PL/SQL stands for Procedural Language/Structured Query Language, which is provided by Oracle as a procedural extension to SQL. SQL is a declarative language. In SQL, the statements have no control to the program and can be executed in any order. PL/SQL, on the other hand, is a procedural language that makes up for all the missing elements in SQL. PL/SQL arose from the desire of programmers to have a language structure that was more familiar than SQL's purely declarative nature.

5.2 Shortcomings in SQL

We know, SQL is a powerful tool for accessing the database but it suffers from some deficiencies as follows:

- (a) SQL statements can be executed only one at a time. Every time to execute a SQL statement, a call is made to Oracle engine, thus it results in an increase in database overheads.

- (b) While processing an SQL statement, if an error occurs, Oracle generates its own error message, which is sometimes difficult to understand. If a user wants to display some other meaningful error message, SQL does not have provision for that.
- (c) SQL is not able to do the conditional query on RDBMS, this means one cannot use the conditions like if . . . then, in a SQL statement. Also looping facility (repeating a set of instructions) is not provided by SQL.

5.3 Structure of PL/SQL

PL/SQL is a 4GL (fourth generation) programming language. It offers all features of advanced programming language such as portability, security, data encapsulation, information hiding, etc. A PL/SQL program may consist of more than one SQL statements, while execution of a PL/SQL program makes only one call to Oracle engine, thus it helps in reducing the database overheads. With PL/SQL, one can use the SQL statements together with the control structures (like if . . . then) for data manipulation. Besides this, user can define his/her own error messages to display. Thus we can say that PL/SQL combines the data manipulation power of SQL with data processing power of procedural language.

PL/SQL is a block structured language. This means a PL/SQL program is made up of *blocks*, where block is a smallest piece of PL/SQL code having logically related statements and declarations. A block consists of three sections namely:

Declare, Begin, and Exception followed by an End statement. We will see the different sections of PL/SQL block.

Declare Section

Declare section declares the variables, constants, processes, functions, etc., to be used in the other parts of program. It is an *optional* section.

Begin Section

It is the executable section. It consists of a set of SQL and PL/SQL statements, which is executed when PL/SQL block runs. It is a compulsory section.

Exception Section

This section handles the errors, which occurs during execution of the PL/SQL block. This section allows the user to define his/her own error messages. This section executes only when an error occurs. It is an *optional* section.

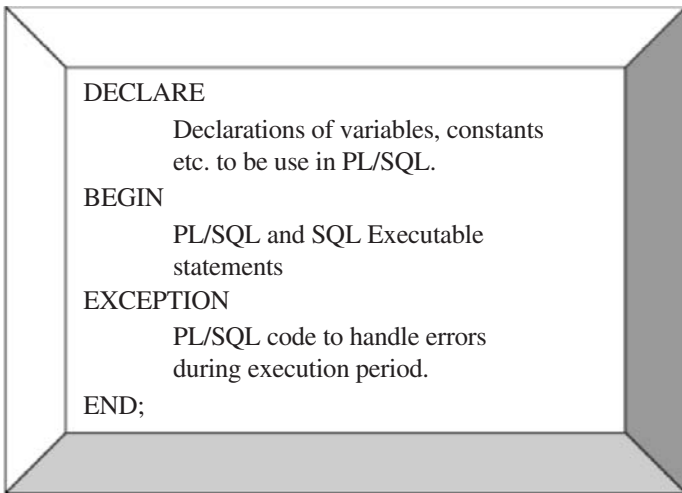


Fig. 5.1. A PL/SQL block

End Section

This section indicates the end of PL/SQL block.

Every PL/SQL program must consist of at least one block, which may consist of any number of nested sub-blocks. Figure 5.1 shows a typical PL/SQL block.

5.4 PL/SQL Language Elements

Let us start from the basic elements of PL/SQL language. Like other programming languages PL/SQL also have specific character sets, operators, indicators, punctuations, identifiers, comments, etc. In the following sections we will discuss about various language elements of PL/SQL.

Character Set

A PL/SQL program consists of text having specific set of characters. Character set may include the following characters:

- Alphabets, both in upper case [A-Z] and lower case [a-z]
- Numeric digits [0-9]
- Special characters () + - * / < > = ! ~ ^ ; : . ' @ % , " # \$ & - | { } ? []
- Blank spaces, tabs, and carriage returns.

PL/SQL is not case sensitive, so lowercase letters are equivalent to corresponding uppercase letters except within string and character literals.

Lexical Units

A line of PL/SQL program contains groups of characters known as lexical units, which can be classified as follows:

- Delimiters
- Identifiers
- Literals
- Comments

Delimiters

A *delimiter* is a simple or compound symbol that has a special meaning to PL/SQL. Simple symbol consists of one character, while compound symbol consists of more than one character. For example, to perform the addition and exponentiation operation in PL/SQL, simple symbol delimiter + and compound symbol delimiter ** is used, respectively. PL/SQL supports following simple symbol delimiters:

+ - * / = > < ; % ' , () @ : "

Compound symbol delimiters legal in PL/SQL are as follows:

<> ! = ~ = ^ = < = > = := ** .. || << >>

In the following sections we will discuss about these delimiters.

Identifiers

Identifiers are used in the PL/SQL programs to name the PL/SQL program items as constants, variables, cursors, cursor variables, subprograms, etc.

Identifiers can consist of alphabets, numerals, dollar signs, underscores, and number signs only. Any other characters like hyphens, slashes, blank spaces, etc. are illegal. An identifier must begin with an alphabetic letter optionally followed by one or more characters (permissible in identifier). An identifier cannot contain more than 30 characters.

Example

Some of the valid identifiers are as follows:

- A - Identifier may consist of a single character
- A1 - identifier may consist of numerals after first character
- Share\$price - dollar sign is permitted
- e_mail - under score is permitted
- phone# - number sign is permitted

The following identifiers are illegal:

- mine&yours - ampersand is illegal
- debit-amount - hyphen is illegal
- on/off - slash is illegal
- user id - space is illegal

However, PL/SQL allows space, slash, hyphen, etc. except double quotes if the identifier is enclosed within double quotes. Thus, the following identifiers are valid:

```

“A&B”
“TATA INFOTECH”
“True/false”
“Student(s)”
“*** BEGIN ***”

```

However, the maximum length of a quoted identifier cannot exceed 30 characters, excluding double quotes.

An identifier can consist of lower, upper, or mixed case letters. PL/SQL is not case sensitive except within string and character literals. So, if the only difference between identifiers is the case of corresponding letters, PL/SQL considers the identifiers to be the same. Take for example, a character string “HUMAN” as an identifier; it will be equivalent to each of the following identifiers:

```

Human
human
hUMAN
hUmAn.

```

An identifier cannot be a reserved word, i.e., the words that have special meaning for PL/SQL. For example, the word DECLARE, which is used for declaring the variables or constants; words BEGIN and END, which enclose the executable part of a block or subprogram are reserved words. An attempt to redefine a reserved word gives an error.

Literals

A *literal* is an explicitly defined character, string, numeric, or Boolean value, which is not represented by an identifier. In the following sections we will discuss about each of these literals in detail:

Numeric Literals

A numeric literal is an integer or a real value. An integer literal may be a positive, negative, or unsigned whole number without a decimal point. Some examples of integer numeric literals are as follows:

```
100  006  -10  0  +10
```

A real literal is a positive, negative, or unsigned whole or fractional number with a decimal point. Some examples of real integer literals are as follows:

```
0.0  -19.0  3.56219  +43.99  .6  7.  -4.56
```

PL/SQL treats a number with decimal point as a real numeric literal, even if the number does not have any numeral after decimal point. Besides integer and real literals, numeric literals can also contain exponential numbers (an optionally signed number suffix with an E (or e) followed by an optionally signed integer). Some examples of exponential numeric literals are as follows:

7E3 2.0E-3 3.14159e1 -2E33 -8.3e-2

where, E stands for “times ten to the power of.” For example the exponential literal 7E3 is equivalent to following numeric literal:

$$7E3 = 7 * 10 ** 3 = 7*10*10*10 = 7000$$

Another exponential literal -8.3e-2 would be equivalent to following numeric literal:

$$-8.3e-2 = -8.3 * 10 ** (-2) = -8.3 *0.01 = -0.083$$

An exponential numeric literal cannot be smaller than 1E-130 and cannot be greater than 10E125. Note that numeric literals cannot contain dollar signs or commas.

Character Literals

A character literal is an individual character enclosed by single quotes (apostrophes). Character literals include all the printable characters in the PL/SQL character set: letters, numerals, spaces, and special symbols. Some examples of character literals are as follows:

“A” “@” “5” “?” “,” “(”

PL/SQL is case sensitive within character literals. For example, PL/SQL considers the literals “A” and “a” to be different. Also, the character literals “0”...“9” are not equivalent to integer literals but can be used in arithmetic expressions because PL/SQL implicitly converts them to integers.

String Literals

A character string can be represented by an identifier or explicitly written as a string literal. A string literal is enclosed within single quotes and may consist of one or more characters. Some examples of string literals are as follows:

“Good Morning!”
 “TATA INFOTECH LTD”
 “04-MAY-00”
 “\$15,000,000”

All string literals are of character data type.

PL/SQL is case sensitive within string literals. For example, PL/SQL considers the following literals to be different:

“HUMAN”
 “Human”

Boolean Literals

Boolean literals are the predefined values TRUE, FALSE, and NULL. Keep in mind Boolean literals are values, not strings. For example a condition: if (x = 10) is TRUE only for the value of x equal to 10, for any other value of x it is FALSE and for no value of x it is NULL.

Comments

Comments are used in the PL/SQL program to improve the readability and understandability of a program. A comment can appear anywhere in the program code. The compiler ignores comments. Generally, comments are used to describe the purpose and use of each code segment. A PL/SQL comment may be a single-line or multiline.

Single-Line Comments

Single-line comments begin with a double hyphen (–) anywhere on a line and extend to the end of the line.

Example

– start calculations

Multiline Comments

Multiline comments begin with a slash-asterisk (/*) and end with an asterisk-slash (*//), and can span multiple lines.

Example

```
/* Hello World! This is an example of multiline comments in PL/SQL */
```

Variables and Constants

Variables and constants can be used within PL/SQL block, in procedural statements and in SQL statements. These are used to store the values. As the program executes, the values of variables can change, but the values of constants cannot. However, it is must to declare the variables and constants, before using these in executable portion of PL/SQL. Let us see how to declare variables and constants in PL/SQL.

Declaration

1Variables and constants are declared in the Declaration section of PL/SQL block. These can be any of the SQL data type like CHAR, NUMBER, DATE, etc.

I. Variables Declaration

The syntax for declaring a variable is as follows:

```
identifier datatype;
```

Example

To declare the variable `name`, `age`, and `joining_date` as datatype `VARCHAR2(10)`, `NUMBER(2)`, `DATE`, respectively; declaration statement is as follows:

```
DECLARE
Name VARCHAR2(10);
Age NUMBER(2);
Joining_date DATE;
```

Initializing the Variable

By default variables are initialized to `NULL` at the time of declaration. If we want to initialize the variable by some other value, syntax would be as follows:

```
Identifier datatype := value;
Or,
Identifier datatype DEFAULT value;
```

Example

If a number of employees have same `joining_date`, say `01-JULY-99`. It is better to initialize the `joining_date` rather than entering the same value individually, any of the following declaration can be used:

```
Joining_date DATE := 01-JULY-99; (or)
Joining_date DATE DEFAULT 01-JULY-99;
```

Constraining a Variable

Variables can be `NOT NULL` constrained at the time of declaring these, for example to constrain the `joining_date NOT NULL`, the declaration statement would be as follows:

```
Joining_date DATE NOT NULL: = 01-JULY-99;
```

(`NOT NULL` constraint must be followed by an initialization clause)
thus following declaration will give an error:

```
Joining_date DATE NOT NULL; – illegal
```

Declaring Constants

Declaration of constant is similar to declaration of variable, except the keyword `CONSTANT` precedes the datatype and it must be initialized by some value. The syntax for declaring a constant is as follows:

```
identifier CONSTANT datatype := value;
```

Example

To define the `age_limit` as a constant, having value 30; the declaration statement would be as follows: `Age_limit CONSTANT NUMBER := 30;`

Restrictions

PL/SQL imposes some restrictions on declaration as follows:

- (a) A list of variables that have the same datatype cannot be declared in the same row

Example

```
A, B, C NUMBER (4,2); – illegal
```

It should be declared in separate lines as follows:

```
A NUMBER (4,2);
```

```
B NUMBER (4,2);
```

```
C NUMBER (4,2);
```

- (b) A variable can reference to other variable if and only if that variable is declared before that variable. The following declaration is illegal:

```
A NUMBER(2) := B;
```

```
B NUMBER(2) := 4;
```

Correct declaration would be as follows:

```
B NUMBER(2) := 4;
```

```
A NUMBER(2) := B;
```

- (c) In a block same identifier cannot be declared by different datatype. The following declaration is illegal:

```
DECLARE
```

```
X NUMBER(4,2);
```

```
X CHAR(4); – illegal
```

5.5 Data Types

Every constant and variable has a datatype. A datatype specifies the space to be reserved in the memory, type of operations that can be performed, and valid range of values. PL/SQL supports all the built-in SQL datatypes. Apart from those datatypes, PL/SQL provides some other datatypes. Some commonly used PL/SQL datatypes are as follows:

BOOLEAN

One of the mostly used datatype is BOOLEAN. A BOOLEAN datatype is assigned to those variables, which are required for logical operations. A BOOLEAN datatype variable can store only logical values, i.e., TRUE, FALSE, or NULL. A BOOLEAN variable value cannot be inserted in a table; also, a table data cannot be selected or fetched into a BOOLEAN variable.

%Type

The %TYPE attribute provides the datatype of a variable or database column. In the following example, %TYPE provides the datatype of a variable:

```
balance NUMBER(8,2);
minimum_balance balance%TYPE;
```

In the above example PL/SQL will treat the minimum_balance of the same datatype as that of balance, i.e., NUMBER(8,2). The next example shows that a %TYPE declaration can include an initialization clause:

```
balance NUMBER(7,2);
minimum_balance balance%TYPE := 500.00;
```

The %TYPE attribute is particularly useful when declaring variables that refer to database columns. Column in a table can be referenced by %TYPE attribute.

Example

To declare a column my_empno of the same datatype as that of empno column of emp table in scott/tiger user, the declaration statement would be as follows:

```
my_empno scott.emp.empno%TYPE;
```

Using %TYPE to declare my_empno has two advantages. First, the knowledge of exact datatype of empno is not required. Second, if the database definition of empno changes, the datatype of my_empno changes accordingly at run time. But %TYPE variables do not inherit the NOT NULL column constraint, even though the database column empno is defined as NOT NULL, one can assign a null to the variable my_empno.

%Rowtype

The %ROWTYPE attribute provides a record type that represents a row in a table (or view). The record can store an entire row of data selected from the table.

Example

emp_rec is declared as a record datatype of emp table. emp_rec can store a row selected from the emp table.

```
emp_rec emp%ROWTYPE;
```

Expressions

Expressions are constructed using operands and operators. PL/SQL supports all the SQL operators; in addition to those operators it has one more operator, named exponentiation (symbol is **). An operand is a variable, constant, literal, or function call that contributes a value to an expression. An example of simple expression follows:

$$A = B * *3$$

where A, B, and 3 are operand; = and ** are operators. B**3 is equivalent to value of thrice multiplying the B, i.e., B*B*B.

Operators may be unary or binary. Unary operators such as the negation operator (−) operate on one operand; binary operators such as the division operator (/) operate on two operands. PL/SQL evaluates (finds the current value of) an expression by combining the values of operands in ways specified by the operators. This always yields a single value and datatype. PL/SQL determines the datatype by examining the expression and the context in which it appears.

5.6 Operators Precedence

The operations within an expression are done in a particular order depending on their precedence (priority). Table 5.1 lists the operator's level of precedence from top to bottom. Operators listed in the same row have equal precedence.

Operators with higher precedence are applied first, but if parentheses are used, expression within innermost parenthesis is evaluated first. For example the expression $8 + 4/2 * *2$ results in a value 9, because exponentiation has the highest priority followed by division and addition. Now in the same expression if we put parentheses, the expression $8 + ((4/2) **2)$ results in a value 12 not 9, because now first it will solve the expression within innermost parentheses.

Table 5.1. Order of operations

operator	operation
** , NOT	exponentiation, logical negation
+, -	identity, negation
*, /	multiplication, division
+, -,	addition, subtraction, concatenation
=, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN	comparison
AND	conjunction
OR	disjunction

5.7 Control Structure

Control structure is an essential part of any programming language. It controls the flow of process. Control structure is broadly divided into three categories:

- Conditional control,
- Iterative control, and
- Sequential control

In the following sections we will discuss about each of these control structures in detail.

Conditional Control

A conditional control structure tests a condition to find out whether it is true or false and accordingly executes the different blocks of SQL statements. Conditional control is generally performed by IF statement. There are three forms of IF statement. IF-THEN, IF-THEN-ELSE, IF-THEN-ELSEIF.

IF-THEN

It is the simplest form of IF condition. The syntax for this statement is as follows:

```
IF condition THEN
Sequence of statements
END IF;
```

Example

To compare the values of two variables A and B and to assign the value of A to HIGH if A is greater than B. The IF construct for this is as follows:

```
IF A > B THEN
HIGH := A;
ENDIF;
```

The sequence of statements is executed only if the condition is true. If the condition is FALSE or NULL, the sequence of statements is skipped and processing continues from statements following END IF statements.

IF-THEN-ELSE

As it is clear with the IF-THEN construct, if condition is FALSE the control exits to next statement out of IF-THEN clause. To execute some other set of statements in case condition evaluates to FALSE, the second form of IF statement is used, it adds the keyword ELSE followed by an alternative sequence of statements, as follows:

```

IF condition THEN
sequence_of_statements1
ELSE
sequence_of_statements2
END IF;

```

Example

To become clear about it, take the previous example, to compare the value of A and B and assign the value of greater number to HIGH. The IF construct for this is as follows:

```

IF A > B THEN
HIGH := A;
ELSE
HIGH := B;
ENDIF;

```

The sequence of statements in the ELSE clause is executed only if the condition is FALSE or NULL.

IF-THEN-ELSIF

In the previous constructs of IF, we can check only one condition, whether it is true or false. There is no provision if we want to check some other conditions if first condition evaluates to FALSE; for this purpose third form of IF statement is used. It selects an action from several mutually exclusive alternatives. The third form of IF statement uses the keyword ELSIF (not ELSEIF) to introduce additional conditions, as follows:

```

IF condition1 THEN
sequence_of_statements1
ELSIF condition2 THEN
sequence_of_statements2
ELSE
sequence_of_statements3
END IF;

```

5.8 Steps to Create a PL/SQL Program

1. First a notepad file can be created as typing in the Oracle SQL editor. Figure 5.2 shows the command to create a file,
2. Then a Notepad file will appear and at the same time background Oracle will be disabled. It is shown in Fig. 5.3
3. We can write our PL/SQL program in that file, save that file, and we can execute that program in the Oracle editor as in Fig. 5.4. In this program Cursor (*Current Set of Records*) concept is used which we will see in the following pages. Here content of EMP table is opened by the cursor and they are displayed by the DBMS.OUTPUT package. Command IF is used to check whether the cursor has been opened successfully by using %Found attribute.
4. Then we can execute that file as follows in Fig. 5.5

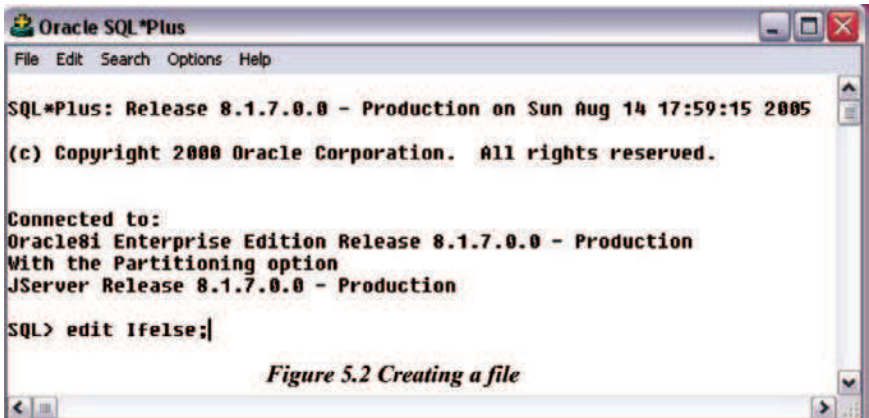


Fig. 5.2. Creating a file

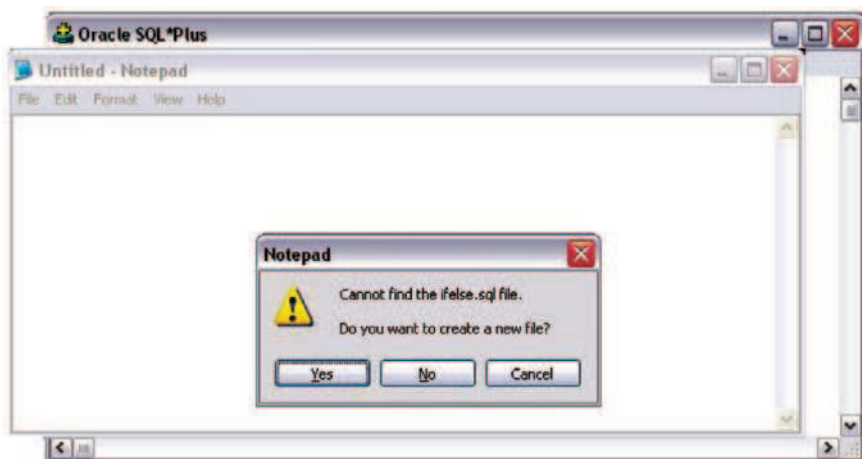


Fig. 5.3. Confirmation for the file created

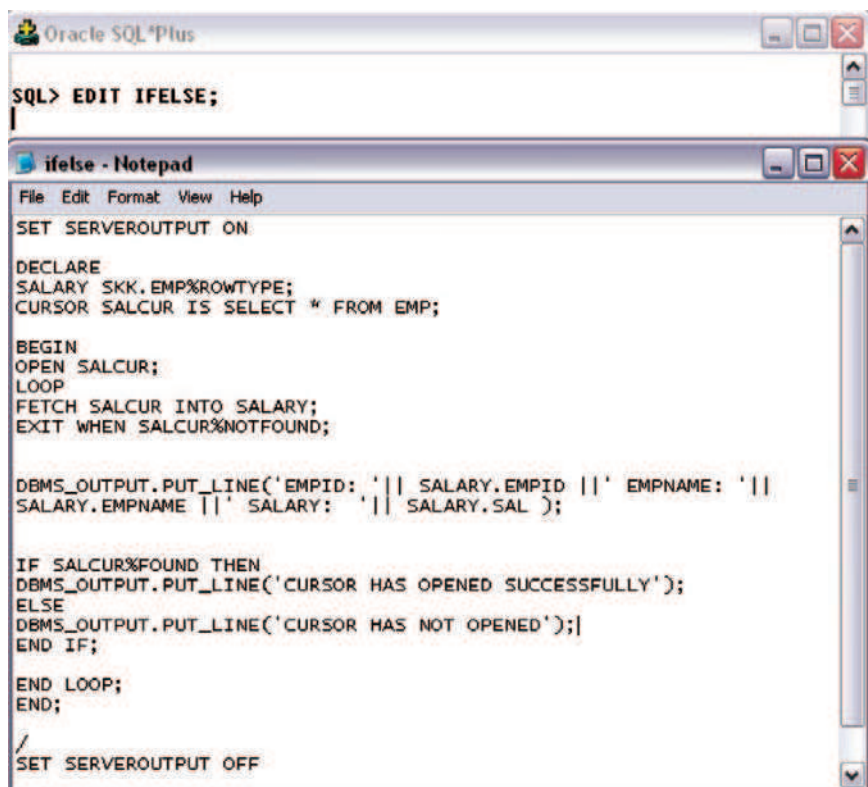


Fig. 5.4. Program writing to the notepad

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select * from emp;

EMPID  EMPNAME                SAL
-----
E101   SKK                     5000
E102   ANAND                    2500
E103   RAJA                     1900

SQL> START IFELSE;
EMPID: E101 EMPNAME: SKK SALARY: 5000
CURSOR HAS OPENED SUCCESSFULLY
EMPID: E102 EMPNAME: ANAND SALARY: 2500
CURSOR HAS OPENED SUCCESSFULLY
EMPID: E103 EMPNAME: RAJA SALARY: 1900
CURSOR HAS OPENED SUCCESSFULLY

PL/SQL procedure successfully completed.

SQL>

```

Fig. 5.5. Program execution

5.9 Iterative Control

In iterative control a group of statements are executed repeatedly till certain condition is true, and control exits from loop to next statement when the condition becomes false. There are mainly three types of loop statements:

LOOP, WHILE-LOOP, FOR-LOOP.

LOOP

LOOP is the simplest form of iterative control. It encloses a sequence of statements between the keywords LOOP and END LOOP. The general syntax for LOOP control is as follows:

```

LOOP
sequence_of_statements
END LOOP;

```

With each iteration of the loop, the sequence of statements gets executed, then control reaches at the top of the loop. But a control structure like this gets entrapped into infinite loop. To avoid this it is must to use the key word EXIT and EXIT-WHEN.

LOOP – EXIT

An EXIT statement within LOOP forces the loop to terminate unconditionally and passes the control to next statements. The general syntax for this is as follows:

```

LOOP
IF condition1 THEN
Sequence of statements1
EXIT;
ELSIF condition2 THEN
Sequence of statements2
EXIT
ELSE
Sequence of statements3
EXIT;
END IF;
END LOOP;

```

LOOP – EXIT WHEN

The EXIT-WHEN statement terminates a loop conditionally. When the EXIT statement is encountered, the condition in the WHEN clause is evaluated. If the condition is true, the loop terminates and control passes to the next statement after the loop. The syntax for this is as follows:

```

LOOP
EXIT WHEN condition
Sequence of statements
END LOOP

```

Example

Figures 5.4 and 5.5 are also the example of LOOP – EXIT WHEN. Condition used here is that the cursor does not return anything by using %NOTFOUND attribute.

WHILE-LOOP

The WHILE statement with LOOP checks the condition. If it is true then only the sequence of statements enclosed within the loop gets executed. Then control resumes at the top of the loop and checks the condition again; if it is true the sequence of statements enclosed within the loop gets executed. The process is repeated till the condition is true. The control passes to the next statement outside the loop for FALSE or NULL condition.

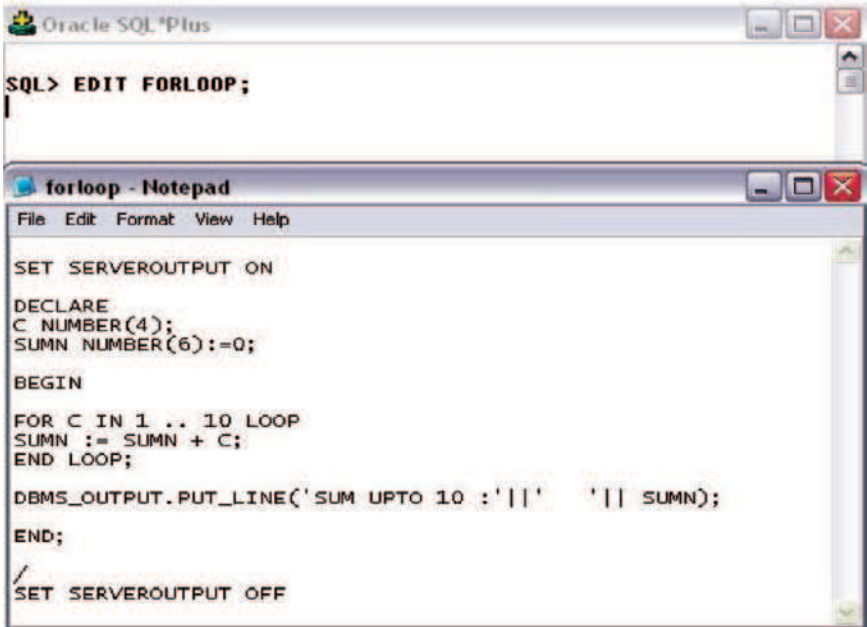


Fig. 5.6. Example for FOR Loop

WHILE condition LOOP
Sequence of statements
END LOOP;

FOR-LOOP

FOR loops iterate over a specified range of integers. The range is part of *iteration scheme*, which is enclosed by the keywords FOR and LOOP. A double dot (..) serves as the range operator. The syntax is as follows:

FOR counter IN lower_limit .. higher_limit LOOP
sequence_of_statements
END LOOP;

The range is evaluated when the FOR loop is first entered and is never re-evaluated. The sequence of statements is executed once for each integer in the range. After every iteration, the loop counter is incremented.

Example

To find the sum of natural numbers up to 10, the following program can be used as in Fig. 5.6.

Sequential Control

The sequential control unconditionally passes the control to specified unique label; it can be in the forward direction or in the backward direction. For sequential control GOTO statement is used. Overuse of GOTO statement may increase the complexity, thus as far as possible avoid the use of GOTO statement.

The syntax is as follows:

```
GOTO label;
.....
.....
<<label>>
Statement
```

5.10 Cursors

Number of rows returned by a query can be zero, one, or many, depending on the query search conditions. In PL/SQL, it is not possible for an SQL statement to return more than one row. In such cases we can use cursors. A cursor is a mechanism that can be used to process the multiple row result sets one row at a time.

In other words, cursors are constructs that enable the user to name a private memory area to hold a specific statement for access at a later time. Cursors are an inherent structure in PL/SQL. Cursors allow users to easily store and process sets of information in PL/SQL program.

Figure 5.7 shows the simple example for the cursor where two rows are selected from the query and they are pointed by the cursor namely All_Lifetime.

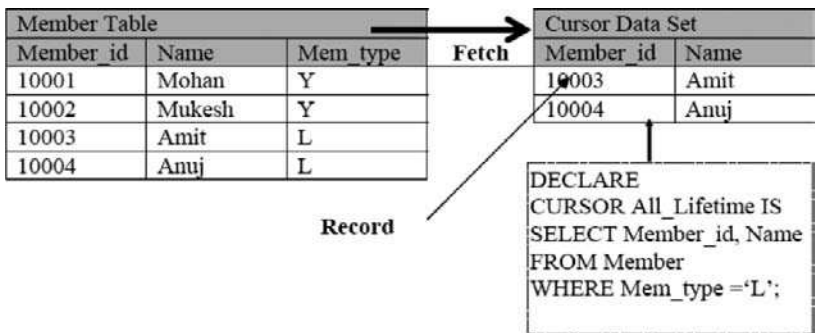


Fig. 5.7. Cursor example

There are two types of cursors in Oracle

1. Implicit cursors
2. Explicit cursors

5.10.1 Implicit Cursors

PL/SQL implicitly declares a cursor for every SQL DML statement, such as INSERT, DELETE, UPDATE, and SELECT statement that is not a part of an explicitly declared cursor, even if the statement processes a single row. PL/SQL allows referencing the most recent cursor or the cursor associated with the most recently executed SQL statement, as the “SQL” cursor. Cursor attributes are used to access information about the most recently executed SQL statement, using SQL cursor.

Implicit Cursor Attributes

In PL/SQL every cursor, implicit or explicit, has four attributes: %NOTFOUND, %FOUND, %ROWCOUNT, and %ISOPEN. These cursor attributes can be used in procedural statements (PL/SQL), but not in SQL statements. These attributes let user access information about the most recent execution of INSERT, UPDATE, SELECT INTO, and DELETE commands. These attributes are associated with the implicit “SQL” cursor and can be accessed by appending the attribute name to the implicit cursor name (SQL). Syntax to use cursor attribute is as follows:

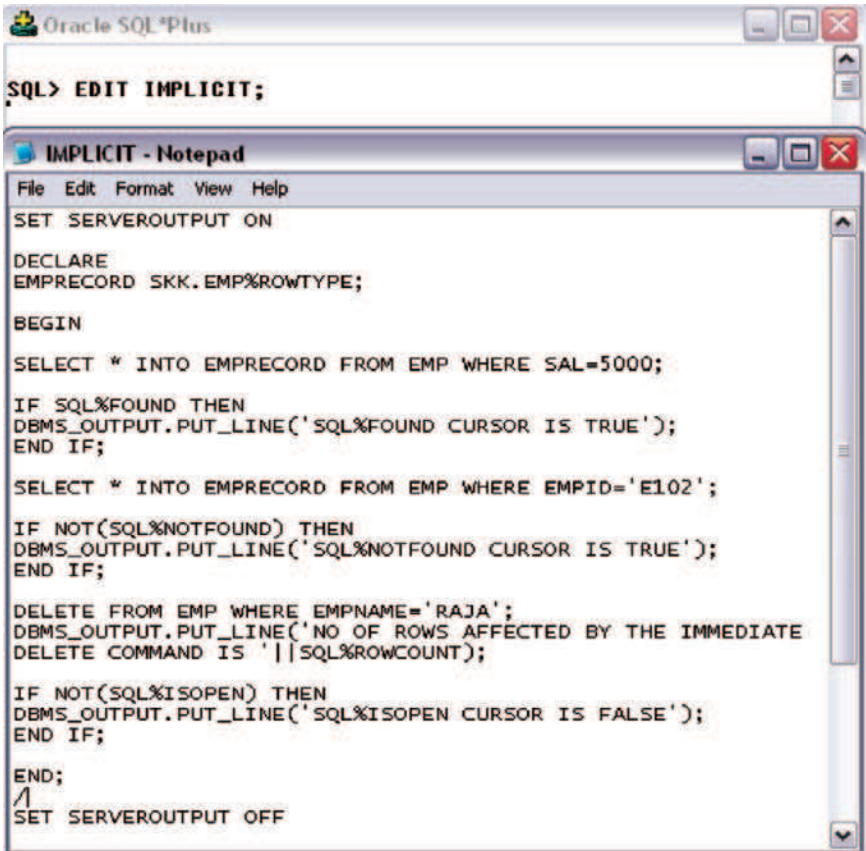
SQL %<attribute name>

%Notfound

This attribute is used to determine if any rows were processed by a SQL DML statement. This attribute evaluates to TRUE if an INSERT, UPDATE, or DELETE affected no rows or a SELECT INTO returned no rows. Otherwise, it returns FALSE. %NOTFOUND attribute can be useful in reporting or processing when no data is affected. If a SELECT statement does not return any data, the predefined exception NO_DATA_FOUND is automatically raised, and program control is sent to an exception handler, if it is present in the program. If a check is made on %NOTFOUND attribute after a SELECT statement, it will be completely skipped when the SELECT statement returns no data.

Example

Figures 5.8 and 5.9 show the example of all the implicit cursor attributes. The program will return the status of each cursor attribute depending on the previously executed DML statement.



```

Oracle SQL*Plus
SQL> EDIT IMPLICIT;

IMPLICIT - Notepad
File Edit Format View Help
SET SERVEROUTPUT ON

DECLARE
EMPRecord SKK.EMP%ROWTYPE;

BEGIN

SELECT * INTO EMPRecord FROM EMP WHERE SAL=5000;

IF SQL%FOUND THEN
DBMS_OUTPUT.PUT_LINE('SQL%FOUND CURSOR IS TRUE');
END IF;

SELECT * INTO EMPRecord FROM EMP WHERE EMPID='E102';

IF NOT(SQL%NOTFOUND) THEN
DBMS_OUTPUT.PUT_LINE('SQL%NOTFOUND CURSOR IS TRUE');
END IF;

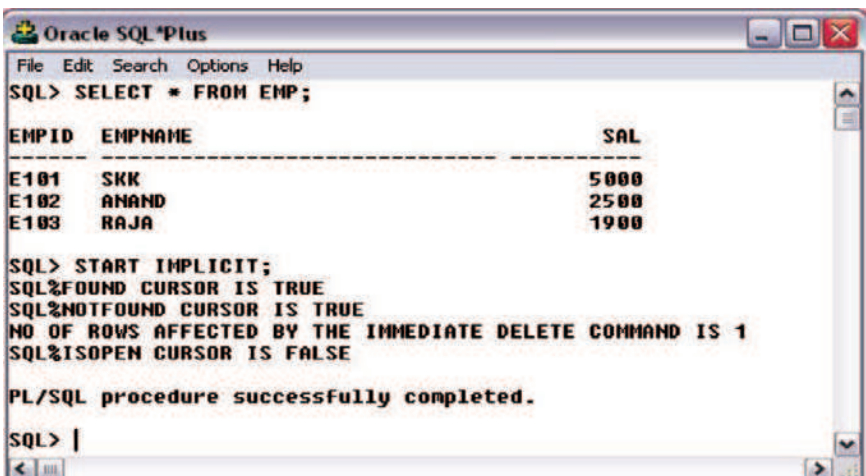
DELETE FROM EMP WHERE EMPNAME='RAJA';
DBMS_OUTPUT.PUT_LINE('NO OF ROWS AFFECTED BY THE IMMEDIATE
DELETE COMMAND IS '||SQL%ROWCOUNT);

IF NOT(SQL%ISOPEN) THEN
DBMS_OUTPUT.PUT_LINE('SQL%ISOPEN CURSOR IS FALSE');
END IF;

END;
/
SET SERVEROUTPUT OFF

```

Fig. 5.8. Implicit cursor example program



```

Oracle SQL*Plus
SQL> SELECT * FROM EMP;

EMPID  EMPNAME          SAL
-----
E101   SKK               5000
E102   ANAND             2500
E103   RAJA              1900

SQL> START IMPLICIT;
SQL%FOUND CURSOR IS TRUE
SQL%NOTFOUND CURSOR IS TRUE
NO OF ROWS AFFECTED BY THE IMMEDIATE DELETE COMMAND IS 1
SQL%ISOPEN CURSOR IS FALSE

PL/SQL procedure successfully completed.

SQL> |

```

Fig. 5.9. Implicit cursor example execution

%Found

This attribute is used to determine if any rows were processed by a SQL DML statement. In fact %FOUND works just the opposite of %NOTFOUND attribute. Until a SQL DML statement is executed, this attribute evaluates to NULL. It equates to TRUE if an INSERT, UPDATE, or DELETE affects one or more rows or select returns one row. If a select statement returns more than one row, the predefined exception TOO_MANY_ROWS is automatically raised and %FOUND attribute is set to FALSE.

%Rowcount

This attribute is used to determine the number of rows that are processed by an SQL statement. It returns the number of rows affected by an INSERT, UPDATE, or DELETE statement or returned by a SELECT INTO statement. %ROWCOUNT returns zero if the SQL statement affects or returns no rows. If a SELECT statement returns more than one row, the predefined exception TOO_MANY_ROWS is raised automatically. In such a case %ROWCOUNT attribute is set to 1 and not the actual number of rows that satisfy the query.

Example

Figures 5.8 and 5.9 show this example.

%Isopen

%ISOPEN is used to determine if a cursor is already open. It always equates to FALSE in an implicit cursor. Oracle automatically closes implicit cursor after executing its associated SQL statements.

Example

Figures 5.8 and 5.9 show this example.

5.10.2 Explicit Cursor

Explicit cursors are declared by the user and are used to process query results that return multiple rows. Multiple rows returned from a query form a set called an **active set**. PL/SQL defines the size of the active set as the number of rows that have met search criteria. Inherent in every cursor is a pointer that keeps track of the multiple rows being accessed, enabling program to process the rows one at a time. An explicit cursor points to the current row in the active set. This allows the program to process one row at a time.

Multirow query processing is somewhat like file processing. For example, a program opens a file to process records, and then closes the file. Likewise,

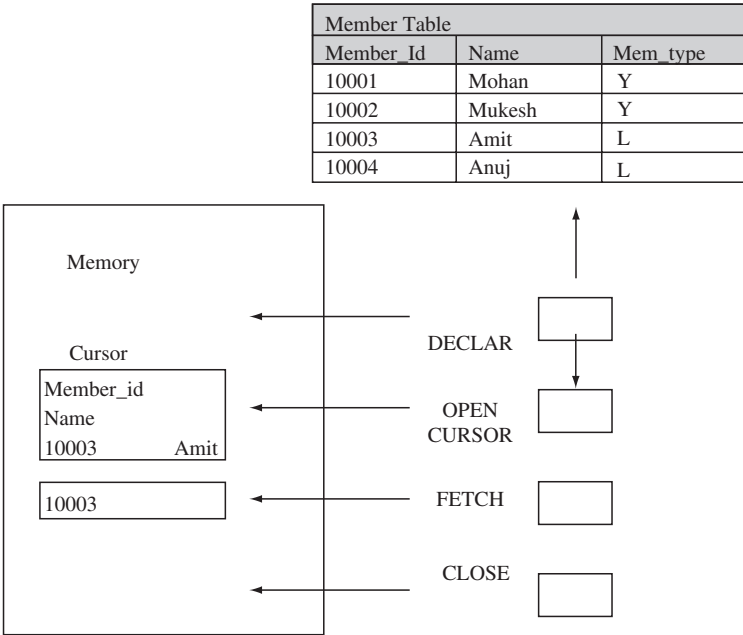


Fig. 5.10. Cursor and memory utilization

a PL/SQL program opens a cursor to process rows returned by a query, and then closes the cursor. Just as a file pointer marks the current position in an open file, a cursor marks the current position in an active set.

After a cursor is declared and opened, the user can FETCH, UPDATE, or DELETE the current row in the active set. The cursor can be CLOSED to disable it and free up any allocated system resources. Three commands are used to control the cursor – OPEN, FETCH, and CLOSE. First the cursor is initialized with an OPEN statement, which identifies the active set. Then, the FETCH statement is used to retrieve the first row. FETCH statement can be executed repeatedly until all rows have been retrieved. When the last row has been processed, the cursor can be released with the CLOSE statement. Figure 5.10 shows the memory utilization by a cursor when each of these statements is given.

5.11 Steps to Create a Cursor

Following are the steps to create a cursor:

5.11.1 Declare the Cursor

In PL/SQL a cursor, like a variable, is declared in the DECLARE section of a PL/SQL block or subprogram. A cursor must be declared before it can be

referenced in other statements. A cursor is defined in the declarative part by naming it and specifying a SELECT query to define the active set.

```
CURSOR <cursor_name> IS
SELECT...
```

The SELECT statement associated with a cursor declaration can reference previously declared variables.

Declaring Parameterized Cursors

PL/SQL allows declaration of cursors that can accept input parameters which can be used in the SELECT statement with WHERE clause to select specified rows. Syntax to declare a parameterized cursor:

```
CURSOR <cursor_name> [(parameter.....)] IS
SELECT.....
WHERE <column_name> = parameter;
```

Parameter is an input parameter defined with the syntax:

```
<variable_name> [IN] <datatype> [{:= | DEFAULT} value]
```

The formal parameters of a cursor must be IN parameters. As in the example above, cursor parameters can be initialized to default values. That way, different numbers of actual parameters can be passed to a cursor, accepting or overriding the default values.

Moreover, new formal parameters can be added without having to change every reference to the cursor. The scope of a cursor parameter is local only to the cursor. A cursor parameter can be referenced only within the SELECT statement associated with the cursor declaration. The values passed to the cursor parameters are used by the SELECT statement when the cursor is opened.

5.11.2 Open the Cursor

After declaration, the cursor is opened with an OPEN statement for processing rows in the cursor. The SELECT statement associated with the cursor is executed when the cursor is opened, and the active set associated with the cursor is created.

The active set is defined when the cursor is declared, and is created when cursor is opened.

The active set consists of all rows that meet the SELECT statement criteria. Syntax of OPEN statement is as follows.

```
OPEN <cursor_name>;
```

5.11.3 Passing Parameters to Cursor

Parameters to a parameterized cursor can be passed when the cursor is opened. For example, given the cursor declaration

```
CURSOR Mem_detail (MType VARCHAR2) IS SELECT...
```

Any of the following statements opens the cursor.

```
OPEN Mem_detail('L');
OPEN Mem_detail(Mem); where Mem is another variable.
```

Unless default values are to be accepted, each formal parameter in the cursor declaration must have a corresponding actual parameter in the OPEN statement. Formal parameters declared with a default value need not have a corresponding actual parameter. They can simply assume their default values when the OPEN statement is executed. The formal parameters of a cursor must be IN parameters. Therefore, they cannot return values to actual parameters. Each actual parameter must belong to a datatype compatible with the datatype of its corresponding formal parameter.

5.11.4 Fetch Data from the Cursor

After a cursor has been opened, the SELECT statement associated with the cursor is executed and the active set is created. To retrieve the rows in the active set one row at a time, the rows must be fetched individually from the cursor. After each FETCH statement, the cursor advances to the next row in the active set and retrieves it. Syntax of FETCH is:

```
FETCH <cursor_name> INTO <variable_name>, <variable_name>...
```

where variable_name is the name of a variable to which a column value is assigned. For each column value returned by the query associated with the cursor, there must be a corresponding variable in the INTO list. This variable datatype must be compatible with the corresponding database column.

5.11.5 Close the Cursor

After processing the rows in the cursor, it is released with the CLOSE statement. To change the active set in a cursor or the values of the variables referenced in the cursor SELECT statement, the cursor must be released with CLOSE statement. Once a cursor is CLOSED, it can be reOPENed. The CLOSE statement disables the cursor, and the active set becomes undefined. For example, to CLOSE Mem_detail close statement will be:

```
CLOSE <cursor_name>;
```

Example

Figures 5.4 and 5.5 show the example of declaring, opening, and fetching the cursor called SALCUR.

Explicit Cursor Attributes

It is used to access useful information about the status of an explicit cursor. Explicit cursors have the same set of cursor attributes %NOTFOUND, %FOUND, %ROWCOUNT, and %ISOPEN. These attributes can be accessed in PL/SQL statements only, not in SQL statements. Syntax to access an explicit cursor attributes:

<code><cursor_name>%<attribute_name></code>

%Notfound

When a cursor is OPENed, the rows that satisfy the associated query are identified and form the active set. Before the first fetch, %NOTFOUND evaluates to NULL. Rows are FETCHed from the active set one at a time. If the last fetch returned a row, %NOTFOUND evaluates to FALSE. If the last fetch failed to return a row because the active set was empty, %NOTFOUND evaluates to TRUE. FETCH is expected to fail eventually, so when that happens, no exception is raised.

Example

Figures 5.4 and 5.5 show the example for this attribute. In this example, it is used for checking whether all the rows have been fetched or not.

%Found

%FOUND is the logical opposite of %NOTFOUND. After an explicit cursor is open but before the first fetch, %FOUND evaluates to NULL. Thereafter, it evaluates to TRUE if the last fetch returned a row or to FALSE if no row was returned. If a cursor is not open, referencing it with %FOUND raises INVALID_CURSOR exception.

Example

Figures 5.4 and 5.5 show the example for this attribute. In this example, it is used for checking whether the cursor has been opened successfully or not.

%Rowcount

When you open a cursor, %ROWCOUNT is initialized to zero. Before the first fetch, %ROWCOUNT returns a zero. Thereafter, it returns the number of rows fetched so far. The number is incremented if the latest fetch returned a row.

Example

Figures 5.8 and 5.9 show the example of this attribute where cursor `updatcur` is used.

%Isopen

`%ISOPEN` evaluates to `TRUE` if the cursor is open; otherwise, `%ISOPEN` evaluates to `FALSE`.

Example

Figures 5.11 and 5.12 show the example of this attribute where cursor `updatcur` is used.

The image shows two overlapping windows. The top window is Oracle SQL*Plus, displaying the command `SQL> EDIT FORUPDATE;`. The bottom window is Notepad, titled "FORUPDATE - Notepad", containing the following PL/SQL code:

```

SET SERVEROUTPUT ON

DECLARE
CURSOR UPDATCUR IS SELECT * FROM EMP WHERE EMPID='E101' FOR
UPDATE OF EMPNAME;
UPDATE_REC EMP%ROWTYPE;

BEGIN
OPEN UPDATCUR;
LOOP
FETCH UPDATCUR INTO UPDATE_REC;
EXIT WHEN UPDATCUR%NOTFOUND;
UPDATE EMP SET EMPNAME='KARTHIKEYAN' WHERE CURRENT OF
UPDATCUR;
END LOOP;

DBMS_OUTPUT.PUT_LINE('NO OF ROWS AFFECTED BY THE IMMEDIATE
UPDATE COMMAND IS '||UPDATCUR%ROWCOUNT);

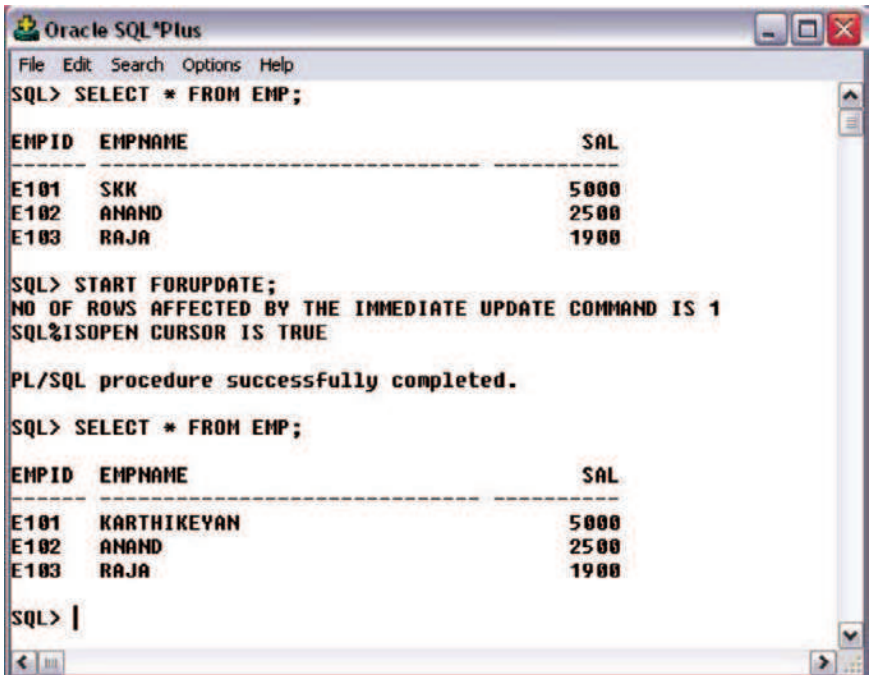
IF UPDATCUR%ISOPEN THEN
DBMS_OUTPUT.PUT_LINE('SQL%ISOPEN CURSOR IS TRUE');
END IF;

CLOSE UPDATCUR;

END;
/

```

Fig. 5.11. Example of FOR UPDATE clause



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT * FROM EMP;

EMPID  EMPNAME                SAL
-----
E101   SKK                     5000
E102   ANAND                   2500
E103   RAJA                    1900

SQL> START FORUPDATE;
NO OF ROWS AFFECTED BY THE IMMEDIATE UPDATE COMMAND IS 1
SQL%ISOPEN CURSOR IS TRUE

PL/SQL procedure successfully completed.

SQL> SELECT * FROM EMP;

EMPID  EMPNAME                SAL
-----
E101   KARTHIKEYAN            5000
E102   ANAND                   2500
E103   RAJA                    1900

SQL> |

```

Fig. 5.12. FOR UPDATE clause execution

Using FOR UPDATE and CURRENT

The FOR UPDATE clause is used to specify that the rows in the active set of a cursor are to be locked for modification. Locking allows the rows in the active set to be modified exclusively by your program. This protects simultaneous modifications until update by one transaction is complete.

```
CURSOR <cursor_name> IS SELECT <column_name> [.....] FROM.....
FOR UPDATE [OF <column_name> .....];
```

FOR UPDATE specifies that the rows of the active set are to be exclusively locked when the cursor is opened and specifies the column names that can be updated. The FOR UPDATE clause must be used in the cursor declaration statement whenever UPDATE or DELETE are to be used after the rows are FETCHed from a cursor.

Syntax of CURRENT clause with UPDATE statement is:

```
UPDATE <table_name> SET <column_name> = expression [.....]
WHERE CURRENT OF <cursor_name>;
```

Syntax of CURRENT OF Clause with DELETE Statement is:

```
DELETE table_name WHERE CURRENT OF cursor_name;
```

Example

Figures 5.11 and 5.12 show this example where a row of id E101 is locked for update and its name of the Employee is changed to Karthikeyan.

Cursor FOR Loop

PL/SQL provides FOR loop to manage cursors effectively in situations where the rows in the active set of cursor are to be repeatedly processed in a looping manner. A cursor FOR loop simplifies all aspects of processing a cursor. Cursor FOR loop can be used instead of the OPEN, FETCH, and CLOSE statements.

A cursor FOR loop implicitly declares its loop index as a %ROWTYPE record, opens a cursor, repeatedly fetches rows of values from the active set into fields in the record, and closes the cursor when all rows have been processed. Syntax to declare and process a cursor in a cursor FOR loop is:

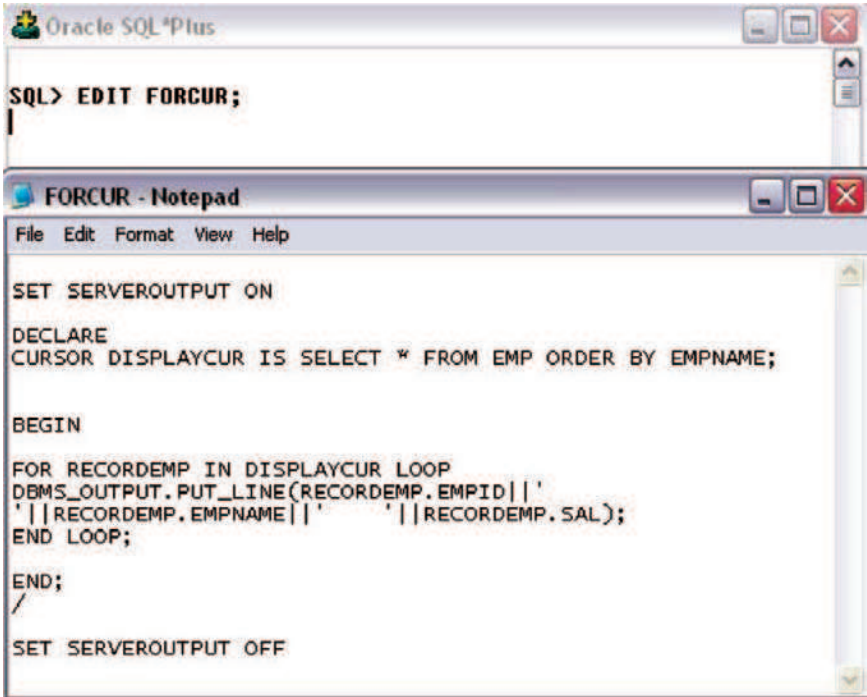
```
FOR <record_name> IN <cursor_name> LOOP
.....
END LOOP;
```

where record_name is the cursor FOR loop index implicitly declared as a record of type %ROWTYPE. Cursor is assumed to be declared in the DECLARE section. In the FOR loop declaration, the FOR loop index is uniquely named and implicitly declared as a record of type %ROWTYPE. This RECORD variable consists of columns referenced in the cursor SELECT statement.

In the FOR loop, the cursor is implicitly opened for processing. No explicit OPEN statement is required. Inside the FOR loop, the column values for each row in the active set can be referenced by the FOR loop index with dot notation in any PL/SQL or SQL statement. Before any iteration of the FOR loop, PL/SQL fetches into the implicitly declared record, which is equivalent to a record declared explicitly. At the end of the active set, the FOR loop implicitly closes the cursor and exits the FOR loop. No explicit CLOSE statement is required. A COMMIT statement is still required to complete the operation. We can pass parameters to a cursor used in a cursor FOR loop. The record is defined only inside the loop. We cannot refer to its fields outside the loop. The sequence of statements inside the loop is executed once for each row that satisfies the query associated with the cursor. On leaving the loop, the cursor is closed automatically. This is true even if an EXIT or GOTO statement is used to leave the loop prematurely or if an exception is raised inside the loop.

Example

Figures 5.13 and 5.14 show the example of cursor execution using FOR loop.



The image shows two overlapping windows. The top window is Oracle SQL*Plus, and the bottom window is Notepad. The Notepad window contains the following PL/SQL code:

```

SET SERVEROUTPUT ON

DECLARE
CURSOR DISPLAYCUR IS SELECT * FROM EMP ORDER BY EMPNAME;

BEGIN

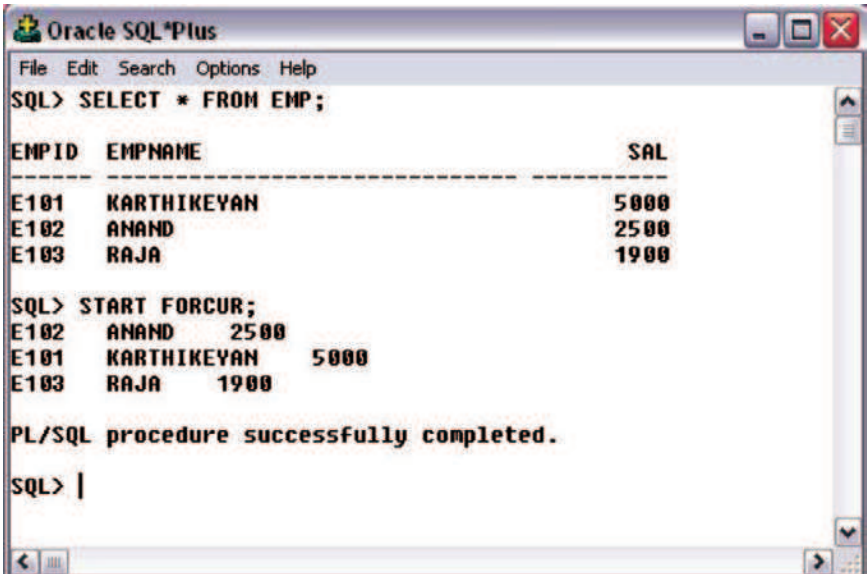
FOR RECORDEMP IN DISPLAYCUR LOOP
DBMS_OUTPUT.PUT_LINE(RECORDEMP.EMPID||'
'||RECORDEMP.EMPNAME||'      '||RECORDEMP.SAL);
END LOOP;

END;
/

SET SERVEROUTPUT OFF

```

Fig. 5.13. Cursor using FOR loop



The image shows the Oracle SQL*Plus window after executing the code from Fig. 5.13. The output is as follows:

```

SQL> SELECT * FROM EMP;

EMPID  EMPNAME                SAL
-----
E101   KARTHIKEYAN            5000
E102   ANAND                   2500
E103   RAJA                    1900

SQL> START FORCUR;
E102 ANAND 2500
E101 KARTHIKEYAN 5000
E103 RAJA 1900

PL/SQL procedure successfully completed.

SQL> |

```

Fig. 5.14. Cursor using FOR loop execution

5.12 Procedure

A procedure is a subprogram that performs some specific task, and stored in the data dictionary. A procedure must have a name, so that it can be invoked or called by any PL/SQL program that appears within an application. Procedures can take parameters from the calling program and perform the specific task. Before the procedure or function is stored, the Oracle engine parses and compiles the procedure or function. When a procedure is created, the Oracle automatically performs the following steps:

1. Compiles the procedure
2. Stores the procedure in the data dictionary

If an error occurs during creation of procedure, Oracle displays a message that procedure is created with compilation errors, but it does not display the errors. To see the errors following statement is used:

```
SELECT * FROM user_errors;
```

When the function is invoked, the Oracle loads the compiled procedure in the memory area called system global area (SGA). Once loaded in the SGA other users can also access the same procedure provided they have granted permission for this.

Benefits of Procedures and Functions

Stored procedures and functions have many benefits in addition to *modularizing* application development.

1. It modifies one routine to affect multiple applications.
2. It modifies one routine to eliminate duplicate testing.
3. It ensures that related actions are performed together, or not at all, by doing the activity through a single path.
4. It avoids PL/SQL parsing at runtime by parsing at compile time.
5. It reduces the number of calls to the database and database network traffic by bundling the commands.

Defining and Creating Procedures

A procedure consists of two parts: specification and body. The specification starts with keyword PROCEDURE and ends with parameter list or procedure name. The procedures may accept parameters or may not. Procedures that do not accept parameters are written parentheses.

The procedure body starts with the keyword IS and ends with keyword END. The procedure body is further subdivided into three parts:

1. Declarative part which consists of local declarations placed between keywords IS and BEGIN.

2. Executable part, which consists of actual logic of the procedure, included between keywords BEGIN and EXCEPTION. At least one executable statement is a must in the executable portion of a procedure. Even a single NULL statement will do the job.
3. Error/Exception handling part, an optional part placed between EXCEPTON and END.

The syntax for creating a procedure is follows:

```
CREATE OR REPLACE PROCEDURE [schema.] package_name
  [(argument {IN, OUT, IN OUT} data type,.....)] {IS, AS}
  [local variable declarations]
BEGIN
  executable statements
EXCEPTION
  exception handlers
END [procedure name];
```

Create: Creates a new procedure, if a procedure of same name already exists, it gives an error.

Replace: Creates a procedure, if a procedure of same name already exists, it replace the older one by the new procedure definition.

Schema: If the schema is not specified then procedure is created in user's current schema.

Figure 5.15 shows the procedure to raise the salary of the employee. The name of the procedure is raise_sal.

```
Oracle SQL*Plus
File Edit Search Options Help
SQL> DESC EMP;
Name                               Null?    Type
-----
EMPID                               NOT NULL VARCHAR2(6)
EMPNAME                             VARCHAR2(30)
SAL                                  NUMBER(6)

SQL> CREATE OR REPLACE PROCEDURE raise_sal
 2 (EID IN EMP.EMPID%TYPE)
 3 IS
 4 BEGIN
 5 UPDATE EMP
 6 SET SAL = SAL * 1.25 WHERE EMPID=EID;
 7 END raise_sal;
 8 /

Procedure created.

SQL> |
```

Fig. 5.15. Procedure creation

Argument: It is the name of the argument to the procedure.

IN: Specifies that a value for the argument must be specified when calling the procedure.

OUT: Specifies that the procedure pass a value for this argument back to its calling environment after execution.

IN OUT: Specifies that a value for the argument must be specified when calling the procedure and that the procedure passes a value for this argument back to its calling environment after execution. If no value is specified then it takes the default value IN.

Datatype: It is the unconstrained datatype of an argument. It supports any data type supported by PL/SQL. No constraints like size constraints or NOT NULL constraints can be imposed on the data type. However, you can put on the size constraint indirectly.

Example

To raise the salary of an employee, we can write a procedure as follows.

Declaring Subprograms

Subprograms can be declared inside any valid PL/SQL block. The only thing to be kept in mind is the declaration of programs must be the last part of declarative section of any PL/SQL block; all other declarations should precede the subprogram declarations.

Like any other programming language, PL/SQL also requires that any identifier that is used in PL/SQL program should be declared first before its use. To avoid problems arising due to such malpractices, forward declarations are used.

System and Object Privileges for Procedures

The creator of a procedure must have CREATE PROCEDURE system privilege in his own schema, if the procedure being created refers to his own schema. To create a procedure in other's schema, the creator must have CREATE ANY PROCEDURE system privilege.

To create a procedure without errors (compiling it without errors), the creator of procedure must have required privileges to all the objects he refer to from his procedure. It must be noted that the owner will not get the required privileges through roles, he must be granted those privileges explicitly.

As soon as the privileges granted to the owner of procedure change, the procedure must be reauthenticated in order to bring into picture the new privileges of the owner. If a necessary privilege to an object referenced by a procedure is revoked/withdrawn from the owner of the procedure, the procedure cannot be run.

To EXECUTE any procedure a user must have EXECUTE ANY PROCEDURE privilege. With this privilege he can execute a procedure which belong to some other user.

Executing/Invoking a Procedure

The syntax used to execute a procedure depends on the environment from which the procedure is being called. From within SQLPLUS, a procedure can be executed by using the EXECUTE command, followed by the procedure name. Any arguments to be passed to the procedure must be enclosed in parentheses following the procedure name.

Example

Figure 5.16 shows the execution of procedure raise_sal.

Removing a Procedure

To remove a procedure completely from the database, following command is used:

DROP PROCEDURE <PROCEDURE NAME>;

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT * FROM EMP;

EMPID  EMPNAME          SAL
-----
E101   KARTHIKEYAN     5000
E102   ANAND           2500
E103   RAJA            1900

SQL> EXECUTE raise_sal('E101');

PL/SQL procedure successfully completed.

SQL> SELECT * FROM EMP;

EMPID  EMPNAME          SAL
-----
E101   KARTHIKEYAN     6250
E102   ANAND           2500
E103   RAJA            1900

SQL>

```

Fig. 5.16. Procedure execution

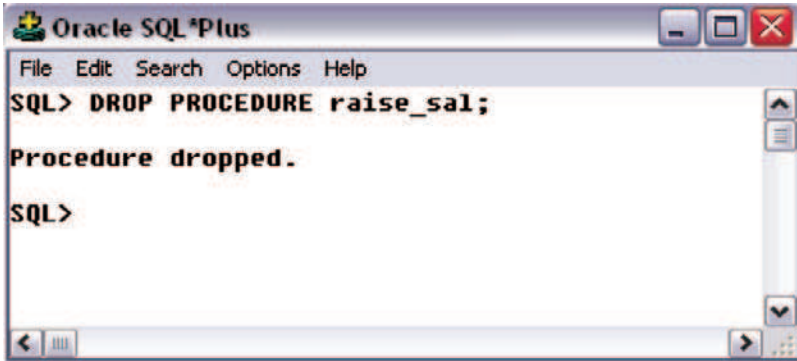


Fig. 5.17. Dropping of a procedure

To remove a procedure, one must own the procedure he is dropping or he must have DROP ANY PROCEDURE privilege.

Example

To drop a procedure raise_sal. Figure 5.17 indicate the dropping of the procedure raise_sal.

5.13 Function

A Function is similar to procedure except that it must return one and only one value to the calling program. Besides this, a function can be used as part of SQL expression, whereas the procedure cannot.

Difference Between Function and Procedure

Before we look at functions in deep, let us first discuss the major differences between a function and a procedure.

1. A procedure never returns a value to the calling portion of code, whereas a function returns exactly *one value* to the calling program.
2. As functions are capable of returning a value, they can be used as elements of SQL expressions, whereas the procedures cannot. However, user-defined functions cannot be used in CHECK or DEFAULT constraints and cannot manipulate database values, to obey function purity rules.
3. It is mandatory for a function to have at least one RETURN statement, whereas for procedures there is no restriction. A procedure may have a RETURN statement or may not. In case of procedures with RETURN statement, simply the control of execution is transferred back to the portion of code that called the procedure.

The exact syntax for defining a function is given below:

```
CREATE OR REPLACE FUNCTION [schema.] functionname
[(argument IN datatype, ...)] RETURN datatype {IS,AS}
[local variable declarations];
BEGIN
executable statements;
EXCEPTION
exception handlers;
END [functionname];
```

where RETURN datatype is the datatype of the function's return value. It can be any PL/SQL datatype.

Thus a function has two parts: function specification and function body. The function specification begins with keyword FUNCTION and ends with RETURN clause which indicates the datatype of the value returned by the function. Function body is enclosed between the keywords IS and END. Sometimes END is followed by function name, but this is optional. Like procedure, a function body also is composed of three parts: declarative part, executable part, and an optional error/exception handling part.

At least one return statement is a must in a function; otherwise PL/SQL raises PROGRAM_ERROR exception at the run time. A function can have multiple return statements, but can return only one value. In procedures, return statement cannot contain any expression, it simply returns control back to the calling code. However in functions, return statement must contain an expression, which is evaluated and sent to the calling code.

Example

To get a salary of an employee, Fig. 5.18 shows a function.

Figure 5.19 shows that how the calling of a function is different from procedure calling.

Purity of a Function

For a function to be eligible for being called in SQL statements, it must satisfy the following requirements, which are known as Purity Rules.

1. When called from a SELECT statement or a parallelized INSERT, UPDATE, or DELETE statement, the function cannot modify any database tables.
2. When called from an INSERT, UPDATE, or DELETE statement, the function cannot query or modify any database tables modified by that statement.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> DESC EMP;
Name                               Null?    Type
-----
EMPID                               NOT NULL VARCHAR2(6)
EMPNAME                             VARCHAR2(30)
SAL                                  NUMBER(6)

SQL> CREATE OR REPLACE FUNCTION get_sal
2 (EID IN EMP.EMPID%TYPE)
3 RETURN NUMBER
4 IS
5 EMPSAL EMP.SAL%TYPE :=0;
6 BEGIN
7 SELECT SAL INTO EMPSAL
8 FROM EMP WHERE EMPID=EID;
9 RETURN(EMPSAL);
10 END;
11 /

Function created.

SQL>

```

Fig. 5.18. Function creation

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT * FROM EMP;
EMPID  EMPNAME          SAL
-----
E101   KARTHIKEYAN     6250
E102   ANAND           2500
E103   RAJA            1900

SQL> SELECT GET_SAL('E102') FROM DUAL;
GET_SAL('E102')
-----
                2500

SQL> |

```

Fig. 5.19. Function execution

- When called from a SELECT, INSERT, UPDATE, or DELETE statement, the function cannot execute SQL transaction control statements (such as COMMIT), session control statements (such as SET ROLE), or system control statements (such as ALTER SYSTEM). Also, it cannot

execute DDL statements (such as CREATE) because they are followed by an automatic commit.

If any of the above rules is violated, the function is said to be not following the Purity Rules and the program using such functions receives run time error.

Removing a Function

To remove a function, use following command:

```
DROP FUNCTION <FUNCTION NAME>;
```

Example

Figure 5.20 illustrates the dropping of a function.

To remove a function, one must own the function to be dropped or he must have DROP ANY FUNCTION privilege.

Parameters

Parameters are the link between a subprogram code and the code calling the subprogram. Lot depends on how the parameters are passed to a subprogram. Hence it is absolutely necessary to know more about parameters, their modes, their default values, and how subprograms can be called without passing all the parameters.

Parameter Modes

Parameter modes define the behavior of formal parameters of subprograms. There are three types of parameter modes: IN, OUT, IN/OUT.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> DROP FUNCTION GET_SAL;

Function dropped.

SQL> SELECT GET_SAL('E102') FROM DUAL;
SELECT GET_SAL('E102') FROM DUAL
*
ERROR at line 1:
ORA-00904: invalid column name

SQL>

```

Fig. 5.20. Dropping the function

IN Mode

IN mode is used to pass values to the called subprogram. In short this is an input to the called subprogram. Inside the called subprogram, an IN parameter acts like a constant and hence it cannot be assigned a new value.

The IN parameter in actual parameter list can be a constant, literal, initialized variable, or an expression. IN parameters can be initialized to default values, which is not the case with IN/OUT or OUT parameters.

It is important to note that IN mode is the default mode of the formal parameters. If we do not specify the mode of a formal parameter it will be treated as an IN mode parameter.

OUT Mode

An OUT parameter returns a value back to the caller subprogram. Inside the subprogram, the parameter specified with OUT mode acts just like any locally declared variable. Its value can be changed or referenced in expressions, just like any other local variables.

The points to be noted for an OUT parameter are:

1. The parameter (in actual argument list) corresponding to OUT parameter must be a variable; it cannot be a constant or literal.
2. Formal OUT parameters are by default initialized to NULL, so we cannot constraint the formal OUT parameters by NOT NULL constraint.
3. The parameter (in actual argument list) corresponding to OUT parameter can have a value before a call to subprogram, but the value is lost as soon as a call is made to the subprogram.

IN/OUT

An IN/OUT parameter performs the duty of both IN parameter as well as OUT parameter. It first passes input value (through actual argument) to the called subprogram and then inside subprogram it receives a new value which will be assigned finally to the actual parameter. In short, inside the called subprogram, the IN/OUT parameter behaves just like an initialized local variable.

Like OUT parameter, the parameter in the actual argument list that corresponds to IN/OUT parameter, must be a variable, it cannot be a constant or an expression. If the subprogram exits successfully, PL/SQL assigns value to actual parameters, however, if the subprogram exits with unhandled exception, PL/SQL does not assign values to actual parameters.

5.14 Packages

A package can be defined as a collection of related program objects such as procedures, functions, and associated cursors and variables together as a unit in the database. In simpler term, a package is a group of related procedures and functions stored together and sharing common variables, as well as local procedures and functions. A package contains two separate parts: the package specification and the package body. The package specification and package body are compiled separately and stored in the data dictionary as two separate objects. The package body is optional and need not to be created if the package specification does not contain any procedures or functions. Applications or users can call packaged procedures and functions explicitly similar to standalone procedures and functions.

Advantages of Packages

Packages offer a lot of advantages. They are as follows.

1. Stored packages allow us to sum up (group logically) related stored procedures, variables, and data types, and so forth in a single-named, stored unit in the database. This provides for better orderliness during the development process. In other words packages and its modules are easily understood because of their logical grouping.
2. Grouping of related procedures, functions, etc. in a package also make privilege management easier. Granting the privilege to use a package makes all components of the package accessible to the grantee.
3. Package helps in achieving data abstraction. Package body hides the details of the package contents and the definition of private program objects so that only the package contents are affected if the package body changes.
4. An entire package is loaded into memory when a procedure within the package is called for the first time. This load is completed in one operation, as opposed to the separate loads required for standalone procedures. Therefore, when calls to related packaged procedures occur, no disk I/O is necessary to execute the compiled code already in memory. This results in faster and efficient operation of programs.
5. Packages provide better performance than stored procedures and functions because public package variables persist in memory for the duration of a session. So that they can be accessed by all procedures and functions that try to access them.
6. Packages allow overloading of its member modules. More than one function in a package can be of same name. The functions are differentiated, depending upon the type and number of parameters it takes.

Units of Packages

As described earlier, a package is used to store together, the logically related PL/SQL units. In general, following units constitute a package.

- Procedures
- Functions
- Triggers
- Cursors
- Variables

Parts of Package

A Package has two parts. They are:

- Package specification
- Package body

Package Specification

The specification declares the types, variables, constants, exceptions, cursors, and subprograms that are public and thus available for use outside the package. In case in the package specification declaration there is only types, constants, exception, or variables, then there is no need for the package body because package specification are sufficient for them. Package body is required when there is subprograms like cursors, functions, etc.

Package Body

The package body fully defines subprograms such as cursors, functions, and procedures. All the private declarations of the package are included in the package body. It implements the package specification. A package specification and the package body are stored separately in the database. This allows calling objects to depend on the specification only, not on both. This separation enables to change the definition of program object in the package body without causing Oracle to interfere with other objects that call or reference the program object. Oracle invalidates the calling object if the package specification is changed.

Creating a Package

A package consists of package specification and package body. Hence creation of a package involves creation of the package specification and then creation of the package body.

The package specification is declared using the CREATE PACKAGE command.

The syntax for package specification declaration is as follows.

```
CREATE[OR REPLACE] PACKAGE <package_name>
[AS/IS]
PL/SQL package specification
```

All the procedures, sub programs, cursors declared in the CREATE PACKAGE command are described and implemented fully in the package body along with private members. The syntax for declaring a package body is as follows:

```
CREATE[OR REPLACE] PACKAGE BODY <package_name>
[AS/IS]
PL/SQL package body
```

Member functions and procedures can be declared in a package and can be made public or private member using the keywords public and private. Use of all the private members of the package is restricted within the package while the public members of the package can be accessed and used outside the package.

Referencing Package Subprograms

Once the package body is created with all members as public, we can access them from outside the program. To access these members outside the packages we have to use the dot operator, by prefixing the package object with the package name. The syntax for referencing any member object is as follows:

```
<PACKAGE_NAME>.<VARIABLE_NAME>
```

To reference procedures we have to use the syntax as follows:

```
EXECUTE <package_name>.<procedure_name(variables)>;
```

But the package member can be referenced by only its name if we reference the member within the package. Moreover the EXECUTE command is not required if procedures are called within PL/SQL. Functions can be referenced similar to that of procedures from outside the package using the dot operator.

Public and Private Members of a Package

A package can consist of public as well as private members. Public members are those members which are accessible outside the package, whereas the private members are accessible only from within the package. Private members are just like local members whose are not visible outside the enclosing code block (in this case, a package).

The place where a package member is declared, also matters in deciding the visibility of that member. Those members whose declaration is found in the package specification are the public members. The package members that are not declared in the package specification but directly defined in the package body become the private members.

Viewing Existing Procedural Objects

The source code for the existing procedures, functions, and packages can be queried from the following data dictionary views.

USER_SOURCE	Procedural objects owned by the user.
ALL_SOURCE	Procedural objects owned by the user or to which the user has been granted access.
DBA_SOURCE	Procedural objects in the database.

Removing a Package

A package can be dropped from the database just like any other table or database object. The exact syntax of the command to be used for dropping a package is:

```
DROP PACKAGE <PACKAGE_NAME>;
```

To drop a package a user either must own the package or he should have DROP ANY PACKAGE privilege.

5.15 Exceptions Handling

During execution of a PL/SQL block of code, Oracle executes every SQL sentence within the PL/SQL block. If an error occurs or an SQL sentence fails, Oracle considers this as an Exception. Oracle engine immediately tries to handle the exception and resolve it, by raising a built-in Exception handler.

Introduction to Exceptions

One can define an EXCEPTION as any error or warning condition that arises during runtime. The main intention of building EXCEPTION technique is to continue the processing of a program even when it encounters runtime error or warning and display suitable messages on console so that user can handle those conditions next time.

In absence of exceptions, unless the error checking is disabled, a program will exit abnormally whenever some runtime error occurs. But with exceptions,

if at all some error situation occurs, the exceptional handler unit will flag an appropriate error/warning message and will continue the execution of program and finally come out of the program successfully.

An exception handler is a code block in memory that attempts to resolve the current exception condition. To handle very common and repetitive exception conditions Oracle has about 20 Named Exception Handlers. In addition to these for other exception conditions Oracle has about 20,000 Numbered Exception Handlers, which are identified by four integers preceded by hyphen. Each exception handler, irrespective of how it is defined, (i.e., by Name or Number) has code attached to it that attempts to resolve the exception condition. This is how Oracle's Internal Exception handling strategy works.

Oracle's internal exception handling code can be overridden. When this is done Oracle's internal exception handling code is not executed but the code block that takes care of the exception condition, in the exception section, of the PL/SQL block is executed. As soon as the Oracle invokes an exception handler the exception handler goes back to the PL/SQL block from which the exception condition was raised. The exception handler scans the PL/SQL block for the existence of exception section within the PL/SQL block. If an exception section within the PL/SQL block exists the exception handler scans the first word, after the key word WHEN, within the exception section. If the first word after the key word WHEN is the exception handler's name then the exception handler executes the code contained in the THEN section of the construct, the syntax follows:

```
EXCEPTION
WHEN exception_name THEN
User defined action to be carried out.
```

Exceptions can be internally defined (by the run-time system) or user defined. Internally defined exceptions are raised implicitly (automatically) by the run-time system. User-defined exceptions must be raised explicitly by RAISE statements, which can also raise internally defined exceptions. Raised exceptions are handled by separate routines called exception handlers. After an exception handler runs, the current block stops executing and the enclosing block resumes with the next statement. If there is no enclosing block, control returns to the host environment.

Advantages of Using Exceptions

1. Control over abnormal exits of executing programs on encountering error conditions, hence the behavior of application becomes more reliable.
2. Meaningful messages can be flagged so that the developer can become aware of error and warning conditions and act upon them.
3. In traditional error checking system, if same error is to be checked at several places, you are required to code the same error check at all those

places. But with exception handling technique, we will write the exception for that particular error only once in the entire code. Whenever that type error occurs at any place in code, the exceptional handler will automatically raise the defined exception.

4. Being a part of PL/SQL, exceptions can be coded at suitable places and can be coded isolated like procedures and functions. This improves the overall readability of a PL/SQL program.
5. Oracle's internal exception mechanism combined with user-defined exceptions, considerably reduce the development efforts required for cumbersome error handling.

Predefined and User-Defined Exceptions

As discussed earlier there are some predefined or internal exceptions, and a developer can also code user-defined exceptions according to his requirement. In next session we will be looking closely at these two types of exceptions.

Internally (Predefined) Defined Exceptions

An internal exception is raised implicitly whenever a PL/SQL program violates an Oracle rule or exceeds a system-dependent limit. Every Oracle error has a number, but exceptions must be handled by name. So, PL/SQL predefines a name for some common errors to raise them as exception. For example, if a SELECT INTO statement returns no rows, PL/SQL raises the predefined exception NO_DATA_FOUND, which has the associated Oracle error number ORA-01403.

Example

Figure 5.21 shows the internally defined exception NO_DATA_FOUND, when we want to get a salary of an employee who is not in the EMP table.

If we execute this query with some emp_name say "XYZ" as input and if emp_name column of employee table does not contain any value "XYZ," Oracle's internal exception handling mechanism will raise NO_DATA_FOUND exception even when we have not coded for it.

PL/SQL declares predefined exceptions globally in package STANDARD, which defines the PL/SQL environment. Some of the commonly used exceptions are as follows:

User Defined Exceptions

Unlike internally defined exceptions, user-defined exceptions must be declared and raised explicitly by RAISE statements. Exceptions can be declared only in the declarative part of a PL/SQL block, subprogram, or package. An exception is declared by introducing its name, followed by the keyword EXCEPTION.

Name of the exception	Raised when ...
ACCESS_INTO_NULL	Your program attempts to assign values to the attributes of an uninitialized (atomically null) object.
COLLECTION_IS_NULL	Your program attempts to apply collection methods, other than EXISTS to an uninitialized (atomically null) nested table or varray, or the program attempts to assign values to the elements of an uninitialized nested table or varray.
CURSOR_ALREADY_OPEN	Your program attempts to open an already open cursor. A cursor must be closed before it can be reopened. A cursor FOR loop automatically opens the cursor to which it refers. So, your program cannot open that cursor inside the loop.
DUP_VAL_ON_INDEX	Your program attempts to store duplicate values in a database column that is constrained by a unique index.
INVALID_CURSOR	Your program attempts an illegal cursor operation such as closing an unopened cursor.
INVALID_NUMBER	In a SQL statement, the conversion of a character string into a number fails because the string does not represent a valid number. (In procedural statements, VALUE_ERROR is raised.)
LOGIN_DENIED	Your program attempts to log on to Oracle with an invalid username and/or password.
NO_DATA_FOUND	A SELECT INTO statement returns no rows, or your program references a deleted element in a nested table or an uninitialized element in an index-by table. SQL aggregate functions such as AVG and SUM always return a value or a null. So, a SELECT INTO statement that calls an aggregate function will never raise NO_DATA_FOUND. The FETCH statement is expected to return no rows eventually, so when that happens, no exception is raised.
NOT_LOGGED_ON	Your program issues a database call without being connected to Oracle.

Continued.

Name of the exception	Raised when ...
ROWTYPE_MISMATCH	The host cursor variable and PL/SQL cursor variable involved in an assignment have incompatible return types. For example, when an open host cursor variable is passed to a stored subprogram, the return types of the actual and formal parameters must be compatible.
PROGRAM_ERROR	PL/SQL has an internal problem.
SELF_IS_NULL	Your program attempts to call a MEMBER method on a null instance. That is, the built-in parameter SELF (which is always the first parameter passed to a MEMBER method) is null.
STORAGE_ERROR	PL/SQL runs out of memory or memory has been corrupted.
SUBSCRIPT_BEYOND_COUNT	Your program references a nested table or varray element using an index number larger than the number of elements in the collection.
SUBSCRIPT_OUTSIDE_LIMIT	Your program references a nested table or varray element using an index number (−1 for example) that is outside the legal range.
SYS_INVALID_ROWID	The conversion of a character string into a universal rowid fails because the character string does not represent a valid rowid.
TIMEOUT_ON_RESOURCE	A time-out occurs while Oracle is waiting for a resource.
TOO_MANY_ROWS	A SELECT INTO statement returns more than one row.
VALUE_ERROR	An arithmetic, conversion, truncation, or size constraint error occurs. For example, when your program selects a column value into a character variable, if the value is longer than the declared length of the variable, PL/SQL aborts the assignment and raises VALUE_ERROR. In procedural statements, VALUE_ERROR is raised if the conversion of a character string into a number fails. (In SQL statements, INVALID_NUMBER is raised.)
ZERO_DIVIDE	Your program attempts to divide a number by zero.

The screenshot shows the Oracle SQL*Plus interface. The command prompt shows the execution of a query: `SQL> SELECT * FROM EMP;`. The result is a table with three columns: EMPID, EMPNAME, and SAL. The data rows are: E101 KARTHIKEYAN 6250, E102 ANAND 3125, and E103 RAJA 1900. Below the query, the user attempts to execute a procedure: `SQL> EXECUTE GET_SAL_PROCEDURE('E666');`. This results in an error: `BEGIN GET_SAL_PROCEDURE('E666'); END;` followed by `* ERROR at line 1: ORA-01403: no data found ORA-06512: at "SKK.GET_SAL_PROCEDURE", line 6 ORA-06512: at line 1`. The prompt returns to `SQL> |`.

EMPID	EMPNAME	SAL
E101	KARTHIKEYAN	6250
E102	ANAND	3125
E103	RAJA	1900

Fig. 5.21. Internally defined exception

The syntax is as follows:

```
DECLARE
<exception_name>EXCEPTION;
```

Exceptions are declared in the same way as the variables. But exceptions cannot be used in assignments or SQL expressions/statements as they are not data items. The visibility of exceptions is governed by the same scope rules which apply to variables also.

Raising User-Defined and Internal Exceptions

As seen in the previous example, one can notice a statement “RAISE Exception1.” This statement is used to explicitly raise the exception “Exception1,” the reason being, unlike internally defined exceptions which are automatically raised by “OracleS” run time engine, user-defined exceptions have to be raised explicitly by using RAISE statement. However, it is always possible to RAISE predefined (internally defined) exceptions, if needed, in the same way as do the user-defined exceptions, which is illustrated in Fig. 5.22

```
RAISE <exception_name>;
```

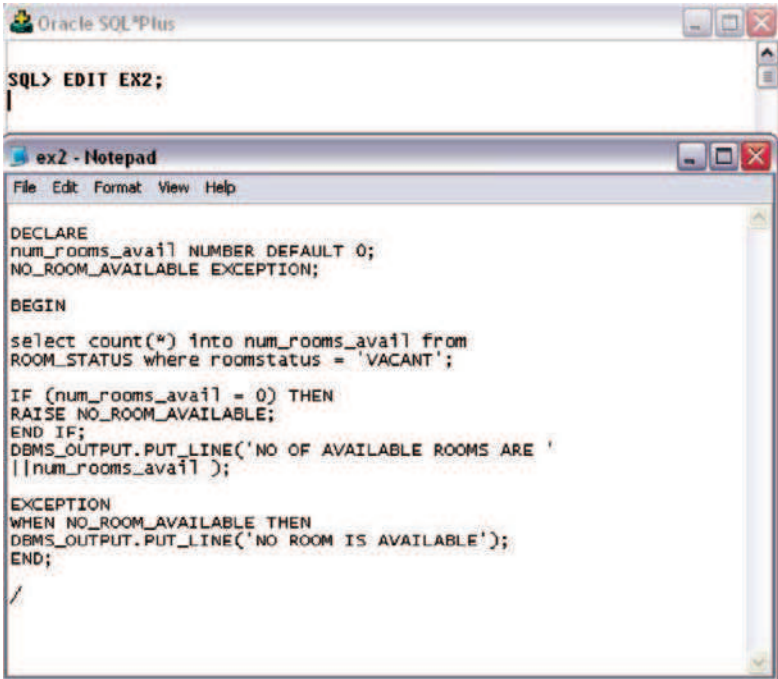


Fig. 5.22. Exception example

Example

Create a table as follows,

```

CREATE TABLE ROOM_STATUS (ROOM_NO NUMBER(5)
PRIMARY KEY,
CAPACITY NUMBER(2),
ROOMSTATUS VARCHAR2(20),
RENT NUMBER(4),
CHECK (ROOMSTATUS IN ('VACANT','BOOKED')));

```

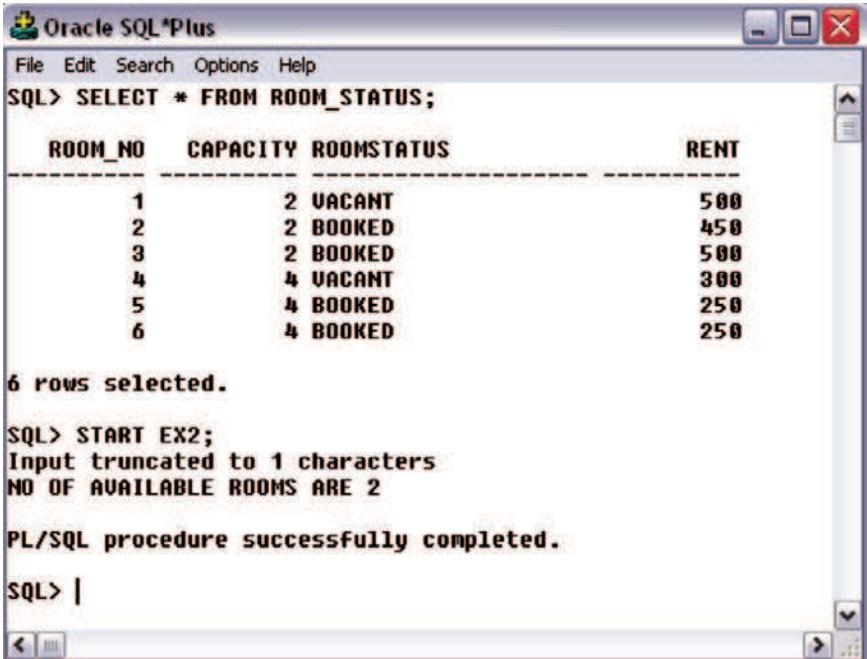
User-Defined Error Reporting – Use of Raise_Application_Error

RAISE_APPLICATION_ERROR lets display the messages we want whenever a standard internal error occurs. RAISE_APPLICATION_ERROR associates an Oracle Standard Error Number with the message we define. The syntax for RAISE_APPLICATION_ERROR is as follows:

```

RAISE_APPLICATION_ERROR (Oracle Error Number,
Error Message, TRUE/FALSE);

```



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT * FROM ROOM_STATUS;

  ROOM_NO  CAPACITY ROOMSTATUS      RENT
-----
         1         2 VACANT                500
         2         2 BOOKED                450
         3         2 BOOKED                500
         4         4 VACANT                300
         5         4 BOOKED                250
         6         4 BOOKED                250

6 rows selected.

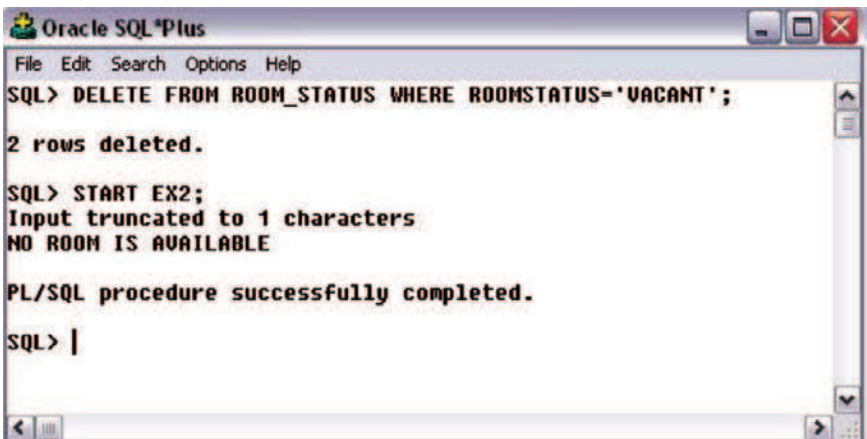
SQL> START EX2;
Input truncated to 1 characters
NO OF AVAILABLE ROOMS ARE 2

PL/SQL procedure successfully completed.

SQL> |

```

Fig. 5.23. Without exception



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> DELETE FROM ROOM_STATUS WHERE ROOMSTATUS='VACANT';

2 rows deleted.

SQL> START EX2;
Input truncated to 1 characters
NO ROOM IS AVAILABLE

PL/SQL procedure successfully completed.

SQL> |

```

Fig. 5.24. Execution of exception

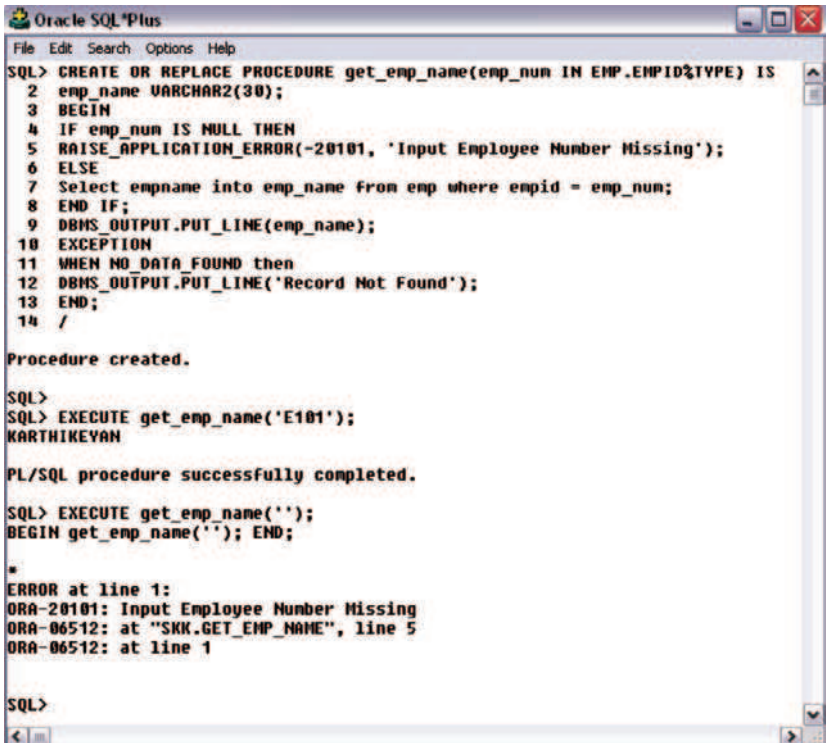
Figures 5.23 and 5.24 show the output for two conditions 'Room Available' and 'Vacant'.

Oracle error number is the standard Oracle error (–20000 to –20999) that we want to associate with the message (max 2,048 kb) defined, TRUE/FALSE indicates whether to place the error message on previous error stack (TRUE) or to replace all the errors with this message (FALSE).

RAISE_APPLICATION_ERROR can be called only from an executing subprogram. As soon as the subprogram encounters RAISE_APPLICATION_ERROR, the subprogram returns control back to the calling PL/SQL code thereby displaying the error message. We can handle the exception raised in the calling portion of PL/SQL block.

Example

Following Fig. 5.25 illustrates the use of RAISE_APPLICATION_ERROR command with the procedure named get_emp_name.



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> CREATE OR REPLACE PROCEDURE get_emp_name(emp_num IN EMP.EMPID%TYPE) IS
 2 emp_name VARCHAR2(30);
 3 BEGIN
 4 IF emp_num IS NULL THEN
 5 RAISE_APPLICATION_ERROR(-20101, 'Input Employee Number Missing');
 6 ELSE
 7 Select empname into emp_name from emp where empid = emp_num;
 8 END IF;
 9 DBMS_OUTPUT.PUT_LINE(emp_name);
10 EXCEPTION
11 WHEN NO_DATA_FOUND then
12 DBMS_OUTPUT.PUT_LINE('Record Not Found');
13 END;
14 /

Procedure created.

SQL>
SQL> EXECUTE get_emp_name('E101');
KARTHIKEYAN

PL/SQL procedure successfully completed.

SQL> EXECUTE get_emp_name('');
BEGIN get_emp_name(''); END;

*
ERROR at line 1:
ORA-20101: Input Employee Number Missing
ORA-06512: at "SKK.GET_EMP_NAME", line 5
ORA-06512: at line 1

SQL>

```

Fig. 5.25. Raise_application_error example

5.16 Database Triggers

A database trigger is a stored PL/SQL program unit associated with a specific database table. It can perform the role of a constraint, which forces the integrity of data. It is the most practical way to implement routines and granting integrity of data. Unlike the stored procedures or functions, which have to be explicitly invoked, these triggers implicitly get fired whenever the table is affected by the SQL operation. For any event that causes a change in the contents of a table, a user can specify an associated action that the DBMS should carry out. Trigger follows the Event-Condition-Action scheme (ECA scheme).

Privileges Required for Triggers

Creation or alteration of a TRIGGER on a specific table requires TRIGGER privileges as well as table privileges. They are:

1. To create TRIGGER in one's own schema, he must have CREATE TRIGGER privilege. To create a trigger in any other's schema, one must have CREATE ANY TRIGGER system privilege.
2. To create a trigger on table, one must own the table or should have ALTER privilege for that table or should have ALTER ANY TABLE privilege.
3. To ALTER a trigger, one must own that trigger or should have ALTER ANY TRIGGER privilege. Also since the trigger will be operating on some table, one also requires ALTER privilege on that table or ALTER ANY TABLE table privilege.
4. To create a TRIGGER on any database level event, one must have ADMINISTER DATABASE TRIGGER system privilege.

Context to Use Triggers

Following are the situations to use the triggers efficiently:

- Use triggers to guarantee that when a specific operation is performed, related actions are performed.
- Do not define triggers that duplicate the functionality already built into Oracle. For example, do not define triggers to enforce data integrity rules that can be easily enforced using declarative integrity constraints.
- Limit the size of triggers. If the logic for our trigger requires much more than 60 lines of PL/SQL code, then it is better to include most of the code in a stored procedure and call the procedure from the trigger.
- Use triggers only for centralized, global operations that should be fired for the triggering statement, regardless of which user or database application issues the statement.
- Do not create recursive triggers which cause the trigger to fire recursively until it has run out of memory.

- Use triggers on DATABASE judiciously. They are executed for every user every time the event occurs on which the trigger is created.

Uniqueness of Trigger

Different types of integrity constraints provide a declarative mechanism to associate “simple” conditions with a table such as a primary key, foreign keys, or domain constraints. Complex integrity constraints that refer to several tables and attributes cannot be specified within table definitions. Triggers, in contrast, provide a procedural technique to specify and maintain integrity constraints.

Triggers even allow users to specify more complex integrity constraints since a trigger essentially is a PL/SQL procedure. Such a procedure is associated with a table and is automatically called by the database system whenever a certain modification (event) occurs on that table.

Simply we can say that trigger is LESS DECLARATIVE AND MORE PROCEDURAL TYPE CONSTRAINT ENFORCEMENT. Triggers are used generally to implement business rules in the database. It is the major difference between Triggers and Integrity Constraints.

Create Trigger Syntax

The Create trigger syntax is as follows:

```
CREATE [OR REPLACE] TRIGGER <trigger_name>
[BEFORE/AFTER/INSTEAD OF]
[INSERT/UPDATE/DELETE [of column,..]] ON <table_name>
[REFERENCING [OLD [AS] <old_name> | NEW [AS]
<new_name>]
[FOR EACH STATEMENT/FOR EACH ROW]
[WHEN <condition>]
[BEGIN
–PL/SQL block
END];
```

This syntax can be explained as follows.

Parts of Trigger

A trigger has three basic parts:

- A triggering event or statement
- A trigger restriction
- A trigger action

Trigger Event or Statement

A triggering event or statement is the SQL statement, database event, or user event like update, delete, insert, etc. that causes a trigger to be fired. It also specifies the table to which the trigger is associated. Trigger statement or an event can be any of the following:

1. INSERT, UPDATE, or DELETE on a specific table or view.
2. CREATE, ALTER, or DROP on any schema object.
3. Database activities like startup and shutdown.
4. User activities like logon and logoff.
5. A specific error message on any error message.

Figure 5.26 shows a database application with some SQL statements that implicitly fire several triggers stored in the database. It shows three triggers, which are associated with the INSERT, UPDATE, and DELETE operation in the database table. When these data manipulation commands are given, the corresponding trigger gets automatically fired performing the task described in the corresponding trigger body.

Trigger Restriction

A trigger restriction is any logical expression whose outcome is TRUE/FALSE/UNKNOWN. For a trigger to fire, this logical expression must evaluate to TRUE. Typically, a restriction is a part of trigger declaration that follows the keyword WHEN.

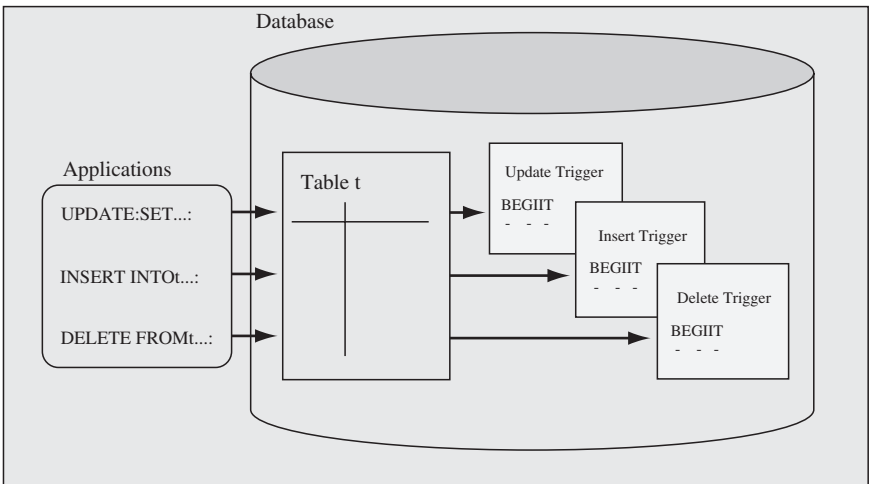


Fig. 5.26. Database application with some SQL statements that implicitly fire several triggers stored in the database

Trigger Action

A trigger action is the PL/SQL block that contains the SQL statements and code to be executed when a triggering statement is issued and the trigger restriction evaluates to TRUE. It is also called the trigger body. Like stored procedures, a trigger action can contain SQL and PL/SQL.

Following statements will explain the various keywords used in the syntax.

BEFORE and AFTER keyword indicates whether the trigger should be executed before or after the trigger event, where a triggering event can be INSERT, UPDATE, or DELETE. Any combination of triggering events can be included in the same database trigger.

When referring the old and new values of columns, we can use the defaults (“old” and “new”) or we can use the REFERENCING clause to specify other names. FOR EACH ROW clause causes the trigger to fire once for each record created, deleted, or modified by the triggering statement. When working with row triggers, the WHEN clause can be used to restrict the records for which the trigger fires.

We can use INSTEAD OF triggers to tell the database what to do instead of performing the actions that invoked the trigger. For example, we can use it on a VIEW to redirect the inserts into a table or to update multiple tables that are parts of the view.

5.17 Types of Triggers

Type of trigger firing, level at which a trigger is executed, and the types of events form the basis classification of triggers into different categories. This section describes the different types of triggers. The broad classification of triggers is as shown below.

On the Basis of Type of Events

- Triggers on System events
- Trigger on User events

On the Basis of the Level at which Triggers are Executed

- Row Level Triggers
- Statement Level Triggers

On the Basis of Type of Trigger/Firing or Triggering Transaction

- BEFORE Triggers
- AFTER Triggers
- INSTEAD OF

Triggers on System Events

System events that can fire triggers are related to instance startup and shutdown and error messages. Triggers created on startup and shutdown events have to be associated with the database; triggers created on error events can be associated with the database or with a schema.

BEFORE Triggers

BEFORE triggers execute the trigger action before the triggering statement is executed. It is used to derive specific column values before completing a triggering DML, DDL statement or to determine whether the triggering statement should be allowed to complete.

Example

We can define a BEFORE trigger on the passengers_detail table that gets fired before deletion of any row. The trigger will check the system date and if the date is Sunday, it will not allow any deletion on the table.

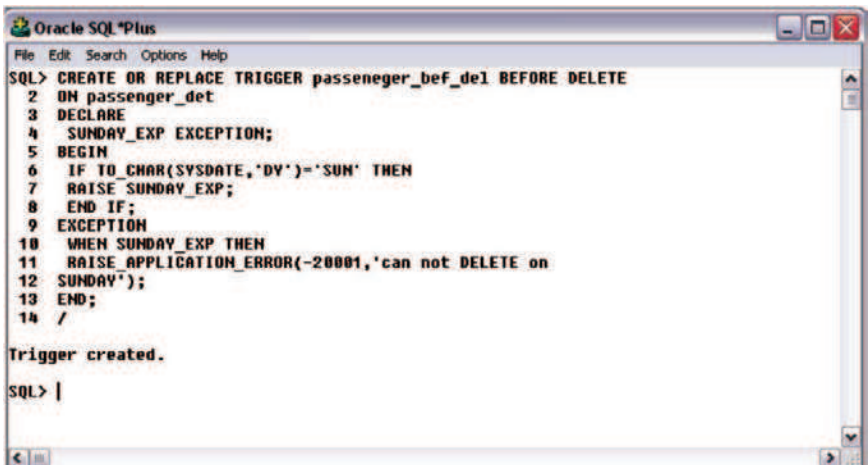
The trigger can be created in Oracle as shown in Fig. 5.27.

The trigger action can be shown as in Fig. 5.28.

As soon as we try to delete a record from passenger_detail table, the above trigger will be fired and due to SUNDAY_EXP fired, all the changes will be rolled back or undone and the record will not be deleted.

AFTER Triggers

AFTER triggers execute the trigger action after the triggering statement is executed. AFTER triggers are used when we want the triggering statement to complete before executing the trigger action, or to execute some additional logic to the before trigger action.



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> CREATE OR REPLACE TRIGGER passeneger_bef_de1 BEFORE DELETE
2 ON passenger_det
3 DECLARE
4 SUNDAY_EXP EXCEPTION;
5 BEGIN
6 IF TO_CHAR(SYSDATE,'DY')='SUN' THEN
7 RAISE SUNDAY_EXP;
8 END IF;
9 EXCEPTION
10 WHEN SUNDAY_EXP THEN
11 RAISE_APPLICATION_ERROR(-20001,'can not DELETE on
12 SUNDAY');
13 END;
14 /

Trigger created.
SQL> |

```

Fig. 5.27. BEFORE trigger creation

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select * from passenger_det;

  PASSNO NAME                CITY
-----
1 skk                          cbe
2 anand                       Madurai
3 Raja                        Nagercoil

SQL> delete from passenger_det where passno=1;
delete from passenger_det where passno=1
*
ERROR at line 1:
ORA-20001: can not DELETE on
SUNDAY

```

Fig. 5.28. BEFORE trigger execution

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> CREATE OR REPLACE TRIGGER PASSENGER_DEL_AFT AFTER DELETE ON RESERV_DET
2 FOR EACH ROW
3 BEGIN
4 DELETE FROM SKK.PASSENGER_DET WHERE PASSENGER_DET.PASSNO=:OLD.PASSNO;
5 END;
6 /

Trigger created.

SQL> |

```

Fig. 5.29. AFTER trigger creation

Example

We can define an AFTER trigger on the reserv_det table that gets fired every time one row is deleted from the table. This trigger will determine the passenger_id of the deleted row and subsequently delete the corresponding row from the passengers_det table with same passenger_id.

Trigger can be created as shown in Fig. 5.29

Trigger action can be shown as in Fig. 5.30. In this figure, the content of the relations passenger_det and reserve_det are shown before and after the triggering event.

Triggers on LOGON and LOGOFF Events

LOGON and LOGOFF triggers can be associated with the database or with a schema. Their attributes include the system event and username, and they can specify simple conditions on USERID and USERNAME.

- LOGON triggers fire after a successful logon of a user.
- LOGOFF triggers fire at the start of a user logoff.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT * FROM PASSENGER_DET;

PASS NAME                                CITY
-----
101 ANAND                                MAGERCOIL
130 RAJA                                  MADURAI
117 SKK                                    COIMBATORE

SQL> SELECT * FROM RESERV_DET;

PASS FLIGHT RESERV_STATUS    CLA
-----
117 R117    RESERVED         F
101 R101    WAITING           F
130 R130    CANCELLED           S

SQL> DELETE FROM RESERV_DET WHERE PASSNO='130';

1 row deleted.

SQL> SELECT * FROM RESERV_DET;

PASS FLIGHT RESERV_STATUS    CLA
-----
117 R117    RESERVED         F
101 R101    WAITING           F

SQL> SELECT * FROM PASSENGER_DET;

PASS NAME                                CITY
-----
101 ANAND                                MAGERCOIL
117 SKK                                    COIMBATORE

SQL>

```

Fig. 5.30. AFTER trigger execution

Example

Let us create a trigger on LOGON event called pub_log, which will store the number, date, and user of login done by different user in that particular database. The trigger will store this information in a table called log_detail. The table log_detail must be created before trigger creation by logging into Administrator login. The trigger can be created as shown in Fig. 5.31.

After logging into another login, if we see the content of the relation log_detail it will show who are all logged into database. The value of the attribute log_times would go on increasing with every login into the database which is indicated in Fig. 5.32.

Note The log_detail relation is visible only in Administrator login.

Triggers on DDL Statements

This trigger gets fired when DDL statement such as CREATE, ALTER, or DROP command is issued. DDL triggers can be associated with the database or with a schema. Moreover depending on the time of firing of trigger, this


```

Oracle SQL*Plus
File Edit Search Options Help
SQL> CREATE TABLE log_detail
 2 (log_times NUMBER,
 3 log_day DATE,
 4 log_name VARCHAR2(25)
 5 );

Table created.

SQL>
SQL>
SQL> CREATE TRIGGER pub_log AFTER LOGON ON DATABASE
 2 DECLARE
 3 j NUMBER;
 4 BEGIN
 5 SELECT COUNT(*) INTO j FROM log_detail;
 6 j := j + 1;
 7 INSERT INTO log_detail
 8 VALUES(j,SYSDATE,SYS.LOGIN_USER);
 9 END;
10 /

Trigger created.

SQL> |

```

Fig. 5.31. Triggers on LOGON event creation

trigger can be classified into BEFORE and AFTER. Hence the triggers on DDL statements can be as follows:

- BEFORE CREATE and AFTER CREATE triggers fire when a schema object is created in the database or schema.
- BEFORE ALTER and AFTER ALTER triggers fire when a schema object is altered in the database or schema.
- BEFORE DROP and AFTER DROP triggers fire when a schema object is dropped from the database or schema.

Example

Let us create a trigger called “no_drop_pass” that fires before dropping any object on the schema of the user with username “skk.” It checks whether the object type and name. If the object name is “passenger_det” and object type is table, it raises an application error and prevents the dropping of the

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> connect skk
Enter password: *****
Connected.
SQL> disconnect
Disconnected from Oracle8i Enterprise Edition Release 8.1.7.0.0 - Production
With the Partitioning option
JServer Release 8.1.7.0.0 - Production
SQL> connect system
Enter password: *****
Connected.
SQL> select * from log_detail;

LOG_TIMES LOG_DAY LOG_NAME
-----
2 08-AUG-05 SYSTEM
1 08-AUG-05 SKK

```

Fig 5.32. Triggers on LOGON Event Execution

Fig. 5.32. Triggers on LOGON event execution

table. The syntax for creating the trigger is as follows. Remember to create the trigger by logging as administrator in the database. The trigger can be created as shown in Fig. 5.33.

The trigger is executed as shown in Fig. 5.34.

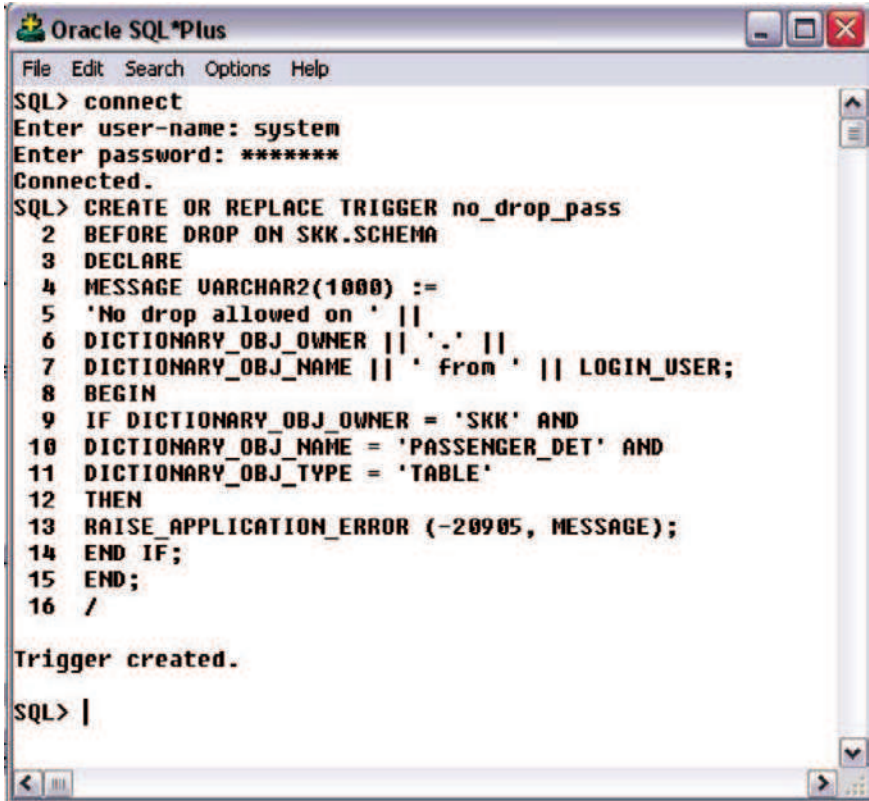
Triggers on DML Statements

This trigger gets fired when DML statement such as INSERT, UPDATE, or DELETE command is issued. DML triggers can be associated with the database or with a schema. Depending on the time of firing of trigger, this trigger can be classified into BEFORE and AFTER. Moreover, when we define a trigger on a DML statement, we can specify the number of times the trigger action is to be executed: once for every row or once for the triggering statement.

Row Level Triggers

A row level trigger, as its name suggests, is fired for each row that will be affected by the SQL statement, which fires the trigger. Suppose for example if an UPDATE statement updates “N” rows of a table, a row level trigger defined for this UPDATE on that particular table will be fired once for each of those “N” affected rows. If a triggering SQL statement affects no rows, a row trigger is not executed at all. To specify a trigger of row type, FOR EACH ROW clause is used after the name of table.

In row level triggers, the statements in a trigger action have access to column values (new and old) of the current row being processed by the trigger. The names of the new and old values are called correlation names. They allow access to new and old values for each column. By means of new, one refers to the new value with which the row in the table is updated or inserted. On



```

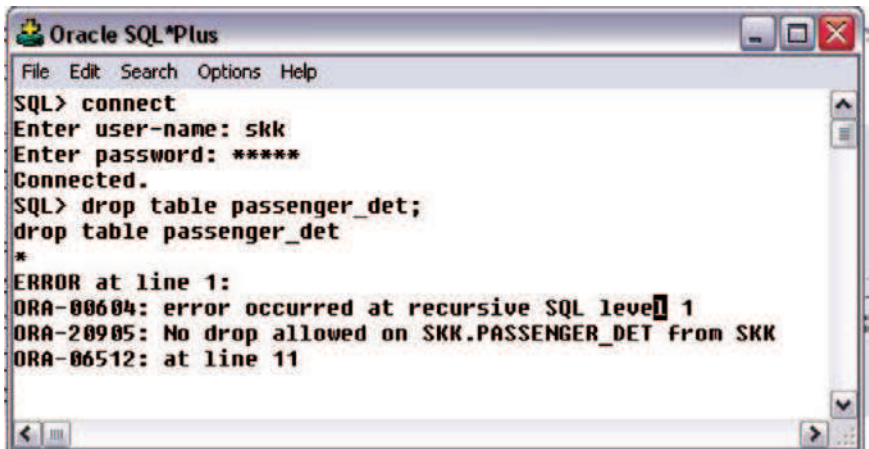
Oracle SQL*Plus
File Edit Search Options Help
SQL> connect
Enter user-name: system
Enter password: *****
Connected.
SQL> CREATE OR REPLACE TRIGGER no_drop_pass
  2 BEFORE DROP ON SKK.SCHEMA
  3 DECLARE
  4 MESSAGE VARCHAR2(1000) :=
  5 'No drop allowed on ' ||
  6 DICTIONARY_OBJ_OWNER || '.' ||
  7 DICTIONARY_OBJ_NAME || ' from ' || LOGIN_USER;
  8 BEGIN
  9 IF DICTIONARY_OBJ_OWNER = 'SKK' AND
 10 DICTIONARY_OBJ_NAME = 'PASSENGER_DET' AND
 11 DICTIONARY_OBJ_TYPE = 'TABLE'
 12 THEN
 13 RAISE APPLICATION_ERROR (-20905, MESSAGE);
 14 END IF;
 15 END;
 16 /

Trigger created.

SQL> |

```

Fig. 5.33. Trigger on DDL statement creation



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> connect
Enter user-name: skk
Enter password: *****
Connected.
SQL> drop table passenger_det;
drop table passenger_det
*
ERROR at line 1:
ORA-00604: error occurred at recursive SQL leve 1
ORA-20905: No drop allowed on SKK.PASSENGER_DET From SKK
ORA-06512: at line 11

```

Fig. 5.34. Trigger on DDL statement execution

the other hand by means of `old`, one refers to the old value, which is being updated or deleted. Row level triggers are useful if the code in the trigger action depends on data provided by the triggering statement or rows that are affected.

Example

The AFTER trigger on `reserv_det` table that deletes all corresponding rows from `passenger_det` table with the same `passenger_id` is a row level trigger as shown in Figs. 5.29 and 5.30, respectively.

Statement Level Triggers

Unlike row level trigger, a statement level trigger is fired only once on behalf of the triggering SQL statement, regardless of the number of rows in the table that the triggering statement affects. Even if the triggering statement affects no rows, the statement level trigger will execute exactly once. For example, if a DELETE statement deletes several rows from a table, a statement-level DELETE trigger is fired only once, regardless of how many rows are deleted from the table. Default type of any trigger is Statement level trigger. Statement level triggers are useful if the code in the trigger action does not depend on the data provided by the triggering statement or the rows affected.

Example

The BEFORE trigger on `passenger_det` table that checks that no row should be deleted on Sunday is a statement level trigger as shown in Figs. 5.27 and 5.28, respectively.

INSTEAD-OF Triggers

INSTEAD-OF triggers are used to tell Oracle what to do instead of performing the actions that executed the trigger. It is applicable to both object views and standard relational database. This trigger can be used to redirect table inserts into a different table or to update different tables that are the part of the view. This trigger is used to perform any action instead of the action that executes the trigger. In simpler words if the task associated with this trigger fails, the trigger is fired. It is used mostly for object views rather than tables. This trigger is used to manipulate the tables through the views.

Enabling and Disabling a Trigger

By default, a trigger is enabled when it is created. Only an enabled trigger gets fired whenever the trigger restriction evaluates to TRUE. Disabled triggers do

not get fired even when the triggering statement is issued. Thus a trigger can be in either of two distinct modes:

- Enabled (an enabled trigger executes its trigger action if a triggering statement is issued and the trigger restriction (if any) evaluates to TRUE).
- Disabled (a disabled trigger does not execute its trigger action, even if a triggering statement is issued and the trigger restriction (if any) would evaluate to TRUE).

The need to disable the trigger is there are some situations like heavy data load or partially succeeded load operations. In case of heavy data load condition, disabling trigger may dramatically improve the performance. After load, one has to do all those data operations manually which otherwise a trigger would have done. In case of partial succeeded load, since a part of load is successful, the triggers are already executed for that part. Now when we start the same load fresh, it may be possible that the same trigger would be executed twice which may cause some undesirable effects. So the best way is to disable the trigger and do the operations manually after the entire load is successful.

For enabled triggers, Oracle automatically does the following:

- Prepares a definite plan for execution of triggers of different types.
- Decides time for integrity constraint checking for each type of trigger and ensures that none of the triggers is violating integrity constraints.
- Manages the dependencies among triggers and schema objects referenced in the code of the trigger action.
- No definite order for firing of multiple triggers of same type.

Syntax

```
ALTER TRIGGER <Trigger name> ENABLE/DISABLE;
```

Example

The `passenger_bef_del` trigger can be disabled and enabled as shown in Fig. 5.35, it shows how Oracle behaves for enabled/disabled triggers.

Replacing Triggers

Triggers cannot be altered explicitly. Triggers have to be replaced with a new definition using `OR REPLACE` option with `CREATE TRIGGER` command. In such case the old definition of the trigger is dropped and the new definition is entered in the data dictionary.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> ALTER TRIGGER PASSENEGER_BEF_DEL DISABLE;

Trigger altered.

SQL> SELECT * FROM PASSENGER_DET;

PASS NAME                                CITY
-----
117 SKK                                    CBE
101 ANAND                                  NAGERCOIL
130 RAJA                                    MADURAI

SQL> DELETE FROM PASSENGER_DET WHERE PASSNO='130';

1 row deleted.

SQL> SELECT * FROM PASSENGER_DET;

PASS NAME                                CITY
-----
117 SKK                                    CBE
101 ANAND                                  NAGERCOIL

SQL> ALTER TRIGGER PASSENEGER_BEF_DEL ENABLE;

Trigger altered.

SQL> DELETE FROM PASSENGER_DET WHERE PASSNO='117';
DELETE FROM PASSENGER_DET WHERE PASSNO='117'
*
ERROR at line 1:
ORA-20001: can not DELETE on
SUNDAY

```

Fig. 5.35. Enabling and disabling the trigger

The exact syntax for replacing the trigger is as follows:

Syntax

```
CREATE OR REPLACE TRIGGER <trigger_name> AS/IS
<trigger_definition>;
```

The `trigger_definition` should be as shown in the definition for creating trigger. Alternately the trigger can be dropped and re-created. On dropping a trigger all grants associated with the trigger are dropped as well.

Dropping Triggers

Triggers can be dropped like tables using the drop trigger command. The drop trigger command removes the trigger structure from the database. User needs

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> DROP TRIGGER PASSENERGER_BEF_DEL;

Trigger dropped.

SQL> SELECT * FROM PASSENGER_DET;

PASS NAME                                CITY
-----
117 SKK                                  CBE
101 ANAND                                NAGERCOIL

SQL> DELETE FROM PASSENGER_DET WHERE PASSNO='117';

1 row deleted.

```

Fig. 5.36. Dropping the trigger

to have DROP ANY TRIGGER system privilege to drop a trigger. The exact syntax for dropping a trigger is as follows.

Syntax

DROP TRIGGER <trigger_name>

Example

We drop the trigger passenger_bef_del as shown in Fig. 5.36.

Summary

This chapter has introduced the concept of PL/SQL. The shortcomings of SQL and the need for PL/SQL are given in detail. PL/SQL combines the data manipulation power of SQL with data processing power of procedural language. The PL/SQL language elements like character sets, operators, indicators, punctuation, identifiers, comments, etc. are introduced with examples in this chapter. The different types of iterative control like FOR loop, WHILE loop, their syntax and concepts are given through examples.

A cursor is a mechanism that can be used to process the multiple row result sets one row at a time. Cursors are an inherent structure in PL/SQL. Cursors allow users to easily store and process sets of information in PL/SQL program. The concept of cursor and different types of cursors like implicit cursor, explicit cursor are given through examples.

A procedure is a subprogram that performs some specific task, and stored in the data dictionary. The concept of procedure, function, the difference between procedure and function are given in this chapter.

A package is a collection of related program objects such as procedures, functions, and associated cursors and variables together as a unit in the database. In simpler term, a package is a group of related procedures and functions stored together and sharing common variables, as well as local procedures and function. In this chapter, the package body and how to create a package are explained with examples.

An EXCEPTION is any error or warning condition that arises during runtime. The main intention of building EXCEPTION technique is to continue the processing of a program even when it encounters runtime error or warning and display suitable messages on console so that user can handle those conditions next time. The advantage of using EXCEPTION, different types of EXCEPTIONS are given through example in this chapter.

A database trigger is a stored PL/SQL program unit associated with a specific database table. It can perform the role of a constraint, which forces the integrity of data. The concept of trigger, the uniqueness of trigger, and the use of trigger are explained with examples in this chapter.

Review Questions

5.1. Mention the key difference between SQL and PL/SQL?

SQL is a declarative language. PL/SQL is a procedural language that makes up for all the missing elements in SQL.

5.2. Mention two drawbacks of SQL?

- SQL statements can be executed only one at a time. Every time to execute a SQL statement, a call is made to Oracle engine, thus it results in an increase in database overheads.
- While processing an SQL statement, if an error occurs, Oracle generates its own error message, which is sometimes difficult to understand. If a user wants to display some other meaningful error message, SQL does not have provision for that.

5.3. Identify which one is not included in PL/SQL Character Set?

- (a) * (b) > (c) ! (d) \

Answer: (d)

5.4. What are Lexical units related with PL/SQL?

A line of PL/SQL program contains groups of characters known as lexical units, which can be classified as follows:

- Delimiters
- Identifiers
- Literals
- Comments

5.5. What is Delimiter?

A delimiter is a simple or compound symbol that has a special meaning to PL/SQL.

5.6. Identify which identifier is not permitted in PL/SQL?

- (a) Bn12 (b) Girt-1 (c) Hay# (d) I am

Answer: (d)

5.7. Give the syntax for single-line comments and multiline comments?

Single line comment: –

Multiline comment: /* Some text..... */

5.8. How you declare a record type variable in PL/SQL?

We can declare record type variable for particular table by using the syntax.

<Variable_Name> <Table_name>%ROWTYPE.

ROWTYPE is a keyword for defining record type variables.

5.9. Find out the error in the following PL/SQL statement?

```
IF condition THEN
sequence_of_statements1
ELSE
sequence_of_statements2
END IF;
```

Answer: No Error in the Statement.

5.10. Mention the facilities available for iterating the statements in PL/SQL?

- (a) For-loop
- (b) While-loop
- (c) Loop-Exit

5.11. What is cursor and mention its types in Oracle?

A cursor is a mechanism that can be used to process the multiple row result sets one row at a time.

In other words, cursors are constructs that enable the user to name a private memory area to hold a specific statement for access at a later time. Cursors are an inherent structure in PL/SQL. Cursors allow users to easily store and process sets of information in PL/SQL program.

There are two types of cursors in Oracle

- (a) Implicit and
- (b) Explicit cursors.

5.12. Mention the syntax for opening and closing a cursor.

For Opening: Open <cursor name>

For Closing: Close <cursor name>

5.13. Mention some implicit and explicit cursor attributes.

Implicit:

%NOTFOUND, %FOUND, %ROWCOUNT, and %ISOPEN

Explicit:

Similar to Implicit.

%NOTFOUND, %FOUND, %ROWCOUNT, and %ISOPEN

5.14. What is Procedure in PL/SQL?

A procedure is a subprogram that performs some specific task, and stored in the data dictionary. A procedure must have a name so that it can be invoked or called by any PL/SQL program that appears within an application. Procedures can take parameters from the calling program and perform the specific task. Before the procedure or function is stored, the Oracle engine parses and compiles the procedure or function.

5.15. Mention any four advantages of procedures and function?

1. It modifies one routine to affect multiple applications.
2. It modifies one routine to eliminate duplicate testing.
3. It ensures that related actions are performed together, or not at all, by doing the activity through a single path.
4. It avoids PL/SQL parsing at runtime by parsing at compile time.

5.16. What is the syntax used in PL/SQL for dropping a procedure?

DROP PROCEDURE <PROCEDURE NAME>

5.17. Mention three differences between functions and procedures?

1. A procedure never returns a value to the calling portion of code, whereas a function returns exactly one value to the calling program.
2. As functions are capable of returning a value, they can be used as elements of SQL expressions, whereas the procedures cannot. However, user defined functions cannot be used in CHECK or DEFAULT constraints and can not manipulate database values, to obey function purity rules.

3. It is mandatory for a function to have at least one RETURN statement, whereas for procedures there is no restriction. A procedure may have a RETURN statement or may not. In case of procedures with RETURN statement, simply the control of execution is transferred back to the portion of code that called the procedure.

5.18. What is Purity rule for functions in PL/SQL?

For a function to be eligible for being called in SQL statements, it must satisfy following requirements, which are known as Purity Rules.

1. When called from a SELECT statement or a parallelized INSERT, UPDATE, or DELETE statement, the function cannot modify any database tables.
2. When called from an INSERT, UPDATE, or DELETE statement, the function cannot query or modify any database tables modified by that statement.
3. When called from a SELECT, INSERT, UPDATE, or DELETE statement, the function cannot execute SQL transaction control statements (such as COMMIT), session control statements (such as SET ROLE), or system control statements (such as ALTER SYSTEM). Also, it cannot execute DDL statements (such as CREATE) because they are followed by an automatic commit.

5.19. What is a syntax for deleting a function in PL/SQL?

DROP FUNCTION <FUNCTION NAME>

5.20. What are parameters?

Parameters are the link between a subprogram code and the code calling the subprogram. Lot depends on how the parameters are passed to a subprogram.

5.21. What are Packages?

A package can be defined as a collection of related program objects such as procedures, functions, and associated cursors and variables together as a unit in the database. In simpler term, a package is a group of related procedures and functions stored together and sharing common variables, as well as local procedures and functions.

5.22. Mention any two advantages of Packages?

1. Stored packages allow you to sum up (group logically) related stored procedures, variables, and datatypes, and so forth in a single-named, stored unit in the database. This provides for better orderliness during the development process. In other words packages and its modules are easily understood because of their logical grouping.

2. Grouping of related procedures, functions, etc. in a package also make privilege management easier. Granting the privilege to use a package makes all components of the package accessible to the grantee.

5.23. Mention how exception handling is done in Oracle?

During execution of a PL/SQL block of code, Oracle executes every SQL sentence within the PL/SQL block. If an error occurs or an SQL sentence fails, Oracle considers this as an Exception. Oracle engine immediately tries to handle the exception and resolve it, by raising a built-in Exception handler.

5.24. Mention two advantages of using exceptions in Oracle?

1. Control over abnormal exits of executing programs on encountering error conditions, hence the behavior of application becomes more reliable.
2. In traditional error checking system, if same error is to be checked at several places, you are required to code the same error check at all those places. But with exception handling technique, you will write the exception for that particular error only once in the entire code. Whenever that type error occurs at any place in code, the exceptional handler will automatically raise the defined exception.

Database Design

Learning Objectives. This chapter deals with various phases in database design, objectives of database design, database design tools. The important concept in database design like functional dependency and normalization are also discussed in this chapter. After completing this chapter the reader should be familiar with the following concepts:

- Various phases in database design
- Database design tools
- Identify modification anomalies in tables
- Functional dependency, Armstrong's axioms
- Concept of normalization and different normal forms
- Denormalization

6.1 Introduction

Database design process integrates relevant data in such a manner that it can be processed through a mechanism for recording the facts. A database of an organization is an information repository that represents facts about the organization. It is manipulated by some software to incorporate the changes that take place in the organization. The database design is a complex process. The complexity arises mainly because of the identification of relationships among individual components and their representation for maintaining correct functionality are highly involved. The degree of complexity increases if there are many-to-many relationships among individual components. The process of database design usually requires a number of steps which are in Fig. 6.1.

Feasibility Study

When designing a database, the purpose for which the database is being designed must be clearly defined. In other words the objective of creating the database must be crystal clear.

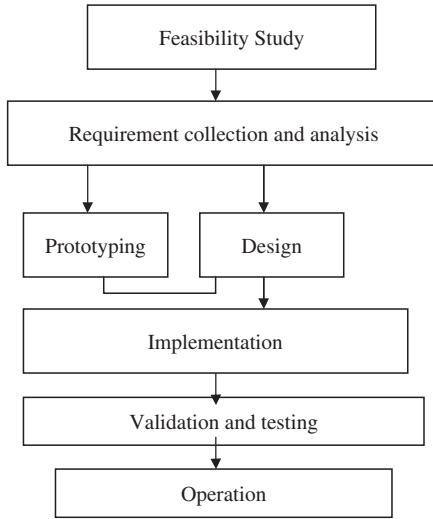


Fig. 6.1. Steps in database design

Requirement Collection and Analysis

In requirement collection, one has to decide what data are to be stored, and to some extent, how that data will be used. The people who are going to use the database must be interviewed repeatedly. Assumptions about the stated relationships between various parts of the data must be questioned again and again. For example, in designing the database about medical records of a patient, the following queries must be clearly defined.

Does a patient have more than one doctor? Is there a separate billing number for each drug ordered by a patient?

Prototyping and Design

Design implies a procedure for analyzing and organizing data into a form suitable to support business requirements and makes use of strategic technology. The three phases in relational database design are conceptual design, logical design, and physical design.

Implementation

Database implement involves development of code for database processing, and also the installation of new database contents, usually form existing data sources.

6.2 Objectives of Database Design

The objectives of database design vary from implementation to implementation. Some of the important factors like efficiency, integrity, privacy, security, implementability, flexibility have to be considered in the design of the database.

Efficiency

Efficiency is generally considered to be the most important. Given a piece of hardware on which the database will run and a piece of software (DBMS) to run it, the design should make full and efficient use of the facilities provided. If the database is made online, then the users should interact with the database without any time delay.

Integrity

The term integrity means that the database should be as accurate as possible. The problem of preserving the integrity of data in a database can be viewed at a number of levels. At a low level it concerns ensuring that the data are not corrupted by hardware or software errors. At a higher level, the problem of preserving database integrity concerns maintaining an accurate representation of the real world.

Privacy

The database should not allow unauthorized access to files. This is very important in the case of financial data. For example the bank balance of one customer should not be revealed to other customers.

Security

The database, once loaded, should be safe from physical corruption whether from hardware or software failure or from unauthorized access. This is a general requirement of most databases.

Implementation

The conceptual model should be simple and effective so that mapping from conceptual model to logical model is easy. Moreover while designing the database, care has to be taken such that application programs should interact effectively with the database.

Flexibility

The database should not be implemented in a rigid way that assumes the business will remain constant forever. Changes will occur and the database must be capable of responding readily to such change.

Other than the factors which were mentioned above, the design of the database should ensure that data redundancy is not there.

6.3 Database Design Tools

Once the objectives of the database design and the various steps in database design is known, it is essential to know the database design tools which are used to automate the task of designing a business system. Using automated design tools is the process of using a GUI tool to assist in the design of a database or database application. Many database design tools are available with a variety of features. The design tools are vendor-specific. CASE tools are software that provides automated support for some portion of the systems development process. Database drawing tools are used in enterprise modeling, conceptual data modeling, logical database design, and physical data modeling.

6.3.1 Need for Database Design Tool

The database design tools increase the overall productivity because the manual tasks are automated and less time is spent in performing tedious tasks and more time is spent in thinking about the actual design of the database. The quality of the end product is improved in using database design tools; because the design tool automates much of the design process as a result the time taken to design a database is reduced. As a result, more time is available to interview the customer, conduct user feedback sessions, and naturally the quality of the product is improved.

6.3.2 Desired Features of Database Design Tools

The database design tools should help the developer to complete the database model of database application in a timely fashion. Some of the features of the database design tools are given below:

- The database design tools should capture the user needs.
- The capability to model the flow of data in an organization.
- The database design tool should have the capability to model entities and their relationships.
- The database design tool should have the capability to generate Data Definition Language (DDL) to create database object.

- The database design tool should support full life cycle database support.
- Database and application version control.
- The database design tool should generate reports for documentation and user-feedback sessions.

6.3.3 Advantages of Database Design Tools

Some of the advantages of using database design tools for system design or application development are given as:

- The amount of code to be written is reduced as a result the database design time is reduced.
- Chances of errors because of manual work are reduced.
- Easy to convert the business model to working database model.
- Easy to ensure that all business requirements are met with.
- A higher quality, more accurate product is produced.

6.3.4 Disadvantages of Database Design Tools

Some of the disadvantages of database design tools are given below:

- More expenses involved for the tool itself.
- Developers might require special training to use the tool.

6.3.5 Commercial Database Design Tools

The database design tools which are commercially popular are given along with their websites.

1. CASE Studio 2 – Powerful database modeling, management, and reporting tool.
<http://www.casestudio.com/enu/default.aspx>
2. Design for Databases V3 – Database development tool using an entity relationship diagram.
<http://www.datanamic.com/dezign>
3. DBDesigner4 – Visual database design system that integrates database design, modeling.
4. ER/Studio – Multilevel data modeling application for logical and physical database design and construction.
<http://www.embarcadero.com/products/erstudio/index.html>
5. Happy Fish Database Designer – Visual database design tool supporting multiple database platforms. Happy Fish generates complete DDL scripts, defining metadata with table creates, indexes, foreign keys.
<http://www.embarcadero.com/products/erstudio/index.html>
6. Oracle Designer 2000 – Provides complete toolset to model, generate, and capture the requirements and design of enterprise applications.
<http://www.Oracle.com/technology/products/designer/index.html>

7. QDesigner – QDesigner is an enterprise modeling and design solution that empowers architects, DBAs, developers, and business analysts to produce IT solutions.
<http://www.quest.com/QDesigner>
8. Power designer – The PowerDesigner product family offers a modeling solution that analysts, DBAs, designers, and developers can tailor. Its modular structure offers affordability and expandability, so the tools can be applied according to the size and scope of the project.
<http://www.sybase.com/products/powerdesigner/>
9. Web Objects – A product from Apple. WebObject helps to develop and deploy enterprise-level web services and java server applications.
<http://www.apple.com/webobjects/>
10. xCase – Database design tools which provides datamodeling environment.
www.xcase.com

6.4 Redundancy and Data Anomaly

Redundant data means storing the same information more than once, i.e., redundant data could be removed without the loss of information. Redundancy can lead to anomalies. The different anomalies are insertion, deletion, and update anomalies.

6.4.1 Problems of Redundancy

Redundancy can cause problems during normal database operations. For example, when data are inserted into the database, the data must be duplicated wherever redundant versions of that data exist. Also when the data are updated, all redundant data must be simultaneously updated to reflect that change.

6.4.2 Insertion, Deletion, and Update Anomaly

A table anomaly is a structure for which a normal database operation cannot be executed without information loss or full search of the data table. The table anomaly can be broadly classified into (1) Insertion Anomaly, (2) Deletion Anomaly, and (3) Update or Modification Anomaly.

Example 1

Staff no.	Job	Dept. no.	Dept. name	City
100	sales man	10	sales	Trichy
101	manager	20	accounts	Coimbatore
102	clerk	30	accounts	Chennai
103	clerk	30	operations	Chennai

Insertion Anomaly

We cannot insert a department without inserting a member of staff that works in that department.

Update Anomaly

We could change the name of the department that “100” works in without simultaneously changing the department that “102” works.

Deletion Anomaly

By removing, employee 100, we have removed all information pertaining to the sales department.

Repeating Group

A repeating group is an attribute (or set of attributes) that can have more than one value for a primary key value.

To understand the concept of repeating group, consider the example of the table STAFF. A staff can have more than one contact number. For each contact number, we have to store the data of the STAFF which leads to more storage space (more memory).

STAFF					
Staff no.	Job	Dept. name	DeptID	City	Contact number
100	sales man	sales	01	Coimbatore	5434, 54221, 54241
101	manager	accounts	02	Chennai	56332, _____
102	clerk	accounts	03	Chennai	____, _____, _____
103	clerk	operations	04	Chennai	____, _____, _____

Repeating groups are not allowed in a relational design, since all attributes have to be atomic, i.e., there can only be one value per cell in a table.

6.5 Functional Dependency

Functional dependencies are the relationships among the attributes within a relation. Functional dependencies provide a formal mechanism to express constraints between attributes. If attribute A functionally depends on attribute B, then for every instance of B you will know the respective value of A. Attribute “B” is functionally dependent upon attribute “A” (or collection of attributes) if a value of “A” determines or single value of attributes “B” at only one time functional dependency helps to identify how attributes are related to each other.

(1) Notation of Functional Dependency

The notation of functional dependency is $A \longrightarrow B$.

The meaning of this notation is:

1. “A” determines “B”
2. “B” is functionally dependent on “A”
3. “A” is called determinant
“B” is called object of the determinant

Student ID \longrightarrow GPA. The meaning is the grade point average (GPA) can be determined if we know the student ID.

Let us consider another example of functional dependency,

Student ID	Name	GPA
------------	------	-----

Child \longrightarrow Mother

Every child has exactly one mother. The attribute mother is functionally dependent on the attribute child. If we specify a child, there is only one possible value for the mother. A functional dependency $A \longrightarrow B$ is said to be trivial if $B \subseteq A$.

(2) Compound Determinants

More than one attribute is necessary to determine another attribute in an entity, and then such a determinant is termed as composite determinant.

For example, the internal marks and the external marks scored by the student determine the grade of the student in a particular subject.

Internal mark, external mark \longrightarrow grade.

Since more than one attribute is necessary to determine the attribute grade it is an example of compound determinant.

(3) Full Functional Dependency

An attribute is fully functionally dependent on a second attribute if and only if it is functionally dependent on the second attribute but not on any subset of the second attribute.

(4) Partial Functional Dependency

This is the situation that exists if it is necessary to only use a subset of the attributes of the composite determinant to identify its object.

Roll No	Subject Number	Hall Number	Grade
---------	----------------	-------------	-------

Full Functional Dependency

The roll number and subject number determines the grade. It implies that a student may be interested in a particular subject; in that subject the grade secured by that student will be good. It is not necessary that the same student get good grade in all the subjects. Hence the grade depends on the subject number.

Roll No, Subject Number \rightarrow Grade

Partial Functional Dependency

With respect to examination schedule, it is not necessary that all the subjects should be held in the same examination hall. Hence hall number depends on both the subject number and the roll number. Hall number depends on subject number is only partial functional dependency because the hall number also depends on the roll number of the student.

Subject Number \rightarrow Hall Number

(5) Transitive Dependency

A transitive dependency exists when there is an intermediate functional dependency.

Notation

$A \rightarrow B$, $B \rightarrow C$, and if $A \rightarrow C$ then it can be stated that the transitive dependency exists.

$A \rightarrow B \rightarrow C$

Example 2

Consider the example of the relation STAFF. The attributes associated with the STAFF are Staff number which is unique to each staff, the designation of the staff like Manager, Deputy Manager, and Managing Director, etc. The last attribute is the salary associated with the staff.

STAFF		
STAFF NUMBER	DESIGNATION	SALARY

It is to be noted that the staff number determines the designation. The designation obviously determines the salary. For example the manager will get more salary than the deputy manager. On the other hand the staff number determines the salary.

STAFF NUMBER \rightarrow DESIGNATION

DESIGNATION \rightarrow SALARY

STAFF NUMBER \rightarrow SALARY

There is a transitive dependency between STAFF NUMBER and SALARY.

6.6 Functional Dependency Inference Rules (Armstrong's Axioms)

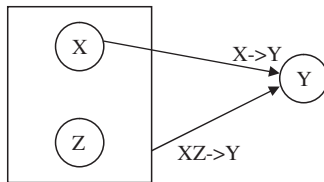
(1) Reflexivity

If $Y \subseteq X$ then, $X \rightarrow Y$. The axiom of reflexivity indicates that given a set of attributes the set itself functionally determines any of its own subsets.

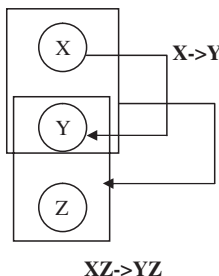
(2) Augmentation

If $X \rightarrow Y$ and Z is a subset of table R (i.e., Z is any set of attributes in R), then $XZ \rightarrow YZ$. The axiom of augmentation indicates that we can augment the left side of the functional dependency or both sides conveniently with one or more attributes. The axiom does not allow augmenting the right-hand side alone. The augmentation rule can be diagrammatically represented as follows:

If $X \rightarrow Y$ then $XZ \rightarrow Y$



A second variation of augmentation is diagrammatically shown below:



(3) Transitivity

If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$. The axiom of transitivity indicates that if one attribute uniquely determines a second attribute and this, in turn, uniquely determines a third one, then the first attribute determines the third one.

Consider three parallel lines X, Y, and Z. The line X is parallel to line Y. The line Y is parallel to line Z then it implies that line X is parallel to line Z. This property is called transitivity.

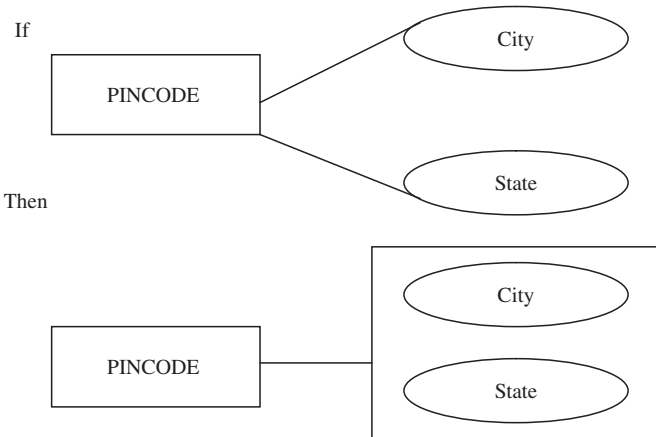
(4) Pseudotransitivity

If $X \rightarrow Y$ and $YW \rightarrow Z$ then $XW \rightarrow Z$. Transitivity is a special case of pseudotransitivity when W is null. The axiom of pseudotransitivity is a generalization of the transitivity axiom.

(5) Union

If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$. The axiom of union indicates that if there are two functional dependencies with the same determinant it is possible to form a new functional dependency that preserves the determinant and has its right-hand side the union of the right-hand sides of the two functional dependencies.

The union rule can be illustrated with the example of PINCODE. The PINCODE is used to identify city as well as PINCODE is used to identify state. This implies that PINCODE determines both city and state

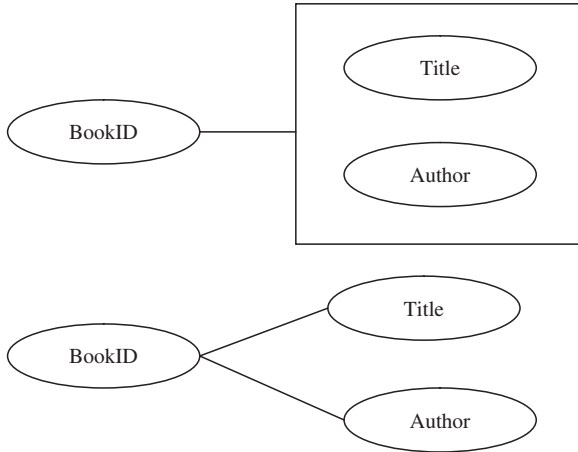


(6) Decomposition

If $X \rightarrow YZ$ then $X \rightarrow Y$ and $X \rightarrow Z$. The axiom of decomposition indicates that the determinant of any functional dependency can uniquely determine any

individual attribute or any combination of attributes of the right-hand side of the functional dependency.

The decomposition can be illustrated with an example of Book ID. The BookID determines the title and the author similar to $(X \rightarrow YZ)$ which implies that BookID determines title $(X \rightarrow Y)$ and BookID determines Author $(X \rightarrow Z)$



6.7 Closure of Set of Functional Dependencies

Given a set F of functional dependencies for a relation R , F^+ , the closure of F , be the set of all functional dependencies that are logically implied by F . Armstrong's axioms are sufficient to compute all of F^+ , which means if we apply Armstrong's rules repeatedly, then we can find all the functional dependencies in F^+ .

6.7.1 Closure of a Set of Attributes

Given a set of attributes A and a set of functional dependencies, the closure of the set of attributes A under F , written as A^+ , is the set of attributes B that can be derived from A by applying the inference axioms to the functional dependencies of F . The closure of A is always nonempty set because $A \rightarrow A$ by the axiom of reflexivity.

Algorithm for Determining the Closure of Attribute

The algorithm determines the closure of the attribute A which is denoted by A^+ , under a given set F of functional dependencies


```

I=0; A[0]=A;
REPEAT
    I=I+1;
    A[I] = A[I - 1];
    FOR ALL Z->W in F
        IF  $Z \subseteq A[I]$ 
            THEN  $A[I] = A[I] \cup W$ ;
    END FOR
UNTIL  $A[I] = A[I - 1]$ ;
RETURN  $A^+ = A[I]$ ;

```

In the above algorithm I is an integer. In the algorithm $A \rightarrow A[I]$ and after finding $Z \rightarrow W$ in F with $Z \subseteq A[I]$, $A[I]$ can be represented as YZ where $Y = A[I] - Z$. We can write $A \rightarrow A[I]$ as $A \rightarrow YZ$. Since F contains $Z \rightarrow W$, it can be concluded by set accumulation rule that $A \rightarrow YZW$, or in other words, $A \rightarrow A[I] \cup W$ and the induction hypothesis $A \rightarrow A[I]$ is maintained.

Covers

If F and G represents two sets of functional dependencies defined over the same relational scheme, F and G are equivalent if $F^+ = G^+$. Whenever $F^+ = G^+$, F covers G and vice versa.

Nonredundant cover

Consider two sets of functional dependencies F and G defined over the same relational scheme, if G covers F and no proper subset H of G is such that $H^+ = G^+$, then G is a nonredundant cover of F.

6.7.2 Minimal Cover

A set of nonredundant functional dependencies, which is obtained by removing all redundant functional dependencies using the functional dependency inference rule (Armstrong axiom), is termed as minimal cover.

Use of Functional Dependency

Functional dependency can be used to test relations to see if the relations are legal under a given set of functional dependencies. If a relation R is legal under a set F of functional dependencies, then the relation R satisfies F.

Functional dependency specifies constraints on the set of legal relations. F holds on R if all legal relations on R satisfy the set of functional dependencies of F.

6.8 Normalization

Normalization is the process of organizing data in a database. This includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating two factors: redundancy and inconsistent dependency. Redundant data wastes disk space and creates maintenance problems. If data that exists in more than one place must be changed, the data must be changed in exactly the same way in all locations. Inconsistent dependencies can make data difficult to access; the path to find the data may be missing.

Normalization is the analysis of functional dependencies between attributes. It is the process of decomposing relations with anomalies to produce well-structured relations. Well-structured relation contains minimal redundancy and allows insertion, modification, and deletion without errors or inconsistencies. Normalization is a formal process for deciding which attributes should be grouped together in a relation. It is the primary tool to validate and improve a logical design so that it satisfies certain constraints that avoid unnecessary duplication of data. Normalization theory is based on the concepts of normal forms. A relational table is said to be a particular normal form if it satisfied a certain set of constraints. There are currently five normal forms that have been defined. Normalization should remove redundancy but not at the expense of data integrity. In general, the normalization process generates many simple entity specifications from a few semantically complex entity specifications. Here entity specification refers to the declaration of entity attribute.

6.8.1 Purpose of Normalization

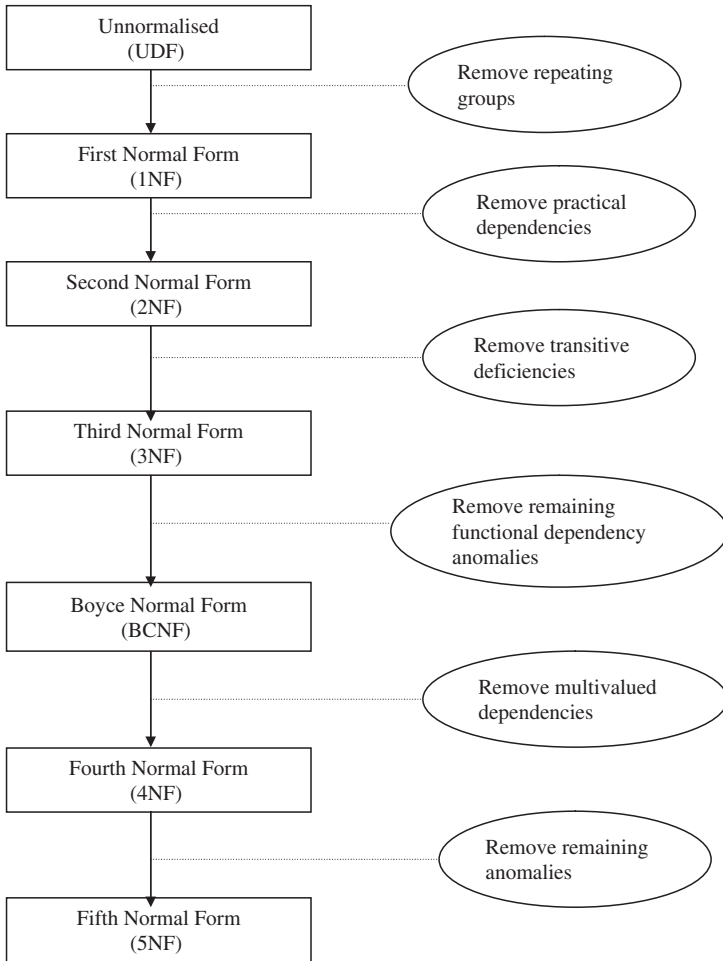
Normalization allows us to minimize insert, update, and delete anomalies and help maintain data consistency in the database.

1. To avoid redundancy by storing each fact within the database only once
2. To put data into the form that is more able to accurately accommodate change
3. To avoid certain updating “anomalies”
4. To facilitate the enforcement of data constraint
5. To avoid unnecessary coding. Extra programming in triggers, stored procedures can be required to handle the non-normalized data and this in turn can impair performance significantly.

6.9 Steps in Normalization

The degree of normalization is defined by normal forms. The normal forms in an increasing level of normalization, are first normal form (1NF), second normal form (2NF), 3NF, Boyce-Codd Normal form, 4NF and 5NF. Each normal

form is a set of conditions on a schema that guarantees certain properties relating to redundancy and update anomalies. In general 3NF is considered good enough. In certain instances, a lower level of normalization, that is the instance where queries take enormous time to execute.



Relational theory defines a number of structure conditions called normal forms that assure that certain data anomalies do not occur in a database.

First Normal Form (1NF)

A table is in first normal form (1NF) if and only if all columns contain only atomic values; that is, there are no repeating groups (columns) within a row. It is to be noted that all entries in a field must be of same kind and each field must have a unique name, but the order of the field (column) is irrelevant. Each record must be unique and the order of the rows is irrelevant.

Second Normal Form (2NF)

A table is in second normal form (2NF) if and only if it is in 1NF and every nonkey attribute is fully dependent on the primary key.

Third Normal Form (3NF)

To be in *Third Normal Form (3NF)* the relation must be in 2NF and no transitive dependencies may exist within the relation.

A *transitive dependency* is when an attribute is indirectly functionally dependent on the key (that is, the dependency is through another nonkey attribute).

Boyce–Codd Normal Form (BCNF)

To be in *Boyce–Codd Normal Form (BCNF)* the relation must be in 3NF and every determinant must be a candidate key.

Fifth Normal Form (5NF)

The *Fifth Normal Form* concerns dependencies that are obscure.

Domain/Key Normal Form (DK/NF)

To be in *Domain/Key Normal Form (DK/NF)* every constraint on the relation must be a logical consequence of the definition of keys and domains.

6.10 Unnormal Form to First Normal Form

Consider a table DEPARTMENT, the table DEPARTMENT is not in normal form because the table DEPARTMENT has repeating group. The table DEPARTMENT is shown in Table 6.1.

Table 6.1. DEPARTMENT (unnormalized form)

Department number	Department name	Location
1	Nilgiris	{Coimbatore, Chennai}
2	Subiksha	{Chennai, Tirunelveli}
3	Krishna	Trichy
4	Kannan	Coimbatore

Table 6.2. DEPARTMENT (first normal form)

Department number	Department name	Location1	Location2
1	Nilgiris	Coimbatore	Chennai
2	Subiksha	Chennai	Tirunelveli
3	Krishna	Trichy	
4	Kannan	Coimbatore	

Table 6.1 is not in normal form because the values are not atomic. The intersection of row with the column should have only one value. But in Table 6.1, the department location value is not atomic. That is the department Nilgiris is located in more than one location (Coimbatore, Chennai).

To convert Table 6.1 from unnormalized form into a normalized form, we have three different ways.

Solution 1

The column location in Table 6.1 is having more than one value. One way is to divide the column location into location1, location2 as shown in Table 6.2.

Drawback of Solution 1

The drawback of solution1 is that if a department is started in many places then more locations like location1, location2.locationN has to be included in the table DEPARTMENT. Moreover some departments will be in only one place, in such a case more NULL values will be there in the table DEPARTMENT.

Solution 2

The second solution is to insert tuples for each location as shown in Table 6.6.

Drawback of Solution 2

The main draw back of solution 2 is that there are more repeating values, hence more number of rows in the Table 6.3.

Solution 3

The third solution is to decompose the relation DEPARTMENT into two tables as shown in Tables 6.4 and 6.5.

Table 6.3. DEPARTMENT table

Department number	Department name	Location
1	Nilgiris	Coimbatore
1	Nilgiris	Chennai
2	Subiksha	Chennai
2	Subiksha	Tirunelveli
3	Krishna	Trichy
4	Kannan	Coimbatore

Table 6.4.

Department number	Department name
1	Nilgiris
2	Subiksha
3	Krishna
4	Kannan

Table 6.5.

Department number	Department name
1	Coimbatore
1	Chennai
2	Chennai
2	Tirunelveli
3	Trichy
4	Coimbatore

In the third solution we have divided the DEPARTMENT table into two tables. The process of splitting the table into more than one table is called normalization.

6.11 First Normal Form to Second Normal Form

Second Normal Form

A table is said to be in second normal form if it is in first normal form and all its nonkey attributes depend on all of the key (no partial dependencies).

Consider the relation EMPLOYEE_PROJECT, the relation EMPLOYEE_PROJECT consists of the attributes EmployeeID, Employee name, Project ID, Project name, Total hours. Here total hours imply the time taken to complete the project.

E_ID	E_NAME	P_ID	P_NAME	Total time
------	--------	------	--------	------------

EMPLOYEE_PROJECT

- E_ID stands for EmployeeID
- P_ID stands for ProjectID
- P_Name stands for Project name
- Total time is the time taken to complete the project

It is to be noted that the “Total Time” attribute depends on the nature of the project and the Employee. If the project is simple, then it can be completed easily and also if the employee is very talented then also the total time required to complete the project is less. Thus total time is determined by the EmployeeID and ProjectID. Clearly the relation EMPLOYEE_PROJECT is not in second normal form. The reason is we have to key attributes E_ID which refers to EmployeeID and P_ID which refers to Project ID. Then each other attribute in the relation EMPLOYEE_PROJECT should dependent on Employee ID alone, Project ID alone or both EmployeeID and ProjectID.

The relation EMPLOYEE_PROJECT can be transformed to second normal form by breaking the relation into two relations EMPLOYEE and HOURS_ASSIGNED.

EMPLOYEE(E_ID, E_NAME)

HOURS_ASSIGNED(E_ID, P_ID, TOTALTIME)

In this relation the attribute TOTAL TIME fully depends on the compositekey E_ID and P_ID.

6.12 Second Normal Form to Third Normal Form

Third Normal Form

A table is in third normal form if it is in second normal form and contains no transitive dependencies.

To understand transitive dependency, let us consider three attributes A, B, and C connected in such a way that $A \rightarrow B$ and $B \rightarrow C$. In other words $A \rightarrow C$. If we know the value of A, we can determine B, which we can use in turn to determine C. This kind of functional dependency is known as transitive dependency.

First let us consider a table HOSTEL which is in second normal form. The attributes of the table HOSTEL are Roll number, Building name, and Fee as shown in Table 6.6.

The table HOSTEL stores information about building in which a student’s room is located, and how much that student pays for the room. Here Student

Table 6.6. HOSTEL

Roll number	Building	Fee
100	main	600
101	additional	500
102	new	650

Table 6.7.

Roll number	Building
100	main
101	additional
102	new

Table 6.8.

Roll number	Building
main	600
additional	500
new	650

Roll number is the key for the table HOSTEL, since the other two columns depend on Student Roll number the table is in second normal form.

The table HOSTEL is not in third normal form because of transitive dependency. Roll Number \rightarrow Building, Building \rightarrow Fee which implies that Roll Number Fees. Because of this transitive dependency, the table is not in third normal form. The table HOSTEL is prone to modification anomalies, since removing the Roll Number 101 from the table HOSTEL also deletes the fact that a room in Additional building costs Rs. 500. The modification anomaly is due to transitive dependency.

Solution to Transitive Dependency

The solution to transitive dependency is to split the HOSTEL table into two as shown in Tables 6.7 and 6.8.

By splitting the table HOSTEL into two separate relations we can observe that the transitive dependency Roll Number \rightarrow Fees is avoided hence the table is in third normal form.

Example 3: Converting a Relation Which is in 2NF to 3NF

Consider a relation SALES which has the attributes CustomerID, Customer name, Sales person, and Region.

SALES (CUSTOMERID, CUSTOMERNAME, SALESPERSON, REGION)

In this relation SALES, the CUSTOMERID determines the CUSTOMERNAME, SALESPERSON, SALESPERSON, and REGION.

CUSTOMERID \longrightarrow CUSTOMERNAME

CUSTOMERID \longrightarrow SALESPERSON

CUSTOMERID \longrightarrow REGION

It is to be noted that SALESPERSON determines the REGION. SALESPERSON \longrightarrow REGION. Thus the relation SALES has transitive dependency which is shown by:



For a relation to be third normal form it has to be in second normal form and there should not be any transitive dependency. Hence the relation SALES has to be splitted into two relations SALES1 and SALES2 to remove transitive dependency.

SALES1 (CUSTOMERID, CUSTOMERNAME, SALESPERSON)

SALES2 (SALESPERSON, REGION)

Example 4: Converting a Relation which is in 2NF to 3NF

Consider a relation SUBJECT with the attributes SUBJECTID, SUBJECTNAME, LECTURER, and DEPARTMENT. The relation SUBJECT is in second normal form. SUBJECT (SUBJECTID, SUBJECTNAME, LECTURER, DEPARTMENT).

The relation SUBJECT has transitive dependency, because the SUBJECTID determines the LECTURER, LECTURER determines the DEPARTMENT. Also the SUBJECTID determines the DEPARTMENT as shown below.



To remove this transitive dependency the relation SUBJECT has to be decomposed into two relations SUBJECT and STAFF as shown below:

SUBJECT(SUBJECTID, SUBJECTNAME, LECTURER)

STAFF(LECTURER, DEPARTMENT)

By splitting the SUBJECT relation into two relations SUBJECT and STAFF, the transitive dependency between the attributes is avoided hence the relations SUBJECT and STAFF is in third normal form.

6.13 Boyce–Codd Normal Form (BCNF)

A relation R is in Boyce-Codd normal form (BCNF) if for every nontrivial functional dependency $X \rightarrow A$, X is a super key. In other words, a relation is in BCNF if and only if every determinant is a candidate key.

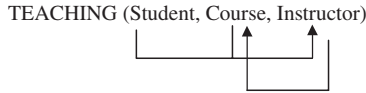
BCNF is a stronger form of normalization than 3NF because it eliminates the second condition for 3NF, which allows the right side of the functional dependency to be a prime attribute.

Third normal form to BCNF:

A relation is in BCNF if and only if every determinant is a candidate key.

Example 5: Converting a Relation to BCNF

Let us consider a relation **TEACHING** which has three attributes: Student, Course, and Instructor.



In the above relation **TEACHING**, Student determines the course (elective subject) which determines the instructor. Also the instructor determines the course which he has to handle. If an instructor is having a command in a particular subject, naturally he would like to handle the subject or course. The relation **TEACHING** can be transformed into BCNF by splitting the relation into two relations R_1 and R_2 .

R_1 (Instructor, Course) and R_2 (Instructor, Student). By splitting the relation **TEACHING** into two relations R_1 and R_2 we have transformed the relation **TEACHING** into BCNF because for the relation to be in BCNF all nonprime attributes must be fully dependent on every key. In the relation R_1 , the nonprime attribute course is fully dependent on the key attribute Instructor.

Example 6: Converting a Relation to BCNF

Consider the relation **ADDRESS** which has three attributes **STREET**, **CITY**, and **ZIP** (Pin code).

ADDRESS (**STREET**, **CITY**, **ZIP**)

ADDRESS		
STREET	CITY	ZIP

The relation **ADDRESS** is not in BCNF, the reason is **ZIP** is not a superkey.

From the relation ADDRESS we can infer that

$$\begin{array}{l} \{CITY, STREET\} \longrightarrow ZIP \\ ZIP \longrightarrow CITY \end{array}$$

The relation ADDRESS has insertion anomaly, that is a city of ZIP code cannot be stored if the street is not given. To overcome this insertion anomaly, the relation ADDRESS has to be split into two relations R1 and R2. The relation R1 has two attributes STREET, ZIP, and the relation R2 has two attributes ZIP, CITY.

R1(STREET, ZIP) and R2(ZIP, CITY). The splitting of the relation ADDRESS into two relations R1 and R2 eliminates insertion anomaly.

Example 7: Converting a Relation to BCNF

In this example, let us consider a relation which is in 3NF but not in BCNF. The relation which we are going to consider is R which has three attributes, PATIENT, DOCTOR, and HOSPITAL.

$$R\{PATIENT, DOCTOR, HOSPITAL\}$$

In this relation HOSPITAL

$$\begin{array}{l} \{PATIENT, HOSPITAL\} \longrightarrow DOCTOR \\ DOCTOR \longrightarrow HOSPITAL \end{array}$$

The relation R is not in BCNF because DOCTOR is not the superkey. To convert the relation R into BCNF, split the relation R into two relations R1 and R2 as shown below:

$$\begin{array}{l} R1\{PATIENT, DOCTOR\} \\ R2\{DOCTOR, HOSPITAL\} \end{array}$$

By splitting the relation R into two relations R1 and R2 we have converted the relation R which is in 3NF to BCNF.

BCNF and Third Normal Form

All BCNF are in 3NF but not all 3NF are in BCNF. BCNF does not make any reference to the concepts of full or partial dependency. BCNF is a stronger form of normalization than 3NF because it eliminates the second condition for 3NF, which allows the right side of the FD to be a prime attribute. Thus, every left side of a FD in a table must be a super key.

Multivalued Dependency

To understand multivalued dependency, consider a relation R which has three attributes: A, B, and C. For each value of A there is a set of values for B and set of values for C. However, the set of values for B and C are independent of each other, and then there exists multivalued dependency between the attributes A, B, and C in the relation R. It is represented by

$A \rightarrow B$ implies that for each value of A there is set of values for B.

$A \rightarrow C$ implies that for each value of A there is set of values for C.

If we have multivalued dependency in a relation we may have to repeat values redundantly in the tuples which is clearly undesirable.

Trivial Multivalued Dependency

Consider a relation R with two attributes A and B. The multivalued dependency between the attributes A and B is denoted by $A \twoheadrightarrow B$ is trivial if B is a subset of A or $A \cup B = R$.

Multivalued Dependency Inference Rules

The inference rules for multivalued dependency are given below:

Reflexivity

The axiom of reflexivity indicates that given a set of attributes the set itself functionally determines any of its own subsets. It is represented by

$$X \rightarrow X$$

Augmentation

The axiom of augmentation indicates that we can augment the left side of the functional dependency or both sides conveniently with one or more attributes. It is represented by

$$\text{If } X \rightarrow Y \text{ then } XZ \rightarrow Y$$

Transitivity

The axiom of transitivity indicates that if one attribute uniquely determines a second attribute and this, in turn, uniquely determines a third one, then the first attribute determines the third one. It is represented by

$$\text{If } X \rightarrow Y \text{ and } Y \rightarrow Z \text{ then } X \rightarrow Z$$

Pseudotransitivity

The axiom of pseudotransitivity is a generalization of the transitivity axiom. Transitivity is a special case of pseudotransitivity when W is null. Pseudotransitivity is represented by

$$\text{If } X \rightarrow Y \text{ and } YW \rightarrow Z \text{ then } XW \rightarrow Z$$

Union

The axiom of union indicates that if there are two functional dependencies with the same determinant it is possible to form a new functional dependency that preserves the determinant and has its right-hand side the union of the right-hand sides of the two functional dependencies. It is represented by

If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$

Decomposition

The axiom of decomposition indicates that the determinant of any functional dependency can uniquely determine any individual attribute or any combination of attributes of the right-hand side of the functional dependency. The decomposition axiom is represented by

If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow Y \cap Z$ and $X \rightarrow (Z - Y)$

6.14 Fourth and Fifth Normal Forms

Normal forms up to BCNF have been defined solely on functional dependency, and for most database practitioners, either 3NF or BCNF is a sufficient level of normalization. However, there are in fact two more normal forms that are needed to eliminate the rest of the currently known anomalies. If multivalued dependency and join dependency do not exist in a table, which is the most common situation, then any table in BCNF is automatically in fourth normal form (4NF) and fifth normal form (5NF) as well. However, when these constraints do exist, there may be further update anomalies that need to be corrected.

6.14.1 Fourth Normal Form

The goal of fourth normal form is to eliminate nontrivial multivalued dependencies from a table by projecting them onto separate smaller tables, thus eliminating the update anomalies associated with the multivalued dependencies. Under 4NF, a record type should not contain two or more independent multivalued facts about an entity.

Definition of Fourth Normal Form

A table R is in fourth normal form (4NF) if and only if it is in BCNF and, whenever there exists a multivalued dependency in R (for example $X \twoheadrightarrow Y$), at least one of the following holds: The multivalued dependency is trivial or X is a super key for relation R .

Example 8: Converting a Relation to Fourth Normal Form

Consider a relation SUBJECT which has the attributes COURSE, INSTRUCTOR, and TEXTBOOK

SUBJECT (COURSE, INSTRUCTOR, TEXTBOOK)

The relation SUBJECT is not in fourth normal form because of multivalued dependency between attributes.

COURSE \twoheadrightarrow INSTRUCTOR which implies that for one course there may be many instructors.

COURSE \twoheadrightarrow TEXTBOOK which implies that for a course that may be many textbooks.

Hence there exists multivalued dependency between attributes in SUBJECT relation. The relation SUBJECT can be converted to fourth normal form by splitting the relation SUBJECT into two relations TEACHER AND TEXT.

TEACHER (COURSE, INSTRUCTOR)

TEXT (COURSE, TEXTBOOK)

The relation TEACHER and TEXT is in fourth normal form.

Example 9: Converting a Relation to Fourth Normal Form

Consider the relation EMPLOYEE with the attributes employee number, project name, and department name as shown below:

EMPLOYEE (ENO, PNAME, DNAME)

where ENO stands for Employee number, PNAME for project name, and DNAME for department name.

The relation EMPLOYEE has the following multivalued dependencies:

ENO \twoheadrightarrow PNAME (One employee can work in several projects)

ENO \twoheadrightarrow DNAME.

ENO is not the superkey of the relation EMPLOYEE. To convert the relation to fourth normal form decompose EMPLOYEE relation into two relations

EMP_PROJ and EMP_DEPT as shown below.

EMP_PROJ (ENO, PNAME) and EMP_DEPT (ENO, DNAME)

Now the relations EMP_PROJ and EMP_DEPT are in fourth normal form.

Preferred Qualities of Decomposition

During normalization, the given relation is split-up into two or more relations. The splitting up of a given relation into two or more relations is known as decomposition. The decomposition should always satisfy the properties of lossless decomposition and dependency preservation

- Lossless decomposition ensures that the information in the original relation can be accurately reconstructed without spurious information.

- Dependency preservation ensures that the decomposed relations have the same capacity to represent the integrity constraints as the original relations and thus to reveal illegal updates.

Lossless-Join Decomposition

A property of a decomposition that ensures that no spurious rows are generated when relations are reunited through a natural join operation. A decomposition $\{R_1, R_2, \dots, R_n\}$ of a relation R is lossless decomposition if the natural join of R_1, R_2, \dots, R_n produces exactly the relation R . This is represented by

$$R = R_1 \bowtie R_2 \dots \bowtie R_n.$$

The decomposition of the relation R which has three attributes X, Y, Z that is $R(X, Y, Z)$ into $R_1(X, Y)$ and $R_2(Y, Z)$ is guaranteed to be nonloss if the attribute that is common to the two projections, Y in this case, has at least one of the two attributes dependent upon it. That is, if $Y \rightarrow X$, or $Y \rightarrow Z$, the decomposition is nonloss.

Lossless decomposition of a table implies that it can be decomposed by two or more projections, followed by a natural join of those projections that results in the original table, without any spurious or missing rows.

Example of Lossy Decomposition

Consider the relation $R(X, Y, Z)$ as shown below:

R(X, Y, Z)		
X	Y	Z
1	2	3
3	2	6
5	4	2

From the neither relation $R(X, Y, Z)$ it is clear that neither X nor Z is functionally depend on Y . Now the relation R is decomposed into two relations $R_1(X, Y)$ and $R_2(Y, Z)$ as shown below:

R1(X, Y)		R2(Y, Z)	
X	Y	Y	Z
1	2	2	3
3	2	2	6
5	4	4	2

Now the natural join of the relation R1 with the relation R2 is shown below:

X	Y	Z
1	2	3
1	2	6
3	2	3
3	2	6
5	4	2

Extra tuples

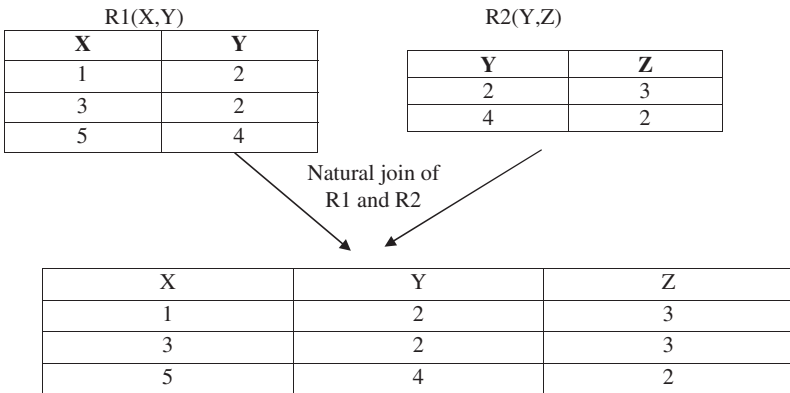
Example of Lossless Decomposition

Consider the relation R(X, Y, Z) as shown below:

X	Y	Z
1	2	3
3	2	3
5	4	2

From the relation R(X, Y, Z) it is clear that $Y \rightarrow Z$.

Now the relation R(X, Y, Z) is decomposed into two relations R1(X, Y) and R2(Y, Z) as shown below:



From the result of natural join of R1 with R2 it is clear that the decomposition is lossless due to the fact that $Y \rightarrow Z$.

6.14.2 Fifth Normal Form

A table R is in fifth normal form (5NF) or project-join normal form (PJ/NF) if and only if every join dependency in R is implied by the keys of R. In other words, a relation is in fifth normal form if it has no join dependency. Join dependency is the term used to indicate the property of a relation that can be decomposed losslessly into “m” simpler relations but cannot be decomposed losslessly into fewer relations.

Domain Key/Normal Form

In 1981 Fagin described a different approach to normalize tables when he proposed domain key/normal form. Domain key/normal form (DKNF) is based on three concepts domain key and constraint. We know that domain is a set of all values of the same datatype, a key is a unique identifier and constraint is a rule governing attribute values. A relation is in domain key/normal form if and only if every constraint on the relation is a logical consequence of the domain constraints and the key constraints that apply to the relation. Donald Fagin was the first person to devise a formal definition in 1981. Domain/key normal form is considered as the perfect normal form because of no insertion or deletion anomalies.

Disadvantages of Normalization

The disadvantage of normalization is that it produces many tables. A query might require retrieval of data from multiple normalized tables. This can result in complicated table joins. Decomposition of tables has two primary impacts. The first is performance. All the joins required to merge data will slow down the process.

6.15 Denormalization

Denormalization is used primarily to improve performance in cases where over-normalized structures are causing overhead to the query processor.

6.15.1 Basic Types of Denormalization

Five basic types of Denormalization are:

1. *Two entities in a many-to-many relationship.* The relationship table resulting from this construct is composed of the primary keys of each

of the associated entities. If we implement the join of this table with one of the entity tables as a single table instead of the original tables, we can avoid certain frequent joins that are based on both keys, but only the monkey data from one of the original entities.

2. *Two entities in a one-to-one relationship.* The table for these entities could be implemented as a single table, thus avoiding frequent joins required by certain applications.
3. *Reference data in a one-to-many relationship.* When artificial primary keys are introduced to tables that either have no primary keys or have keys that are very large composites, they can be added to the child entity in a one-to-many relationship as a foreign key and avoid certain joins in current applications.
4. *Entities with the most detailed data.* Multivalued attributes are usually implemented as entities and are thus represented as separate records in a table. Sometimes it is more efficient to implement them as individually named columns as an extension of the parent entity when the number of replications is a small fixed number for all instances of the parent entity.
5. *Derived attributes.* If one attribute is derived from another at execution time, then in some cases it is more efficient to store both the original value and the derived value directly in the database. This adds at least one extra column to the original table and avoids repetitive computation.

6.15.2 Table Denormalization Algorithm

The strategy for table Denormalization is to select only the most dominant process to determine those modifications that will most likely improve performance. The basic modification is to add attributes to existing tables to reduce join operations. The steps of strategy are as follows:

1. Select the dominant processes based on such criteria as high frequency of execution, high volume of data accessed, response time constraints, or explicit high priority. It can be considered as a rule of thumb as any process whose frequency of execution or data volume accessed is ten times that of another process is considered dominant.
2. Define join tables, when appropriate, for the dominant processes.
3. Evaluate total cost for storage, query, and update for the database schema, with and without the extended table, and determine which configuration minimizes total cost.
4. Consider also the possibility of Denormalization due to a join table and its side effects. If a join table schema appears to have lower storage and processing cost and insignificant side effects, then consider using that schema for physical design in addition to the original candidate table schema. Otherwise, use only the original schema.

In general, it is advisable to avoid joins based on nonkeys. They are likely to produce very large tables, thus greatly increasing storage and update costs.

Summary

This chapter introduced the various steps in database design. The concepts of functional dependency were discussed with suitable examples. The different types of functional dependency like full functional dependency, partial functional dependency, and transitive dependency were discussed with practical examples. This chapter also focused on the concept of normalization, which is very vital to minimize data redundancy. The different normal forms, like first normal form, second normal form, third normal form, BCNF, fourth normal form, fifth normal form, and Domain key normal form, conversion from one normal form to the other were discussed with suitable examples. Some of the drawbacks of normalization and its solution like denormalization were explained in this chapter.

Review Questions

6.1. Explain why the table given below is not in first normal form?

PERSON				
PERSON ID	PERSON ADDRESS	PERSON AGE	PERSON SALARY	PERSON CONTACT NUMBER
C100	12, Anna Nagar, Coimbatore	43	15,000	26185649, 23176247
C101	22, Peelamedu, Coimbatore	34	12,000	28976127
C102	15, Gandhipuram, Coimbatore	37	13,000	24379012, 21783251

Answer: The table PERSON is not in first normal form, because for the table to be in first normal form, the column value has to be atomic (only one value). Whereas in PERSON table, the column (or) attribute PERSON CONTACT NUMBER is not atomic (because a person can have more than contact number). Hence the table PERSON is not in first normal form.

6.2. Describe the purpose of normalizing the data? Describe the types of anomalies that may occur on a relation that has redundant data?

The purpose of normalization is given below:

1. To avoid redundancy by storing each fact with in the database only once
2. To put data into the form that is more able to accurately accommodate change

3. To avoid certain updating “anomalies”
4. To facilitate the enforcement of data constraint

The types of anomalies that may occur on a relation that has redundant data are

(a) Insertion, (b) Deletion, and (c) Update anomalies

6.3. Give an example of a relation which is in 3NF but not in BCNF? How will you convert that relation to BCNF?

Consider the example of the relation TEAM, which consists of entities Employee name, team name, and leader name as shown below:

TEAM		
Employee name	Team name	Leader name
Anand	Blue star	Rajan
Siva	Red star	Ramakrishnan
Anand	Green star	Ravi
Madhavan	Red star	Ramakrishnan

If Anand drops out of Green star team, then we have no record of Ravi leading the Green star team. This table has deletion anomaly. Even though the relation TEAM is in third normal form, it has deletion anomaly. To overcome the deletion anomaly, we have to split the table TEAM into two separate relations, TEAM1 with attributes Employee name and Team name and TEAM2 with attributes Team name and Leader name.

6.4. Show that every two attribute relation is in BCNF.

Let us consider a relation R with two attributes A and B that is R (A, B). If A is the sole key of the relation then the nontrivial dependency $A \rightarrow B$ has A as a superkey since $A \subset A$. On the other hand, if B is the sole key of the relation, then the nontrivial dependency $B \rightarrow A$ has B as a superkey since $B \subset B$. If both $A \rightarrow B$ and $B \rightarrow A$ simultaneously then whatever primary key we consider for the relation R we will have either A or B as its determinant. Hence every two attribute relation is in BCNF.

6.5. Given a relation R(S, B, C, D) with key={S, B, D} and F={S → C}. Identify the normal form of the relation R?

The relation R is in First normal form. The relation R is not in second normal form because the attribute C does not depend on the whole key. Hence the relation R is in first normal form.

6.6. Given a relation $R(S, B, C, D)$ with $\text{key}=\{S, B, D, CBD\}$ and $F=\{S \rightarrow C\}$. Identify the normal form of the relation R ?

The relation R is now in third normal form. The reason is C is now a key attribute. But the relation R is not in BCNF because the determinant is not the key.

6.7. A company obtains parts from a number of suppliers. Each supplier is located in one city. A city can have more than one supplier located there and each city has a status code associated with it. Each supplier may provide many parts. The company creates a simple relational table to store this information

FIRST (**s#**, **status**, **city**, **p#**, **qty**)

s# Supplier identification number
 status Status code assigned to city
 City City where supplier is located
 p# Part number of part supplied
 Qty Qty of parts supplied to date

Composite primary key is (s#, p#)

Identify in which normal form the table FIRST belongs to and normalize it to third normal form?

Solution: The table FIRST is shown by:

FIRST				
s#	city	status	p#	qty
s1	Chennai	20	p1	300
s1	Chennai	20	p2	100
s1	Chennai	20	p3	200
s1	Chennai	20	p4	100
s2	Delhi	10	p1	250
s2	Delhi	10	p3	100
s3	Mumbai	30	p2	300
s3	Mumbai	30	p4	200

Step 1 : First let us analyze whether the relation FIRST is in first normal form. For the relation FIRST to be in first normal form, all the values of the columns should be atomic. Since all the values of the column are atomic, the relation FIRST is atomic and no repeating values the relation FIRST is in first normal form.

Step 2 : Now let us analyze whether the relation FIRST is in second normal form. For the relation to be in second normal form, it should be in first normal form and every nonkey column is fully dependent upon the primary key.

The relation FIRST is in 1NF but not in 2NF because status and city are functionally dependent upon only on the column s# of the composite key (s#, p#).

To convert the relation FIRST into second normal form, we split the relation FIRST into two separate relations PARTS and SECOND as shown below:

PARTS		
s#	p#	qty
s1	p1	300
s1	p2	100
s1	p3	200
s1	p4	100
s2	p1	250
s2	p3	100
s3	p2	300
s3	p4	200

SUPPLIER		
s#	city	status
s1	Chennai	20
s2	Delhi	10
s3	Mumbai	30

Step 3 : Second normal form to third normal form:

For the relation to be in third normal form, the relation should be in second normal form and every nonkey column is nontransitively dependent upon its primary key.

The table supplier is in 2NF but not in 3NF because it contains a *transitive dependency*

SUPPLIER.s# \rightarrow SUPPLIER.city
 SUPPLIER.city \rightarrow SUPPLIER.status
 SUPPLIER.s# \rightarrow SUPPLIER.status

To convert the relation SUPPLIER into third normal form, we split the relation SUPPLIER into two relations SUPPLIER and CITY_STATUS as shown below to avoid transitive dependency.

SUPPLIER		CITY_STATUS	
s#	city	city	status
s1	Chennai	Chennai	20
s2	Delhi	Delhi	10
s3	Mumbai	Mumbai	30
s4	Pune	Madurai	50
s5	Madurai		

PARTS		
s#	p#	qty
s1	p1	300
s1	p2	100
s1	p3	200
s1	p4	100
s2	p1	250
s2	p3	100
s3	p2	300

Thus the given table is split up into three tables PARTS, SUPPLIER, and CITY_STATUS to convert the relation FIRST into third normal form.

Transaction Processing and Query Optimization

Learning Objectives. This chapter deals with various concepts in transaction processing. The ACID property that is necessary in transaction processing is discussed in detail. The anomalies in interleaved transactions like Write–Read Conflicts (WR Conflicts), Read–Write Conflicts (RW Conflicts), and Write–Write Conflicts (WW Conflicts) are illustrated with examples. This chapter also discusses different query evaluation plans in query optimization. This chapter throws light on advanced concept of query optimization using Genetic Algorithm. After completing this chapter the reader should be familiar with the following concepts.

- Principle of Transaction Management System
- Concept of Lock-Based Concurrency Control
- Dead Lock, Two Phase Locking Scheme
- Need for Query Optimization
- Query Optimizer Architecture
- Query Evaluation Plans
- Query Optimization Using Genetic Algorithm

7.1 Transaction Processing

7.1.1 Introduction

Managing Data is the critical role in each and every organization. To achieve the hike in business they need to manage data efficiently. DBMS provides a better environment to store and retrieve the data in an economical and efficient manner. User can store and retrieve data through various sets of instructions. These sets of instructions do several read and write operations on database. These processes are denoted by a special term “Transaction” in DBMS.

Transaction is the execution of user program in DBMS. It is different from the execution of the program external to DBMS. In other words it can be stated as the various read and write operations done by the user program on the DBMS, when it is executed in DBMS environment.

Transaction Management plays a crucial role in DBMS. It is responsible for the efficiency and consistency of the DBMS. Partial transaction let the database in an inconsistency state, so they should be avoided.

7.1.2 Key Notations in Transaction Management

The key notations in transaction management are as follows:

Object. The smallest Data item which is read or updated by the Transaction is called as Object in this case.

Transaction. Transaction is represented by the symbol T. It is termed as the execution of query in DBMS.

Read Operation. Read operation on particular object is notated by symbol R (object-name).

Write Operation. Write operation on particular object is notated by symbol W (object-name).

Commit. This term used to denote the successful completion of one Transaction.

Abort. This term used to denote the unsuccessful interrupted Transaction.

7.1.3 Concept of Transaction Management

User program executed by DBMS may claim several transactions. In the web environment, there is a possibility to several users' attempt to access the data stored in same database. To maintain the accuracy and the consistency of the database several scheduling algorithms are used.

To improve the effective throughput of the DBMS we need to enforce certain concurrent executions in DBMS. Transaction Manager is responsible for scheduling the Transactions and providing the safest path to complete the task. To maintain the data in the phase of concurrent access and system failure, DBMS need to ensure four important properties. These properties are called as ACID properties.

ACID Properties of DBMS

ACID is an acronym for Atomicity, Consistency, Isolation, and Durability.

A – Atomicity

C – Consistency

I – Isolation

D – Durability

Atomicity and Durability are closely related.

Consistency and Isolation are closely related.

The ACID properties are explained as follows.

Atomicity and Durability

Atomicity

Either all Transactions are carried out or none are. The meaning is the transaction cannot be subdivided, and hence, it must be processed in its entirety or not at all. Users should not have to worry about the effect of incomplete Transactions in case of any system crash occurs.

Transactions can be incomplete for three kinds of reasons:

1. *Transaction can be aborted, or terminated unsuccessfully.* This happens due to some anomalies arises during execution. If a transaction is aborted by the DBMS for some internal reason, it is automatically restarted and executed as new.
2. *Due to system crash.* This may be happen due to Power Supply failure while one or more Transactions in execution.
3. *Due to unexpected situations.* This may be happen due to unexpected data value or be unable to access some disk. So the transaction will decide to abort. (Terminate itself).

Durability

If the System crashes before the changes made by a completed Transaction are written to disk, then it should be remembered and restored during the system restart phase.

Partial Transaction

If the Transaction is interrupted in the middle way it leaves the database in the inconsistency state. These types of transactions are called as Partial Transactions.

Partial Transactions should be avoided to gain consistency of database. To undo the operations done by the Partial Transactions DBMS maintains certain log files. Each and every moment of disk writes are recorded in this log files before they are reflected to disk. These are used to undo the operations done when the system failure occurs.

Consistency and Isolation

Consistency

Users are responsible for ensuring transaction consistency. User who submits the transaction should make sure the transaction will leave the database in a consistent state.

Example 1

If the transaction of money between two accounts “A” and “B” is manually done by the user, then first thing he has to do is, he deducts the amount (say \$100) from the account “A” and add it with the account “B.” DBMS do not know whether the user subtracted the exact amount from account “B.”

User has to do it correctly. If the user subtracted \$99 from account “B” instead of \$100 DBMS is not responsible for that. This will leave DBMS in inconsistency state.

Isolation

In DBMS system, there are many transaction may be executed simultaneously. These transactions should be isolated to each other. One’s execution should not affect the execution of the other transactions. To enforce this concept DBMS has to maintain certain scheduling algorithms. One of the scheduling algorithms used is Serial Scheduling.

Serial Scheduling

In this scheduling method, transactions are executed one by one from the start to finish. An important technique used in this serial scheduling is interleaved execution.

Interleaved Execution

In DBMS to enforce concurrent Transactions, several Transactions are ordered in a serial manner and executed one by one according to the schedule. So there will be the switching over of execution between the Transactions. This is called as Interleaved Execution.

Example 2

The example for the serial schedule is illustrated in Fig. 7.1.

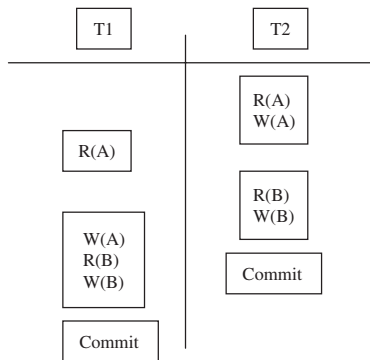


Fig. 7.1. Serial scheduling

Explanation

In the above example, two Transactions T1, T2 and two Objects A, B are taken into account. Commit denotes successful completion of both Transactions.

First one read and one write operation are done on the object A by Transaction T2. This is followed by T1. It does one write operation on object A. The same procedure followed by others for further. Finally both Transactions are ended successfully.

Anomalies due to Interleaved Transactions

If all the transactions in DBMS systems are doing read operation on the Database then no problem will arise. When the read and write operations done alternatively there is a possibility of some type of anomalies. These are classified into three categories.

1. Write–Read Conflicts (WR Conflicts)
2. Read–Write Conflicts (RW Conflicts)
3. Write–Write Conflicts (WW Conflicts)

*WR Conflicts***Dirty Read**

This happens when the Transaction T2 is trying to read the object A that has been modified by another Transaction T1, which has not yet completed (committed). This type read is called as dirty read.

Example 3

Consider two Transactions T1 and T2, each of which, run alone, preserves database consistency. T1 transfers \$200 from A to B, and T2 increments both A and B by 6%.

If the Transaction is scheduled as illustrated in Fig. 7.2.

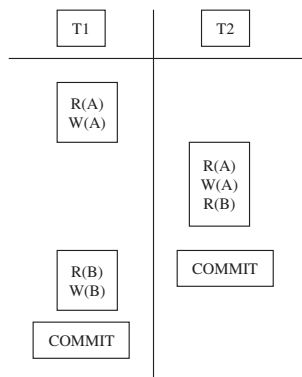


Fig. 7.2. Reading uncommitted data

Explanation

Suppose if the transactions are interleaved according to the above schedule then the account transfer program T1 deducts \$100 from account A, then the interest deposit program T2 reads the current values of accounts A and B and adds 6% interest to each, and then the account transfer program credits \$100 to account B. The outcome of this execution will be different from the normal execution like if the two instructions are executed one by one. This type of anomalies leaves the database in inconsistency state.

RW Conflicts

Unrepeatable Read
 In this case anomalous behavior could result is that a Transaction T2 could change the value of an object A that has been read by a Transaction T1, while T2 is still in progress. If T1 tries to read A again it will get different results. This type of read is called as Unrepeatable Read.

Example 4
 If “A” denotes an account. Consider two Transactions T1 and T2. Duty of T1 and T2 are reducing account A by \$100. Consider the following Serial Schedule as shown in Fig. 7.3.

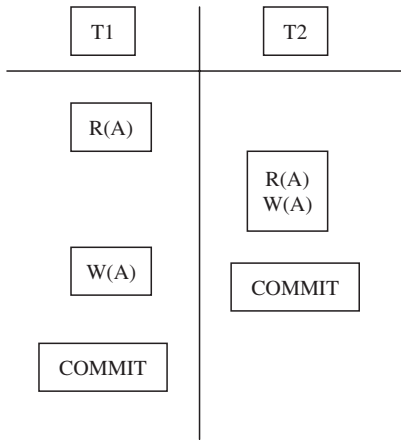


Fig. 7.3. RW conflicts

Explanation

At first, T1 checks whether the money in account A is more than \$100. Immediately it is interleaved and T2 also checks the account for money and

reduce it by \$100. After T2, T1 is tried to reduce the same account A. If the initial amount in A is \$101 then, after the execution of T2 only \$1 will remain in account A. Now T1 will try to reduce it by \$100. This makes the Database inconsistent.

WW Conflicts

The third type of anomalous behavior is that one Transaction is updating an object while another one is also in progress.

Example 5

Consider the two Transactions T1, T2.

Consider the two objects A, B.

Consider the following Serial Schedule as illustrated in Fig. 7.4.

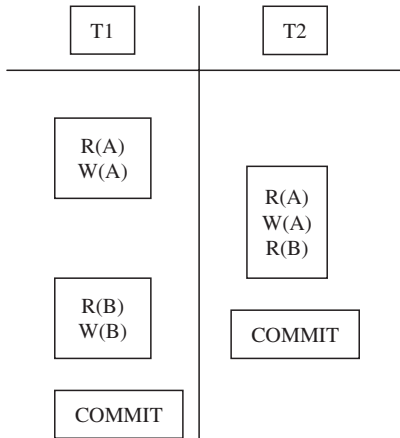


Fig. 7.4. WW conflicts

Explanation

If A and B are two accounts and their values have to be kept equal always, Transaction T1 updates both objects to 3,000 and T2 updates both objects to 2,000. At first T1 updates the value of object A to 3,000. Immediately T2 makes A as 2,000 and B as 2,000 and committed.

After the completion of T2, T1 updates B to 3,000. Now the value of A is 2,000 and value of B is 3,000, they are not equal. Constraint violated in this case due to serial scheduling.

Durability

Durable means the changes are permanent. Once a transaction is committed, no subsequent failure of the database can reverse the effect of the transaction.

7.1.4 Lock-Based Concurrency Control

Concurrency Control is the control on the Database and Transactions which are executed concurrently to ensure that each Transaction completed healthy. Concurrency control is concerned with preventing loss of data integrity due to interference between users in a multiuser environment.

Need for Concurrency Control

In database management system several transactions are executed simultaneously. In order to achieve concurrent transactions, the technique of interleaved execution is used. But in this technique there is a possibility for the occurrence of certain anomalies, due to the overriding of one transaction on the particular Database Object which is already referred by another Transaction.

Lock-Based Concurrency Control

It is the best method to control the concurrent access to the Database Objects by providing suitable permissions to the Transactions. Also it is the only method which takes less cost of Time and less program complexity in terms of code development.

Key Terms in Lock-Based Concurrency Control

Database Object

Database Object is the small data element, the value of which one is altered during the execution of transactions.

Lock

Lock is a small object associated with Database Object which gives the information about the type of operations allowed on a particular Database Object.

Lock can be termed as a type of permission provided by the transaction manager to the transactions to do a particular operation on a Database Object. The transaction must get this permission from Transaction Manager to access any Database Object for alteration. Locking mechanisms are the most common type of concurrency control mechanism. With locking, any data that is retrieved by a user for updating must be locked, or denied to other users, until the update is complete.

Locking Protocol

It is the set of rules to be followed by each transaction, to ensure that the net effect of execution of each Transaction in interleaved fashion will be same as,

the result obtained when the Transactions executed in serial fashion. Generally locks can be classified into two. First one is related to what already told in the previous paragraph. Next one is the unwanted effect when we implement lock of the first type.

The two types of Lock are:

1. Strict Two-Phase Locking (Strict 2PL)
2. Deadlock

Strict Two-Phase Locking (Strict 2PL)

It is a most widely used locking protocol. It provides few rules to the Transactions to access the Database Objects. They are:

Rule 1:

If a Transaction “T” wants to read, modify an object, it first requests a shared, exclusive lock on the Database Object respectively.

Rule 2:

All Locks held by the Transaction will be released when it is completed.

Shared Lock. It is type of lock established on a Database Object. It is like a component which is sharable within all active transactions. A Database Object can be shared locked by more than one number of transactions. To get a shared lock on particular Database Object the Database Object should satisfy the following condition.

Condition. It should not be exclusively locked by any of the other Transactions.

Example 6

If a person updates his bank account then the Database will lock that Database Object exclusively to avoid RW conflicts. So the Transactions which are requesting to read that Database Object will be suspended until the completion of updating.

Exclusive Lock. It is type of lock established on a Database Object. It is like a component which cannot be shared within all active Transactions. It is dedicated to particular transaction; only that particular transaction can access and modify that object.

Condition. It should not be exclusively locked by any one of the other Transactions.

Example 7

Assume the situation in reservation of Bus tickets in KPN Travels agencies. Assume number of tickets remain in bus no. 664 AV is only one. Two persons who are in different places are trying to reserve that particular ticket. See the situation here that only one of them should be allowed to access the Database while the other should wait until previous one is completed. Otherwise one terminal will check the number of seats available and due to interleaved actions next terminal will do the same and finally both of them will try to modify the Database (Decrement the seats available) this leads to more anomalies in Database and finally Database will be left into inconsistent state.

Deadlock

Deadlock occurs within the Transactions in DBMS system. Due to this neither one will be committed. This is the dead end to the execution of transactions. DBMS has to use suitable recovery systems to overcome Deadlocks.

Reasons for Deadlock Occurrence. Deadlock occurs mainly due to the Lock-Based Concurrency Control. The exclusive lock type will isolate one particular Database Object from the access of other transactions. This will suspend all the transactions who request Shared lock on that particular Database Object until the transaction which holds Exclusive lock on that object is completed. This will create a loop in Database which leads to Deadlock within transactions. This will leave the Database in inconsistent state.

Example 8

Assume, Transactions T1, T2, T3 as illustrated in Fig. 7.5. Database Objects are O1, O2, O3.

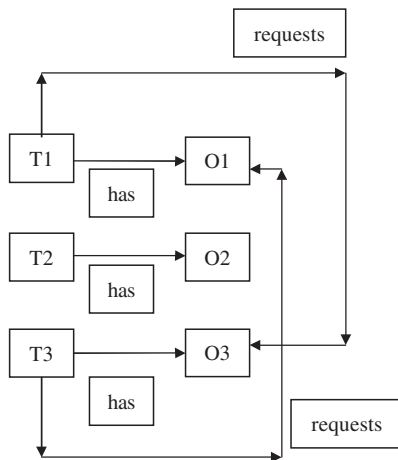


Fig. 7.5. Deadlock

Explanation

Here we can observe that the loop occurs between T1 and T3. Neither T1 nor T3 are completed.

Methods to Overcome Deadlock

Mostly it is not possible to avoid the occurrence of Deadlock. Most of the methods available are detection and recovery based.

Detection from the Average Waiting Time of the Transaction

If more than two transactions are waiting for the long time, then it implies that at some part of the database, deadlock has occurred. So we can detect Deadlock easily.

Deadlock Detection algorithm

Deadlock detection algorithms are used to find any loops in the Database. The main advantage is that we can find the locked transactions quickly and it is enough to restart only those particular transactions.

Recovery Mechanism

Once if Deadlock is found then we have several methods to release locked Transactions.

Method 1: Release of Objects from Big Transaction

In this method the transaction which holds more number of Database Object will be taken and all Database Objects associated with that Big Transaction will be removed.

Example 9

If Transaction say, T1 holds exclusive lock on four objects, T2 holds same on three objects and T3 holds same on two objects then if Deadlock occurred within these transactions then T1 will be restarted.

Method 2: Restarting the Locked Transactions

In this method Database Objects associated with all Transactions are released and they will be restarted.

Example 10

If Transaction say, T1 holds exclusive lock on four objects, T2 holds same on three objects and T3 holds same on two objects then if Deadlock occurred within these all Transactions are restarted.

Sample Deadlock Detection Program in “C” (Pseudocode)

The sample pseudocode for dead lock detection is as follows. The program flow and the process block for the dead lock detection are illustrated in Figs. 7.6 and 7.7, respectively.

```

/*  NTRANS = Number of Transactions
    NOREQUEST = Number of Objects Requested
    OALLOC = Object Allocated
    ORTT = Object Requested to Transaction
    OATT = Object Allocated to Transaction
*/
Now =0;
m = 0;
n = 0;
LOOPNO = 0;
    for (i = 0; i < NTRANS ; i++){
InnerLoop:    for (j = now ; j < NOREQUEST[i] ; j++) {
                if ( OALLOC[ORTT[i][j]] == TRUE){
                    for (k = 0 ; k < LOOPNO ; k++){
                        if (LOOP[k] == OATT[ORTT[i][j]]){
                            printf (“DEAD LOCK”);
                            goto end;}}
                    LOOP [LOOPNO] = i; JPUSH [m] = j; IPUSH [m] = I;
                    LOOPNO++; m++; i = OATT [ORTT[i][j]]; j = -1; }}
    if (m != 0){
        now = JPUSH [m-1] +1; i = IPUSH [m-1]; m -;
        LOOPNO -; goto InnerLoop;}}
[18pt] printf(“No Dead Lock”);
end:

```

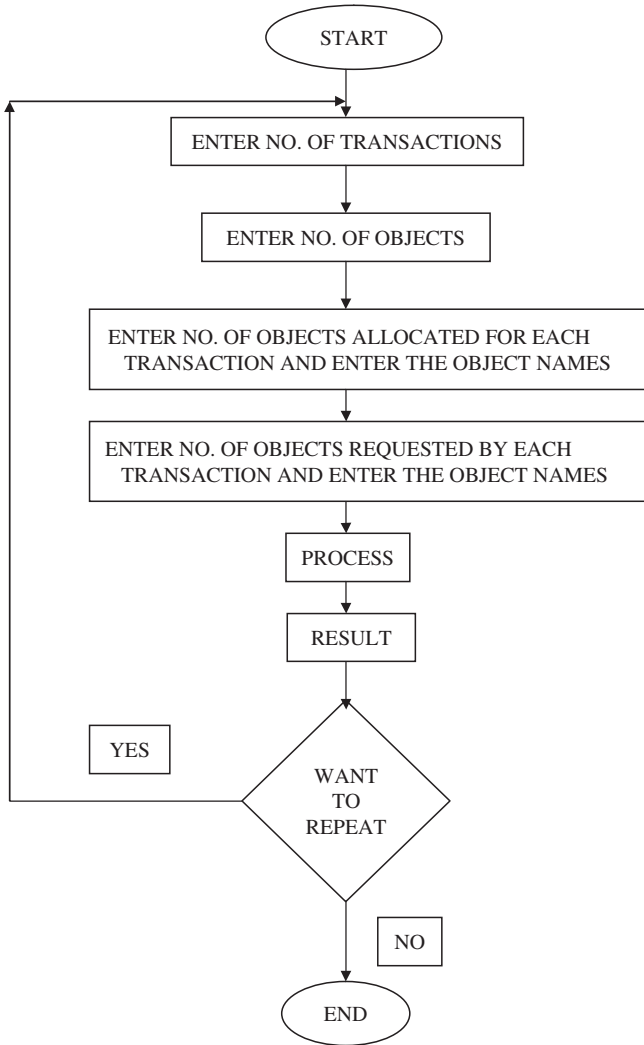


Fig. 7.6. Program flow

IN THE FLOW CHART:

NTR = No. of Transactions

NOR = No. of Objects Requested

OAL = Object Allocated

ORTT = Object Requested To Transaction

OATT = Object Allocated To Transaction

PROCESS BLOCK

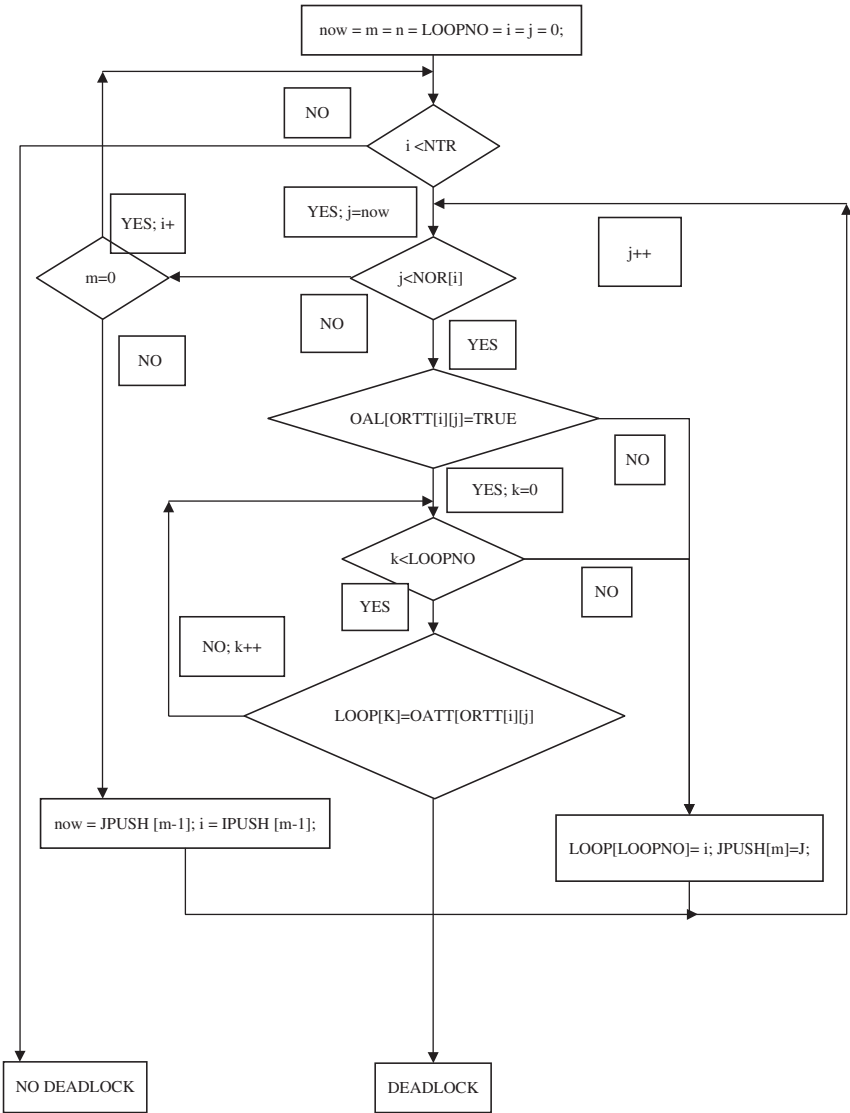


Fig. 7.7. Process block

7.2 Query Optimization

As we are in the comfortable age of information technology, databases have become a necessary and fundamental tool for managing and exploiting the power of information. Because the amount of data in a database grows larger

and larger as time passes, one of the most important characteristics of a database is its ability to maintain a consistent and acceptable level of performance. The principal mechanism through which a database maintains an optimal level of performance is known as the database query optimizer; without a well-designed query optimizer, even small databases would be noticeably sluggish. The query optimizers for some of the most popular commercial-quality databases are estimated to have required about 50 man-years of development. It should therefore go without saying that the specific processes involved in designing the internal structure of a real-world optimizer can be overwhelmingly complex. Nevertheless, because of the optimizer's paramount importance to the robustness and flexibility of a database, it is worthwhile to engage in a survey of the theory behind the rudimentary components of a basic, cost-based query optimizer.

7.2.1 Query Processing

The activities involved in retrieving data from the database are called as query processing. The aims of query processing are to transform a query written in a high-level language typically SQL, into a correct and efficient execution strategy expressed in a low-level language (implementing relational algebra), and to execute the strategy to retrieve the required data. An important aspect of query processing is Query Optimization.

The activity of choosing an efficient execution strategy for processing a query is called as query optimization. As there are many equivalent transformations of the same high-level query, the aim of query optimization is to choose the one that minimizes the resource usage.

A DBMS uses different techniques to process, optimize, and execute high-level queries (SQL). A query expressed in high-level query language must be first scanned, parsed, and validated.

The scanner identifies the language components (tokens) in the text of the query, while the parser checks the correctness of the query syntax. The query is also validated (by accessing the system catalog) whether the attribute names and relation names are valid. An internal representation (tree or graph) of the query is created.

Queries are parsed and then presented to a query optimizer, which is responsible for identifying an efficient plan. The optimizer generates alternative plans and chooses the plan with the least estimated cost.

7.2.2 Need for Query Optimization

In high-level query languages, any given query can be processed in different ways. Resources required by each query will be different.

DBMS has the responsibility to select the optimized way to process the query. Query optimizers do not “optimize” – just try to find “reasonably good” evaluation strategies. Query optimizer uses relational algebra expressions.

7.2.3 Basic Steps in Query Optimization

The two basic steps involved in query optimization are:

- Enumerating alternative plans for evaluating the expression. Because number of alternative plans are large.
- Estimating the cost of each enumerated plan and choosing the plan with least estimated cost.

Taking SQL query for query Optimization, when it is given as input to the following system of processes, the Query undergoes to the following stages as illustrated in Fig. 7.8.

As in Fig. 7.8, the DBMS begins by *parsing* the SQL statement. It breaks the statement into individual words, makes sure that the statement has a valid verb, legal clauses, and so on. Syntax errors and misspellings can be detected in this step. The DBMS *validates* the statement. It checks the statement against the system catalog. It checks whether all the tables referred exists in the database and their definition exists in the catalog.

The optimizer optimizes the statement. It explores various ways to carry out the statement. After exploring alternatives, it chooses one of them. The

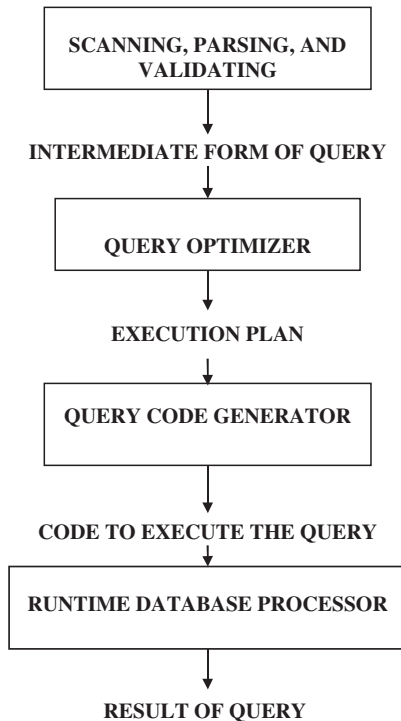


Fig. 7.8. Query processing steps

optimizer then generates an *execution plan* for the statement. The plan is a binary representation of the steps that are required to carry out the statement; it is the DBMS equivalent of “executable code.” It is carried out in Runtime Database Processor. Finally, the DBMS carries out the statement by executing the execution plan.

7.2.4 Query Optimizer Architecture

In the query optimizer architecture we provide an abstraction of the query optimization process in a DBMS. Given a database and a query on it, several execution plans exist that can be employed to answer the query. In principle, all the alternatives need to be considered so that the one with the best-estimated performance is chosen. An abstraction of the process of generating and testing these alternatives is shown in Fig. 7.9, which is essentially a modular architecture of a query optimizer. Based on the figure, the entire query optimization process involves two stages: Rewriting and Planning. There is only one module in the first stage, the Rewriter, whereas all other modules are in the second stage.

Module Functionality

The functionalities of each module in the Query optimizer are discussed in this section.

Rewriter

This module applies transformations to a given query and produces equivalent queries that are hopefully more efficient, e.g., replacement of views with their definition. The transformations performed by the Rewriter depend only on the declarative, i.e., static characteristics of queries do not take into account the actual query costs for the specific DBMS and database concerned. If the rewriting is known or assumed to always be beneficial, the original query is discarded; otherwise, it is sent to the next stage as well. By the nature of the rewriting transformations, this stage operates at the declarative level.

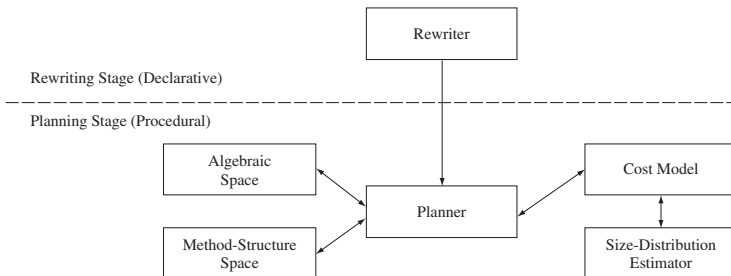


Fig. 7.9. Modular architecture of a query optimizer

Planner

This is the main module of the ordering stage. It examines all possible execution plans for each query produced in the previous stage and selects the overall cheapest one to be used to generate the answer of the original query. It employs a search strategy, which examines the space of execution plans in a particular fashion. This space is determined by two other modules of the optimizer, the Algebraic Space and the Method-Structure Space. For the most part, these two modules and the search strategy determine the cost, i.e., running time, of the optimizer itself, which should be as low as possible. The execution plans examined by the Planner are compared based on estimates of their cost so that the cheapest may be chosen. These costs are derived by the last two modules of the optimizer namely the Cost Model and the Size-Distribution Estimator.

Algebraic Space

This module determines the action execution orders that are to be considered by the Planner for each query sent to it. All such series of actions produce the same query answer, but usually differ in performance. They are usually represented in relational algebra as formulas or in tree form. Because of the algorithmic nature of the objects generated by this module and sent to the Planner, the overall planning stage is characterized as operating at the procedural level.

Method-Structure Space

This module determines the implementation choices that exist for the execution of each ordered series of actions specified by the Algebraic Space. This choice is related to the available join methods for each join (e.g., nested loops, merge scan, and hash join). This choice is also related to the available indices for accessing each relation, which is determined by the physical schema of each database stored in its catalogs. Given an algebraic formula or tree from the Algebraic Space, this module produces all corresponding complete execution plans, which specify the implementation of each algebraic operator and the use of any indices.

Cost Model

This module specifies the arithmetic formulas that are used to estimate the cost of execution plans. For every different join method, for every different index type access, and in general for every distinct kind of step that can be found in an execution plan, there is a formula that gives its cost. Given the complexity of many of these steps, most of these formulas are simple approximations of what the system actually does and are based on certain assumptions regarding issues like buffer management, disk-CPU overlap, sequential vs. random I/O, etc. The most important input parameters

to a formula are the size of the buffer pool used by the corresponding step, the sizes of relations or indices accessed, and possibly various distributions of values in these relations. While the first one is determined by the DBMS for each query, the other two are estimated by the Size-Distribution Estimator.

Size-Distribution Estimator

This module specifies how the sizes (and possibly frequency distributions of attribute values) of database relations and indices as well as (sub) query results are estimated. As mentioned above, these estimates are needed by the Cost Model. The specific estimation approach adopted in this module also determines the form of statistics that need to be maintained in the catalogs of each database, if any.

Detailed Description

This section provides a detailed description of the Algebraic Space, the Planner, and Size-Distribution Estimator, respectively.

Algebraic Space

SQL query corresponds to a select-project-join query in relational algebra. Typically, such an algebraic query is represented by a query tree whose leaves are database relations and nonleaf nodes are algebraic operators like selections (denoted by σ) projections (denoted by π), and joins (denoted by \bowtie). An intermediate node indicates the application of the corresponding operator on the relations generated by its children, the result of which is then sent further up. Thus, the edges of a tree represent data flow from bottom to top, i.e., from the leaves, which correspond to data in the database, to the root, which is the final operator producing the query answer. Figure 7.10 gives three examples of query trees for the query.

SELECT name, floor FROM emp, dept WHERE emp.dno = dept.dno AND sal > 100 K

For a complicated query, the number of all query trees may be enormous. To reduce the size of the space that the search strategy has to explore, DBMSs usually restrict the space in several ways. The first typical restriction deals with selections and projections:

R1 Selections and projections are processed on the “y” and almost never generate intermediate relations. Selections are processed as relations are accessed for the first time. Projections are processed as the results of other operators are generated.

For example, plan P1 of Sect. 7.1.1 satisfies restriction R1: the index scan of emp finds emp tuples that satisfy the selection on emp.sal on the “y”

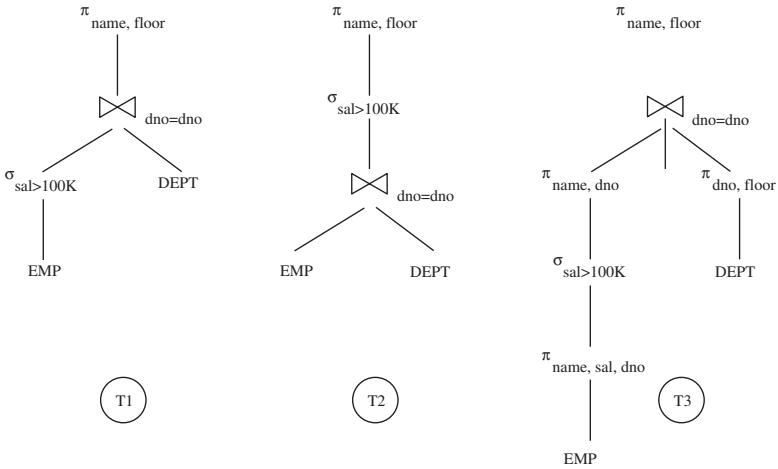


Fig. 7.10. Examples of general query trees

and attempts to join only those; furthermore, the projection on the result attributes occurs as the join tuples are generated. For queries with no join, R1 is root. For queries with joins, however, it implies that all operations are dealt with as part of join execution. Restriction R1 eliminates only suboptimal query trees, since separate processing of selections and projections incurs additional costs. Hence, the Algebraic Space module specifies alternative query trees with join operators only, selections and projections being implicit.

Given a set of relations to be combined in a query, the set of all alternative join trees is determined by two algebraic properties of join: commutativity ($R1 \bowtie R2 \equiv R2 \bowtie R1$) and associativity ($(R1 \bowtie R2) \bowtie R3 \equiv R1 \bowtie (R2 \bowtie R3)$). The first determines which relation will be inner and outer in the join execution. The second determines the order in which joins will be executed. Even with the R1 restriction, the alternative join trees that are generated by commutativity and associativity is very large, $(N!)$ for N relations. Thus, DBMSs usually further restrict the space that must be explored. In particular, the second typical restriction deals with cross products.

R2 Cross products are never formed, unless the query itself asks for them. Relations are combined always through joins in the query.

For example, consider the following query:

```

SELECT name, floor, balance
FROM emp, dept, acnt
WHERE emp.dno=dept.dno AND dept.ano=acnt.ano.
    
```

Figure 7.11 shows the three possible join trees (modulo join commutativity) that can be used to combine the emp, dept, and acnt relations to answer the query.

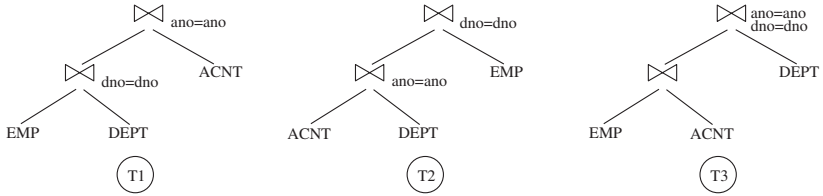


Fig. 7.11. Examples of join trees; T3 has a cross product

Of the three trees in the Fig. 7.11, tree T3 has a cross product, since its lower join involves relations emp and acnt, which are not explicitly joined in the query. Restriction R2 almost always eliminates suboptimal join trees, due to the large size of the results typically generated by cross products. The exceptions are very few and there are cases where the relations forming cross products are extremely small. Hence, the Algebraic Space module specifies alternative join trees that involve no cross product. The exclusion of unnecessary cross products reduces the size of the space to be explored, but that still remains very large. Although some systems restrict the space no further (e.g., Ingres and DB2-Client/Server), others require an even smaller space (e.g., DB2/MVS). In particular, the third typical restriction deals with the shape of join trees. *R3* The inner operand of each join is a database relation, never an intermediate result.

For example, consider the following query:

```
SELECT name, floor, balance, address
FROM emp, dept, acnt, bank
WHERE emp.dno=dept.dno AND dept.ano=acnt.ano AND
acnt.bno=bank.bno
```

Figure 7.12 shows three possible cross-product-free join trees that can be used to combine the emp, dept, acnt, and bank relations to answer the query. Tree T1 satisfies restriction R3, whereas trees T2 and T3 do not, since they have at least one join with an intermediate result as the inner relation. Because of their shape, join trees that satisfy restriction R3, e.g., tree T1, are called left-deep. Trees that have their outer relation always being a database relation, e.g., tree T2, are called right-deep. Trees with at least one join between two intermediate results, e.g., tree T3 is called bushy. Restriction R3 is of a more heuristic nature than R1 and R2 and may well eliminate the optimal plan in several cases. It has been claimed that most often the optimal left-deep tree is not much more expensive than the optimal tree overall. The two typical arguments used are:

- Having original database relations as inners increases the use of any pre-existing indices.
- Having intermediate relations as outers allows sequences of nested loops joins to be executed in a pipelined fashion.

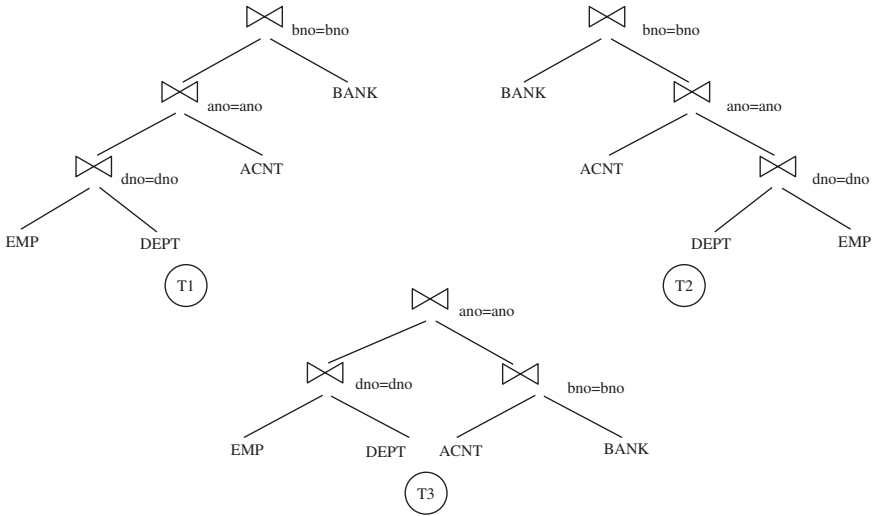


Fig. 7.12. Examples of left-deep (T1), right-deep (T2), and bushy (T3) join trees

Both index usage and pipelining reduce the cost of join trees. Moreover, restriction R3 significantly reduces the number of alternative join trees, to $O(2N)$ for many queries with N relations. Hence, the Algebraic Space module of the typical query optimizer only specifies join trees that are left-deep. In summary, typical query optimizers make restrictions R1, R2, and R3 to reduce the size of the space they explore.

Planner

The role of the Planner is to explore the set of alternative execution plans, as specified by the Algebraic Space and the Method-Structure space, and find the cheapest one, as determined by the Cost Model and the Size-Distribution Estimator. This section deals with different types of search strategies that the Planner may employ for its exploration. The first one focuses on the most important strategy, dynamic programming, which is the one used by essentially all commercial systems. The second one discusses a promising approach based on randomized algorithms, and the third one talks about other search strategies that have been proposed.

Size-Distribution Estimator

The final module of the query optimizer that we examine in detail is the Size-Distribution Estimator. Given a query, it estimates the sizes of the results of (sub) queries and the frequency distributions of values in attributes of these results.

7.2.5 Basic Algorithms for Executing Query Operations

A query basically consists of following operations. The query can be analyzed by analyzing these operations separately.

- Selection Operation
- Join Operation
- Projection Operation
- Set Operations

Select Operation

A query can have condition to select the required data. For that selection it may use several ways to search for the data. The following are some of the ways to search:

- File Scan
- Index Scan

File Scan

A number of search algorithms are possible for selecting the records from a file. The selection of records from a file is known as *file scan*, as the algorithm scans the records of a file to search for and retrieve records that satisfy the selection condition.

Index Scan

If the search algorithm uses an index to perform the search, then it is referred as *Index Scan*.

- *Linear Search*: This is also known as Brute Force method. In this method, every record in the file is retrieved and tested as to whether its attribute values satisfy the selection condition.
- *Binary Search*: If a selection condition involves an equality comparison on a key attribute on which the file is ordered, a binary search can be used. Binary search is more efficient than linear search.
- *Primary Index Single Record Retrieval*: If a selection condition involves an equality comparison on a primary key attribute with a primary index, we can use the primary index to retrieve that record.
- *Secondary Index*: This search method can be used to retrieve a single record if the indexing field has unique values (key field) or to retrieve multiple records if the indexing field is not a key. This can also be used for comparisons involving $<$, $>$, $<=$ or $>=$

- *Primary Index Multiple Records Retrieval*: A search condition uses comparison condition $<$, $>$, etc. on a key field with primary index is known as Primary index multiple records retrieval.
- *Clustering Index*: If the selection condition involves an equality comparison on a nonkey attribute with a clustering index, we can use that index to retrieve all the records satisfying the condition.

Conjunctive Condition Selection

If the selection condition of the SELECT operation is a conjunctive condition (a condition that is made up of several simple conditions with the AND operator), then the DBMS can use the following additional methods to perform the selection.

Conjunctive selection: If an attribute in any single simple condition in the conjunctive condition has an access path that permits use of binary search or an index search, use that condition to retrieve the records and then check if that record satisfies the remaining condition.

Conjunctive selection using composite index: If two or more attributes are involved in equality conditions in the conjunctive condition and a composite index exists on the combined fields we can use the index.

Query Optimization for Select Operation

Following method gives query optimization for selection operation:

1. If more than one of the attributes has an access path, then use the one that retrieves fewer disk blocks.
2. *Need to consider the selectivity of a condition*: Selectivity of a condition is the number of tuples that satisfy the condition divided by total number of tuples. The smaller the selectivity the fewer the number of tuples retrieved and higher the desirability of using that condition to retrieve the records.

Join Operation

Join is one of the most time-consuming operations in query processing. Here we consider only equijoin or natural join. Two-way join is a join of two relations, and there are many ways to perform the join. Multiway join is a join of more than two relations and number of ways to execute multiway joins increases rapidly with number of relations.

We use methods for implementing two-way joins of form

$$R \text{JN}(a = b)S$$

where, R and S are relations which need to be joined, a and b are attributes used for conditions, and $JN \rightarrow \text{Type of join}$.

Methods for Implementing Joins

Following are various methods to implement join operations.

Nested (Inner–Outer) Loop

For each record t in R (outer loop), retrieve every record s from S (inner loop) and test whether the two records satisfy the join condition $t[A] = s[B]$.

Access Structure Based Retrieval of Matching Records

If an index (or hash key) exists for one of the two join attributes – say, B of S – retrieve each record t in R, one at a time, and then use the access structure to retrieve directly all matching records from S that satisfy $s[B] = t[A]$.

Sort-merge join

If the records of R and S are physically sorted (ordered) by value of the join attributes A and B, respectively, then both the relations are scanned in order to the join attributes, matching the records that have same values for A and B. In this case, the relations R and S are only scanned once.

Hash-join

The tuples of relations R and S are both hashed to the same hash file, using the same hashing function on the join attributes A of R and B of S as hash keys. A single pass through the relation with fewer records (say, R) hashes its tuples to the hash file buckets. A single pass through the other relation (S) then hashes each of its tuples to the appropriate bucket, where the record is combined with all matching records from R.

In practice all the above techniques are implemented by accessing whole disk blocks of a relation, rather than individual records. Depending on the availability of buffer space in memory the number of blocks read in from the file can be adjusted. It is advantageous to use the relation with fewer blocks as the outer loop relation in nested loop method.

For method using access structures to retrieve matching tuples, either the smaller relation or the file that has a match for every record (high join selection factor) should be used in the outer loop. In some cases an index may be created specifically for performing the join operation if one does not exist already.

Sort-Merge algorithm is the most efficient, sometimes the relations are sorted before merging. Hash join method is efficient if the hash file can be kept in the main memory.

Projection Operation

If the projected list of attributes has the key of the relation, then the whole relation has to be scanned. If the projected list does not have key then duplicate tuples need to be eliminated, this is done by sorting or hashing.

7.2.6 Query Evaluation Plans

Query evaluation plan consists of an extended relational algebra tree, with additional information at each node indicating the access methods for each table and implementation method for relational operator.

Example:

Considering this query,

```

SELECT c.custname
FROM customer c, account a
WHERE c.custid = a.custid and a.balance > 5000;
    
```

This query can be expressed in relational algebra as,

$$\Pi_{\text{custname}} (\sigma(\text{customer.custid}=\text{account.custid}) \wedge \text{balance} > 5000 (\text{customer} \times \text{account})).$$

The relational algebra tree representation for the above query is illustrated in Fig. 7.13.

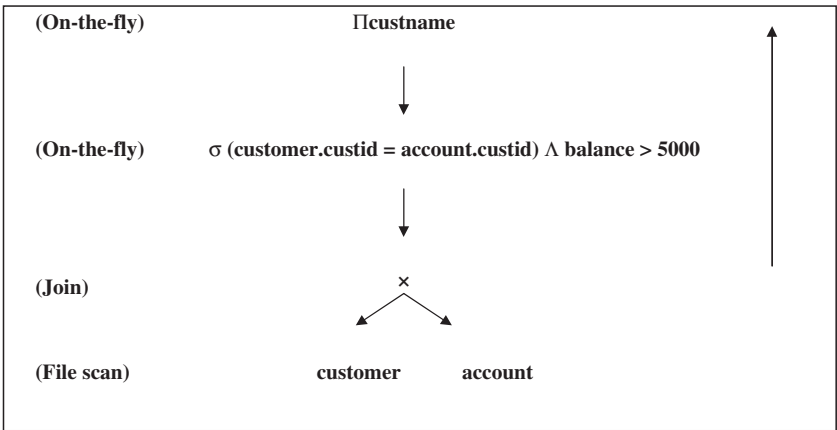


Fig. 7.13. Relational algebra tree representation

Pipelined Evaluation

From Fig. 7.13, we can visualize that each operation is carried out and inputted to the next section. In this type of evaluation is called as Pipelined evaluation which is more efficient than conventional method. In conventional method we use temporary memory to hold the evaluated operation and from that, memory next operation will be carried out.

So, pipelined evaluation has the advantage of not using temporary memory and this method is thus a control strategy.

On-the-fly

When the input table to a unary operator (for e.g., Selection or projection) is pipelined into it, which is referred as *on-the-fly*.

The Iterator Interface

To simplify the code responsible for coordinating the execution of a plan, the relational operators that form the nodes of a plan tree typically support uniform iterator interface, hiding internal implementation details of each operator.

The iterator interfaces for an operator includes the functions **open**, **get next**, **close**.

Open()

It initializes the state of the iterator by allocating buffers for its inputs and output.

It calls for the operator specific code to process the input tuples.

Close()

To deallocate the memory allocated during the process.

The iterator interface is also used to encapsulate access methods such as B+ trees and Hash-based indexes.

Pushing Selection Method

In this method, it is considered that the selection operator can be applied before the join operation to be performed. So, it is only necessary to join the selected tuples which reduces considerable memory requirement as shown in Fig. 7.14.

In our previous example we can apply the “balance” condition before join operation is to be applied. We push the selection before applying join operation.

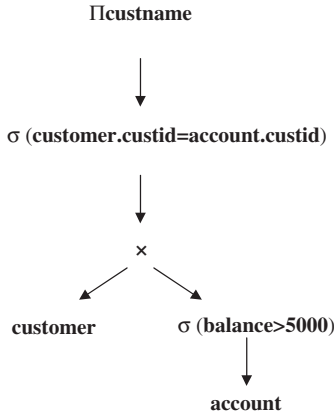


Fig. 7.14. Relational algebra tree representation

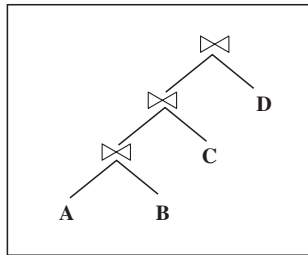


Fig. 7.15. Left-deep tree

Queries Over Multiple Relations

As the number of joins increases, the number of alternative plans grows rapidly; we need to restrict the search space. Left-deep trees have the advantages of reducing search space for the optimum strategy, and allowing the query optimizer to be based on dynamic processing techniques. Their main disadvantage is that in reducing the search space many alternative execution strategies are not considered, some of which may be of lower cost than the one found using the linear tree. As in Fig. 7.15, where A, B, C, and D are relations. Left-deep trees allow us to generate all fully pipelined plans. Intermediate results are not written to temporary files and not all left-deep trees are fully pipelined.

7.2.7 Optimization by Genetic Algorithms

Genetic Algorithms (GAs) are search techniques that are based on the mechanics of natural selection and genetics, involving a structured yet randomized information exchange resulting in a survival of the fittest amongst a population of string structures. The GA operates on a population of structures

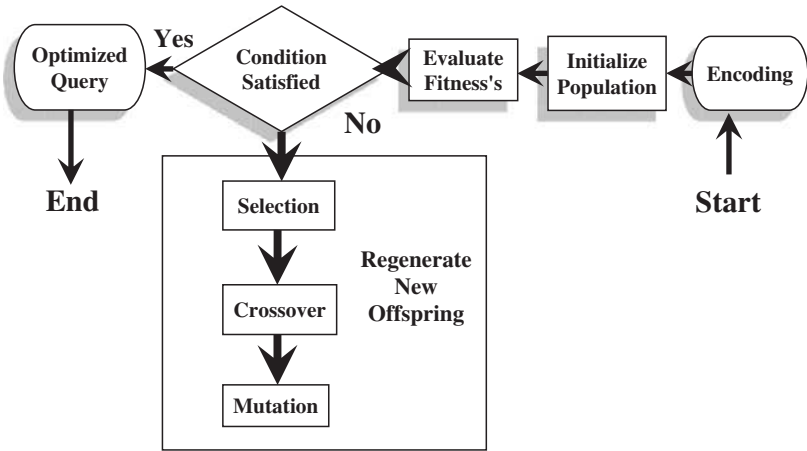


Fig. 7.16. Flowchart diagram

that are fixed length strings representing all possible solutions to a problem domain. A binary expression can be used to encode a parameter as a bit string. Using such a representation, an initial population is randomly generated. For each structure in the population, a fitness value is assigned. Each structure is then assigned a probability measure based on the fitness value that decides the contribution that structure would make to the next generation. This phase is called as the Reproduction phase as shown in Fig. 7.16. Each of the offspring generated by the reproduction phase is then modified using genetic operators of *Crossover and Mutation*.

In the Crossover operation, substrings of two individual strings selected randomly from the population are swapped resulting in new two strings. The Crossover operation is governed by a crossover probability. The Mutation operator generates a new string by independently modifying the values at each location of an existing string with a certain probability of mutation.

GA used Darwinian Evolution to extract optimization strategies nature and uses successfully and transforms them for application in mathematical optimization theory to find the global optimum in defined phase space. GA is used in Information Retrieval problems especially in optimizing a query. Selection, Fitness function, Crossover, and Mutation are the GA operators used for Query optimizer.

The GA can be represented by an 8-tuples as follows:

$$GA = \{P(0), \lambda, l, f, s, c, m, i\}$$

where,

$P(0)$ => Initial Population,

λ => Population Size,

l => Length of Each String,

f => Fitness Function,

s => Selection Operator,
 c => Crossover Operator,
 m => Mutation Operator,
 i => Inversion Operator.

Flowchart Description

This section deals with the operators of GA, which are Selection, Crossover, and Mutation.

Selection Operation

The selection string decides which of the strings in a population are selected for further genetic operations. Each string i of a population is assigned a fitness value f_i . The fitness value f_i is used to assign a probability value p_i to each string. The probability value p_i assigned to a string is calculated as

$$p_i = f_i / \sum f_i \quad [i = 1 \text{ to } \lambda]$$

Thus, from the above equation it can be seen that strings with a large fitness value have a large value of probability of selection. Using the probability distribution defined by above equation, strings are selected for further genetic operations.

Crossover Operation

This operation has GAs most of the exploratory power. The parameters defining the crossover operation are the probability of crossover (p_c) and the crossover point. The crossover operator works as follows:

From a population, two strings are drawn at random. If the crossover probability is satisfied, a crossover point is selected at random so as to lie between the defining length of a string, i.e., crossover point in 1 to $(L-1)$ th range. The substring to the left of the first string and to right of the second string is swapped to create a new string. A similar operation is performed with the two remaining substrings. Thus, two new substrings are generated from the parent string.

The operation is illustrated by means of an example given below:

Before Crossover

0 0 1 1 | 0 1 1

1 1 1 0 | 1 1 0

After Crossover

0 0 1 1 | 1 1 0

1 1 1 0 | 0 1 1

The usual value used for the crossover probability (P_c) lies between 0.6 and 0.8.

Mutation Operation

In GAs mutation is usually assigned as secondary role. It is primarily used as a background operator to guard against total premature loss. Use of the crossover operation by itself would not recover this loss. The mutation operator allows for this by changing the bit value at each locus with a certain probability. Thus, every locus on the binary string has a finite probability of assuming either a value of “0” or “1.”

The probability of this change is the defining parameter of the operation and is referred to as the probability of mutation (Pm) and is assigned a very small value of 0.001. The operation is explained below with an example:

Before Mutation
0 0 1 1 0 1 1
After Mutation
1 0 1 1 0 0 1

The bit values that have been affected by the mutation process are shown in italics. These operators form the basis for GA-based query optimization. As shown in Fig. 7.16, our query is encoded into any conventional modeling and then processed into the population step in which all the possible ways of obtaining results are produced. Each of the way is being checked for optimization. They are sent to the GAs Regeneration phase in which selection, crossover and mutation are processed. When the optimized one is found, required result can be obtained by the optimized query. Thus Genetic Algorithm can be used for query optimization effectively.

Summary

This chapter introduced Principle of Transaction Management System. The concept of Transaction Management System is discussed with suitable examples. ACID Properties of DBMS such as Atomicity, Durability, Consistency, and Isolation are discussed clearly. Importance of Crash Recovery and various methods for Crash Recovery is discussed with Examples. IBM’s System R Architecture is discussed clearly and various Query facilities such as Data Manipulation Facilities, Data Definition Facilities, and Data Control Facilities are described.

In this chapter we have presented the basic idea of query optimization and different query evaluation schemes. This chapter also gives the basic idea of the role of Genetic algorithm in query optimization.

Review Questions

7.1. What is Transaction?

Transaction is the execution of user program in DBMS. It is different from the execution of the program external to DBMS. In other words it can be

stated as the various read and write operations done by the user program on the DBMS, when it is executed in DBMS environment.

7.2. What are ACID properties of the DBMS?

ACID is an acronym for Atomicity, Consistency, Isolation, and Durability.

A – Atomicity

C – Consistency

I – Isolation

D – Durability

Atomicity and Durability are closely related. Consistency and Isolation are closely related.

7.3. What is strict 2PL? Explain its role in Lock-Based Concurrency Control.

It is a most widely used locking protocol.

It provides few rules to the Transactions to access the Database Objects.

They are:

Rule 1

If a Transaction say T wants to read, modify an object, it first requests a shared, exclusive lock on the Database Object, respectively.

Rule 2

All Locks held by the Transaction will be released when it is completed.

7.4. What is WR conflict?

This happens when the Transaction T2 is trying to read the object A that has been modified by another Transaction T1, which has not yet completed (committed). This type read is called as dirty read or Write read conflict.

7.5. What is Deadlock?

It is the lock that occurs within the Transactions in DBMS system. Due to this neither one will be committed. This is the dead end to the execution of Transactions. DBMS has to use suitable recovery systems to overcome Deadlocks.

7.6. Why Deadlock Occurs?

It occurs mainly due to the Lock Based Concurrency Control. In the exclusive lock type it will isolate one particular Database Object from the access to the other Transactions. This will suspend all the Transactions who request Shared lock on that particular Database Object until the Transaction which holds Exclusive lock on that object is completed. This will create a loop in Database. This leads to Deadlock within Transactions. This will leave the Database in inconsistent state.

7.7. Mention the three major methods used to handle Deadlock?

- (a) Deadlock Prevention
- (b) Deadlock Avoidance
- (c) Deadlock Detection

Deadlock prevention: Transaction aborts if there is a possibility of deadlock occurring. If the transaction aborts, it must be rollback and all locks it has are released.

Deadlock detection: DBMS occasionally checks for deadlock. If there is deadlock, it randomly picks one of the transactions to kill (i.e., rollback) and the other continues.

Deadlock avoidance: A transaction must obtain all its locks before it can begin so deadlock will never occur.

7.8. What is Interleaved Execution?

In DBMS to enforce concurrent Transactions, several Transactions are ordered in a serial manner and executed one by one according to the schedule. So there will be switching over of execution between the Transactions. This is called as Interleaved Execution.

7.9. What is Partial Transaction?

If the Transaction is interrupted in the middle way it leaves the database in the inconsistency state. These types of transactions are called as Partial Transactions.

7.10. What is Unrepeatable Read?

In this case anomalous behavior could result in that a transaction T2 could change the value of an object A that has been read by a Transaction T1, while T2 is still in progress. If T1 tries to read A again it will get different results. This type of read is called as Unrepeatable Read.

7.11. Find out whether the following system is in Deadlock or not?

Total Number of Processes 3 (P1, P2, P3)
 Total Number of Resources 3 (R1, R2, R3)
 P1 holds R1 and waiting for R2.
 P2 holds R2 and waiting for R3.
 P3 holds nothing and waiting for R3.

Answer: System is in safe state.

Since R3 is not assigned yet it can be assigned to P3 and it can be finished. After P3, P2, and P1 can be completed sequentially.

7.12. What is meant by query optimization?

The activity of choosing an efficient execution strategy for processing a query is called as query optimization. As there are many equivalent transformations of the same high-level query, the aim of query optimization is to choose the one that minimizes the resource usage.

7.13. What is the advantage of pipelined query evaluation?

Pipelined evaluation has the advantage of not using temporary memory and this method is thus a control strategy.

7.14. What is conjunctive condition selection?

A condition that is made up of several simple conditions with the AND operator can be termed as conjunctive condition selection.

7.15. Mention the pros and cons of Left-deep tree based query evaluation?

Left-deep trees have the advantages of reducing search space for the optimum strategy, and allowing the query optimizer to be based on dynamic processing techniques.

The main disadvantage is that in reducing the search space many alternative execution strategies are not considered, some of which may be of lower cost than the one found using the linear tree.

7.16. Illustrate the concept of crossover and mutation operation in Genetic Algorithm.

The operation of crossover is illustrated by means of an example as given below:

Before Crossover

0 0 1 1 | 0 1 1

1 1 1 0 | 1 1 0

After Crossover

0 0 1 1 | 1 1 0

1 1 1 0 | 0 1 1

Mutation is primarily used as a background operator to guard against total premature loss. The operation of mutation is explained below with an example:

Before Mutation

0 0 1 1 0 1 1

After Mutation

1 0 1 1 0 0 1

Database Security and Recovery

Learning Objectives. This chapter provides an overview of database security and recovery. In this chapter the need for database security, the classification of database security, and different types of database failures are discussed with suitable examples. Advanced concept in database recovery like ARIES algorithm is illustrated. After completing this chapter the reader should be familiar with the following concepts:

- Need for database security
- Classification of database security
- Database security at design and maintenance level
- Types of failure
- ARIES recovery algorithm

8.1 Database Security

8.1.1 Introduction

Database security issues are often lumped together with data integrity issues, but the two concepts are really quite distinct. Security refers to the protection of data against unauthorized disclosure, alteration, or destruction; integrity refers to the accuracy or validity of that data. To put it a little glibly:

- *Security* means protecting the data against unauthorized users.
- *Integrity* means protecting the data against authorized users.

In both cases, system needs to be aware of certain constraints that users must not violate; in both cases those constraints must be specified (typically by the DBA) in some suitable language, and must be maintained in the system catalog; and in both cases the DBMS must monitor user operations in order to ensure that the constraints are enforced. The main reason to clearly separate the discussion of the two topics is that integrity is regarded as absolutely fundamental but security as more of a secondary issue.

Data are the most valuable resource for an organization. Security in a database involves mechanisms to protect the data and ensure that it is not accessed, altered, or deleted without proper authorization. The database in Defense Research Development Organization (DRDO), Atomic Research Centre, and Space Research Centre contains vital data and it should not be revealed to unauthorized persons. To protect the secret data there should be restriction to data access. This ensures the confidentiality of the data. Also the data should be protected from accidental destruction. Due to advancement in information technology, people share the data through World Wide Web. As a result the data become vulnerable to hackers. A database should not only provide the end user with the data needed to function, but also it should provide protection for the data.

8.1.2 Need for Database Security

The need for database security is given below:

- In the case of shared data, multiple users try to access the data at the same time. In order to maintain the consistency of the data in the database, database security is needed.
- Due to the advancement of internet, data are accessed through World Wide Web, to protect the data against hackers, database security is needed.
- The plastic money (Credit card) is more popular. The money transaction has to be safe. More specialized software both to enter the system illegally to extract data and to analyze the information obtained is available. Hence, it is necessary to protect the data/money.

8.1.3 General Considerations

There are numerous aspects to the security problem, some of them are:

- Legal, social, and ethical aspects
- Physical controls
- Policy questions
- Operational problems
- Hardware control
- Operating system support
- Issues that are the specific concern of the database system itself

There are two broad approaches to data security. The approaches are known as discretionary and mandatory control, respectively. In both cases, the unit of data or “data object” that might need to be protected can range all the way from an entire database on the one hand to a specific component within a specific tuple on the other. How the two approaches differ is indicated by the following brief outline.

In the case of discretionary control, a given user will typically have different access rights (also known as privileges) on different objects; further, there are very few – inherent limitations, that is – regarding which users can have which rights on which object (for example, user U1 might be able to see A but not B, while user U2 might be able to see B but not A). Discretionary schemes are thus very flexible.

In the case of mandatory control, each data object is labeled with a certain classification level, and each user is given a certain clearance level. A given data object can then be accessed only by users with the appropriate clearance. Mandatory schemes thus tend to be hierarchic in nature and are hence comparatively rigid. (If user U1 can see A but not B, then the classification of B must be higher than that of A, and so no user U2 can see B but not A.)

Regardless of whether we are dealing with a discretionary scheme or a mandatory one, all decisions as to which users are allowed to perform which operations on which objects are policy decisions, not technical ones. As such, they are clearly outside the jurisdiction of the DBMS as such; all the DBMS can do is enforcing those decisions once they are made. It follows that:

- The results of policy decisions made must be known to the system (this is done by means of statement in some appropriate definitional language).
- There must be a means of checking a given access request against the applicable security constraint in the catalog. (By “access request” here we mean the combination of requested operation plus requested object plus requesting user, in general.) That checking is done by the DBMS’s security subsystem, also known as the authorization subsystem.

In order to decide which security constraints are applicable to a given access request, the system must be able to recognize the source of that request, i.e., it must be able to recognize the requesting user. For that reason, when users sign on to the system, they are typically required to supply, not only their user ID (to say who they are), but also a password (to prove they are who they say they are). The password is supposedly known only to the system and to legitimate users of the user ID concerned.

Regarding the last point, incidentally, note that any number of distinct users might be able to share the same ID. In this way the system can support user groups, and can thus provide a way of allowing everyone in the accounting department to share the same privileges on the same objects. The operations of adding individual users to or removing individual users from a given group can then be performed independently of the operation of specifying which privileges on which objects apply to that group. Note, however, that the obvious place to keep a record of which users are in which groups is once again the catalog (or perhaps the database itself).

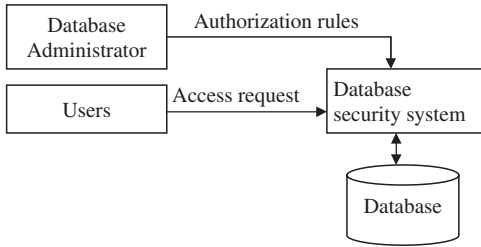


Fig. 8.1. Database security system

8.1.4 Database Security System

The person responsible for security of the database is usually database administrator (DBA). The database administrator must consider variety of potential threats to the system. Database administrators create authorization rules that define who can access what parts of database for what operations. Enforcement of authorization rules involves authenticating the user and ensuring that authorization rules are not violated by access requests. DBMS should support creation and storage of authorization rules and enforcement of authorization rules when users access a database. The database security system through the enforcement of authorization rules is shown in Fig. 8.1.

The database security system stores authorization rules and enforces them for each database access. The authorization rules define authorized users, allowable operations, and accessible parts of a database. When a group of users access the data in the database, then privileges can be assigned to groups rather than individual users. Users are assigned to groups and given passwords. In a nutshell, database security involves allowing and disallowing users from performing actions on the database and the objects within it. Database security is about controlling access to information. That is, some information should be available freely and other information should only be available to certain authorized people or groups.

8.1.5 Database Security Goals and Threats

Some of the goals and threats of database security are given below:

- *Goal.* Confidentiality (secrecy or privacy). Data are only accessible (read-type) by authorized subjects (users or processes).
- *Threat.* Improper release of information caused by reading of data through intentional or accidental access by improper users. This includes inferring of unauthorized data from authorized observations from data.
- *Goal.* To ensure data integrity which means data can only be modified by authorized subjects.
- *Threat.* Improper handling or modification of data.

- *Goal.* Availability (denial of service). Data are accessible to authorized subjects.
- *Threat.* Action could prevent subjects from accessing data for which they are authorized.

Security Threat Classification

Security threat can be broadly classified into accidental, intentional according to the way they occur.

The accidental threats include human errors, errors in software, and natural or accidental disasters:

- *Human errors* include giving incorrect input, incorrect use of applications.
- *Errors in software* include incorrect application of security policies, denial of access to authorized users.
- *Natural or accidental disasters* include the damage of hardware or software.

The intentional threat includes authorized users who abuse their privileges and authority, hostile agents like improper users executing improper reading or writing of data, legal use of applications can mask fraudulent purpose.

8.1.6 Classification of Database Security

The database security can be broadly classified into physical and logical security. Database recovery refers to the process of restoring database to a correct state in the event of a failure.

Physical security. Physical security refers to the security of the hardware associated with the system and the protection of the site where the computer resides. Natural events such as fire, floods, and earthquakes can be considered as some of the physical threats. It is advisable to have backup copies of databases in the face of massive disasters.

Logical security. Logical security refers to the security measures residing in the operating system or the DBMS designed to handle threats to the data. Logical security is far more difficult to accomplish.

Database Security at Design Level

It is necessary to take care of the database security at the stage of database design. Few guidelines to build the most secure system are:

1. The database design should be simple. If the database is simple and easier to use, then the possibility that the data being corrupted by the authorized user is less.

2. The database has to be normalized. The normalized database is almost free from update anomalies. It is harder to impose normalization on the relations after the database is in use. Hence, it is necessary to normalize the database at the design stage itself.
3. The designer of the database should decide the privilege for each group of users. If no privileges are assumed by any user, there is less likelihood that a user will be able to gain illegal access.
4. Create unique view for each user or group of users. Although “VIEW” promotes security by restricting user access to data, they are not adequate security measures, because unauthorized persons may gain knowledge of or access to a particular view.

Database Security at the Maintenance Level

Once the database is designed, the database administrator is playing a crucial role in the maintenance of the database. The security issues with respect to maintenance can be classified into:

1. Operating system issues and availability
2. Confidentiality and accountability through authorization rules
3. Encryption
4. Authentication schemes

(1) Operating System Issues and Availability

The system administrator normally takes care of the operating system security. The database administrator is playing a key role in the physical security issues. The operating system should verify that users and application programs attempting to access the system are authorized. Accounts and passwords for the entire database system are handled by the database administrator.

(2) Confidentiality and Accountability

Accountability means that the system does not allow illegal entry. Accountability is related to both prevention and detection of illegal actions. Accountability is assured by monitoring the authentication and authorization of users.

Authorization rules are controls incorporated in the data management system that restrict access to data and also restrict the actions that people may take when they access data.

Authentication can be carried out by the operating system level or by the relational database management system (RDBMS). In case, the system administrator or the database administrator creates for every user an individual account or username. In addition to these accounts, users are also assigned passwords.

(3) Encryption

Encryption can be used for highly sensitive data like financial data, military data. Encryption is the coding of data so that they cannot be read and understood easily. Some DBMS products include encryption routines that automatically encode sensitive data when they are stored or transmitted over communication channel. Any system that provides encryption facilities must also provide complementary routines for decoding the data. These decoding routines must be protected by adequate security, or else the advantage of encryption is lost.

(4) Authentication Schemes

Authentication schemes are the mechanisms that determine whether a user is who he or she claims to be. Authentication can be carried out at the operating system level or by the RDBMS. The database administrator creates for every user an individual account or user name. In addition to these accounts, users are also assigned passwords. A password is a sequence of characters, numbers, or a combination of both which is known only to the system and its legitimate user. Since the password is the first line of defense against unauthorized use by outsiders, it needs to be kept confidential by its legitimate user. It is highly recommended that users change their password frequently. The password needs to be hard to guess, but easy for the user to remember. Passwords cannot, of themselves, ensure the security of a computer and its databases, because they give no indication of who is trying to gain access.

The password can also be tapped; hence mere password cannot ensure the security of the database. To circumvent this problem, the industry is developing devices and techniques to positively identify any prospective user. The most promising of these appear to be biometric devices, which measure or detect personal characteristics such as fingerprints, voice prints, retina prints, or signature dynamics. To implement this approach, several companies have developed a smart card which is a thin plastic card with an embedded microprocessor. An individual's unique biometric data are stored permanently on the card. To access the database the user inserts the card and the biometric device reads the person's unique feature. The actual biometric data are then compared with the stored data, and the two must match for the user to gain computer access. A lost or stolen card would be useless to another person, since biometric data would not match.

Database Security Through Access Control

A database for an enterprise contains a great deal of information and usually has several groups of users. Most users need to access only a small portion of the database which is allocated to them. Allowing users unrestricted access

to all the data can be undesirable, and a DBMS should provide mechanisms to access the data. Especially, it is a way to control the data accessible by a given user.

Two main mechanisms of access control at the DBMS level are:

- Discretionary access control
- Mandatory access control

In fact it would be more accurate to say that most systems support discretionary control, and some systems support mandatory control as well; discretionary control is thus more likely to be encountered in practice, and so we deal with it first.

Discretionary Access Control

Discretionary access control regulates all user access to named objects through privileges, based on the concept of access rights or privileges for objects (tables and views), and mechanisms for giving users' privileges (and revoking privileges). A privilege allows a user to access some data object in a manner (to read or modify). Creator of a table or a view automatically gets all privileges on it. DBMS keeps track of who subsequently gains and loses privileges, and ensures that only requests from users who have the necessary privileges (at the time the request is issued) are allowed.

There needs to be a language that supports the definition of (discretionary) security constraints. For fairly obvious reasons, however, it is easier to state what is allowed rather than what is not allowed; languages therefore typically support the definition, not of security constraints as such, but rather of authorities, which are effectively the opposite of security constraints (if something is authorized, it is not constrained). We therefore begin by briefly describing a language for defining authorities with a simple example:

AUTHORITY SA3

```
GRANT RETRIEVE (S#, SNAME, CITY), DELETE  
ON S  
TO Jim, Fred, Mary;
```

This example is intended to illustrate the point that (in general) authorities have four components, as follows:

1. A name (SA3 – “suppliers authority three” – in the example). The authority will be registered in the catalog under this name.
2. One or more privileges (RETRIEVE – on certain attributes only – and DELETE, in the example), specified by means of the GRANT clause.
3. The relvar to which the authority applies (relvar S in the example), specified by means of the ON clause.

4. One or more “users” (more accurately, user IDs) who are to be granted the specified privileges over the specified relvar, specified by means of the TO clause.

Here is the general syntax:

```
AUTHORITY  <authority name>
GRANT      <privilege commalist>
ON         <relvar name>
TO         <user ID commalist>;
```

Explanation The <authority name>, <relvar name>, and <user ID commalist> are self-explanatory (except that we regard ALL, meaning all known users, as a legal “user ID” in this context). Each <privilege> is one of the following:

```
RETRIEVE   [( <attribute name commalist> )]
INSERT     [( <attribute name commalist> )]
UPDATE     [( <attribute name commalist> )]
ALL
```

RETRIEVE (unqualified), INSERT (unqualified), UPDATE (unqualified), and DELETE are self-explanatory. If a commalist of attribute names is specified with RETRIEVE, then the privilege applies only to the attributes specified; INSERT and UPDATE with a commalist of attribute names are defined analogously. The specification ALL is shorthand for all privileges: RETRIEVE (all attributes), INSERT (all attributes), UPDATE (all attributes), and DELETE.

Note. For simplicity, we ignore the question of whether any special privileges are required in order to perform general relational assignment operations. Also, we deliberately limit our attention only to data manipulation operations; in practice, of course, there are many other operations that we would want, to be subject to authorization checking as well, such as the operations of defining and dropping relvars and the operations of defining and dropping authorities themselves. We omit detailed consideration of such operations here which is beyond the scope of this book.

What should happen if some user attempts some operation on some object for which he or she is not authorized? The simplest option is obviously just to reject the attempt (and to provide suitable diagnostic information, of course); such a response will surely be the one most commonly required in practice. So we might as well make it as default. In more sensitive situations, however, some other action might be more appropriate; for example, it might be necessary to terminate the program or lock the user’s keyboard. It might also be desirable to record such attempts in a special log (threat monitoring), in order to permit subsequent analysis of attempted security breaches and also to serve in itself as a deterrent against illegal infiltration (see the discussion of audit trails at the end of this section).

Of course, we also need a way of dropping authorities:

DROP AUTHORITY <authority name>;

For example:

DROP AUTHORITY SA3;

For simplicity, we assume that dropping a given relvar will automatically drop any authorities that apply to that relvar.

Here are some further examples of authorities, most of them are fairly self-explanatory.

1. AUTHORITY EX1

```
GRANT RETRIEVE (P#, PNAME, WEIGHT)
ON P
TO Jacques, Anne, Charley;
```

Users Jacques, Anne, and Charley can see a “vertical subset” of base relvar P. This is an example of a value-independent authority.

2. AUTHORITY EX2

```
GRANT RETRIEVE, UPDATE (SNAME, STATUS),
DELETE
ON LS
TO Dan, Misha;
```

Relvar LS here is a view. Users Dan and Misha can thus see a “horizontal subset” of base relvar S. This is an example of a value-dependent authority. Note too that although users Dan and Misha can DELETE certain supplier tuples (via view LS), they cannot INSERT them, and they cannot UPDATE attributes S# or CITY.

3. VAR SSPR VIEW

```
(S JOIN SP JOIN (P WHERE CITY = 'Rome') {P#})
{ALL BUT P#, QTY};
```

AUTHORITY EX3

```
GRANT RETRIEVE
ON SSPR
TO Giovanni;
```

This is another value-dependent example. User Giovanni can retrieve supplier information, but only for suppliers who supply some stored in Rome.

4. VAR SSQ VIEW

```
SUMMARIZE SP PER S {S#} ADD SUM {QTY}
AS SQ;
```

AUTHORITY EX4

```

GRANT    RETRIEVE
ON       SSQ
TO       Fidel;

```

User Fidel can see total shipment quantities per supplier, but not individual shipment quantities. User Fidel thus sees a statistical summary of the underlying base data.

5. AUTHORITY EX5

```

GRANT RETRIEVE, UPDATE (STATUS)
ON     S
WHEN   DAY () IN ('Mon', 'Tue', 'Wed', 'Thu', 'Fri')
        AND NOW () >=TIME '09:00:00'
        AND NOW () >=TIME '17:00:00'
TO     Purchasing;

```

Here, we are extending our **AUTHORITY** syntax to include a **WHEN** clause to specify certain “context controls”; we are also assuming that the system provides two niladic operators – i.e., operators that take no operands – called **DAY ()** and **NOW ()**, with the obvious interpretations. Authority EX5 guarantees that supplier status values can be changed by the user “Purchasing” (presumably meaning anyone in the purchasing department) only on a weekday, and only during working hours. This is an example of context-dependent authority, because a given access request will or will not be allowed depending on the context – here the combination of day of the week and time of day – in which it is issued.

Other examples of built-in operators, that the system probably ought to support anyway and could be useful for context-dependent authorities, include:

```

TODAY () value = the current date
USER () value  = the ID of the current user
TERMINAL value = the ID of the originating terminal for the current
                request

```

By conceptually speaking, authorities are all “ORed” together. In other words, a given access request (meaning, to repeat, the combination of requested operation plus requested object plus requesting user) is acceptable if and only if at least one authority permits it. Note, however, that (for example) if one authority lets user Nancy retrieve part colors and another lets her retrieve part weights, it does not follow that she can retrieve part colors and weights together (a separate authority is required for the combination).

Finally, we have implied, but never quite said as much, that users can do only the things they are explicitly allowed to do by the defined authorities. Anything not explicitly authorized is implicitly outlawed.

Request Modification In order to illustrate some of the ideas introduced above, we now briefly describe the security aspects of the university ingress prototype and its query language QUEL, since they adopt an interesting approach to the problem. Basically, any given QUEL request is automatically modified before execution in such a way that it cannot possibly violate any specified security constraint. For example, suppose user U is allowed to retrieve parts stored in London only:

```
DEFINE PERMIT RETRIEVE ON P TO U
WHERE P.CITY = "London"
```

(See below for details of the DEFINE PERMIT operation.) Now suppose user U issues the QUEL request:

```
RETRIEVE (P.P#, P.WEIGHT)
WHERE P.COLOR = "Red"
```

Using the "Permit" for the combination of relvar P and user U as stored in the catalog, the system automatically modifies this request so that it looks like this:

```
RETRIEVE (P.P#, P.WEIGHT)
WHERE P.COLOR = "Red"
AND P.CITY = "London"
```

And of course this method request cannot possibly violate the security constraint. Note, incidentally, that the modification process is "silent": user U is not informed that the system has in fact executed a statement that is somewhat different from the original request, because that fact in itself might be sensitive (user U might even be allowed to know there are any non-London parts).

The process of request modification just outlined is actually identical to the technique used for the implementation of views and also – in the case of the ingress prototype specially – integrity constraint. So, one advantage of the scheme is that it is very easy to implement – much of the necessary code exists in the system already. Another is that it is comparatively efficient – the security enforcement overhead occurs at compile time instead of run time, at least in part. Yet another advantage is that some of the awkwardness that can occur with the SQL approach when a given user needs different privileges over different portions of the same relvar does not arise.

One disadvantage is that not all security constraints can be handled in this simple fashion. As a trivial counterexample, suppose user U is not allowed to access relvar P at all. Then no simple "modified" form of the RETRIEVE shown above can preserve the illusion that relvar P does not exist. Instead, an explicit error message along the lines of "You are not allowed to access this relvar" must necessarily be produced. (Or perhaps the system could simply lie and say "No such relvar exists.")

Here then is the syntax of DEFINE PERMIT:

```

DEFINE PERMIT      <operation name commalist>
  ON      <relvar name> [( <attribute name commalist> )]
  TO      <user ID>
[ AT      <terminal name commalist> ]
[ FROM    <time> TO <time> ]
[ ON      <day> TO <day> ]
[ WHERE <Boolean expression> ]

```

This statement is conceptually rather similar to our **AUTHORITY** statement, except that it supports a **WHERE** clause. Here is an example.

```

DEFINE PERMIT APPEND, RETRIEVE, REPLACE
ON      S(S#, CITY)
TO      Joe
AT      TTA4
FROM    9:00 TO 17:00
ON      Sat TO Sun
WHERE   S.STATUS < 50
AND     S.S# = SP.P#
AND     SP.P# = P.P#
AND     P.COLOR = "Red"

```

Note. **APPEND** and **REPLACE** are the QUEL analogs of our **INSERT** and **UPDATE**, respectively.

Audit Trails It is important not to assume that the security system is perfect. An infiltrator who is sufficiently determined will usually find a way of breaking through the controls, especially if the payoff for doing so is high. In situations where the data are sufficiently sensitive, therefore, or where the processing performed on the data is sufficiently critical, an audit trail becomes a necessity. If, for example, data discrepancies lead to a suspicion that the database has been tampered with, the audit trail can be used to examine what has been going on and to verify that matters are under control (or to help pinpoint the wrongdoer if not).

An audit trail is essentially a special file or database in which the system automatically keeps track of all operations performed by users on the regular data. In some systems, the audit trail might be physically integrated with the recovery log, in others the two might be distinct; either way, users should be able to interrogate the audit trail using their regular query language. A typical audit trail entry might contain the following information.

Request (source text)

Terminal from which the operation was invoked

User who invoked the operation

Date and time of the operation

Relvar(s), tuple(s), attribute(s) affected**Old values****New values**

As mentioned earlier in this section, the very fact that an audit trail is being maintained might be sufficient in itself to detect an infiltrator in some situations.

SQL supports discretionary access control through the GRANT and REVOKE commands. The GRANT command gives users privileges to base tables and tables and REVOKE command takes away privileges.

Grant Command This command is used to give privileges to other users of the database by the administrator.

Syntax**GRANT privileges ON object TO users [WITH GRANT OPTION]**

where objects is either a base table or views.

Mandatory Access Control

It is based on system-wide policies that cannot be changed by individual users. In this each DB object is assigned a security class. Each subject (user or user program) is assigned a clearance for a security class. Rules based on security classes and clearances govern who can read/write which objects. Most commercial systems do not support mandatory access control. Versions of some DBMSs do support it; used for specialized (e.g., military) applications.

Mandatory controls are applicable to databases in which the data have a rather static and rigid classification structure, as might be the case in certain military or government environments. As explained briefly in previous section the basic idea is that each data object has a classification level (e.g., top secret, secret, confidential, etc.), and each user has a clearance level (with the same possibilities as for the classification levels). The levels are assumed to form a strict ordering (e.g., top secret > secret > confidential, etc.). The following simple rules, due to Bell and La Padula, are then imposed:

1. User I can retrieve object j only if the clearance level of I is greater or equal to the classification level of j (the “simple security property”).
2. User I can update object j only if the clearance of I is equal to the classification level of j (the “star property”).

The first rule here is obvious enough, but the second requires a word of explanation. Observe first that another way of stating that second rule is to say that, by definition, anything written by user I automatically acquires a classification level equal to I’s clearance level. Such a rule is necessary in order to prevent a user with, e.g., “secret” classification scheme.

Note. From the point of view of pure “write” (INSERT) operations only, it would be sufficient for the classification for the second rule to say that the clearance level of I must be less than or equal to the classification level of j, and the rule is often stated in this form in the literature.

Mandatory controls began to receive a lot of attention in the database world in the early 1990s, because that was when the US Department of Defense (DoD) began to require any system it purchased to support such controls. As a consequence, DBMS vendors have been vying with one another to implement them. The controls in question are documented in two important DoD publications known informally as the Orange Book and the Lavender Book, respectively; the Orange Book defines a set of security requirements for any “Trusted Computing Base” (TCB), and the Lavender Book defines and “interpretation” of the TCB requirements for database systems specifically.

First of all, the documents define four security classes (D, C, B, and A); broadly speaking, class D is the least secure, class C is more secure than class D, and so on. Class D is said to provide minimal protection, class C discretionary protection, class B mandatory protection, and class A verified protection.

Discretionary Protection Class C is divided into two subclasses C1 and C2 (where C1 is less secure than C2), each supports discretionary controls, meaning that access is subject to the discretion of the data owner. In addition:

1. Class C1 distinguishes between ownership and access, i.e., it supports the concept of shared data, while allowing users to have private data of their own as well.
2. Class C2 additionally requires accountability support through sign-on procedures, auditing, and resource isolation.

Mandatory Protection Class B is the class that deals mandatory controls. It is further divided into subclasses B1, B2, and B3, as follows:

1. Class B1 requires “labeled security protection” (i.e., it requires each data object to be labeled with its classification level – secret, confidential, etc.). It also requires an informal statement of the security policy in effect.
2. Class B2 additionally requires a formal statement of the same thing. It also requires that covert channels be identified and eliminated. Examples of covert channels might be the possibility of inferring the answer to an illegal query from the answer to a legal one.
3. Class B3 specifically requires audit and recovery support as well as a designated security administrator.

Verified Protection Class A, the most secure, requires a mathematical proof that the security mechanism is consistent and that it is adequate to support the specified security policy.

Several commercial DBMS products currently provide mandatory controls at the B1 level. They also typically provide discretionary controls at the C2

level. Terminology: DBMS's that support mandatory controls are sometimes called multilevel secure systems. The term trusted system is also used with much the same meaning.

Suppose we want to apply the ideas of mandatory access control to the suppliers relvar S. For definiteness and simplicity, suppose the unit of data we wish to control access to the individual tuple within that relvar. Then each tuple needs to be labeled with its classification level.

Advantages of Mandatory Access Control Discretionary control has some flaws, e.g., the *Trojan horse* problem. In this, a devious unauthorized user can trick an authorized user into disclosing sensitive data. The modification of the code is beyond the DBMSs control, but it can try and prevent the use of the database as a channel for secret information.

8.2 Database Recovery

Recovery brings the database from the temporary inconsistent state to a consistent state. Database recovery can also be defined as mechanisms for restoring a database quickly and accurately after loss or damage. Databases are damaged due to human error, hardware failure, incorrect or invalid data, program errors, computer viruses, or natural catastrophes. Since the organization depends on its database, the database management system must provide mechanisms for restoring a database quickly and accurately after loss or damage.

8.2.1 Different Types of Database Failures

A wide variety of failures can occur in processing a database, ranging from the input of an incorrect data value or complete loss or destruction of the database. Some of the types of failures are listed below:

1. System crashes, resulting in loss of main memory
2. Media failures, resulting in loss of parts of secondary storage
3. Application software errors
4. Natural physical disasters
5. Carelessness or unintentional destruction of data or facilities
6. Sabotage

8.2.2 Recovery Facilities

DBMS should provide following facilities to assist with recovery.

1. Backup mechanism, which makes periodic backup copies of database
2. Logging facilities, which keep track of current state of transactions and database changes

3. Checkpoint facility, which enables updates to database in progress to be made permanent
4. Recovery manager, which allows DBMS to restore the database to a consistent state following a failure

Backup Mechanism

The DBMS should provide backup facilities that produce a backup copy of the entire database. Typically, a backup copy is produced at least once per day. The copy should be stored in a secured location where it is protected from loss or damage. The backup copy is used to restore the database in the event of hardware failure, catastrophic loss, or damage.

With large databases, regular backups may be impractical, as the time required to perform the backup may exceed that available. As a result, backups may be taken of dynamic data regularly but backups of static data, which do not change frequently, may be taken less often.

Logging Facilities

Basically there are two types of log, “transaction log” and “database change log.” A transaction log is a record of the essential data for each transaction that is processed against the database. In database change log, there are before and after images of records that have been modified.

Transaction log. Transaction log contains a record of the essential data for each transaction that is processed against the database. Data that are typically recorded for each transaction include the transaction code or identification, action or type of transaction, time of the transaction, terminal number or user ID, input data values, table and records accessed, records modified, and possibly the old and new field values.

Database change log. The database change log contains before and after images of records that have been modified by transactions. A before-image is a copy of a record before it has been modified, and an after-image is a copy of the same record after it has been modified.

Checkpoint Facility

A checkpoint facility in a DBMS periodically refuses to accept any new transactions. All transactions in progress are completed, and the journal files are brought up to date. At this point, the system is in a quiet state, and the database and transaction logs are synchronized. The DBMS writes a special record (called a checkpoint record) to the log file, which is like a snapshot of the state of the database. The checkpoint record contains information necessary to restart the system. Any dirty data blocks are written from memory to disk storage, thus ensuring that all changes made prior to taking the

checkpoint have been written to long-term storage. A DBMS may perform checkpoints automatically or in response to commands in user application programs. Checkpoints should be taken frequently.

Recovery Manager

The recovery manager is a module of the DBMS which restores the database to a correct condition when a failure occurs and which resumes processing user requests. The recovery manager uses the logs to restore the database.

8.2.3 Main Recovery Techniques

Three main recovery techniques that are commonly employed are:

1. Deferred update
2. Immediate update
3. Shadow paging

Deferred update. Deferred updates are not written to the database until after a transaction has reached its commit point. If transaction fails before commit, it will not have modified database and so no undoing of changes are required. Deferred update may be necessary to redo updates of committed transactions as their effect may not have reached database.

Immediate update. In the case of immediate update, updates are applied to database as they occur. There is a need to redo updates of committed transactions following a failure. Also there may be need to undo effects of transactions that had not committed at time of failure. It is essential that log records are written before write to database. If no “transaction commit” record in log, then that transaction was active at failure and must be undone. Undo operations are performed in reverse order in which they were written to log.

Shadow paging. Shadow paging maintains two page tables during life of a transaction, current page and shadow page table. When transaction starts, two pages are the same. Shadow page table is never changed thereafter and is used to restore database in the event of failure. During transaction, current page table records all updates to database. When transaction completes, current page table becomes shadow page table.

8.2.4 Crash Recovery

Crash recovery is the process of protecting the database from catastrophic system failures and media failures. Recovery manager of a DBMS is responsible for ensuring transaction atomicity and durability. Atomicity is attained by undoing the actions of transactions that do not commit. Durability is attained by making sure that all actions of committed transactions survive system crashes.

Recovery Manager

Recovery manager is responsible for ensuring transaction atomicity and durability. To save the state of database for the period of times it performs few operations. They are:

1. Saving checkpoints
2. Stealing frames
3. Forcing pages

Saving checkpoints. It will save the status of the database in the period of time duration. So if any crashes occur, then database can be restored into last saved check point.

Steal approach. In this case, the object can be written into disk before the transaction which holds the object is committed. This is happening when the buffer manager chooses the same place to replace by some other page and at the same time another transaction require the same page. This method is called as stealing frames.

Forcing pages. In this case, once if the transaction completed the entire objects associated with it should be forced to disk or written to the disk. But it will result in more I/O cost. So normally we use only no-force approach.

8.2.5 ARIES Algorithm

ARIES is a recovery algorithm designed to work with a steal, no-force approach. It is more simple and flexible than other algorithms.

ARIES algorithm is used by the recovery manager which is invoked after a crash. Recovery manager will perform restart operations.

Main Key Terms Used in ARIES

Log. These are all file which contain several records. These records contain the information about the state of database at any time. These records are written by the DBMS while any changes done in the database. Normally copy of the log file placed in the different parts of the disk for safety.

LSN. The abbreviation of LSN is log sequence number. It is the ID given to each record in the log file. It will be in monotonically ascending order.

Page LSN. For recovery purpose, every page in the database contains the LSN of the most recent log record that describes a change to this page. This LSN is called the page LSN.

CLR. The abbreviation of CLR is compensation log record. It is written just before the change recorded in an update log record U is undone.

WAL. The abbreviation of WAL is write-ahead log. Before updating a page to disk, every update log record that describes a change to this page must be forced to stable storage. This is accomplished by forcing all log records up

to and including the one with LSN equal to the page LSN to stable storage before writing the page to disk.

There are three phases in restarting process, they are:

1. Analysis
2. Redo
3. Undo

Analysis. In this phase, it will identify whether any page present in the buffer pool is not written into disk and activate the transactions which are in active at the time of crash.

Redo. In this phase, all the operations are restarted and the state of database at the time of crash is obtained. This is done by the help of log files.

Undo. In this phase, the actions of uncommitted transactions are undone. So only committed are taken into account.

Example

Consider the following log history as shown in Fig. 8.2.

Explanation. When the system is restarted, the analysis phase identifies T1 and T3 as transactions active at the time of crash and therefore to be undone; T2 as a committed transaction, and all its actions therefore to be written to

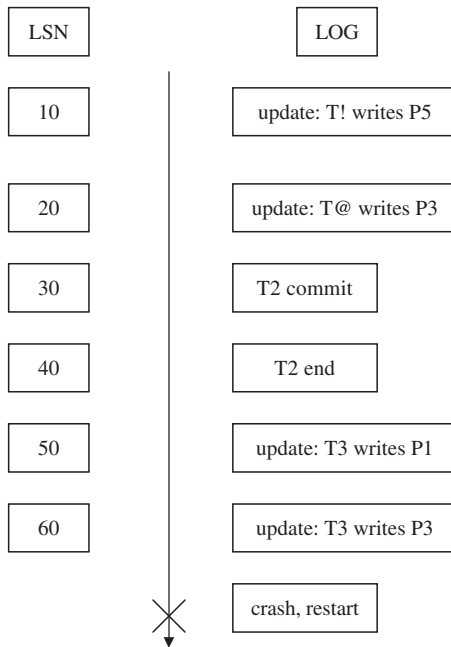


Fig. 8.2. Program flow

disk, and P1, P3, and P5 as potentially dirty pages (not yet written into disk). All the updates are reapplied in the order shown during the redo phase. Finally the actions of T1 and T3 are undone in reverse order during the undo phase; that is, T3's write of P3 is undone, T3's write of P1 is undone, and then T1's write of P5 is undone.

ARIES algorithm has three main principles:

1. Write-ahead logging
2. Repeating history during redo
3. Logging changes during undo

Write-ahead logging. This states that any change to a database object is first recorded in the log; the record in the log must be written to stable storage before the change to the database object is written to disk.

Repeating history during redo. On restart after a crash, ARIES retraces all actions of the DBMS before the crash and brings the system back to the exact state that it was in at the time of crash. Then, it undoes the actions of transactions still active at the time of crash.

Logging changes during undo. Changes made to the database while undoing a transaction are logged to ensure such an action is not repeated in the event of repeated restarts.

Elements of ARIES

1. The log
2. Tables
3. The write-ahead log protocol
4. Checkpointing

The Log

It records a history of actions that are executed by DBMS. The most recent portion of the log is called as log tail. This page will be kept at main memory and periodically it will be forced to disk. It contains several records. Each record is uniquely identified by LSN.

A log record is written for each of the following actions:

1. Updating a page
2. Commit
3. Abort
4. End
5. Undoing an update

Updating a page. After modifying the page, an update type record is appended to the log tail. The page LSN of this page is then set to the update log record as illustrated in the Fig. 8.3.

Before-image is the value of the changed bytes before the change. After-image is the value of the changed bytes after the change.

Example

Previous LSN	Transaction ID	Type	Page ID	Length	Offset	Before-Image	After-Image
--------------	----------------	------	---------	--------	--------	--------------	-------------

Fig. 8.3. Structure of update type record**Example**

Consider the following example as illustrated in Fig. 8.4.

Previous LSN	Transaction ID	Type	Page ID	Length	Offset	Before-Image	After-Image
	T1	Update	P1	4	15	HIL	SIL
	T2	Update	P2	4	23	BIL	WIL
	T2	Update	P1	4	14	TIL	VIL
	T1	Update	P4	4	15	RIL	JIL

Fig. 8.4. Log records*Tables*

In addition to log ARIES, it maintains the following two tables to maintain recovery related information:

1. Transaction table
2. Dirty page table

Transaction table. This table contains one entry for each active transaction. The entry contains the transaction ID, the status and a field called last LSN, which is the LSN of the most recent log record for this transaction. The status of a transaction can be that it is in progress, committed, or aborted.

Dirty page table. This table contains one entry for each dirty page in the buffer pool, i.e., each page with changes not yet reflected on disk. The entry

TRANSACTION TABLE

Transaction ID	Last LSN
T1	
T2	

Fig. 8.5. Transition table

DIRTY PAGE TABLE

PAGE ID	Record LSN
P1	
P2	
P4	

Fig. 8.6. Dirty page table

contains a field record LSN, which is the LSN of the first log record that caused the page to become dirty.

Now the content of transaction table will be as illustrated in Fig. 8.5.

Content of dirty page table will be as shown in Fig. 8.6.

Record LSN in the dirty page table and last LSN in the transaction table are pointing to the corresponding records in the log table.

Write-Ahead Log Protocol

WAL is the fundamental rule that ensures that a record of every change to the database is available while attempting to recover from crash. If a transaction made a change and committed, the no-force approach means that some of these changes may not have been written to disk at the time of a subsequent

crash. Without a record of these changes, there would be no way to ensure that the changes of a committed transaction survive crashes.

According to its rules when a transaction is completed its log tail is forced to disk, even a no-force approach is used.

Checkpointing

A checkpoint is used to reduce amount of work to be done during restart in the event of a subsequent crash.

Checkpointing in ARIES has three steps:

1. Begin checkpoint
2. End checkpoint
3. Fuzzy checkpoint

Begin checkpoint. It is written to indicate the checkpoint is starts.

End checkpoint. It is written after begin checkpoint. It contains current contents of transaction table and the dirty page table, and appended to the log.

Fuzzy checkpoint. It is written after end checkpoint is forced to the disk. While the end checkpoint is being constructed, the DBMS continues executing transactions and writing other log records; the only guarantee we have is that the transaction table and dirty page table are accurate as the time of the begin checkpoint.

Summary

Database security is concerned with protecting a database against accidental or intentional loss, destruction, or misuse. A comprehensive data security plan will address all of these potential threats, partly through the establishment of views, authorization rules, user-defined procedures, and encryption procedures. DBMS software provides security control through facilities such as user views, authorization rules, encryption, and authentication schemes. Set of security mechanisms presented in this chapter includes user with password and complete authorization, encryption of data.

Database recovery procedures are required to restore a database quickly after loss or damage. Basic recovery facilities that should be in place include backup facilities, checkpoint facilities, and a recovery manager. Since the organization depends so heavily on its database, the database management system must provide mechanisms for restoring a database quickly and accurately after loss or damage. In this chapter the concept of crash recovery was presented in a lucid manner. ARIES recovery algorithm was illustrated with example in this chapter.

Review Questions

8.1. List the security guidelines that a conscientious database designer should follow?

Some of the guidelines that the database designer should follow to ensure the security of the database system are:

- Keep the database simple
- Normalize the database
- Always follow the principle of assuming privileges must be explicitly granted rather than excluded
- Create unique views for each user or group of users

8.2. Why is encryption an important step in securing databases?

Encryption is a method of modifying the original information according to some code so that it can be read only if the user knows the decryption. Encryption can be used to transmit information from one computer to another. Information stored on a computer also can be encrypted. Encryption is important when transmitting data across networks.

8.3. What are the types of authorization?

The database should have sound security system so that each and every transaction is carried out by an authorized user. The types of authorization are:

- (a) User with password and complete authorization
- (b) User with password and limited authorization
- (c) Encryption of data

8.4. List four common types of database failure?

The four common types database failures are:

- (a) Aborted transactions
- (b) Incorrect data
- (c) System failure
- (d) Database destruction

Aborted transaction refers to a transaction that is in progress terminates abnormally.

Second common type of database failure is the database has been updated with *incorrect*, but valid data.

In *system failure*, some component of the system fails, but the database is not damaged. Some causes of system failure are power loss, operator error, and loss of communication in the case of network transaction.

Database destruction means the database itself is lost, or destroyed, or cannot be read.

Types of failure	Recovery techniques
Aborted transaction Incorrect data	Rollback (1) Backward recovery (2) Compensating transactions (3) Restart from checkpoint
System failure	 (1) Rollback (2) Restart from checkpoint
Database destruction	Roll forward

8.5. Mention the recovery techniques that can be applied to the common types of database failure discussed in question 8.3?

8.6. What are the security features that are commonly used in data management software?

The important security features in data management software are:

- Views which restrict user views of the database
- Authorization rules which identify users and restrict the actions they may take against database
- Encryption procedures, which encode data in an unrecognizable form
- Authentication schemes, which positively identify a person attempting to gain access to a database

8.7. What is meant by crash recovery?

Crash recovery is the process of protecting the database from catastrophic system failures and media failures. Recovery manager of a DBMS is responsible for ensuring transaction atomicity and durability. Atomicity is attained by undoing the actions of transactions that do not commit. Durability is attained by making sure that all actions of committed transactions survive system crashes.

8.8. What is the role of recovery manager in database recovery?

Recovery manager is a module of DBMS which restores the database to a correct condition when a failure occurs and which resumes processing user requests.

8.9. Discuss the importance of database recovery?

A database is a centralized facility for the entire organization. When it is accessed and used by several users of the organizations, several types of failures are bound to occur. These failures will affect the content of the database which is highly sensitive. If there is damage to the database, then one can identify the activities which are performed just prior to the point of failure of the database system. Based on this, the database is to be restored as quickly as possible to the state just prior to the occurrence of the damage.

8.10. Distinguish between logical and physical security of the database?

Physical security. Physical security refers to the security of the hardware associated with the system and the protection of the site where the computer resides. Natural events such as fire, floods, and earthquakes can be considered as some of the physical threats. It is advisable to have backup copies of databases in the face of massive disasters.

Logical security. Logical security refers to the security measures residing in the operating system or the DBMS designed to handle threats to the data. Logical security is far more difficult to accomplish.

Physical Database Design

Learning Objectives. This chapter describes physical database design which is the final phase of the database development process. During physical database design, the designer translates the logical description of data into the technical specifications for storing and retrieving data. The goal of physical database design is to create a design for storing data that will provide adequate performance and ensure database integrity. This chapter also throws light in different types of file organization. File organization is a technique for physically arranging the records of a file on secondary storage devices. The different types of file organization discussed in this chapter include sequential file organization, heap file organization, hash file organization, and index file organization.

Different types of data storage devices are discussed in this chapter. More emphasis is given to Redundant Array of Inexpensive Disk (RAID) technology. RAID is array of physical disk drives that appear to the database set as if they form one large logical storage unit. Different levels of RAID are illustrated in this chapter.

- Physical database design concept
- Access methods
- Different types of file organization
- Data storage devices
- RAID concepts and different levels of RAID

9.1 Introduction

Physical database design describes the storage structures and access methods used in system. The goal of physical database design is to specify all identifying and operational characteristics of the data that will be recorded in the information system. The physical database design specifies how database records are stored, accessed, and related to ensure adequate performance. The physical database design specifies the base relations, file organizations, and indexes used to achieve efficient access to the data, and any associated integrity constraints and security measures. The physical organization of data has a major impact on database system performance because it is the level at which actual implementation takes place in physical storage.

9.2 Goals of Physical Database Design

The goal of physical database design is to create a design providing the best response time at the lowest cost. Here response time refers to the time required to access the data, and the cost associated with CPU, memory disk input/output. The main goals of good physical database design are summarized as:

- A good physical database design should achieve high packing density, which implies minimum wastage space.
- A good physical database design should achieve fast response time.
- The physical database design should also support a high volume of transactions.

9.2.1 Physical Design Steps

The various steps in physical database design are:

1. Stored record format design
2. Stored record clustering
3. Access method design
4. Program design

Step 1: Stored Record Format Design

The visible component of the physical database structure is the stored record format design. Stored record format design addresses the problem of formatting stored data by analysis of the characteristics of data item types, distribution of their values, and their usage by various applications. Decisions on redundancy of data, derived vs. explicitly stored values of data, and data compression are made here. Certain data items are often accessed far more frequently than others, but each time a particular piece of data is needed, the entire stored record, and all stored records in a physical block as well, must be accessed. Record partitioning defines an allocation of individual data items to separate physical devices of the same or different type, or separate extents on the same device, so that total cost of accessing data for a given set of user applications is minimized. Logically, data items related to a single entity are still considered to be connected, and physically they can still be retrieved together when necessary. An extent is a contiguous area of physical storage on a particular device.

Step 2: Stored Record Clustering

One of the most important physical design considerations is the physical allocations of stored records, as a whole, to physical extents. Record clustering

refers to the allocation of records of different types into physical clusters to take advantage of physical sequentiality whenever possible. Analysis of record clustering must take access path configuration into account to avoid access time degradation due to new placement of records. Associated with both record clustering and record partitioning is the selection of physical block size. Blocks in a given clustered extent are influenced to some extent by stored record size, storage characteristics of the physical devices. Larger blocks are typically associated with sequential processing and smaller blocks with random processing.

Step 3: Access Method Design

The critical components of an access method are storage structure and search mechanisms. Storage structure defines the limits of possible access paths through indexes and stored records, and the search mechanisms define which paths are to be taken for a given applications. Access method design is often defined in terms of primary and secondary access path structure. The primary access paths are associated with initial record loading, or placement, and usually involve retrieval via the primary key. Individual files are first designed in this manner to process the dominant application most efficiently. Access time can be greatly reduced through secondary indexes, but at the expense of increased storage space overhead and index maintenance.

Step 4: Program Design

Standard DBMS routines should be used for all accessing, and query or update transaction optimization should be performed at the systems software level. Consequently, application program design should be completed when the logical database structure is known.

9.2.2 Implementation of Physical Model

The implementation of the physical model is dependent on the hardware and software being used by the company. The hardware can determine what type of software can be used because software is normally developed according to common hardware and operating system platforms. Some database software might only be available for Windows NT systems, whereas other software products such as Oracle are available on a wider range of operating system platforms, such as UNIX. The available hardware is also important during the implementation of the physical model because data are physically distributed into one or more physical disk drives. Normally, the more physical drives available, the better the performance of the database after the implementation.

9.3 File Organization

A database is stored as collection of files. A file is a set of records or relations of same type. This definition includes files of text, which can be regarded as files of 1-byte records. A record can be seen as a collection of related fields containing elementary data. Data files can exist in both primary and secondary memory, they are almost always held in secondary memory because data files are often voluminous, and the capacity of a data file can easily exceed the capacity of primary memory. Consequently, only portions of large files can be held in main memory at one time. File organization is a technique for arranging records of a file in secondary storage.

9.3.1 Factors to be Considered in File Organization

The problem in selecting a particular file organization is to choose a structure that will satisfy certain requirements. For example, a user may need to retrieve records in sequence and may also need fast access to a particular record. In this case, suitable organizations include hash files in which the key order is preserved, B-trees and indexed sequential access method (ISAM) files.

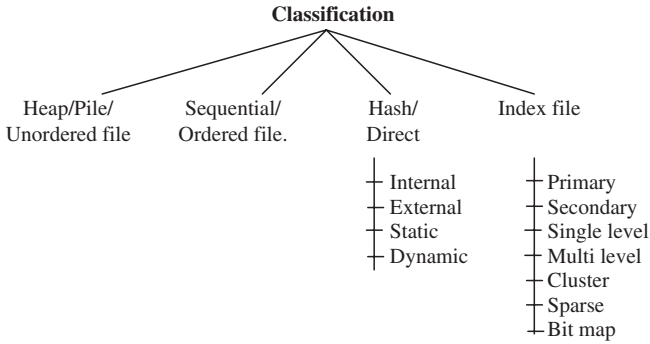
Some of the factors which are preferred in choosing the file organization are given below:

- Access should be fast. Here access refers to data access
- Storage space has to be efficiently used
- Minimizing the need for reorganization
- Accommodating growth

It is difficult to determine the best file organization and the most efficient access technique for a given situation. A good approach is to simulate the behavior of a number of candidate organizations. The simulator requires three sets of input parameters: file characteristics, user requirements, and hardware characteristics. File characteristics are logical properties of the file. File characteristics include the number of records in the file and the average attribute length. User requirements are concerned with the accesses and changes to the file. This includes the number of deletions per day and the number of times a month the whole file is read serially. Hardware characteristics are parameters of the available storage devices. These characteristics include block size, tracks per cylinder, and the storage cost per megabyte.

9.3.2 File Organization Classification

File organization can be broadly classified into two types (1) primary file organization and (2) secondary file organization as shown below.



9.4 Heap File Organization

Heap file is otherwise known as random file or pile file. In heap file organization, file records are inserted at the end of the file or in any file block with free space, hence insertion of record is efficient. Data in a file are collected in the order that it arrives. It is not analyzed, categorized, or forced to fit field definitions or field sizes. At best, the order of the records may be chronological. Records may be of variable length and need not have similar sets of data elements.

9.4.1 Uses of Heap File Organization

Heap files are used in situations where data are collected prior to processing, where data are not easy to organize, and in some research on file structures. Since much of the data collected in real-world situations are in the form of piles, this file organization is considered as the base for other evaluations.

9.4.2 Drawback of Heap File Organization

In heap file organization, data analysis can become very expensive because of the time required for retrieval of a statistically adequate number of sample records:

1. Searching of record is difficult. Normally linear search is used to locate the record.
2. Deletion of record is difficult. Because if we want to delete a particular record, first we have to locate the file and delete.

Good in the Following Situations

Heap file is good under the following situations:

- Bulk data have to be loaded.
- Relations which are always few pages long. Because search is easy.

Bad Under Situations

Heap file is bad under the following situation:

- Output is in sorted order.

9.4.3 Example of Heap File Organization

Consider the attendance register maintained by the faculty. Usually in the attendance register the students name are arranged as per the increasing order of Roll number. On the other hand if it is arranged, as per their registration to the course (like first come first serve) as shown in Table 9.1 then it is heap file organization.

From this table, it is clear that the student names are not arranged in the increasing order of Roll number. As discussed earlier, the main advantage of heap file organization is insertion of record is efficient, but if want to retrieve the data in the increasing order or decreasing order of Roll number of the student, then it is inefficient. That is searching and retrieving of record are difficult in heap file organization.

9.5 Sequential File Organization

Sequential files are also called as ordered files. In sequential file, the file records are kept sorted by the value of an ordering key. The ordering field has unique value. A sequential file is a set of contiguously stored records on a physical device such as a disk, tape, or CD-ROM. Let us consider a sequential file of “n” records. To be stored on disk, these “n” records must be grouped into physical blocks as shown in Fig. 9.1.

Table 9.1. Example of heap file organization

Roll No.	Name
S4	Krishnan
S2	Chitra
S3	Dilip
S1	Alex

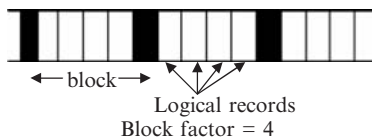


Fig. 9.1. Sequential file parameters

A block is the basic unit of input/output from disk to RAM. It can range in size from a fraction of a record to hundreds or thousands of records. Typically a block size ranges from one to hundred records. If a database has normalized records, i.e., records of constant size, then the number of records in a block is called blocking factor. For consistency and ease of programming, block sizes are usually constant for a whole system. On the other hand, almost all relational systems allow variable-size records, hence average record size can be used for simplicity.

9.5.1 Sequential Processing of File

If we have a file of “n” records then the basic performance measures for sequential search of the entire file is given below:

- Logical record accesses = n, where n is the number of records in a file.
- Sequential block accesses = $\text{ceil}(n/\text{blocking factor})$.
- Random block accesses = 0.
- Once a sequential file is created, records can be added at the end of the file.
- It is not possible to insert records in the middle of the file without rewriting the file.
- Searching method used in sequential file organization is binary search.

9.5.2 Draw Back

If new record is inserted or some record is deleted the file has to be reorganized, which is time consuming.

9.6 Hash File Organization

Hash files are also called direct files. Hashing is nothing but a method of distributing data evenly to different areas of memory. In hash file organization, the records are organized using hashing algorithm. Hash file organization is suitable for random access. In hash file organization, a hash function is applied to each record key, which returns a number which is used to indicate the position of the record in the file. The hash function must be used for both reading and writing.

9.6.1 Hashing Function

Hash function is used to locate record for access, insertion, and deletion. The hash function is given by:

Hashing function = $K \bmod B$.

$K \rightarrow$ Key value.

$B \rightarrow$ No of buckets.

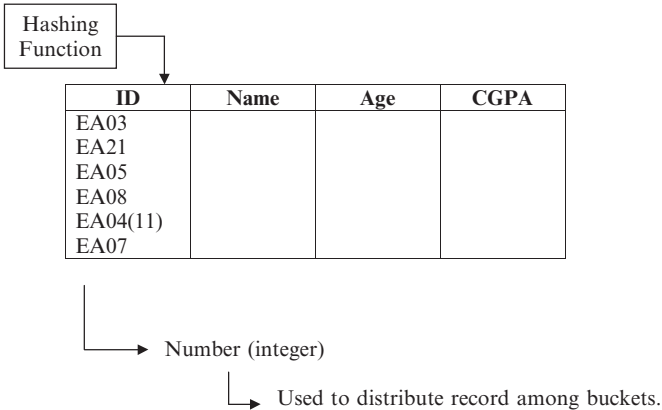
Example

Consider the example STUDENT RECORD. The attributes in the relation STUDENT RECORD are student ID, name of the student, age of the student, and the cumulative grade point average (CGPA) of the student.

In the STUDENT RECORD relation, the key value is the student ID and the number of buckets which is denoted by B is chosen as 3 as shown in Sect. 9.6.2.

STUDENT RECORD

Student ID	Name	Age	CGPA
EA03			
EA21			
EA05			
EA08			
EA04			
EA07			



9.6.2 Bucket

A bucket is a unit of storage containing one or more records (a bucket is typically a disk block). In a hash file organization, the bucket of a record is obtained directly from its search key value using a hash function.

Hashing function = $K \bmod B$.

$K \rightarrow$ Key value.

$B \rightarrow$ No of buckets.

If $B = 3$,

$$3 \bmod 3 \longrightarrow 0$$

$$21 \bmod 3 \longrightarrow 0$$

$$5 \bmod 3 \longrightarrow 2$$

$$8 \bmod 3 \longrightarrow 2$$

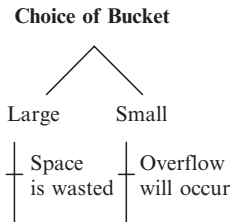
$$4 \bmod 3 \longrightarrow 1$$

$$7 \bmod 3 \longrightarrow 1$$

EA03	
EA21	Bucket 0
EA04	
EA07	Bucket 1
EA05	
EA08	Bucket 2

9.6.3 Choice of Bucket

Here the choice of the bucket refers to the fact that whether the bucket size is large or small. If the bucket size is large then, storage space required is large; on the other hand if the bucket size is small, then there is a chance of overflow.



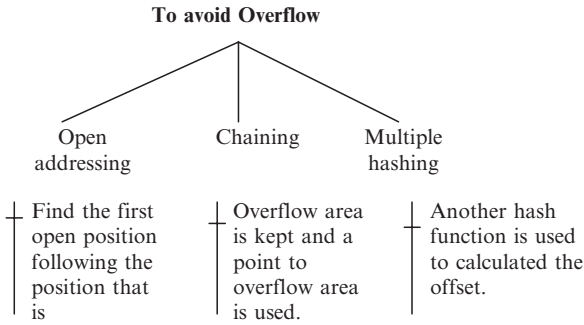
As the bucket size increases, the probability of overflow decreases but the time taken to search for a record in the bucket may increase.

Collision

There is a collision during an insertion when two records hash to the same bucket address. Two keys that hash to the same address are sometimes termed synonyms. Collisions are not a problem if the bucket is not full, because the records are simply stored in available spaces in the bucket. Overflow occurs during insertion when a record is hashed to a bucket that is already full.

Methods to Avoid Overflow

Overflow occurs when a record hashes to a full bucket. In open addressing, the addresses of buckets to search are calculated dynamically. In chaining, chains of overflow records are rooted in the home buckets.



Open Addressing

Open addressing is a technique to avoid overflow. From the name *open addressing* it is clear that we have to generate a list of bucket addresses. The list of bucket addresses is generated from the key of the record. The processes of generation of list of bucket addresses is denoted by

$$A_i = f(i, \text{key}) \quad i = 0, 1, 2, 3, \dots$$

where A_i is the list of bucket address and i is an integer.

If the bucket A_i is full, then the bucket A_{i+1} are examined. When retrieving a record, buckets are examined until one is found that either contains the required record or has an empty space. An empty space indicates that the record being searched for is not in the file. This method of resolving overflow was proposed by Peterson who termed it open addressing. A good function “ f ” should ensure that each integer 0 to $N-1$ (where N is the number of home buckets) appears in the generated list of bucket addresses.

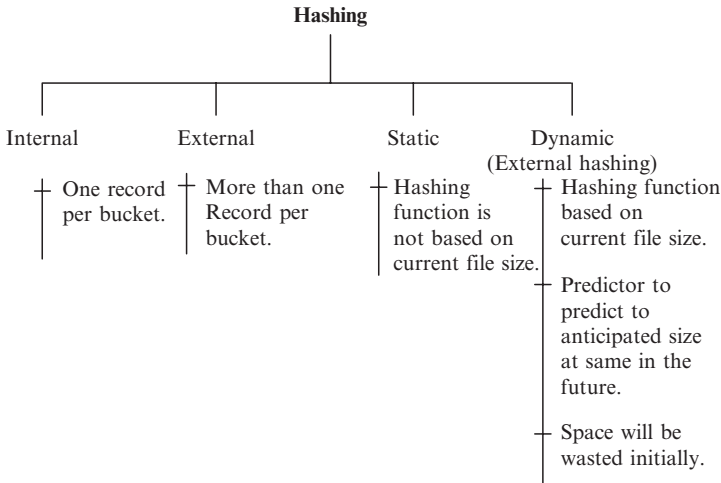
Chaining

A second solution to the problem of overflows is called chaining or closed addressing. In this method, lists of records that have overflowed are rooted in appropriate home buckets. The overflow records are put in any convenient place, perhaps in a separate overflow area or in an arbitrary bucket that is not full. In contrast to open addressing, the location used does not typically depend on the contents of the record. Let us consider three variations of the basic idea outlined above (1) separate lists, (2) coalescing lists, and (3) directory methods.

Separate lists. In the separate lists method we link together records overflowing from the same home bucket. The list is rooted in that bucket. Only the records on the list need to be examined when searching, and because any one might be the one looked for, comparisons are minimized. Deletions are straightforward. If we delete a record from a home bucket, we can replace it by one on the overflow list. If we delete a record on a list, it is removed from the list in the conventional way.

Coalescing lists. The separate list method requires a comparatively large amount of space for pointers. A second possibility is to store records in spare space in home buckets. Each bucket has a single pointer pointing to the next bucket to try when searching. Pointers are established as records overflow. This method reduces pointer overhead, but many more records may have to be examined when searching.

Directory methods. In methods involving directories, room is allocated in a home bucket beyond that needed to store records. The extra space is used to hold pointers to records overflowing from the bucket and their keys. As long as all overflows can be pointed to in this way, this method, is fast.



Hash-based index is good for equality search.

9.6.4 Extendible Hashing

Techniques that combine basic hashing with dynamic file expansion are popular today. In extendible hashing, the number of buckets grows or contracts depending on the need. When a bucket becomes full, it splits into two buckets, and records in the bucket are reallocated to the two new buckets. Thus, collisions are resolved immediately and dynamically, and long sequential searches, long overflow chains, and multiple hashing computations are avoided. The basic architecture of extendible hashing is shown in Fig. 9.2.

The primary key is sent to a hash function that produces a hash address pointing to an entry in the bucket address table (BAT), which normally resides in RAM. The BAT contains pointers to the respective physical (disk) buckets that hold the actual data records. The BAT is initialized with space for one entry and expands as the database records are inserted and more differentiation is needed to allocate records to the buckets. If “k” bits of the hash address are used to determine the bucket to store or retrieve from, the

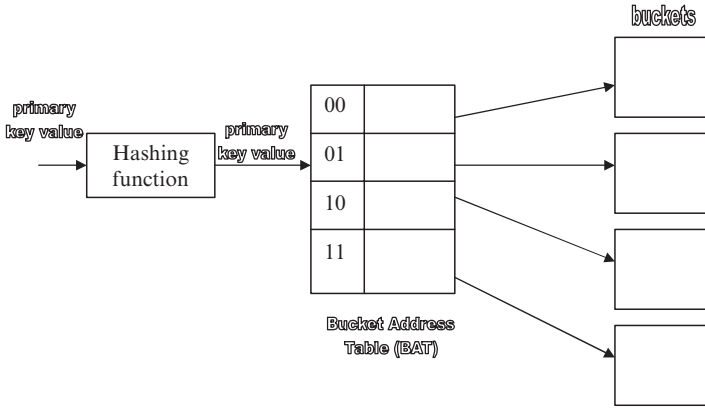


Fig. 9.2. Extensible hashing architecture

BAT contains 2^k entries. Thus, if the 8-bits are needed to allocate records to buckets, the BAT contains 256 entries, and therefore up to 256 buckets can be defined and pointed to from these entries.

9.7 Index File Organization

Index is a collection of data entries plus a way to quickly find entries with given key values. Index is a mechanism for efficiently locating row(s) without having to scan entire table.

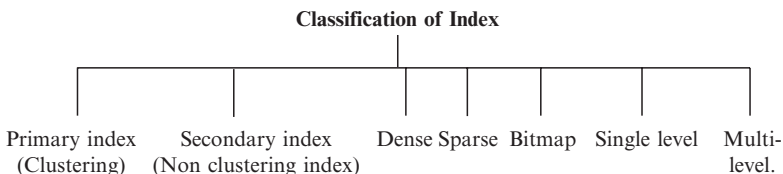
9.7.1 Advantage of Indexing

Index speeds up selections on the search key field. Search key is an attribute or set of attributes used to look up records in a file. It is possible to build more than one index for the same table:

- Index in the book allows us to locate specific page in the book.
- Index in the record allows us to find specific record in the file.

9.7.2 Classification of Index

Indexes can be broadly classified into primary index, secondary index, dense index, sparse index, bitmap, and single and multilevel as illustrated below:



Primary index. Primary index is one whose search key specifies the sequential order of the file.

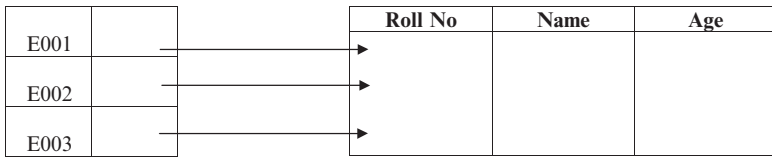
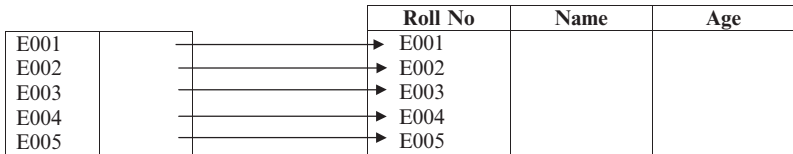
Secondary index. Secondary index improves the performance of queries that use keys other than primary search key.

Dense index. Dense index has index entry for each data record.

Sparse index. Sparse index has index entry for each page of data file.

9.7.3 Search Key

Search key is attribute or set attributes used to look up records in a file. In the example shown below, the Roll number associated with the student is the search key as it is unique for each and every student.



Dense Vs Sparse Index

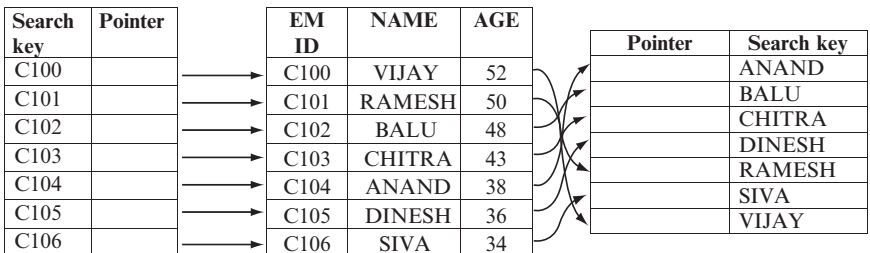
- 1. Fast search.
- 2. Space is more.

- 1. Access time is increase.
- 2. Less storage space.

PRIMARY INDEX ON EMPID

DATA FILE

SECONDARY INDEX ON NAME



9.8 Tree-Structured Indexes

A tree is a structure in which each node has at most one parent except for the root or top node. Tree-structured indexes are ideal for range-searches, and also good for equality searches. This tree-structured indexes can be classified into (1) ISAM, (2) B-tree and, (3) B⁺ tree.

9.8.1 ISAM

ISAM stands for indexed sequential access method. ISAM is a static index structure that is effective when the file is not frequently updated. In ISAM new entries are inserted in overflow pages.

9.8.2 B-Tree

The B-tree is a very popular structure for organizing and maintaining large indexes. B-trees were studied in early 1970s by Bayer, McCreight, and Comer. B-tree is a generalization of binary tree in which two or more branches may be taken from each node. B-tree is called balanced tree because the access paths to different records of equal length. B-tree has the ability to quickly search huge quantities of data. B-tree adapts well to insertions and deletions. One of the earliest B-tree search mechanisms was used at Boeing Labs. Later, the original B-tree spawned several variants, including the B⁺ developed by Prof. Donald Knuth. An index provides fast access to data when the data can be searched by the value that is the index key.

B-Tree Properties

A B-tree is a generalization of binary tree in which two or more branches may be taken from each node. A B-tree of order k has the following properties:

- Each path from the root node to a leaf node has the same length, h , also called the height of the B-tree (i.e., h is the number of nodes from the root to the leaf, inclusive).
- Each node, except the root and leaves, has at least $k + 1$ child nodes and no more than $2k + 1$ child nodes.
- The root node may have as few as two child nodes, but no more than $2k + 1$ child nodes.
- Each node, except the root, has at least k keys and no more than $2k$ keys. The root may have as few as one key. In general, any nonleaf (branch) node with j keys must have $j + 1$ child nodes.

9.8.3 Building a B⁺ Tree

The following points are useful in building a B⁺ tree. In the case of B⁺ tree, only the nodes at the bottom of the tree point to records, and all

other nodes point to the other nodes. Nodes which point to records are called leaf nodes:

- If a node is empty, then the data are added on the left.

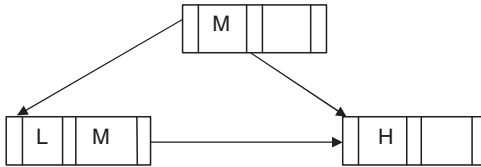


- If a node has one entry, then the left takes the smallest valued key and the right takes the biggest.



In this example, 30 is the small value hence it takes the left position and 60 is the higher value hence it takes the right position.

- If a node is full and is a leaf node, classify the keys as L (lowest), M (middle value) and H (highest), and split the node.
- If a node is full and is not a leaf node, classify the keys L (Lowest), M (middle value) and H (highest), and split the node.



Differences Between B and B⁺ Tree

B-tree	B ⁺ tree
In B-tree, nonleaf nodes are larger than leaf nodes	In B ⁺ tree leaf and nonleaf nodes are of same size
Deletion in B-tree is complicated	In B ⁺ tree, deleted entry always appears in a leaf, hence it is easy to delete an entry
Pointers to data records exist at all levels of the tree	Pointers to data records exist only at the leaves

Advantages of B-Trees

The major advantages of B-trees are summarized below:

- Secondary storage space utilization is better than 50% at all times. Storage space is dynamically allocated and reclaimed, and no service degradation occurs when storage utilization becomes very high.
- Random access requires very few steps and is comparable to hashing and multiple index methods.

- Record insertions and deletions are efficiently handled on the average, allowing maintenance of the natural order of keys for sequential processing and proper tree balance to maintain fast random retrieval.
- Allows efficient batch processing by maintaining key order.

9.8.4 Bitmap Index

Bitmap index is optimal for indexing a column containing few unique values. For example, a gender column in an application form can take just three possible values. They are “M,” “F,” and “U.” Here “M” stands for male, “F” stands for female, and “U” stands for unknown.

In order to understand how bitmap index organizes records let us consider a database table APPLICANT as shown below and the corresponding bitmap index:

APPLICANT		
ID	NAME	GENDER
	Krishnan	M
2	Radha	F
3	Mohan	M
4	Sudan	U

Bitmap index on gender			
ID	FEMALE	MALE	UNKNOWN
1		1	
2	1		
3		1	
4			1

The person corresponding to ID 1 is Krishnan who is a male hence a “1” is inserted in MALE in bitmap index. In a bitmap index, a bitmap for each key value is used. Each bit in the bit map corresponds to a possible rowed, and if the bit is set, it means that the row with the corresponding rowed contains the key value.

Benefits of Bitmap Index

The benefits of bitmap index are summarized as:

- Reduced response time for large classes of ad hoc queries.
- A substantial reduction of space usage compared to other indexing techniques.

Fully indexing a large table with a normal index can be expensive in terms of space since the index can be several times larger than the data in the table. Bitmap indexes are typically only a fraction of the size of the indexed data in the table.

9.9 Data Storage Devices

The data stored by an organization double in every 3 or 4 years. Hence the selection of data storage devices is a key consideration for data managers.

9.9.1 Factors to be Considered in Selecting Data Storage Devices

The following factors have to be considered while evaluating data storage options:

- Online storage
- Backup files
- Archival storage

When the device is used to store online data, then one has to give importance to access speed and capacity, because many firms require rapid response to large volumes of data.

Backup files are required to provide security against data loss. Ideally, backup storage is a high volume capacity at low cost.

Archived data may need to be stored for many years; so archival medium should be highly reliable, with no data decay over extended periods, and low cost.

The following factors have to be considered in storing the data in a particular medium:

- Volume of data
- Volatility of data
- Required speed of access to data
- Cost of data storage
- Reliability of data storage medium

9.9.2 Magnetic Technology

Magnetic technology is based on magnetization and demagnetization of spots on a magnetic recording surface. The same spot can be magnetized and demagnetized repeatedly. Magnetic recording materials may be coated on rigid plotters (hard disks), flexible circular substrates (floppy disks), thin ribbons of material (magnetic tapes), or rectangular sheets (magnetic cards).

The main advantages of magnetic technology are its relative maturity and widespread use. A major disadvantage is susceptibility to strong magnetic

fields that can corrupt data stored on a disk. Also magnetization decays with time. Hence it is not a preferable medium to store legal documents, archival data.

9.9.3 Fixed Magnetic Disk

A fixed magnetic disk contains one or more recording surfaces that are permanently mounted in the disk drive and cannot be removed. Fixed disk is the medium of choice from personal computers to super computers. Fixed disks gives rapid, direct access to large volumes of data, and is ideal for highly volatile files. The major disadvantage of magnetic disk is the possibility of head crash that destroys the disk surface and data hence it is necessary to regularly make backup copies of hard disk files.

9.9.4 Removable Magnetic Disk

A removable disk comes in two formats: single disk and disk pack. Disk packs consist of multiple disks mounted together on a common spindle in a stack, usually on a disk drive with a retractable read/write heads. The disk's removability is its primary advantage making it ideal for backup.

9.9.5 Floppy Disk

Floppy low cost makes them ideal for storing and transporting small files and programs. But the reliability is not so good. A speck of dust can cause read error.

9.9.6 Magnetic Tape

In magnetic tape, the data storage and retrieval are in sequential manner. Hence the access time, which refers to data access, is high. Magnetic tape was used extensively for archiving and backup in early database systems.

9.10 Redundant Array of Inexpensive Disk

RAID stands for Redundant Array of Inexpensive (Independent) Disk. A disk array comprises of several disks managed by a controller. Disks and controllers can be joined together in RAID combinations. First, they can provide fault tolerance by introducing redundancy across multiple disks. Second, they can provide increased throughput because disk array controller supports parallel access to multiple disks. Instead of using one massive drive, RAID technology stores several smaller drives in one container.

Stripping. Stripping is an important concept for RAID storage. Stripping involves the allocation of physical records to different disks. A stripe is the set of physical records that can be read or written in parallel. Normally, a stripe contains a set of adjacent physical records.

The different types of disk arrays are known by their RAID Levels. Some of the RAID Levels are:

- (1) RAID Level 0 + 1
- (2) RAID Level 0
- (3) RAID Level 1
- (4) RAID Level 2
- (5) RAID Level 3
- (4) RAID Level 4
- (5) RAID Level 5
- (6) RAID Level 6
- (7) RAID Level 10
- (8) RAID Level 50

9.10.1 RAID Level 0 + 1

RAID Level 0 + 1 requires a minimum of four drives to implement. RAID 0 + 1 is implemented as mirrored arrays as shown in Fig. 9.3 whose segments are RAID 0 arrays. High input/output rates are achieved due to multiple stripe segments.

Disadvantages of RAID Level 0 + 1

The disadvantages of RAID Level 0 + 1 are:

- Limited scalability at a very high inherent cost.
- Very expensive/high overhead.
- A single drive failure will cause the whole array to become RAID Level 0 array.
- All drives must move in parallel to proper track lowering sustained performance.

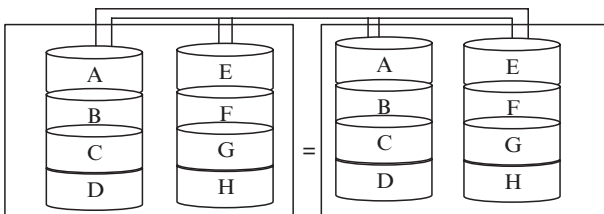


Fig. 9.3. Mirroring

Recommended Applications of RAID Level 0 + 1

The recommended applications of RAID Level 0 + 1 are:

- Imaging applications
- File server

9.10.2 RAID Level 0

RAID Level 0 requires a minimum of two drives to implement. RAID Level 0 implements a striped disk array, the data are broken down into blocks and each block is written to a separate disk drive. The striped disk array concept is shown in the Fig. 9.4.

Advantages of RAID Level 0

The main advantages of RAID Level 0 are:

- Very simple design and easy to implement.
- No parity calculation overhead is involved.
- Best performance is achieved when data are striped across multiple controllers with only one drive per controller.
- Input/output performance is greatly improved by spreading the input/output load across many channels and drives.

Drawbacks of RAID Level 0

Some of the drawbacks of RAID Level 0 are:

- RAID Level 0 is not a “true” RAID because it is not fault-tolerant.
- The failure of just one drive will result in all data in an array being lost.
- RAID Level 0 should never be used in mission critical environments.

Recommended Applications of RAID Level 0

- Image, video editing
- Prepress applications
- Applications that require high bandwidth

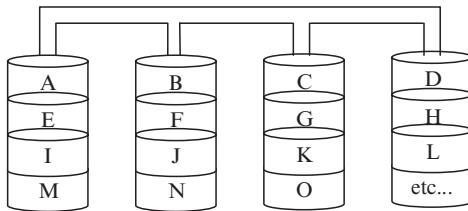


Fig. 9.4. Stripped disk array

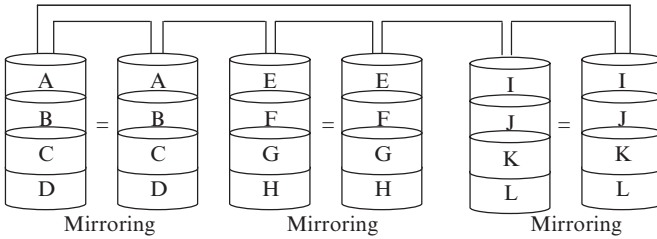


Fig. 9.5. Mirroring and duplexing

9.10.3 RAID Level 1

RAID Level 1 requires a minimum of two drives to implement. The characteristics of RAID Level 1 are mirroring and duplexing which is shown in Fig. 9.5.

Advantages of RAID Level 1

The main advantages of RAID Level 1 are:

- Simplest RAID storage subsystem design.
- Under certain circumstances, RAID 1 can sustain multiple simultaneous drive failures.
- One hundred percent redundancy of data means no rebuild is necessary in case of a disk failure, just a copy to the replacement disk.

Disadvantages of RAID Level 1

Some of the drawbacks of RAID Level 1 are:

- Highest disk overhead.
- May not support hot swap of failed disk when implemented in “software.”
- Hardware implementation is strongly recommended.

Recommended Applications of RAID Level 1

- Accounting
- Payroll
- Financial
- Any application requiring high availability

9.10.4 RAID Level 2

A RAID Level 2 system would normally have as many data disks as the word size of the computer, typically 32. In addition, RAID 2 requires the use of extra

disks to store an error-correcting code for redundancy. With 32 data disks, a RAID 2 system would require seven additional disks for a Hamming-code ECC.

For a number of reasons, including the fact that modern disk drives contain their own internal ECC, RAID 2 is not a practical disk array scheme.

Advantages of RAID Level 2

The main advantages of RAID Level 2 are:

- Extremely high data transfer rates possible.
- Relatively simple controller design compared to RAID Levels 3-5.

Disadvantages of RAID Level 2

Some of the disadvantages of RAID Level 2 are:

- Entry level cost is very high.
- No practical use; same performance can be achieved by RAID 3 at lower cost.

9.10.5 RAID Level 3

RAID Level 3 is characterized by parallel transfer with parity. The idea of parallel transfer with parity is illustrated in the Fig. 9.6.

In RAID Level 1, data are striped (subdivided) and written on the data disks. Stripe parity is generated on Writes, recorded on the parity disk, and checked on Reads. RAID Level 3 requires a minimum of three drives to implement.

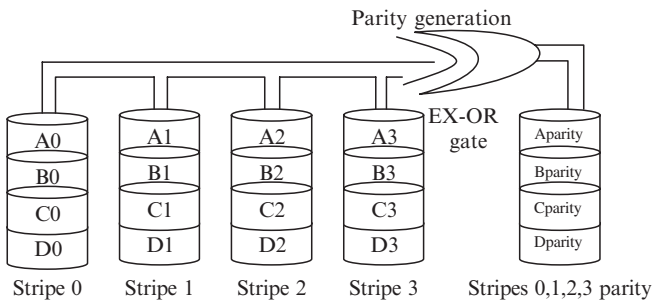


Fig. 9.6. Parallel transfer with parity

Advantages of RAID Level 3

The advantages of RAID Level 3 are:

- Very high data transfer rate.
- Disk failure has an insignificant impact on throughput.
- High efficiency because of low ratio of parity disks to data disks.

Disadvantages of RAID Level 3

- Controller design is fairly complex.
- Transaction rate is equal to that of a single disk drive at best.
- Very difficult and resource intensive to do as a “software” RAID.

Recommended Applications

- Video production and live streaming
- Image editing, video editing
- Any application requiring high throughput

9.10.6 RAID Level 4

RAID Level 4 is characterized by independent data disks with shared parity disks as shown in Fig. 9.7. Each entire block is written onto a data disk. Parity for same rank blocks is generated on Writes, recorded on the parity disk, and checked on Reads. RAID Level 4 requires a minimum of three drives to implement.

Advantages of RAID Level 4

Some of the advantages of RAID Level 4 are:

- Very high Read data transaction rate.
- Low ratio of parity disks to data disks means high efficiency.
- High aggregate Read transfer rate.

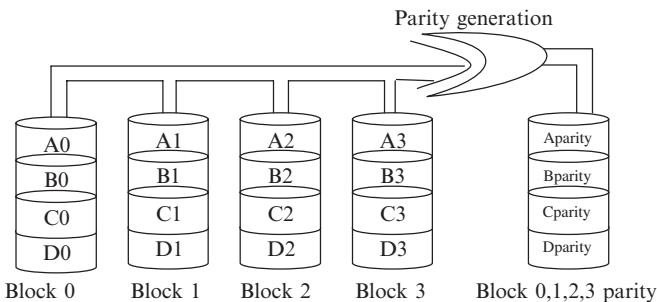


Fig. 9.7. Independent disk with shared parity

Disadvantages of RAID Level 4

Some of the disadvantages of RAID Level 4 are:

- Quite complex controller design.
- Worst write transaction rate and Write aggregate transfer rate.
- Difficult and inefficient data rebuild in the event of disk failure.

9.10.7 RAID Level 5

In RAID Level 5, each entire data block is written on a data disk, parity for blocks in the same rank is generated on Writes, recorded in a distributed location and checked on Reads. RAID Level 5 requires a minimum of three drives to implement. RAID Level 5 is characterized by independent data disks with distributed parity blocks as shown in Fig. 9.8.

Advantages of RAID Level 5

The main advantages of RAID Level 5 are:

- Highest Read transaction rate
- Medium Write data transaction rate
- Good aggregate transfer rate

Disadvantages of RAID Level 5

Some of the disadvantages of RAID Level 5 are:

- Most complex controller design
- Difficult to rebuild in the event of disk failure
- Individual block transfer rate is same as single disk

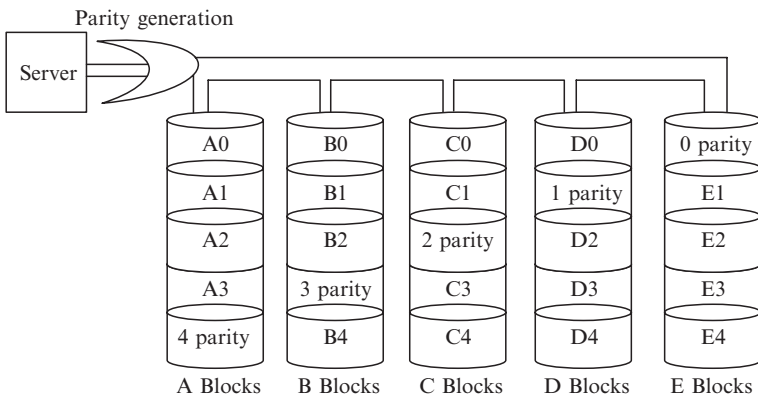


Fig. 9.8. Independent disk with distributed parity

Recommended Applications

- File and application servers
- Database servers
- Intranet servers

9.10.8 RAID Level 6

RAID Level 6 is characterized by independent data disks with two independent distributed parity schemes as shown in Fig. 9.9. Two independent parity computations must be used in order to provide protection against double disk failure. Two different algorithms are employed to achieve this purpose. RAID Level 6 requires a minimum of four drives to implement.

Advantages of RAID Level 6

The main advantages of RAID Level 6 are:

- RAID Level 6 provides high fault tolerance and can sustain multiple simultaneous drive failures.
- Perfect solution for critical applications.

Drawbacks of RAID Level 6

Some of the drawbacks of RAID Level 6 are:

- More complex controller design.
- Controller overhead to compute parity addresses is extremely high.

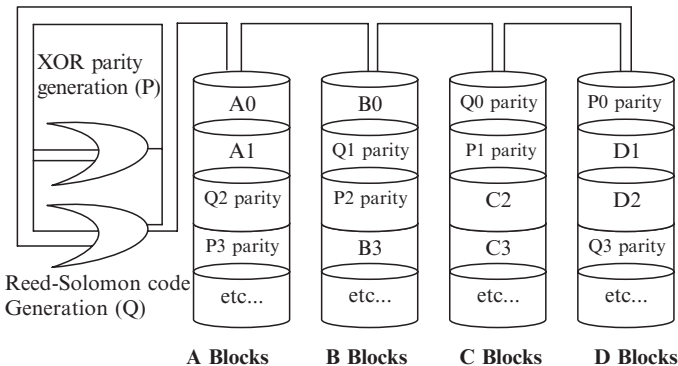


Fig. 9.9. Independent data disks with two independent distributed parity schemes

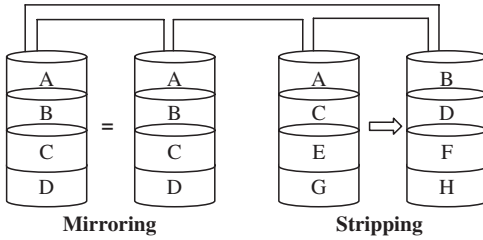


Fig. 9.10. Mirroring and stripping

Recommended Applications

- File and application servers
- Web and E-mail servers
- Intranet servers

9.10.9 RAID Level 10

RAID Level 10 has very high reliability combined with high performance. RAID Level 10 is implemented as a striped array whose segments are RAID 1 arrays as shown in Fig. 9.10.

Advantages of RAID Level 10

The main advantages of RAID Level 10 are:

- High input/output rates are achieved by striping RAID 1 segments.

Drawbacks of RAID Level 10

Some of the drawbacks of RAID Level 10 are:

- Very expensive/high overhead
- Very limited scalability at a very high inherent cost

Recommended Application

- Database server requiring high performance and fault tolerance

9.11 Software-Based RAID

Primarily used with entry-level servers, software-based arrays rely on a standard host adapter and execute all I/O commands and mathematically intensive RAID algorithms in the host server CPU. This can slow system

performance by increasing host PCI bus traffic, CPU utilization, and CPU interrupts. Some network operating system (NOS) such as NetWare and Windows NT include embedded RAID software. The chief advantage of this embedded RAID software has been its lower cost compared to higher-priced RAID alternatives. However, this advantage is disappearing with the advent of lower-cost, bus-based array adapters. The major advantages are low cost and it requires only a standard controller.

9.12 Hardware-Based RAID

Unlike software-based arrays, bus-based array adapters/controllers plug into a host bus slot (typically a 133 MByte (MB) s^{-1} PCI bus) and offload some or all of the I/O commands and RAID operations to one or more secondary processors as shown in Fig. 9.11. Originally used only with mid- to high-end servers due to cost, lower-cost bus-based array adapters are now available specifically for entry-level server network applications.

9.12.1 RAID Controller

The RAID controller is a device in which servers and storage intersect. The controller can be internal to the server, in which case it is a card or chip, or external, in which case it is an independent enclosure, such as a network-attached storage (NAS). In either case, the RAID controller manages the physical storage units in a RAID system and delivers them to the server in logical units.

While a RAID controller is almost never purchased separately from the RAID itself, the controller is a vital piece of the puzzle and therefore not as much a commodity purchase as the array.

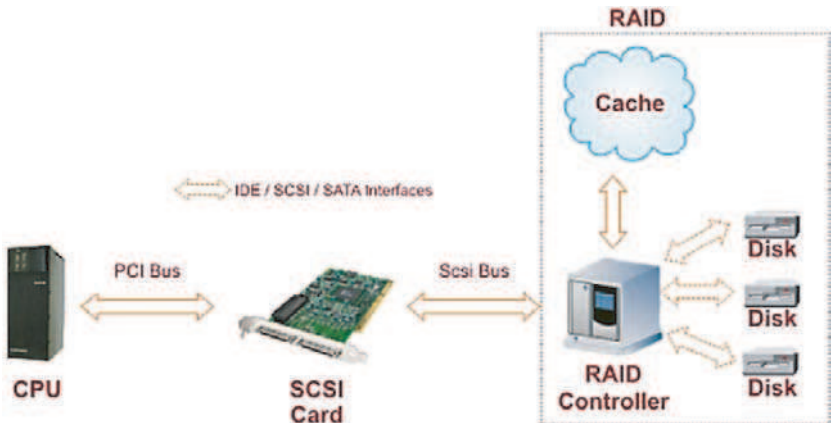


Fig. 9.11. Hardware-based RAID

In addition to offering the fault-tolerant benefits of RAID, bus-based array adapters/controllers perform connectivity functions that are similar to standard host adapters. By residing directly on a host PCI bus, they provide the highest performance of all array types. Bus-based arrays also deliver more robust fault-tolerant features than embedded NOS RAID software.

9.12.2 Types of Hardware RAID

There are two main types of hardware RAID, differing primarily in how they interface the array to the system.

Bus-Based or Controller Card Hardware RAID

This is the more conventional type of hardware RAID, and the type most commonly used, particularly for lower-end systems. A specialized RAID controller is installed into the PC or server, and the array drives are connected to it. It essentially takes the place of the small computer system interface (SCSI) host adapter or integrated development environment (IDE) controller that would normally be used for interfacing between the system and the hard disks; it interfaces to the drives using SCSI or IDE/ATA, and sends data to the rest of the PC over the system bus. Some motherboards, particularly those intended for server systems, come with some variant of *integrated* RAID controller. These are built into the motherboard, but function in precisely the same manner as an add-in bus-based card. (This is analogous to the way that the integrated IDE/ATA controllers on all modern motherboards function the same way that add-in IDE/ATA controllers once did on older systems.) The only difference is that integrated controllers can reduce overall cost at the price of flexibility.

Intelligent, External RAID Controller

In this higher-end design, the RAID controller is removed completely from the system to a separate box. Within the box the RAID controller manages the drives in the array, typically using SCSI, and then presents the logical drives of the array over a standard interface (again, typically a variant of SCSI) to the server using the array. The server sees the array or arrays as just one or more very fast hard disks; the RAID is completely hidden from the machine. In essence, one of these units really *is* an entire computer unto itself, with a dedicated processor that manages the RAID array and acts as a conduit between the server and the array.

Advantages are data protection and performance benefits of RAID and more robust fault-tolerant features and increased performance vs. software-based RAID.

9.13 Optical Technology

Optical storage systems work by reflecting beams of laser light off a rotating disk with a minutely pitted surface. As the disk rotates, the amount of light reflected back to a sensor varies, generating a stream of ones and zeros. The advantages of optical technology are high storage densities, low cost media, and direct access.

There are four storage media based on optical technology. They are CD-ROM, WORM, magneto-optical, and DVD. Optical technology is highly reliable because it is not susceptible to head crashes.

CD-ROM. CD-ROM stands for compact disk-read only memory. CD-ROM is a compact, robust, high capacity medium for the storage of permanent data. Once the data are written, it cannot be altered.

CD-R. CD-R stands for CD-recordable. CD-R writers are used to prepare disks for CD mastering, test prototype applications, and backup systems.

CD-RW. CD-RW stands for CD-rewritable. This format allows erasing and rewriting to a disk many times.

WORM. WORM stands for write once read many. WORM is the major storage device for images. Information once written to a blank disk cannot be altered. WORM jukeboxes are used to store high volumes of data. A jukebox may contain up to 2,000 WORM disks. A WORM jukebox makes terabytes of data available in about 10 s.

9.13.1 Advantages of Optical Disks

The main advantages of optical disk are given below:

1. *Physical.* An optical disk is much sturdier than tape or a floppy disk. It is physically harder to break, melt, or warp.
2. *Delicacy.* It is not sensitive to being touched, though it can get too dirty or scratched to be read. It can be cleaned.
3. *Magnetic.* It is entirely unaffected by magnetic fields.
4. *Capacity.* Optical disks hold much more data than floppy disks.

9.13.2 Disadvantages of Optical Disks

Some of the disadvantages of optical disks are:

Cost. The cost of the optical disk is high. But due to the advancement in technology, the price has come down drastically. Hence cost cannot be considered as drawback.

Duplication. It is not easy to copy an optical disk as it is a floppy disk. Software and hardware is necessary for writing disks. This is balanced by the fact that it is not as necessary to have extra copies since the disk is so much sturdier than other media.

Summary

The primary goal of physical database design is data processing efficiency. Today, with ever-decreasing costs of computer technology per unit of measure, it is typically important that the physical database design must minimize the time required by users to interact with the information system. During physical database design, the database designer translates the logical description of data into the technical specifications for storing and retrieving data.

A physical file is a named portion of secondary memory allocated for the purpose of storing physical records. Data within a physical file are organized through a combination of sequential storage and pointers. A file organization arranges the records of a file on a secondary storage device. The three major categories of file organizations are sequential file organization, index file organization, and hash file organization. In sequential file organization, records are stored in a sequence to a primary key value. In index file organization, records are stored sequentially or nonsequentially and an index is used to keep track of where the records are stored. In hash file organization, the address of each record is determined using an algorithm that converts a primary key value into a record address. In this chapter the different types of file organization are explained through illustrative examples.

File access efficiency and file reliability can be enhanced by the use of a RAID, which allows blocks of data from one or several programs to be read and written in parallel to different disks, thus reducing the input/output delays with traditional sequential I/O operations on a single disk drive. In this chapter, the basic concept of RAID and different levels of RAID are explained. Various levels of RAID allow a file and database designer to choose the combination of access efficiency, space utilization, and fault tolerance best suited for the database applications.

Review Questions

9.1. What are the main differences between ISAM and B⁺ tree indexes?

The main difference between ISAM and B⁺ tree indexes is that ISAM is static while B⁺ tree is dynamic. Another difference between the two indexes is that ISAM's leaf pages are allocated in sequence.

9.2. What is the order of B⁺ tree?

The order of a B⁺ tree is denoted by which is a measure of the capacity of the tree node. Every node in the B⁺ tree contains “m” entries where $d \leq m \leq 2d$.

9.3. How many nodes must be examined for equality search in a B⁺ tree? How many for a range selection? Compare this with ISAM?

For equality search in a B⁺ tree, l nodes must be examined, where l = height of the tree. For range selection, number of nodes examined = l + m - 1, where

m is the number of nodes that contains elements in the range selection. For ISAM, the number of nodes examined is the same as B^+ tree plus any overflow pages that exist.

9.4. Define static, extensible, and linear hashing? Describe the advantages and disadvantages?

Static hashing is a hashing technique where the buckets that hold data entries are statically allocated. If too many records are inserted for a given bucket, the system creates overflow pages. While this technique is simple, it can require many I/Os to find a specific data record if that record is in a bucket with many other records.

Extensible hashing is a hashing technique that does not require overflow pages. Extensible hashing uses a directory of pointers to buckets. When a page for a bucket overflows, the bucket is split. This splitting occasionally requires the doubling of the directory structure. As stated above, this technique does not require overflow pages. However, it requires the space overhead of the directory, and possibly (but not likely) an extra I/O for the directory lookup.

Linear hashing is a dynamic hashing technique that handles the problem of long overflow chains without a directory. Linear hashing uses temporary overflow pages, and chooses the buckets to split in a round-robin fashion. Linear hashing requires no dynamic directory structure, only a counter for “next” and “level.” However, it does have some overflow pages, and it may require more than 1–2 I/Os for a lookup.

9.5. In extensible hashing, why do you use the least significant bits of the hash value to determine the directory slot of a data item?

If the least significant bits are used, the system can copy the old directory, and use this new copy as the second half of the new directory. After the copy, only one of the directory pointers for the bucket that split needs to be updated.

9.6. Compare the merits and demerits of the different types of secondary storage devices in tabular form?

Type	Advantage	Disadvantage	Typical use
Floppy disk	Inexpensive, direct access, removable	Low capacity, slow access	Store files for word processors and spreadsheets
Hard disk	Fast, direct access	Limited capacity	Store programs and data
CD-ROM	High capacity, direct access	Slow access	Reference material
Magnetic tape	High capacity	Slow sequential data access	Backup programs and data

9.7. What is RAID system and what are its benefits in a database application?

A relatively recent innovation in disk drives is dramatically improving capabilities of DBMS. In RAID technology, instead of using one massive drive, several smaller drives are there in one container.

The primary advantage of storing data on separate drives is that each drive can store or retrieve data at the same time. This parallel processing significantly improves the system performance. A second advantage to the system is that it can automatically duplicate each portion of the data and store it on a different disk. If one of the disks is destroyed, all of the data are still available on the other disks and can be recovered automatically.

9.8. What are the benefits and costs of using indexes?

Indexed tables provide fast random and sequential access to tables from any predetermined sort condition. The time taken to retrieve data from the database is considerable reduced by using indexes.

It is not advisable to index each column of the table as it takes more space. If the index is stored sequentially, then it is necessary to copy huge chunks of the index whenever a row is inserted into the table.

9.9. Construct a B⁺ tree for the following set of key values?

(2, 3, 5, 7, 11, 17, 19, 23, 29, 31). Assume that the number of pointers that will fit in one node is 4. It is also assumed that the tree is initially empty and the values are added in ascending order.

Solution:

Given data:

Number of pointer in one node = 4

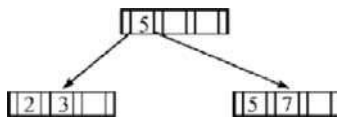
=> The number of keys = 3

To ensure the property that everything is at least half full, all leaf nodes must have at least one key, all nonleaf nodes must have at least two pointers.

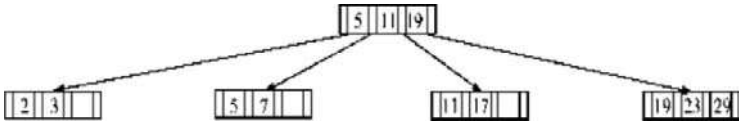
Since the values are assumed to be entered in ascending order, insert 2, 3, and 5.



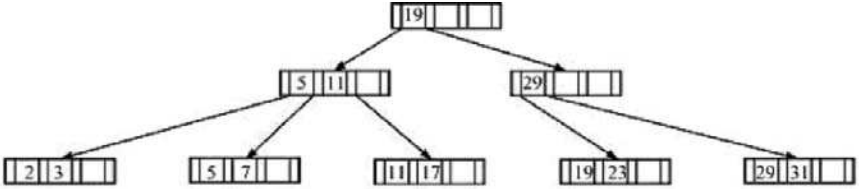
Insert 7, by splitting into two nodes and adding an index node, copying middle value up.



Insert 11, insert 17 and split, insert 19, insert 23 and split, insert 29.



Finally, insert 31 and split the node, copy up the 29 into the index node which now needs to be split creating a parent (new root) index node with the middle index pushed up.



9.10. What is the function of RAID controller?

The RAID controller is a device in which servers and storage intersect. The controller can be internal to the server, in which case it is a card or chip, or external, in which case it is an independent enclosure, such as a NAS. In either case, the RAID controller manages the physical storage units in a RAID system and delivers them to the server in logical units.

Data Mining and Data Warehousing

Learning Objectives. This chapter provides an overview of advanced concepts in database management system, which includes concepts related to data mining and data warehousing. This chapter deals with classification of data mining system, data mining issues, prediction, clustering, association rules, trends, and applications of data mining system. The data warehouse architectures, design, and user interface concepts are discussed. After completing this chapter the reader should be familiar with the following concepts:

- Need for data mining
- Data mining functionalities
- Classification and prediction of data mining
- Performance issues in data mining
- Data mining association rules
- Application and trends in data mining
- Goals of Data Warehousing
- Characteristics of Data in Data Warehouse
- Data Warehouse Architectures
- Data Warehouse Design
- The User Interface

10.1 Data Mining

10.1.1 Introduction

Data mining refers to extracting or “mining” knowledge from large amounts of data. The data mining is appropriately named as “Knowledge Mining.” There are many other terms carrying a similar or slightly different meaning to data mining, such as Knowledge Mining from Databases, Knowledge Extraction, Data/Pattern Analysis, Data Archaeology, and Data dredge.

Data mining is an essential process where intelligent methods are applied in order to extract data patterns. Data mining is the process of discovering interesting knowledge from large amounts of data stored in databases, data warehouses, or other information repositories.

10.1.2 Architecture of Data Mining Systems

Data mining is an essential process where intelligent methods are applied in order to extract data patterns. The architecture of data mining is shown in Fig. 10.1. The major components are described as follows.

Data Warehouse. This is one or a set of databases, spreadsheets, or other kinds of information repositories. Data cleaning and Data integration techniques may be performed on the data.

Database. The database server is responsible for fetching the relevant data based on the user data-mining request.

Knowledge Base. This can be used to guide the search, or evaluate the interestingness of the resulting patterns. Such knowledge include concept hierarchies, used to organize attributes or attribute values into different levels of abstraction, knowledge such as user beliefs, which can be used to assess the pattern interestingness based on its unexpectedness may also be included.

Data Mining Engine. This is essential to the data mining system and ideally consists of set of functional modules for tasks such as characterization, association, classification, cluster analysis, evaluation, and deviation analysis.

Pattern Evaluation Modules. This component typically employs interestingness measures and interacts with the data mining modules so as to focus the search toward interesting patterns. It may use interestingness thresholds to filter out discover patterns. Alternatively, this module may be integrated with the mining module depending on the implementation of the data mining method used.

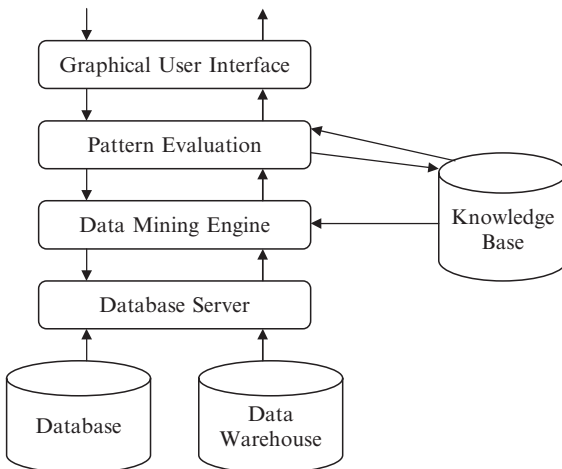


Fig. 10.1. Architecture of data mining

Graphical User Interface. This module communicates between the user and the data mining system, allowing the user to interact with the system by specifying the data mining query or task, providing the information to help focus the search, and performing exploratory data mining based on the intermediate data mining results.

10.1.3 Data Mining Functionalities

Functionalities of data mining are used to specify the kind of patterns to be found in data mining tasks. It can be classified into two categories such as Descriptive and Predictive. Descriptive mining task characterize the general properties of data in the database, whereas predictive mining task perform inference on the current data in order to make predictions.

These functionalities are classified as follows:

- Characterization and discrimination
- Association analysis
- Classification and prediction
- Cluster analysis
- Outlier analysis
- Evolution analysis

10.1.4 Classification of Data Mining Systems

Data mining is an interdisciplinary field, the confluence of set of disciplines, including database system statistics, machine learning, visualization, and information science. Moreover, depending on the data mining approach used, techniques from other disciplines may be applied. Data mining research is expected to generate a large variety of data mining systems. It can be described as follows.

Classification According to the Kinds of Database Mined

Database system themselves can be classified according to different criteria such as data models, each of which may require its own data mining techniques. If classifying according to the special types of data handled, we may have a spatial, time series, text, or world wide mining system.

Classification According to the Kinds of Knowledge Mined

It can be categorized according to the kinds of knowledge they mine, i.e., based on data mining functionalities, such as characterization, discrimination, association, classification, clustering, cluster outlier analysis, and evolution analysis. It can be based on granularity or levels of abstraction of the knowledge mined.

Classification According to the Kinds of Techniques Utilized

Data mining techniques can be categorized according to the degree of user interaction involved or methods of data analysis employed. A sophisticated data mining system will often adopt multiple data mining techniques or work out an effective integrated technique that combines the merits of a few individual approaches.

10.1.5 Major Issues in Data Mining

Major issues in data mining are mining methodology, user interaction, performance, and diverse data types. These are described follows.

Mining Methodologies and User Interaction Issues

These reflect the kind of knowledge mine, the ability to mined knowledge at multiple granularities, the user domain knowledge, and knowledge visualization.

Mining Different Kind of Knowledge in Database

Since different users can be interested in different kinds of knowledge, data mining should cover a wide spectrum of data analysis and knowledge discovery task, including data characterization discrimination, association, classification, clustering, trend and deviation analysis, and similarity analysis. These tasks may be used in the same database in different ways and require the development of numerous data mining techniques.

Incorporation of Background Knowledge

Background knowledge or information regarding the domain under study may be used to guide the discovery process and allows discovered patterns to be expressed in concise terms and at different levels of abstraction.

Presentation and Visualization of Data Mining Results

Discover knowledge should be expressed in high-level languages, visual representations, or other expressive forms so that knowledge can be easily understood and directly used by humans. This is especially crucial if the data mining system is to be interactive.

Handling Noisy or Incomplete Data

The data stored in database may reflect noise, exceptional cases, or incomplete data objects. When mining data regularities, these objects may confuse the process, causing knowledge model constructed to over fit the data. As a result, the accuracy of the discovered pattern can be poor.

10.1.6 Performance Issues

The performance issues in data mining include efficiency, scalability, and parallelization of data mining algorithms.

Efficiency and Scalability of Data Mining Algorithms

To effectively extract information from a huge amount of data in databases, data mining algorithm must be efficient and scalable. Many of the issues are followed under mining methodology, and user interaction must consider efficiency and scalability.

Parallel, Distributed, and Incremental Mining Algorithms

The huge size of many databases, the wide distribution of data, and computational complexity of some data mining methods are factors motivating the development of parallel and distributed data mining algorithm. Such algorithms divide the data into partitions, which are processed in parallel. The results from the partitions are then merged. Therefore, this algorithm performs the knowledge modification incrementally to amend and strengthened.

Issues Relating to the Diversity of Database Types

The main issues related to the diversity of the database types are handling of relational and complex types of data and mining information from heterogeneous database.

Handling of Relational and Complex Types of Data

Relational databases are widely used, the development of efficient and effective data mining systems for such data are important. However, other database may contain complex data object, hypertext and multimedia data, spatial data, temporal data, or transaction data. It is unrealistic to expect one system to mine all kinds of data, given the diversity of data types and different goals of data mining.

Mining Information from Heterogeneous Database and Global Information Systems

LAN connects many sources of data, forming huge, distributed, and heterogeneous databases. The discovery of knowledge from different sources of structure and semistructured or unstructured data with diverse data semantics poses great challenges to data mining. Web mining, which uncovers interesting knowledge about Web contents, Web usage became a very challenging and highly dynamic field in data mining.

10.1.7 Data Preprocessing

Today's real world databases are highly susceptible to noisy, missing, and inconsistent data due to their typically huge size, often several giga bytes or more. To improve the quality of the data and efficiency, data preprocessing is introduced.

Real world data tends to be dirty incomplete and inconsistent. This technique can improve the quality of data, thereby improving accuracy and efficiency of the subsequent data mining process. It is an important step in the knowledge discovery process. Since quality decisions must be based on quality data. Detecting data anomalies, rectifying them early, and reducing the data to be analyzed can lead to huge payoffs for decision making.

There are a number of data preprocessing techniques. They are:

- Data Cleaning
- Data Integration
- Data Transformation
- Data Reduction

Data Cleaning

Data cleaning routines attempt to fill in the missing values, smooth out noise while identifying outliers, and correct inconsistencies in the data.

Noisy Data

Noise is a random error or variance in a measured variable. Smooth out the data to remove the noise. The smoothing techniques are as follows:

Binning. This method smoothen a sorted data value by consulting its “neighborhood,” that is, the values around it. The sorted values are distributed into a number of “buckets,” or bins:

- A. Smoothing by bin means each value in a bin is replace by the mean value of the bin.
- B. Smoothing by bin medians means each bin value is replaced by bin median.

Clustering. Outliers may be detected by clustering, where similar values are organized into groups or clusters. The values outside the set of clusters are outliers.

Combined Computer and Human Inspection. Outliers may be identified through a combination of computer and human inspection. A human can sort through the patterns in the list to identify the actual garbage ones (e.g., Miss Labeled Character). This is much faster than manually searching through the entire database.

Regressions. Data can be smoothed by fitting the data to a function such as with regression.

- a. *Linear Regression.* This involves finding the “best” line to fit two variables so that one variable can be used to predict the other.
- b. *Multiple Linear Regression.* It is an extension of linear regressions, where more than two variables are involved and the data are fit to a multi-dimensional surface.

Inconsistent Data

There may be inconsistencies in the data recorded for some transactions. Some data inconsistencies may be corrected manually using external references.

Data Integration

Data mining often requires data integration, the merging of data from multiple data stores. There are a number of issues to consider during data integration, which combines data from multiple sources into a coherent data store. These sources may include multiple databases, data cubes, or flat files. The issues in data integration are:

- A. *Schema Integration.* It is referred to as entity identification problem. So the databases typically have metadata, that is, data about the data. The metadata can be used to help avoid errors in Schema integration.
- B. *Redundancy.* An attribute may be redundant if it can be derived from another table. Inconsistencies in attribute or dimension naming can also cause redundancies in the resulting data set. Some redundancies can be detected by correlation analysis. The correlation between attributes A and B can be measured by:

$$r_{A,B} = \frac{\sum (A - \bar{A})(B - \bar{B})}{(n - 1)\sigma_A\sigma_B}$$

where n is the number of tuples, A and B are the respective mean values of A and B, and σ

Careful integration of the data from multiple sources can help reduce and avoid redundancies and inconsistencies in the resulting data set.

Data Transformation

In data transformation, the data are transformed or consolidated into forms appropriate for mining. Data transformations can involve the following:

1. *Smoothing.* Smoothing works to remove the noise from data, such technique include binning, clustering, and regression.

2. *Aggregation.* Aggregation operations are applied to the data, this step is typically used in constructing a data cube for analysis of the data at multiple granulaires.
3. *Generalization.* Generalization of the data, where low-level or “primitive” data are replaced by higher-level concepts through the use of concept hierarchies.
4. *Normalization.* Normalization implies that the attribute data are scaled so as to fall within a small specified range, such as $-1.0-1.0$ or $0.0-1.0$
5. *Attribute Construction.* In attribute construction, new attributes are constructed and added from the given set of attributes to help the mining process.

Data Reduction

Data reduction technique can be applied to obtain a reduced representation of the data set that is much smaller in volume. It maintains the integration of original data that is mining on the reduced data set more efficiently.

There are several types of data reduction, which are given as follows:

1. *Data Cube Aggregation.* The aggregation operations are applied to the data in the construction of a data cube.
2. *Dimension Reduction.* In dimension reduction, the irrelevant, weakly relevant or redundant attributes or dimensions may be detected and removed.
3. *Data Compression.* In data compression, the encoding mechanisms are used to reduce the data set size.
4. *Numerosity Reduction.* In numerosity reduction, the data are replaced or estimated by alternative, smaller data representations such as parametric models or nonparametric methods such as clustering, sampling, and use of histograms.
5. *Discretization and Concept Hierarchy Generation.* The raw data values for attributes are replaced by ranges or higher conceptual levels. Concept hierarchies allow the mining of data at multiple levels of abstraction and are a powerful tool for data mining.

Data Mining Primitives

A popular misconception about data mining is to expect that data mining systems can autonomously dig out all of the valuable knowledge that is embedded in a given large database, without human intervention or guidance. It may be the first sound appealing to have an autonomous data mining system. A more realistic scenario is to expect that users can communicate with the data mining system using a set of data mining primitives designed in order to facilitate efficient and fruitful knowledge discovery. Such primitives include the specification of the portions of the database.

These primitives allow the user to interactively communicate with the data mining system during discovery in order to examine the findings from different angles or depths, and direct the mining process. A data mining query language can be designed to incorporate these primitives, allowing users to flexibly interact with data mining systems. Data mining query language provides a foundation on which user-friendly graphical interfaces can be built.

10.1.8 Data Mining Task

Data mining task can be specified in the form of a data mining query, which is input to the data mining system. A data mining query is defined in terms of the following primitives.

The Kinds of Knowledge to be Mined

This specifies the data mining function to be performed, such as characterization, discrimination, association, classification, clustering or evaluation analysis. In addition to specifying the kind of knowledge to be mined for a given data mining task, the user can be more specific and provide pattern templates that all discovered patterns must match. These templates are Meta patterns (meta rules or meta queries) and can be used to guide the discovery process.

Background Knowledge

User can specify background knowledge or knowledge about the domain to be mined. This knowledge is useful for guiding the knowledge discovery process and evaluating the patterns found. There are several kinds of background knowledge such as concept hierarchies, schema hierarchies, and operation derived hierarchies.

Concept hierarchy defines a sequence of mapping from a set of low-level concepts to higher-level, more general concepts. It is useful to allow the data to be mined with multiple levels of abstraction. It is represented as a set of nodes organized in a tree, where each node in itself represents a concept as illustrated in Fig. 10.2. A special node, all, is reserved for root of tree. It denotes the most generalized value of the given dimension.

A Schema Hierarchy is a total or partial order among attributes in the database schema. This hierarchy may formally express existing schematic relationships between attributes. Typically, it specifies a data warehouse dimension.

An Operation Derived Hierarchy is based on operation specified by users, experts, or the data mining system. Operation can include the decoding of information-encoded strings, information, and extraction from complex data objects and data clustering.

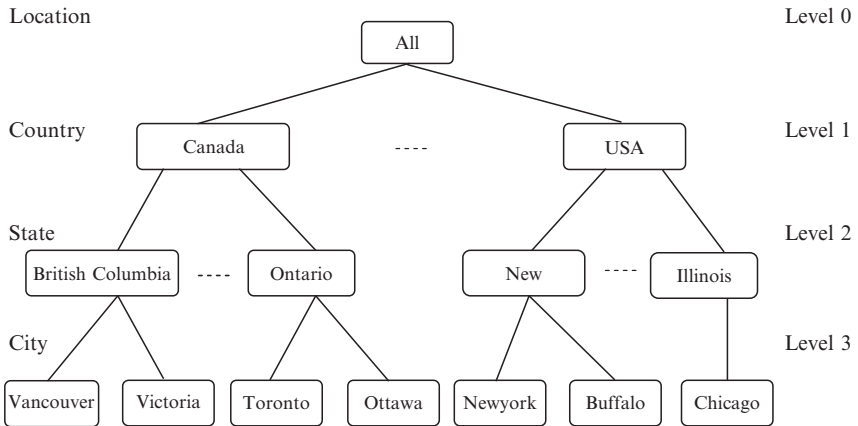


Fig. 10.2. A concept hierarchy for dimension

Interestingness Measures

These functions are used to separate uninteresting patterns from knowledge. They may be used to guide the mining process or after discovery to evaluate discovered patterns. Different kinds of knowledge may have different interestingness measures. Some objective measures pattern interestingness, such objectives are based on the structures of the patterns and the statistic underlying them.

- *Simplicity.* A factor contributing to the interestingness of a pattern is the pattern overall simplicity for human comprehension. Objective measures of patterns simplicity can be viewed as function of the pattern structure defined in terms of pattern size in bits or the number of attributes or operator appearing in the pattern.
- *Certainty.* Each discovered pattern should have a measure of certainty associated with it that assesses the validity or “Trustworthiness” of a pattern. Certainty measure for association rules of the form “ $A \Rightarrow B$,” where A and B are sets of items, is confidence.
- *Novelty.* Novel patterns are those that attribute new information or increase performance to the given pattern set. The strategy for detecting the novelty is to remove redundant patterns if the discovered rule can be implied by another rule that is already in the knowledge base or in the derived rule set, and then either rule should be reexamined in order to remove the potential redundancy.

Presentation and Visualization of Discovered Patterns

This refers to the form in which discovered patterns are to be displayed. User can choose from different forms for knowledge presentation, such as rules, tables, charts, graphs, decision trees, and cubes.

Allowing the visualization of discovered patterns can help users with a different backgrounds to identify patterns of interest and to interact or guide the system in further discovery. A user should be able to specify the forms of presentation to be used for displaying the discovered patterns. The use of concept hierarchies plays an important role in aiding the user to visualize the discovered patterns. The mining with concept hierarchies allows the representation to discover knowledge in high-level concepts, which may be more understandable to user than rules expressed in terms of primitive data such as functional or multivalued dependency rules or integrity constants.

10.1.9 Data Mining Query Language

The importance of design of a good data mining query language can also be seen by observing history of relational database system. Relational database systems have dominated the database market for decades. The standardization of relational query language, which occurred at early stages of relational database development, is widely credited for success of the relational database field. Hence having a good query language may help standardize the development of platforms for data mining system. Designing a comprehensive data mining language is challenging because data mining covers wide spectrum of task, from data characterization to mining association rules, data classification, and evaluation analysis.

Designing the data mining query language is specified by the following primitives:

- The kind of knowledge to be mined.
- The background knowledge to be used in the discovery process.
- The Interestingness measures and threshold or pattern evaluation.

Syntax for Specifying the Kind of Knowledge to be Mined

This statement is used to specify the kind of knowledge to be mined. Its syntax is defined later for characterization, association, and classification.

Characterization

```
⟨Mine_Knowledge_Specification⟩ ::= mine characteristics [as ⟨pattern_name⟩]
Analyze ⟨measure(s)⟩
```

This specifies the characteristic descriptions are to be mined, the analyzed class is used for characterization, and specific aggregate measures.

Association

```
⟨Mine_Knowledge_Specification⟩ ::= mine association [as ⟨pattern_name⟩]
[Matching ⟨metapattern⟩]
```

This specifies the mining patterns of association, the user providing the option with the matching clause. The Meta patterns can be used to focus

the discovery toward the patterns that match the given Meta patterns, thereby enforcing additional syntactic constraints for the mining task.

Classification

$\langle \text{Mine_Knowledge_Specification} \rangle ::= \text{mine classification [as } \langle \text{pattern_name} \rangle \text{] Analyze } \langle \text{classifying_attribute_or_dimension} \rangle$

It specifies that the patterns for data classification are to be mined; the analyze clause specifies that the classification is performed according to the values of classifying_attribute_or_dimension.

Syntax for Concept Hierarchy Specification

It allows mining of knowledge at multiple levels of abstraction. To accommodate a different viewpoint of uses with regard to the data, there may be more than one concept hierarchy per attribute or dimension. For instance, some users prefer organizing branch locations by provinces and states, while others organizing them according to languages used.

Syntax: use hierarchy $\langle \text{hierarchy_name} \rangle$ for $\langle \text{attribute_or_dimension} \rangle$

Syntax for Interestingness Measure Specification

The user can help control the number uninteresting patterns returned by the data mining system by specifying measures of pattern interestingness and their corresponding threshold. Interestingness measure includes the confidence, support, noise, and novelty.

Syntax: with $[(\text{interest_measure_name})]$ threshold = $\langle \text{threshold_value} \rangle$

10.1.10 Architecture Issues in Data Mining System

The architecture and design of data mining system is critically important. Based on the different architecture designs data mining systems can be integrated with a DB/DW (database/data warehouse) system using the following coupling schemes:

- *No coupling.* No coupling means that a data mining system will not utilize any function of a DB/DW system. It may fetch data from a particular source, process data using some data mining algorithms, and then store the mining results in another file. Moreover, most data have been or will be stored in DB/DW system. Without any coupling of such systems, a data mining system will need to use other tools to extract data, making it difficult to integrate such a system into an information processing environment. This represents a poor design.

- *Loose coupling*. It means the data mining system will use some facilities of a DB or DW system, fetching data from a data repository managed by these systems, performing data mining, and then storing the mining results either in a file or in a designated place in a database. Loose coupling is better than no coupling since it can fetch any portion of data stored in a database by using query processing, indexing, and other system facilities.
- *Semitight coupling*. It means that besides linking a data mining system to a DB/DW system, efficient implementations of a few essentials data mining primitives can be provided in the DB/DW system. These primitives can include sorting, indexing, aggregation, histogram analysis, multiway joint, and precomputation of some essential statistical measures such as sum, count, max, minimum, and standard deviation.
- *Tight coupling*. It means that data mining system is smoothing integrators in to the DB/DW system. The data mining subsystem is treated as one functional component of an information system. Data mining queries and functions are optimized based on mining query analysis, data structures, indexing scheme, and query process methods of DB or DW system.

This approach is highly desirable since it facilitates efficient implementation of data mining functions, high system performances, and an integrated information processing environment.

With these analysis data mining system should be coupled with the DB/DW system. Loose coupling is, though not efficient, is better than no coupling. Since it makes use of both delta and system facilities of a DB/DW system. Tight coupling is highly desirable but its implementation is nontrivial and more research is needed in this area. Semitight coupling is a compromise between loose and tight coupling. It is important to identify commonly used data mining primitives and provide efficient implementation of such primitives in DB/DW system.

10.1.11 Mining Association Rules in Large Databases

Association rule mining finds interesting association or correlation relationships among a large set of data items, with a massive amounts of data continuously being collected and stored, many industries are becoming interested in mining association rules from their databases.

The discovery of interesting association relationships among huge amounts of business transaction records can help in many business decision making process, such as catalog design, cross marketing, and loss-leader analysis.

Basic Concepts

Let $j = \{i_1, i_2, i_m\}$ be a set of items, let D , the task-relevant data, be a set of database transactions where each transaction T is a set of items such that $T \subseteq J$. Each transaction is associated with an identifier called TID. Let A be a set of items, A transaction T is said to contain A if and only if $A \subseteq T$.

An association rule is an implication of the form $A \Rightarrow B$, where $A \subset J$, $B \subset J$, and $A \cap B = \phi$, so the rule is an implication of the form $A \Rightarrow B$ holds in the transaction set D with support S , where S is percentage of transactions in D that contain $A \cup B$.

Association Rule Mining is a Two-step Process

1. *Find all frequent itemsets.* Each of these itemsets will occur at least as frequently as a predetermined minimum support count.
2. *Generate strong association rules from the frequent itemsets.* These rules must satisfy minimum support and minimum confidence, it is the easiest process of the two methods. So, the overall performance of mining association rule is determined by the first step.

The Apriori Algorithm

A. Finding Frequent Itemsets Using Candidate Generation

The Apriori algorithm is an influential algorithm for mining frequent item sets for Boolean association rules. This algorithm uses prior knowledge of frequent itemset properties. Apriori employs an iterative approach known as level-wise search, where k itemsets are used to explore $(k+1)$ itemsets. It also improves the efficiency of level-wise generation of frequent itemsets, which is an important property and is called Apriori property and it is used to reduce the search space.

Improving the Efficiency of Apriori The techniques for improving the efficiency of Apriori is summarized later:

1. *Hash-based technique (Hashing itemset counts).* This technique can be used to reduce the size of the candidate k -itemsets, C_k for $k > 1$.
2. *Transaction reduction.* A transaction that does not contain any frequent k -itemsets cannot contain any frequent $(k+1)$ itemsets. Therefore such a transaction can be marked or removed from further consideration to the subsequent for j -itemsets, where $j > k$.
3. *Partitioning.* A partitioning technique that requires just two database scans to mine the frequent itemsets can be used. It consists of two phases as shown in Fig. 10.3, in phase one, the algorithm subdivides the transactions of D into n nonoverlapping partitions. If the minimum support threshold for transactions in D is min_sup then the minimum itemset support count for a partition is $\text{min_sup} \times \text{number of transaction in the partition}$. For each partition, all frequent itemset within the partition are found. This refers to a local frequent itemsets. These itemsets are candidate itemsets with respect to D .

In phase two, a second scan of D is conducted in which the actual support of each candidate is assessed to determine the frequent itemsets. Since the

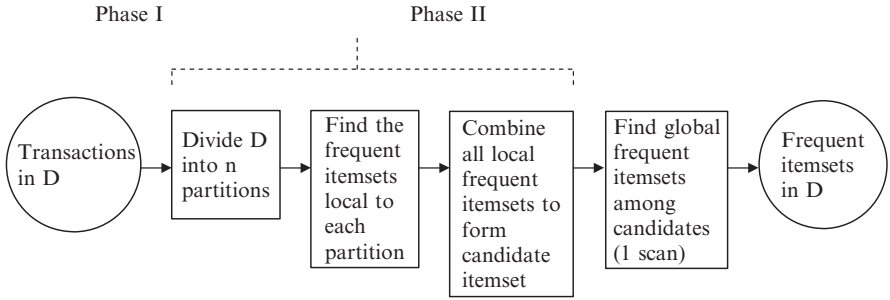


Fig. 10.3. Mining by partitioning the data

partition size and the number of partition are each said to partition to fit a main memory, they read only once in each memory.

4. *Sampling.* The basic idea of sampling approach is to pick random sample S of the given data D , and then search for frequent itemsets in S instead of D . In this way, we trade off some degree of accuracy against efficiency. A sample size S search for frequent itemsets S can be done in main memory, so only one scan of transaction is required overall. A sampling approach is especially beneficial when efficiency is of utmost importance, such as in computationally intensive application that must be run on a very frequent basis.
5. *Dynamic Itemset Counting.* This technique is proposed in which the database is partitioned into blocks marked by start points. In this variation, new candidate itemset can be added at any start point, unlike in Apriori, in which determines a new candidate itemset only immediately prior to each complete database scan. It estimates a support of all the itemsets that have been counted so far, adding new candidate itemsets to be frequent.

B. Mining Frequent Itemsets Without Candidate Generation

In many cases, the Apriori candidate generation and test method reduces the size of the candidate sets significantly and leads to good performance gain. It may suffer from two nontrivial cost:

1. It may need to generate a huge number of candidate sets. (It must generate more than $2^{100} \approx 10^{30}$ candidates in total).
2. It may need to repeatedly scan the database and check the large sets of candidate by pattern matching. (This is especially the case for long mining patterns).

Iceberg Queries The Apriori algorithm can be used to improve the efficiency of answering iceberg queries. Iceberg queries are commonly used in data mining, particularly for market basket analysis. This query computes an aggregate

function over an attribute or set of attributes in order to find aggregate values above specified threshold.

10.1.12 Mining Multilevel Association From Transaction Databases

This section deals with multilevel and multidimensional association rules of data mining.

Multilevel Association Rules

For many applications, it is difficult to find strong association among data items at low or primitive levels of abstraction due to the sparsity of data in multidimensional space. Strong association discovered at high-concept levels may represent common sense knowledge. However, what may represent common sense to one user may seem novel to another. Therefore, data mining system should provide capabilities to mine association rules at multiple levels of abstraction and traverse easily among different abstraction spaces.

Approaches to Mining Multilevel Association Rules

The approaches to mining multilevel association rules are summarized later:

1. *Using Uniform Minimum Support for all Levels.* The same minimum support threshold is used when mining at each level of abstraction. When a uniform minimum support threshold is used, the search procedure is simplified. The method is also simple in that users are required to specify only one minimum support threshold. This approach has some difficulties; it is unlikely that items at lower levels of abstraction will occur as frequently as those at higher levels of abstraction. If the minimum support threshold is set to high, it could miss several meaningful associations occurring at low abstraction levels. If the threshold is set to low, it may generate many uninteresting associations occurring at high abstraction levels.
2. *Using Reduced Minimum Support at Lower Levels.* Each level of abstraction has its own minimum support threshold. The lower the abstraction level, the smaller the corresponding threshold. For mining multiple level associations with reduced support, there are numbers of alternative search strategies:
 - *Level by level independent.* This is a full breadth search, where no background knowledge of frequent itemsets is used for pruning.
 - *Level cross-filtering by single item.* An item at the i^{th} level is examined if and only if parent node at the $(i - 1)^{\text{th}}$ level is frequent.
 - *Level cross-filtering by k-itemset.* A k-itemset at the i^{th} level is examined if and only if its corresponding parent k-itemsets at the $(i - 1)^{\text{th}}$ level is frequent.

Multidimensional Association Rules

For example, consider a Samsung Electronics database; we may discover the Boolean association rule, “IBM desktop computer \Rightarrow Samsung color printer,” which can be written as:

Buys (X, “IBM Desktop computer”) \Rightarrow Buys (X, “Samsung Color printer”)

where X is a variable representing customers who purchased items in Samsung Electronics transactions. Following the terminology used in multidimensional databases, we refer to each distinct predicate in a rule as a dimension. Hence, we can refer to rule as a single-dimensional or intradimensional association rule since it contains a single distinct predicate (e.g., buys) with multiple occurrences (i.e., the predicate occurs more than once within the rule). Techniques for mining multidimensional association rules can be categorized according to three basic approaches regarding the treatment of quantitative attributes.

In the first approach, quantitative attributes are discretized using predefined concept hierarchies. This discretization occurs prior to mining. A concept hierarchy for income may be used to replace the original numeric values of this attribute by ranges, such as “0–20 K,” “21–30 K,” “31–40 K,” and so on. The discretization is static here and it can be predetermined. The discretized numeric attributes, with their range values, can be treated as categorical attributes. This is referred as Mining Multidimensional Association rules using static discretization of quantitative attributes.

In the second approach, quantitative attributes are discretized into “bins” based on the distribution of the data. These bins may be further combined during the mining process. The discretization process is dynamic and established so as to satisfy some mining criteria, such as maximizing the confidence of the rules mined. This strategy treats the numeric attribute values as quantities rather than as predefined ranges or categories, association rules mined from this approach are also referred to as Quantitative Association Rules.

In the third approach, quantitative attributes are discretized so as to capture the semantic meaning of such interval data. This dynamic discretization procedure considers the distance between data points. Hence, such quantitative association rules are also referred to as Distance-Based Association rules.

Mining Quantitative Association Rules

Quantitative association rules are multidimensional association rules in which the numeric attributes are dynamically discretized during the mining process so as to satisfy some mining criteria, such as maximizing the confidence or compactness of the rules mined. The quantitative association rules are focused by the two quantitative attributes on the left side of the rule and one categorical attribute on the right side of the rule.

Binning. The grids in the ranges of Quantitative attributes are partitioned into intervals. These intervals are dynamic in that they may later be further combined during mining process. This partitioning process is referred to as Binning and the intervals are referred as bins. The three common binning strategies are:

1. Equiwidth Binning, where the interval size of each bin is the same.
2. Equidepth Binning, where each bin has approximately the same number of tuples assigned to it.
3. Homogeneity-based Binning, where bin size is determined so that the tuples in each bin are uniformly distributed.

From Association Mining to Correlation Analysis. Most of the association rule-mining algorithm employs a support-confidence framework. In spite of using minimum support and confidence thresholds to help weed out or exclude the exploration of uninteresting rules, many rules that are not interesting to the user may still be produced. Even the strong association rules can be uninteresting and misleading, and then discuss additional measures based on statistical independence and correlation analysis.

Strong Rules are not Necessarily Interesting. In data mining, whether a rule is interesting or not can be judged subjectively or objectively. Ultimately, only the user can judge if a given rule is interesting or not, and this judgments, being subjective, may differ from one user to another. However, objective interestingness measures, based on the statistics behind the data, can be used as one step toward the goal of weeding out uninteresting rules from presentation to the user.

From Association Analysis to Correlation Analysis. Association rules mined using a support and support-confidence frameworks are useful for many applications. However, the support-confidence framework can be misleading that is it may identify a rule $A \Rightarrow B$ as interesting when, in fact, the occurrence of A does not imply the occurrence of B.

The occurrence of itemset A is independent of the occurrence of itemset B if $P(A \cup B) = P(A)P(B)$; otherwise itemsets A and B are dependent and correlated as events. This definition can be easily extended to more itemsets. The correlation between the occurrence of A and B can be measured by computing,

$$\text{corr}_{A,B} = \frac{P(A \cup B)}{P(A)P(B)}$$

If the resulting value of the equation is less than 1, then the occurrence of A is negatively correlated with the occurrence of B. If the resulting value is greater than 1, then A and B are positively correlated, meaning the occurrence of the other. If the resulting value is equal to 1, then A and B are independent and there is no correlation between them.

Constraint-Based Association Mining. For a given set of task-relevant data, the data mining process may uncover thousands of rules, many of which are uninteresting to the user. In constraint-based mining, mining is performed under the guidance of various kinds of constraints provided by the user. These constraints include the following:

- *Knowledge type constraints.* These specify the type knowledge to be mined, such as association.
- *Data Constraints.* These specify the set of task-relevant data.
- *Dimension/level Constraints.* These specify the dimension of the data, or levels of the concept hierarchies, to be used.
- *Interestingness Constraints.* These specify thresholds on statistical measures of rule interestingness, such as support and confidence.

10.1.13 Rule Constraints

These specify the form of rules to be mined. Such constraints may be expressed as metarules (rule templates), as the maximum or minimum number of predicates that can occur in the rule antecedent or consequent, or as relationships among attributes, attribute values, and/or aggregates. The use of rule constraint is to focus the mining task. This form of constraint-based mining allows the user to specify the rules to be mined according to their intention, thereby making the data mining process more effective.

Metarules-guided Mining of Association Rules

Metarules allow users to specify the syntactic form of rules that they are interested in mining. It can be used as constraints to help improve the efficiency of the mining process. It may be based on the analyst's experience, expectations, or intuition regarding the data, or automatically generated based on the database scheme.

Mining Guided by Additional Rule Constraints

Rule constraints specifying set/subset relationships, constant intuition of variables, and aggregate functions can be specified by the user. These may be used together with, or as an alternative to, metarules guided mining. Rule constraints can be classified into five categories with respect to frequent itemsets mining, namely as antimonotone, monotone, succinct, convertible, and inconvertible.

- If an itemsets does not satisfy this rule constraint, none of its supersets can satisfy the constraint. If rule constraint obeys this property, it is called *antimonotone*. Pruning by antimonotone constraints can be applied at each iteration of Apriori style algorithms to help improve the efficiency of the overall mining process.

- If itemsets satisfies this rule constraint, so do all of its supersets. If a rule constraint obeys this property, it is called *monotone*.
- In this category, we can enumerate all and only those sets that are guaranteed to satisfy the constraint. If a rule is *succinct* we can directly generate precisely the sets that satisfy it, even before support counting begins. It avoids the substantial overhead of the generation and test paradigm.
- Some constraints belong to none of the earlier three categories. However, if the items in the itemsets are arranged in a particular order, the constraint may become monotone or antimonotone with regard to a frequent itemsets mining process, these constraints are called as *convertible constraints*.

For example average price is not more than 100, aside from “ $\text{avg}(s) \leq v$,” and “ $\text{avg}(s) \geq v$,” there are many other convertible constraints, such as “ $\text{variance}(s) \geq v$,” “ $\text{standard deviation}(s) \geq v$ ”

- For example, “ $\text{sum}(G) \theta v$,” where $\theta \in \{\leq, \geq\}$ and each element in G could be of any real value, which is not convertible and this is called *inconvertible constraints*. Although there still exist some tough constraints that are not convertible. So the SQL aggregate belongs to one of the four categories to which efficient constraint mining methods can be applied.

10.1.14 Classification and Prediction

Databases are rich with hidden information that can be used for making intelligent business decisions. Classification and prediction are two forms of data analysis that can be used to extract models describing important data classes or predict future data trends. Whereas classification predicts categorical labels, prediction models continuous-valued functions.

Data Classification

It is a two-step process and in the first step, a model is built by describing a predetermined set of data classes or concepts. The model is constructed by analyzing database tuples described by attributes. Each tuple is assumed to belong to a predefined class, as determined by one of the attributes, called the class-label attribute. The data tuples analyzed to build the model collectively form the training data set. The individual tuples making up the training set are referred to as training samples and are randomly selected from the same population. Since the class label of each training sample is provided, this step is also known as supervised learning.

In the second step, the model is used for classification. First, the predictive accuracy of the model is estimated. The hold-out method is a simple technique for estimating classifier accuracy that uses a test set of class-labeled samples. These samples are randomly selected and are independent of the training samples as shown in Fig. 10.4. The accuracy of a model on a given test set is

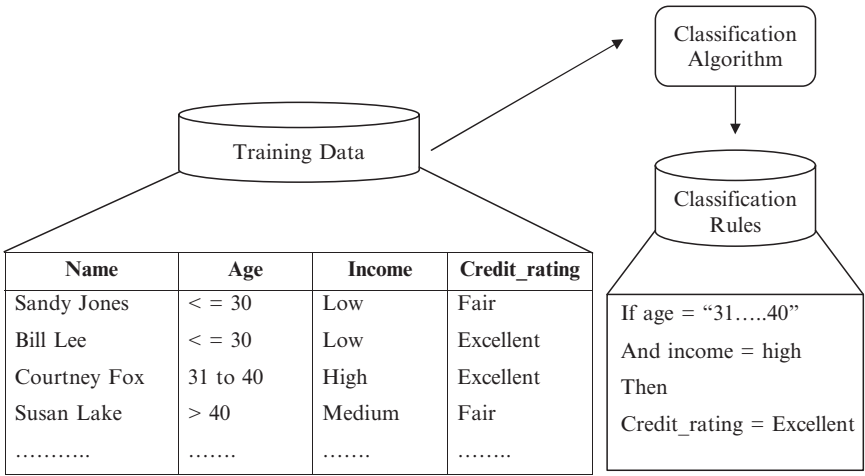


Fig. 10.4. Training data are analyzed by a classification algorithm

the percentage of test set samples that are correctly classified by the model. For each test sample, the known class label is compared with the learned model’s class prediction for that sample.

Prediction

It can be viewed as the construction and use of a model to assess the class of an unlabeled sample, or to assess the value or value ranges of an attribute that a given sample is likely to have. In this view, classification and regression are the two major types of prediction problems where classification is used to predict discrete or nominal values, while regression is used to predict continuous or ordered values.

Preparing the Data for Classification and Prediction

The following steps are applied to the data for improving the accuracy, efficiency, and scalability of the classification or prediction process.

Data Cleaning This refers to the preprocessing of data in order to remove or reduce noise and the treatment of missing values. Although most classification algorithms have some mechanisms for handling noisy or missing data, this step can help reduce confusion during learning.

Relevance Analysis Many of the attributes in the data may be irrelevant to the classification or prediction task. For example, data recording the day of the week on which a bank loan application was filed is unlikely to be relevant to the success of the application. Furthermore, other attributes may be redundant. Hence, relevance analysis may be performed on the data with the aim of removing any irrelevant or redundant attributes from the learning process. In machine process, this step is known as feature selection.

Data Transformation The data can be generalized to higher-level concepts. Concept hierarchies may be used for this purpose. This is particularly useful for continuous-valued attributes. The data may also be normalized, particularly when neural networks or methods involving distance measurements are used in the learning step. Normalization involves scaling all values for a given attribute so that they fall within a small-specified range, such as $-1.0-1.0$ or $0.0-1.0$.

10.1.15 Comparison of Classification Methods

Classification and prediction methods can be compared and evaluated according to the following criteria:

- *Predictive Accuracy*. This refers to the ability of the model to correctly predict the class label of new or previously unseen data.
- *Speed*. This refers to the computation costs involved in generating and using the model.
- *Robustness*. This refers to the ability of the model to make correct predictions given noisy data or data with missing values.
- *Scalability*. This refers to the ability to construct the model efficiently given large amounts of data.
- *Interpretability*. This refers to the level of understanding and insight that is provided by the model.

Classification by Decision Tree Induction

A decision tree is a flow-chart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and leaf nodes represent classes or class distributions. The topmost node is the root node.

In order to classify an unknown sample, the attribute values of the sample are tested against the decision tree. A path is traced from the root to a leaf node that holds the class prediction for that sample. Decision trees can easily be converted to classification rules.

The basic algorithm for decision-tree induction is a greedy algorithm that constructs decision trees in a top-down recursive divide and conquer manner. The basic strategy is as follows:

1. The tree starts as a single node representing the training samples.
2. If the samples are all of the same class, then the node becomes a leaf and is labeled with that class.
3. Otherwise, the algorithm uses an entropy-based measure known as information gain as a heuristic for selecting the attribute that will best separate the samples into individual classes.

4. This attribute becomes the “test” or “decision” attribute at the node. All attributes used in this algorithm are categorized into discrete valued. So, continuous-valued attributes must be discretized.
5. A branch is created for each known value of the test attribute, and the samples are partitioned.
6. The algorithm uses the same process recursively to form a decision tree for the samples at each partition. Once an attribute has occurred at a node, it need not be considered in any of the node’s descendents.
7. The recursive partitioning stops only when any one of the following conditions is true:
 - (a) All samples for a given node belong to the same class.
 - (b) There are no remaining attributes on which the samples may be further partitioned.
 - (c) There are no samples for the branch test-attribute = a_i . in this case, a leaf is created with the majority class in samples.

Tree Pruning

When a decision tree is built, many of the branches will reflect anomalies in the training data due to noise or outliers. Tree-pruning methods address this problem of over fitting the data. Such methods typically use statistical measures to remove the least reliable branches, generally resulting in faster classification and an improvement in the ability of the tree to correctly classify independent test data.

There are two common approaches to tree pruning. The first approach is the prepruning approach; a tree is “pruned” by halting its construction earlier. Upon halting, the node becomes a leaf. The leaf may hold the most frequent class among the subset samples. If partitioning the samples at a node would result in a split that falls below a prespecified threshold, then further partitioning of the given subset is halted. High thresholds would result in over-simplified trees, while low thresholds could result in very little simplification.

The second approach, postpruning, removes branches from a “fully grown” tree. A tree node is pruned by removing its branches. The cost complexity pruning is an example of the postpruning approach. If pruning the node leads to a greater expected error rate, then the subtree is kept. Otherwise, it is pruned. Alternatively prepruning and postpruning may be interleaved for a combined approach. Postpruning requires more computation than prepruning, yet generally leads to a more reliable tree.

Extracting Classification Rules from Decision Trees

The knowledge represented in decision trees can be extracted and represented in the form of classification, IF-THEN rules. One rule is created from each path from the root to a leaf node. Each attribute-value pair along a given path forms a conjunction in the rule antecedent (“IF” part). The leaf node

holds the class prediction, forming the rule consequent (“THEN” part). The IF-THEN rules may be easier for humans to understand, particularly if the given tree is very large.

Scalability and Decision Tree Induction

The efficiency of existing decision tree algorithms, such as ID3 and C4.5, has been well established for relatively small data sets. Efficiency and Scalability become issues of concern when these algorithms are applied to the mining of very large real-world databases. Most decision tree algorithms have the restriction that the training samples should reside in main memory. In data mining applications, very large training sets of millions of samples are common. Hence, this restriction limits the scalability of such algorithms, where the decision tree construction can become inefficient due to swapping of all the training samples in and out of main cache memories.

More recent decision tree algorithms that address the scalability issue have been proposed. Algorithms for the induction of decision trees from very large training sets include SLIQ and SPRINT, both of which can handle categorical and continuous-valued attributes. Both algorithms propose presorting techniques on disk-resident data sets that are too large to fit in memory.

Bayesian Classification

Bayesian classifiers are statistical classifiers. They can predict class membership probabilities, that is a given sample belongs to a particular class. Studies comparing classification algorithms have found a simple Bayesian classifier known as the native Bayesian classifier to be comparable in performance with decision tree and neural network classifiers. Bayesian classifiers have also exhibited high accuracy and speed when applied to large databases.

Bayesian belief networks are graphical models, which unlike native Bayesian classifiers allow the representation of dependencies among subsets of attributes. Bayesian belief networks can also be used for classification. Bayesian classification is based on Bayes theorem, described later.

Bayes Theorem

Assume X is a data sample whose class label is unknown and H be some hypothesis, such that the data sample X belongs to a specified class C . For classification problems, we want to determine $P(H|X)$, the probability that the hypothesis H holds for the given observed data sample X . $P(H|X)$ is the posterior probability, or a posteriori probability, of H conditions on X . In contrast, $P(H)$ is the prior probability, or a priori probability, of H . $P(X)$, $P(H)$ and $P(X|H)$ may be estimated from the given data.

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)}$$

Naive Bayesian Classification

The naive Bayesian classifier, or simple Bayesian classifier, works as follows:

- Each data sample is represented by an n -dimensional feature vector, $X = (x_1, x_2, \dots, x_n)$, depicting n measurements made on the sample from n attributes, respectively, A_1, A_2, \dots, A_n .
- As $P(X)$ is constant for all classes, only $P(X | C_i) P(C_i)$ needs to be maximized. If the class prior probabilities, are not known, then it is commonly assumed that the classes are equally likely, that is $P(C_1) = P(C_2) = \dots = P(C_n)$. Therefore maximize $P(X | C_i)$.

Bayesian Belief Networks

This network classifier makes the assumption of class conditional independence, that is, given the class label of a sample, the values of the attributes are conditionally independent of one another, and it simplifies the computation. When the assumption holds true this network classifier is most accurate in comparison with all other classifiers. This network specifies joint conditional probability distributions and it is defined by two components. The first is directed acyclic graph, where each node represents a random variable and each arc represents a probabilistic dependence. The second component is defining a belief network consists of one Conditional Probability Table (CPT). The CPT for a variable Z specifies the conditional distributions $P(Z | \text{Parents}(Z))$.

Training Bayesian Belief Networks

In the learning or training of this network, a number of scenarios are possible. In the network, structure may be given in advance inferred from the data. The network variables may be observable or hidden in all or some of the training samples. The hidden data are also referred to as missing or incomplete data.

If the network structure is known and the variables are observable, then the training network is straightforward. It consists of computing the CPT entries, as is similarly done when computing the probabilities involved in this network when the network structure is given and some of the variables are hidden then the method of gradient descent can be used to train this network.

Classification Based on Concepts from Association Rule Mining (ARC)

Association rule mining is an important and highly active area of data mining research. Recently, data mining techniques that apply concepts used in association rule mining to the problem of classification have been developed. The clustered association rules generated by ARCS were applied to classification, and then accuracy was compared to C4.5.

The first network mines association rules based on clustering and then employs the rules for classification. In general, ARC is empirically found to

be slightly more accurate than C4.5 when there are outliers in the data. The accuracy of ARCS is related degree of discretization used. The second method is referred to as Associative Classification. It mines rules of the form $\text{condset} \Rightarrow y$, where condset is a set of items and y is a class label. The third method, Classification by Aggregating Emerging Patterns (CAEP), uses the notation of itemset support to mine Emerging Patterns (EP), which are used to construct the classifier. EP is an itemset whose support increases significantly from one class of data to another. The ratio of two supports is called growth rate of EP.

Other Classification Methods

There are several other classification methods, which are discussed as follows:

K -Nearest Neighbors classifiers. This classifier can also be used for prediction, that is, to return a real-valued prediction for a given unknown sample. In this case, the classifier returns the average value of the real-valued labels associated with the k -nearest neighbor of the unknown sample.

Case-Based Reasoning

These classifiers are instance based. The case-based reasoned may employ background knowledge and problem-solving strategies in order to propose a feasible combined solution. It include finding a good similarity metric, developing efficient techniques for indexing training cases, and methods for combining solutions.

Genetic Algorithm

Genetic algorithms attempt to incorporate ideas of natural evolution. It is easily parallelizable and has been used for classification as well as optimization problems. In data mining, they may be used to evaluate the fitness of other algorithms.

Rough Set Approach

Rough set theory is based on the establishment of equivalence classes within the given training data. All of the data samples forming an equivalence class are indiscernible, that is, the samples are identical with respect to the attributes describing the data. This Rough set can also be used for feature reduction and relevance analysis, however, algorithms to reduce the computation intensity have been proposed.

Fuzzy Set Approach

Rule-based Systems for classification have the disadvantage that they involve sharp cutoffs for continuous attributes, so fuzzy logic is useful for data mining systems performing classification. It provides the advantage of working at

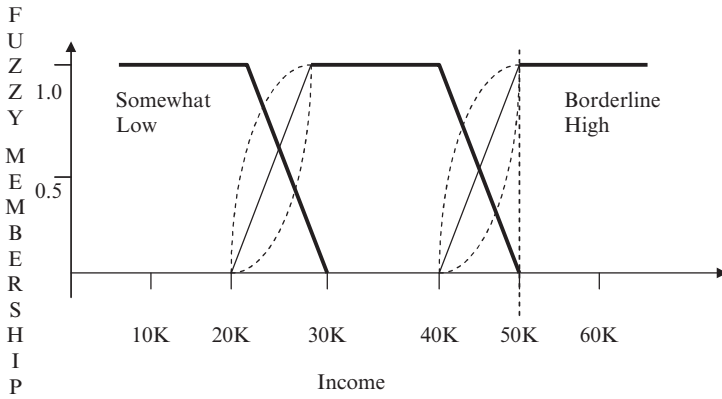


Fig. 10.5. Fuzzy values for income

a high level of abstraction. For a given new sample, more than one fuzzy rule may apply. Each applicable rule contributes a vote for membership in the categories as illustrated with the example of income in Fig. 10.5. Typically, the truth-values for each predicted category are summed.

10.1.16 Prediction

The prediction of continuous values can be modeled by statistical techniques of regression. Many problems can be solved by linear regression, and even more can be tackled by applying transformations to the variables so that a nonlinear problem can be converted to linear one.

Linear and Multiple Regression. In linear regression, data are modeled using a straight line. It is a simplest form of regression. Bivariate linear regression models a random variable, y (response variable), as a linear function of another random variable, x (predictor variable)

$$Y = \alpha + \beta x$$

where the variance of y is assumed to be constant, and α and β are regression coefficients.

Multiple Regression

It is an extension of linear regression involving more than one predictor variable. It allows response variable y to be modeled as a linear function of a multidimensional feature vector

$$Y = \alpha + \beta_1 X_1 + \beta_2 X_2$$

The method of leastsquares can also be applied here to solve for α , β_1 , β_2 .

Nonlinear Regression

Polynomial regression can be modeled by adding polynomial terms to the basic linear model. By applying transformation to the variables, we convert the nonlinear model into linear one that can then be solved by method of least squares.

Other Regression Models

Linear regression is used to model continuous valued functions. It is widely used, owing largely to its simplicity. Generalized linear model represent the theoretical foundation on which linear regression can be applied to the modeling of categorical response variables. Common types of generalized linear models include logistic and poisson regression. Logistic regression models the probability of some event occurring as a linear function of set of predictor variables. Count data frequency exhibits a poisson distribution and is commonly modeled using poisson regression. Log linear models approximate discrete multidimensional probability distribution. They may be used to estimate the probability value associated with the data cube cells.

10.1.17 Cluster Analysis

Imagine that we are given a set of data objects for analysis where, unlike in classification, the class label of each object is not known. Clustering is the process of grouping the data into classes or clusters so that objects within a cluster have high similarity in comparison to one another, but are very dissimilar to objects in other clusters. Dissimilarities are assessed based on the attribute values describing the objects.

The process of grouping a set of physical or abstract objects into classes of similar objects is called clustering. A cluster is a collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in other clusters. A cluster of data objects can be treated collectively as one group in many applications.

Some typical application of clustering are in business, clustering can help marketers discover distinct groups in their customer bases and characterize customer groups based on purchasing patterns. In biology, it can be used to derive plant and animal taxonomies, categorize genes with similar functionality, and gain insight into structures inherent in populations.

As a data mining function, cluster analysis can be used as a stand-alone tool to gain insight into the distribution of data, to observe the characteristics of each cluster, and to focus on a particular set of clusters for further analysis.

Data clustering is under vigorous development. Contributing areas of research include data mining, statistics, machine learning, spatial database technology, biology, and marketing.

Cluster analysis tools based on k-means, k-medoids, and several other methods have also been built into many statistical analysis software packages or systems, such as S-plus, SPSS, and SAS. Clustering is an example for unsupervised learning.

Requirements of Clustering in Data Mining

Scalability. Many clustering algorithms work well on small data sets containing fewer than 200 data objects; however, a large database may contain millions of objects. Clustering on a sample of a given large set may lead to biased results. Highly scalable clustering algorithms are needed.

Ability to deal with different types of Attributes. Many algorithms are designed to cluster interval-based (numerical) data. However, applications may require clustering other types of data, such as binary, categorical (nominal), and ordinal data, or mixtures of these data types.

Discovery of Clusters with Arbitrary Shape

Many clustering algorithms determine clusters based on Euclidean or Manhattan distance measures. Algorithms based on such distance measures tend to find spherical clusters with similar size and density. However, a cluster could be of any shape. It is important to develop algorithms that can detect clusters of arbitrary shape.

Minimal Requirements for Domain Knowledge to Determine Input Parameters

Many clustering algorithms require users to input certain parameters in cluster analysis (such as the number of desired clusters). The clustering results can be quite sensitive to input parameters. Parameters are often hard to determine, especially for data sets containing high-dimensional objects. This not only burdens users, but also makes the quality of clustering difficult to control.

Ability to Deal with Noisy Data

Most real-world databases contain outliers or missing, unknown, or erroneous data. Some clustering algorithms are sensitive to such data and may lead to clusters of poor quality.

Insensitivity to the Order of Input Records

Some clustering algorithms are sensitive to the order of input data; for example, the same set of data, when presented with different orderings to such an algorithm, may generate dramatically different clusters. It is important to develop algorithm that are insensitive to the order of input.

High Dimensionality

A database or a data warehouse can contain several dimensions or attributes. Many clustering algorithms are good at low-dimensional data, involving only two or three dimensions. Human eyes are good at judging the quality of clustering for up to three dimensions. It is challenging to cluster data objects in high-dimensional space, especially considering that such data can be very sparse and highly skewed.

Types of Data in Cluster Analysis

A data set to be clustered contains n objects, which may represent persons, houses, documents, countries, and so on. Therefore, we are going to study the types of data that often occur in cluster analysis and preprocess of them. Main memory-based clustering algorithms typically operate on either of the following two data structures.

1. *Data matrix (objects-by-variable structure)*. Data matrix represents n objects, such as persons, with p variables (also called measurements or attributes), such as age, height, weight, gender, race, and so on. The structure is in the form of a relational table, or n -by- p matrix (n objects \times p variables) is shown later in (10.1)

$$\begin{bmatrix} X_{11} & \dots & X_{1f} & \dots & X_{1p} \\ \vdots & & \vdots & & \vdots \\ X_{i1} & \dots & X_{if} & \dots & X_{ip} \\ \vdots & & \vdots & & \vdots \\ X_{N1} & \dots & X_{Nf} & \dots & X_{Np} \end{bmatrix} \quad (10.1)$$

2. *Dissimilarity matrix (or object-by-object structure)*. Dissimilarity matrix stores a collection of proximities that are available for all pairs of n objects. It is represented by n -by- n table is shown later in (10.2)

$$\begin{bmatrix} 0 & & & & & \\ d(2,1) & 0 & & & & \\ d(3,1) & d(3,2) & 0 & & & \\ \vdots & \vdots & \vdots & & & \\ d(n,1) & d(n,2) & \dots & \dots & 0 & \end{bmatrix} \quad (10.2)$$

where $d(i, j)$ is the measured difference or dissimilarity between objects i and j .

The data matrix is often called a one-node matrix, where as the dissimilarity matrix is called a one-mode matrix since the rows and columns of the former represent different entities, while those of the latter represent the same entity. Many clustering algorithms operate on a dissimilarity matrix. If the data are presented in the form of a data matrix, it can first be transformed into dissimilarity matrix before applying such clustering algorithms.

Computation of Dissimilarity

1. *Interval-Scaled Variables.* It describes distance measures that are commonly used for computing the dissimilarity of objects described by such variables. These measures include the Euclidean, Manhattan, and Minkowski distances.

These variables are continuous measurements of a roughly linear scale. Typical examples include weight, height, latitude and longitude, and weather temperatures. The measurement unit used can affect the clustering analysis. For example, changing measurement units from meters to inch for height, or from kilograms to pounds for weight, may lead to a very different cluster in structure.

To help avoid dependence on the choice of measurement units, the data should be standardized. The standardizing measurement attempts to give all variables an equal weight. In the standardizing measurement, one choice is to convert the original measurements to unit less variable (f):

1. The mean absolute deviation (s_f) is calculated by:

$$s_f = \frac{1}{n} (|x_{1f} - m_f| + |x_{2f} - m_f| + \dots + |x_{nf} - m_f|)$$

where X_{1f}, \dots, X_{nf} are n measurements of f , and m_f is the mean value of f .

$$m_f = \frac{1}{n} (x_{1f} + x_{2f} + \dots + x_{nf})$$

2. The standardized measurement or z score is calculated by:

$$z_{if} = \frac{x_{if} - m_f}{s_f}$$

The interval-scaled variables are typically computed based on distance between each pair of objects. The most popular distance measure is Euclidean distance, which is defined by:

$$d(i, j) = \sqrt{|x_{i1} - x_{j1}|^2 + |x_{i2} - x_{j2}|^2 + \dots + |x_{ip} - x_{jp}|^2}$$

where $i = (X_{i1}, X_{i2}, \dots, X_{ip})$ and $j = (X_{j1}, X_{j2}, \dots, X_{jp})$ are two p -dimensional objects

2. *Binary Variable.* A binary variable has only two states: 0 or 1 where 0 means the variable is absent, and 1 means the variable is present. Given the variable smoker describing a patient, for instance, 1 indicates that the patient smokes, while 0 indicates the patient does not. Treating binary variables as if they are interval scaled can lead to misleading clustering results. Therefore, these methods specific to binary data are necessarily for computing dissimilarities.

The binary variable is symmetric if both of its states are equally valuable and carry the same weight; that is, there is no preference on which outcome should be coded as 0 or 1. Similarity based on symmetric binary variables is called invariant similarity in that the result does not change when some or all of the binary variables are coded differently.

A binary variable is asymmetric if the outcomes of the states are not equally important, such as the positive and negative outcomes of a disease test. It is usually the rarest one by 1 (HIV positive) and other by 0 (HIV negative).

3. *Nominal Variables.* A nominal variable is a generalization of the binary variables in that it can take one more than two states. Dissimilarity between two objects i and j can be computed using simple matching approach,

$$d(i, j) = \frac{p - m}{p}$$

where m is the number of matches, p is the total number of variables.

Nominal variables can be encoded by asymmetric binary variables by creating a new binary variable for each of the M nominal state. For an object with a given state value, the binary variable representing that state is set to 1, while the remaining binary variables are set to 0.

4. *Ordinal Variables.* A discrete ordinal variable resembles a nominal variable, except that the M states of the ordinal value are ordered in a meaningful sequence. It is very useful for registering subjective assessments of qualities that cannot be measured objectively. For example, professional ranks are often enumerated in a sequential order, such as assistant, associate, and full.

A continuous ordinal variable looks like a set of continuous data of an unknown scale that is the relative ordering of the values is essential but not their actual magnitude. The treatment of ordinal variable is quite similar to that of interval-scaled variables when computing the dissimilarity between objects.

5. *Ratio-Scaled Variables.* A ratio-scaled variable makes a positive measurement on a nonlinear scale such as exponential scale by:

$$Ae^{BT} \text{ or } Ae^{-BT},$$

where A and B are positive constants. Typical examples include the growth of a bacteria population, or the decay of radioactive element.

There are three methods to handle the ratio scaled variable for computing the dissimilarity between the objects:

- Treat ratio-scaled variables like interval scaled variables. This is not usually a good choice since it is likely that scale may be distorted.
- Apply logarithmic transformation to a ratio-scaled variable f having value x_{if} for object i by using the formula $y_{if} = \log(x_{if})$.
- Treat x_{if} as continuous ordinal data and treat the ranks as interval-valued.

Major Clustering Methods

There exist a large number of clustering algorithms in the literature. The choice of clustering algorithm depends both on the type of data available and on the particular purpose and application. In general, major clustering methods can be classified as follows

1. *Partitioning methods.* Given a database of n objects and k the number of cluster to form a partitioning algorithm organizes the objects into k partitions ($k \leq n$), where each partition represents a cluster. The clusters are formed to optimize an objective-partitioning criterion, often called a similarity function, such as distance, so that the objects within a cluster are “similar,” whereas the objects of different clusters are “dissimilar” in terms of database attributes

Algorithm

The K-means algorithm for partitioning based on the mean value of the objects in the cluster

Input: The number of clusters k and a database containing n -objects.

Output: A set of k clusters that minimizes a squared error criterion.

Method:

1. Arbitrarily choose k objects as the initial cluster centers.
2. Repeat.
3. Assign/Reassign each object to the cluster to which the object is the most similar based on the mean value of the objects in the cluster.
4. Update the cluster means value (Calculate the mean value of each cluster).
5. Until no change.

The k means algorithm takes the input parameter k and partitions the set of n objects into k clusters, so that the resulting intercluster similarity is high but the intercluster similarity is low. Cluster similarity is measured in regard to the mean value of the objects in a cluster, which can be viewed as a cluster center of gravity. Typically, the squared error is defined as:

$$E = \sum_{i=1}^k \sum_{p \in C_i} |p - m_i|^2$$

where E is the sum of squared error for all objects in the database, p is point in space representing in a given object, and m_i is mean of cluster C_i . The k -means method is not suitable for discovering with nonconvex shape or clusters of very different size. Moreover, it is sensitive to noise and outlier data points since a small number of such data can substantially influence the mean value.

2. *Partitioning Around Medoids (PAM)*. It is the one of the first k-medoids algorithms introduced. It attempts to determine k partition for n objects. After an initial random selection of k-medoids the algorithm repeatedly tries to make a better choice of medoids.

Algorithm: k-Medoids algorithms for partitioning based on method or central objects.

Input: The number of clusters k and a database containing n objects.

Output: In a set of k cluster that minimize the sum of the dissimilarities of all the objects to the nearest medoids.

Methods

1. Arbitrarily choose k objects as the initial medoids.
2. Repeat.
3. Assign each remaining object to the cluster to the nearest medoids.
4. Randomly select the nonmedoid object, O_{random} .
5. Complete the total cost S, of swapping O_j with O_{random} .
6. If $S < 0$ then swap O_j with O_{random} to form the new set of k-medoids.
7. Until no change.

The set of best objects for each cluster in a single iteration forms the medoids for the next iteration. For large values of n and k, such computation becomes very costly. The k medoids method is more robust than the k means in the presence of noise and outliers because the medoids is less influenced by outliers or other extreme values than a mean. However it is processing is more costly than the k means method. Both methods require the user to specify k, the number of clusters.

3. *Partitioning Methods in Large Database*. A typical k medoids-partitioning algorithm like PAM works effectively for small data set, but does not scale well for large data sets. To deal with large data sets, a sampling-based method called Clustering LARge Application (CLARA) can be used.

The effectiveness of CLARA depends on sample size to notice that PAM searches for the best k-medoids among given data set, whereas CLARA searches for the best k-medoids among the selected sample of the data set. A k-medoids type algorithm called Clustering Large Application based upon Randomized Search (CLARANS) was proposed that combines the sampling technique with PAM.

However unlike, CLARA, CLARANS does not confine itself to any sample at any given time. While CLARA has fixed sample at each stage of search, CLARANS draws sample with some randomness in each step of search, it has been experimentally shown to be more effective than PAM and CLARA. It can be used to find the most “Natural” number of clusters using a silhouette coefficient – a property of an object that specifies how much the object truly belongs to the cluster.

4. *Hierarchical Methods*. A hierarchical clustering method works on the grouping data objects into a tree of clusters. This method can be further

classified into agglomerative and divisive hierarchical clustering, depending on whether the hierarchical decomposition is formed in a bottom-up or top-down fashion. The quality of this method suffers from its inability to perform adjustment once a merge or split decision has been executed.

Agglomerative Hierarchical Clustering. This bottom-up strategy starts by placing each object in its own cluster and then merges these atomic clusters into larger clusters, until all of the objects are in a single cluster or until certain termination conditions are satisfied. Most hierarchical clustering methods belong to this category. They differ only in their definition of intercluster similarity.

Divisive Hierarchical Clustering. This top-down strategy does the reverse of agglomerative hierarchical clustering by starting with all objects in one cluster. It subdivides the cluster into smaller pieces, until each object forms a cluster on its own or until it satisfies certain termination conditions, such as a desired number of clusters is obtained or the distance between the two closest clusters is above a certain threshold distance.

5. *CURE: Clustering Using Representatives.* Most clustering algorithms either favor clusters with spherical shape and similar sizes, or are fragile in the presence of outliers. CURE overcomes this problem by using more robust spherical shapes and similar sizes with respect to their outliers. To handle large databases, CURE employs a combination of random sampling and partitioning, a random sample is first partitioned, and each partition is partially clustered. The partial clusters are then clustered in a second pass to yield the desired clusters. The following steps outline the spirit of the CURE algorithm:
 1. Draw a random sample, S , of the original objects.
 2. Partition sample S into a set of partitions.
 3. Partially cluster each partition.
 4. Eliminate outliers by random sampling. If a cluster grows too slowly, remove it.
 5. Cluster the partial clusters. The representative points falling in each newly formed cluster are “shrunked” or moved toward the cluster center by a user specified fraction, or shrinking factor, α . These points then represent and capture the shape of the cluster.
 6. Mark the data with the corresponding cluster labels.

10.1.18 Mining Complex Types of Data

An increasingly important task in data mining is to mine complex types of data, including complex objects, spatial data, multimedia data, and worldwide data. To further, develop the essential data mining techniques (such as Characterization, Association) and how to develop new ones to cope with complex types of data and perform fruitful knowledge mining in complex information repositories.

Multidimensional Analysis and Descriptive Mining of Complex Data Objects

To introduce data mining and multidimensional data analysis for complex objects, this examines how to perform generalization on complex structured objects and construct object cubes for OLAP and mining in object databases. This system organizes a large set of complex data objects into classes, which are in turn organized into class/subclass hierarchies. Each object in a class is associated with an object identifier, a set of attributes that may contain sophisticated data structures, set or list value data, class composition hierarchies, multimedia data, and a set of methods that specify the computational routines or rules associated with the object class.

Generalization of Structured Data

An important feature of object relational and object oriented database are their capability of storing, accessing, and modeling complex structured valued data, such as set valued and list valued data.

A set valued attribute may be of homogeneous or heterogeneous type. Typically, this can be generalized by generalization of each value in the set into its corresponding higher level concepts and derivation of the general behavior of the set, such as number of elements in the types or value ranges in the set or weighted average for numerical data.

A list valued or sequence valued attribute can be generalized in a manner similar to that for set valued attributes except that the order of the elements in the sequence should be observed in the generalization. Each value in the list can be generalized into its corresponding higher-level concepts.

Generalization of Object Identifiers and Class/Subclass Hierarchies

Objects in an object oriented databases are organized into classes, which in turn are organized into class/subclass hierarchies. The generalization of an object can be performed by referring to its associated hierarchy. The object identifier is generalized to the identifier of the lowest subclass to which the object belongs. The identifier of the subclass can be generalized into higher-level class/subclass identifier by climbing up the class/subclass hierarchy. This method is usually defined by a computational procedure with a set of deduction rules; it is impossible to perform generalization on the method itself.

Construction and Mining of Object Cubes

Consider that a generalization of base data mining process can be view as application of a sequence of class-based generalization operators on different attributes. Generalization can continue until the resulting class contains a small number of generalized objects that can be summarized as a concise generalized rule in high-level terms. For efficient implementation, the generalization of multidimensional attributes of a complex objects can be performed

by examining each attribute, generalizing each attribute to simple valued data and constructing multidimensional data cube called an object cube.

Mining Spatial Databases

Spatial data mining refers to the extraction of knowledge, spatial relationship, or other interesting patterns not explicitly stored in spatial databases. It can be used for understanding spatial data, discovering spatial relationship and relationship between spatial and nonspatial data, constructing spatial knowledge bases, reorganizing spatial data spaces, and optimizing spatial queries.

A crucial challenge to spatial data mining is the exploration of efficient spatial data mining techniques due to the huge amount of spatial data the complexity of spatial data types, and spatial access methods. Spatial data mining allows the extension of traditional spatial analysis method by placing emphasis on efficiency, scalability, co-operation with database systems, improved interaction with the user, and discovery of new types of knowledge.

Spatial Association Analysis

A Spatial association rule is of the form $A \Rightarrow B [s\%, c\%]$, where A and B are sets of spatial or nonspatial predicates, $s\%$ is the support of the rule, and $c\%$ is the confidence of the rule.

An interesting mining optimization method called progressive refinement can be adapted in spatial association analysis. This method first mines large data sets roughly using a fast algorithm and then improves the qualities of mining in a pruned data sets using a more expensive algorithm, an important requirement for the rough mining algorithm applied in the early stage superset coverage property that is it preserve all of the potentials answers. For mining spatial association related to the spatial predicate, we can first collect the candidate that pass the minimum of threshold by:

- Applying certain rough spatial evolution algorithm
- Evaluating the relax spatial predicate

Mining Multimedia Databases

The multimedia database system stores and manages a large collection multimedia objects, such as audio data, image data, video data sequence data, and hypertext data, which contains text, text markups and linkages. Multimedia database systems are increasingly common owing to the popular use of audio-video equipment, CD-ROMs, and the Internet.

Classification and Prediction Analysis of Multimedia Data

Classification and prediction modeling have been used for mining multimedia data, especially in scientific research, such as astronomy, seismology, and

geoscientific research. Decision tree classification is an essential data mining method in reported image data mining applications.

Data preprocessing is important when mining such image data and can include data cleaning, data focusing, and feature extraction. The image data are often in huge volumes and may require substantial processing power, parallel, and distributed processing. Image data mining classification and clustering are closely linked to image analysis and scientific data mining, and thus many image analysis techniques and scientific data analysis methods can be applied to image data mining.

Mining Associations in Multimedia Data

Association rules involving multimedia objects can be mined in image and video databases. At least three categories should be observed as follows:

1. *Association between image content and nonimage content features.* A rule like “If at least 50% of the upper part of the picture is blue, it is likely to represent sky” belongs to this category since it links the image content to the keyword sky.
2. *Association among image contents that are not related to spatial relationships.* A rule like “If a picture contains two blue squares, it is likely to contain one red circle as well” belongs to this category since the associations are regarding image contents.
3. *Association among image contents related to spatial relationships.* A rule like “If a red triangle is in between two yellow squares, it is likely there is a big oval shaped underneath” belongs to this category since it associates objects in the image with spatial relationships.

A progressive resolution refinement approach is essential in mining the multimedia data. We should first mine frequently occurring patterns at a relatively rough resolution level, and then focus only on those that have passed the minimum threshold when mining at a finer resolution level. The efficiency will be improved because the overall data mining cost is reduced without loss of the quality.

Secondly, the picture containing multiple recurrent objects is an important feature in image analysis, recurrence of the same objects should not be ignored in association analysis. Therefore, the definition of multimedia association and its measurements, such as support and confidence, should be adjusted accordingly.

Thirdly, there exists an important spatial relationship among multimedia objects, such as beneath, above, between, nearby, left-of, and so on. These features are very useful for exploring object associations and correlations. Thus, spatial data mining methods and properties of topological spatial relationships become quite important for multimedia mining.

Mining the World Wide Web

The Web contains a rich and dynamic collection of hyperlink information, Web page access, and usage information, providing rich sources for data mining. The Web also poses great challenges for effective resource and knowledge discovery that are as follows:

- The Web seems to be huge for effective data mining
- The complexity of pages is greater than that of traditional text document collection
- The Web is a highly dynamic information source
- The Web serves a broad diversity of user communities
- Only a small portion of the information on the Web is truly relevant or useful

Web Usage Mining

Besides mining Web contents and Web linkage structures, another important task for Web mining is Web usage mining, which mines Web log records to discover user access patterns of Web pages. Analyzing and exploring regularities in Weblog records can identify potential customers for electronic commerce, enhance the quality and delivery of Internet information services to the end user, and improve Web server system performance. In developing techniques for Web usage mining, we should consider three important factors.

First, it is important to know about the application and it depends on what and how much valid and reliable knowledge can be discovered from the large raw log data. Often, raw Weblog data need to be cleaned, condensed, and transformed in order to retrieve and analyze significant and useful information.

Second, with the available URL, time, IP address, and Web page content information, a multidimensional view can be constructed on the Weblog database, and multidimensional OLAP analysis can be performed to find the top N users, top N accessed Web pages, most frequently accessed time periods, and so on, which will help discover potential customers, users, markets, and others.

Third, data mining can be performed on Weblog records to find association patterns, sequential patterns, and trends of Web accessing. For Web access pattern mining, it is often necessary to take further measures to obtain additional information of user traversal to facilitate detailed Weblog analysis. Such additional information may include user-browsing sequences of the Web pages in the Web server buffer, and so on.

10.1.19 Applications and Trends in Data Mining

Data mining is a young discipline with wide and diverse applications, there is still a nontrivial gap between general principles of data mining and domain specific for effective data mining tools for particular applications.

Data Mining for Biomedical and DNA Data Analysis

Biomedical research, ranges from the development of pharmaceutical and advances in cancer therapies, identification, and study of the human genome by discovering large-scale sequencing patterns. A gene is usually comprised of hundreds of individual nucleotides arranged in a particular order. There are almost an unlimited number of ways that the nucleotides can be ordered and sequenced to form distinct genes. Data Mining has become a powerful tool and contributes substantially to DNA analysis in the following ways.

Semantic Integration of Heterogeneous, Distributed Genome Databases

Due to the highly distributed, uncontrolled generation and use of a wide variety of DNA data, the semantic integration of such heterogeneous and widely distributed genome databases becomes an important task for systematic and coordinated analysis of DNA databases. Data cleaning and data integration methods will help the integration of genetic data and the construction of data warehouses for genetic data analysis.

Similarity Search and Comparison Among DNA Sequences

One of the most important search problems in genetic analysis is similarity search and comparison among DNA sequences. The techniques needed here is quite different from that used for time series data: For example, data transformation methods such as scaling, normalization, and window stitching, which are popularly used in the analysis of time series data, are ineffective for genetic data.

Association Analysis: Identification of Co-occurring Gene Sequences

Association analysis methods can be used to help determine the kinds of genes that are likely to co-occur in target samples. Such analysis would facilitate the discovery of groups of genes and the study of interactions and relationships between them.

Path Analysis: Linking Genes to Different Stages of Disease Development

While a group of genes may contribute to a disease process, different genes may become active at different stages of the disease, therefore achieving more effective treatment of the disease. Such path analysis is expected to play an important role in genetic studies.

Visualization Tools and Genetic Data Analysis

Complex structures and sequencing patterns of genes are most effectively presented in graphs, trees, cuboids, and chains by various kinds of visualization tools. Visualization therefore plays an important role in biomedical data mining.

Data Mining for Financial Data Analysis

Financial data collected in the banking and financial industry is often relatively complete, reliable, and of high quality, which facilitates systematic data analysis and data mining. Here we present a few typical cases:

Loan payment prediction and customer credit policy analysis

Loan payment prediction and customer credit analysis are critical to the business of a bank. Many factors can strongly or weakly influence loan payment performance and customer credit rating. Data mining methods, such as feature selection and attribute relevance ranking, may help identify important factors and eliminate irrelevant ones.

Classification and clustering of customers for customer group identification targeted marketing

Customers with similar behaviors regarding banking and loan payments may be grouped together by multidimensional clustering techniques. Effective clustering and collaborative filtering methods can help identify customer groups, associate a new customer with an appropriate customer group, and facilitate targeted marketing.

Data Mining for the Retail Industry

The retail industry is a major application area for data mining since it collects huge amounts of data on sales, customer shopping history, goods transportation, consumption and service records, and so on. The quantity of data collected continues to expand rapidly, especially due to the increasing ease, availability, and popularity of business conducted on the Web; or e-commerce.

Retail data mining can help identify customer buying behaviors, discover customer shopping patterns and trend, improve the quality of customer service, achieve better customer retention and satisfaction, enhance goods consumption ratios, design more effective goods transportation and distribution policies, and reduce the cost of business.

A few examples of data mining in the retail industry are:

- Design and construction of data warehouses based on the benefits of data mining

- Multidimensional analysis of sales, customers, products, time, and religion
- Analysis of the effectiveness of sales campaigns
- Customer retention – analysis of customer loyalty
- Purchase recommendation and cross reference of items

Data Mining for the Telecommunication Industry

The telecommunication industry has quickly evolved from offering local and long distance telephone services for providing many other comprehensive communication services including voice, fax, pager, cellular phone, images, e-mail, computer and Web data transmission, and other data traffic. The integration of telecommunication, computer network, Internet, and numerous other means of communication and computing is also underway.

The following are a few scenarios where data mining may improve telecommunication services.

Multidimensional analysis of telecommunication data

Telecommunication data are intrinsically multidimensional with dimensions such as calling time, duration, and location of caller, location of called, and type of call. The multidimensional analysis of such data can be used to identify and compare the data traffic, system workload, resource usage, user group behavior, profit, and so on.

Multidimensional association and sequential pattern analysis

The discovery of association and sequential patterns in multidimensional analysis can be used to promote telecommunication services. The calling records may be grouped by customer in the following form:

(Customer_id, residence, office, time, date, service_1, service_2, ...)

A sequential pattern can help to promote the sales of specific long distance and cellular phone combinations, and improve the availability of particular services in the region.

Use of visualization tools in telecommunication data analysis

Tools for OLAP, linkage, association, clustering, and outlier visualization have been shown to be very useful for telecommunication and data analysis.

10.1.20 How to Choose a Data Mining System

To choose a data mining system that is appropriate for your task, it is important to have a multiple dimensional view of data mining systems. In general, data mining systems should be assessed based on the following multiple dimensional features.

Data Types

Most data mining systems that are available on the market handle formatted, record-based, relational-like data with numerical, categorical, and symbolic attributes. The data could be in the form of ASCII text, relational database data, or data warehouse data. It is important to check what exact format(s) each system we are considering can handle. Moreover, many data mining companies offer customized data mining solutions that incorporate essential data mining functions or methodologies.

System Issues

A given data mining system may run on only one or several operating systems. The most popular operating systems that host data mining software are UNIX and Microsoft Windows (including 95, 98, 2000, and NT). There are also data mining systems that run on OS/2, Macintosh, and Linux.

Data Mining Functions and Methodologies

Data mining functions form the core of a data mining system. Some data mining systems provide only one data mining function, such as classification. Others may support multiple data mining functions, such as description, discovery-driven OLAP analysis, association, classification, prediction, clustering, outlier analysis, similarity search, sequential pattern analysis, and visual data mining.

Coupling Data Mining with Database and/or Data Warehouse Systems

A data mining system should be coupled with a database and/or data warehouse system, where the coupled components are seamlessly integrated into a uniform information-processing environment.

Ideally, a data mining system should be tightly coupled with a database system in the sense that the data mining and data retrieval processes are integrated by optimizing data mining queries deep into the iterative mining and retrieval process. Tight coupling of data mining with OLAP-based data warehouse systems is also desirable so that data mining and OLAP operations can be integrated to provide OLAP mining features.

Scalability. Data mining has two kinds of scalability issues: row (or database size) and column (or dimension) scalability. A data mining system is considered row scalable if, when the number of rows is enlarged ten times, it takes no more than ten times to execute the same data mining queries. A data mining system is considered column scalable than row scalable.

Visualization tools. “A picture is worth a thousand words” – this is very true in data mining. Visualization in data mining can be categorized into data visualization, mining result visualization, mining process visualizations,

and visual data mining. The variety, quality, and flexibility of visualization tools may strongly influence the usability, interpretability, and attractiveness of a data mining system.

10.1.21 Theoretical Foundations of Data Mining

Research on the theoretical foundations of data mining has yet to mature. A solid and systematic theoretical foundation is important because it can help provide a coherent framework for the development, evaluation, and practice of data mining technology. There are a number of theories for the basis of data mining, such as the following.

Data Reduction. In this theory, the basis of data mining is to reduce the data representation. Data reduction trades accuracy for speed in response to the need to obtain quick approximate answers to queries on very large databases. Data reduction techniques include singular value decomposition (the driving element behind principal components analysis), wavelets, regression, long-linear models, histograms, clustering sampling, and the construction of index trees.

Data Compression. According to this theory, the basis of data mining is to discover patterns occurring in the database, such as associations, classification models, sequential patterns, and so on.

Profitability Theory. This is based on statistical theory. In this theory, the basis of data mining is to discover joint probability distributions of random variables, for example, Bayesian belief networks or hierarchical Bayesian models.

Microeconomic Views. The microeconomic view considers data mining as the task of finding patterns that are interesting only to the extent that they can be used in the decision-making process of some enterprise (e.g., regarding marketing strategies, production plans, etc.). This view is one of the utilities in which patterns are considered interesting if they can be acted on. Enterprises are regarded as facing optimization problems where the object is to maximize the utility or value of a decision. In this theory, data mining becomes a nonlinear optimization problem.

Inductive Databases. According to this theory, a database scheme consists of data and patterns that are stored in the database. Data mining is therefore the problem of performing induction on databases, where the task is to query the data and the theory (i.e., patterns) of the database. This view is popular among many researchers in database systems.

Is Data Mining a Threat to Privacy and Data Security?

With more information accessible in electronic forms and available on the Web, and with increasingly powerful data mining tools being developed and

put into use. Since data mining may disclose patterns and various kinds of knowledge that are difficult to find otherwise, it may pose a threat to privacy and information security if not done or used properly.

Most consumers do not mind providing companies with personal information if they think it will enable the companies to better service their needs.

“Will the data be sold to other companies?”

“Can I find out what is recorded about me?”

“Will the information about me be “anonymized,” or will it be traceable to me ?”

“How to secure are the data?”

“How accountable is the company who collects or stores my data, if these data are stolen or misused?”

This includes the following principles:

Purpose Specification and Use Limitation

The purposes for which personal data are collected should be specified at the time of collection, and the data collected should not exceed the stated purpose. Data mining is typically a secondary purpose of the data collection.

Openness

Individuals have the right to know what information is collected about them who have access to the data, and how the data are being used.

Companies should provide consumers with multiple opt-out choices, allowing consumers to specify limitations on the use of their personal data, such as (1) the consumer’s personal data are not to be used at all for data mining; (2) the consumer’s data can be used for data mining, but the identity of each consumer or any information that may lead to disclosure of a person’s identity should be removed.

The field of database systems was initially met with some opposition, as many individuals were concerned about the security risks associated with large on-line data storage. Many data security-enhancing techniques have been developed so that, although some “hacker infractions” do occur, people are generally secure about the safety of their data and now accept the benefits offered by database management systems. Such data security enhancing techniques can be used to anonymous information and securely protect privacy in data mining.

Data mining may pose a threat to our privacy and data security. However, as we have seen, many solutions are being developed to help prevent misuse of the data collected. In addition, the field of database systems has many data security enhancing techniques that can be used to guard the security of data collected for and resulting from data mining.

Trends in Data Mining

The diversity of data, data mining tasks, and data mining approaches poses many challenging research issues in data mining. The design of data mining languages, the development of efficient and effective data mining methods and systems, the construction of interactive and integrated data mining environments, and the application of data mining techniques to solve large application problems are important tasks for data mining researchers and data mining system and application developers.

Application Exploration

Early data mining applications focused mainly on helping businesses gaining competitive edge. As data mining became more popular, it is increasingly used for the exploration of applications in other areas, such as biomedicine, financial analysis, and telecommunications.

Scalable data mining methods. In contrast with traditional data analysis methods, data mining must be able to handle huge amounts of data efficiently and, if possible, interactively. Since the amount of data, being collected continues to increase rapidly, scalable algorithms for individual and integrated data mining functions become essential.

Integration of data mining with database, data warehouse, and Web database systems. Database systems, data warehouse systems, and the WWW have become mainstream information processing systems. It is important to ensure that data mining serves as an essential data analysis component that can be smoothly integrated into such an information-processing environment. This will ensure data availability, data mining portability scalability, high performance, and an integrated information-processing environment for multi-dimensional data analysis and exploration.

Standardization of data mining language. A standard data mining language or other standardization efforts will facilitate the systematic development of data mining solutions, improve interoperability among multiple data mining systems and functions, and promote the education and use of data mining systems in industry and society.

Visual data mining. Visual data mining is an effective way to discover knowledge from huge amounts of data. The systematic study and development of visual data mining techniques will facilitate the promotion and use of data mining as a tool for data analysis.

New methods for mining complex types of data. Mining complex types of data are an important research frontier in data mining. Although progress has been made at mining geospatial, multimedia, time-series, sequence, and text data, there is still a huge gap between the needs for these applications and the available technology.

Web mining. The huge amounts of information available on the Web and the increasingly important role that the Web plays in today's society, Web content mining, Weblog mining, and data mining services on the Internet will become one of the most important and flourishing subfields in data mining.

Privacy protection and information security in data mining. With the increasingly popular use of data mining tools and telecommunication and computer networks, an important issue to face in data mining is privacy protection and information security. Further methods should be developed to ensure privacy protection and information security while facilitating proper information access and mining.

10.2 Data Warehousing

A Data Warehouse (DW) is a database that stores information oriented to satisfy decision-making requests. It is a database with some particular features concerning the data it contains and its utilization. A very frequent problem in enterprises is the impossibility for accessing to corporate, complete, and integrated information of the enterprise that can satisfy decision-making requests. A paradox occurs: data exists but information cannot be obtained. In general, a DW is constructed with the goal of storing and providing all the relevant information that is generated along the different databases of an enterprise. A data warehouse helps turn data into information. In today's business world, data warehouses are increasingly being used to make strategic business decisions.

10.2.1 Goals of Data Warehousing

Data warehousing technology comprises a set of new concepts and tools which support the knowledge worker like executive, manager, and analyst with information material for decision making. The fundamental reason for building a data warehouse is to improve the quality of information in the organization. The key issues are the provision of access to a company-wide view of data whenever it resides. Data coming from internal and external sources, existing in a variety of forms from traditional structural data to unstructured data like text files or multimedia is cleaned and integrated into a single repository. A data warehouse is the consistent store of this data which is made available to end users in a way they can understand and use in a business context.

The need for data warehousing originated in the mid-to-late 1980s with the fundamental recognition that information systems must be distinguished into operational and informational systems. Operational systems support the day-to-day conduct of the business, and are optimized for fast response time of predefined transactions, with a focus on update transactions. Operational data are a current and real-time representation of the business state. In contrast,

informational systems are used to manage and control the business. They support the analysis of data for decision making about how the enterprise will operate now and in the future. They are designed mainly for ad hoc, complex, and mostly read-only queries over data obtained from a variety of sources. Information data are historical, i.e., they represent a stable view of the business over a period of time.

Limitations of current technology to bring together information from many disparate systems hinder the development of informational systems. Data warehousing technology aims at providing a solution for these problems.

10.2.2 Characteristics of Data in Data Warehouse

Data in the Data Warehouse is integrated from various, heterogeneous operational systems like database systems, flat files, etc. Before the integration, structural and semantic differences have to be reconciled, i.e., data have to be “homogenized” according to a uniform data model. Furthermore, data values from operational systems have to be cleaned in order to get correct data into the data warehouse. Since a data warehouse is used for decision making, it is important that the data in the warehouse be correct. However, large volumes of data from multiple sources are involved; there is a high probability of errors and anomalies in the data. Therefore, tools that help to detect data anomalies and correct them can have a high payoff. Some examples where data cleaning becomes necessary are: inconsistent field lengths, inconsistent descriptions, inconsistent value assignments, missing entries, and violation of integrity constraints.

The need to access historical data are one of the primary incentives for adopting the data warehouse approach. Historical data are necessary for business trend analysis which can be expressed in terms of understanding the differences between several views of the real-time data. Maintaining historical data means that periodical snapshots of the corresponding operational data are propagated and stored in the warehouse without overriding previous warehouse states. However, the potential volume of historical data and the associated storage costs must always be considered in relation to their business benefits.

Data warehouse contains usually additional data, not explicitly stored in the operational sources, but derived through some process from operational data. For example, operational sales data could be stored in several aggregation levels in the warehouse.

10.2.3 Data Warehouse Architectures

Data warehouses and their architectures vary depending upon the specifics of an organization’s situation. Three common data warehouse architectures which are discussed in this section are:

1. Basic Data Warehouse Architecture
2. Data Warehouse Architecture with a Staging Area
3. Data Warehouse Architecture with a Staging Area and Data Marts

Basic Data Warehouse Architecture

The basic data warehouse architecture is shown in Fig. 10.1. End users directly access data derived from several source systems through data warehouse.

The data obtained from the warehouse can be used for analysis, reporting, and mining information as illustrated in the Fig. 10.6. The data sources include operational system and flat files. Here a flat file is one in which the fields of records are simple atomic values.

Data Warehouse Architecture with a Staging Area

The architecture of data warehouse with staging area is shown in Fig. 10.7.

In this architecture, the operational data must be cleaned and processed before putting into the warehouse. This can be done programmatically although most data warehouses use a staging instead. A staging area simplifies building summaries and general warehouse management.

Data Warehouse Architecture with Staging Area and Data Marts

The data warehouse architecture with staging area and data marts is illustrated in Fig. 10.8. The basic difference between this architecture and the architecture discussed earlier is the inclusion of data mart. It is necessary to customize the data warehouse's architecture for different groups within

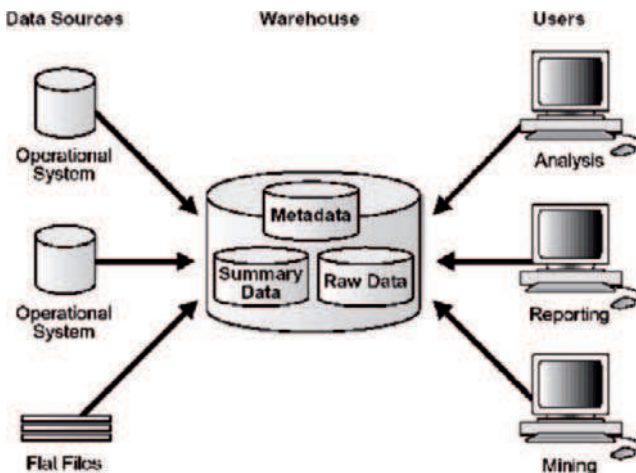


Fig. 10.6. Basic data warehouse architecture

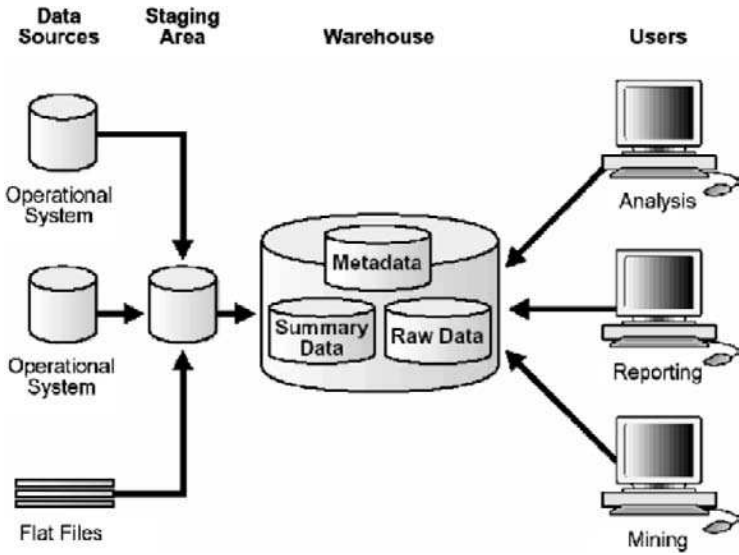


Fig. 10.7. Data warehouse architecture with staging area

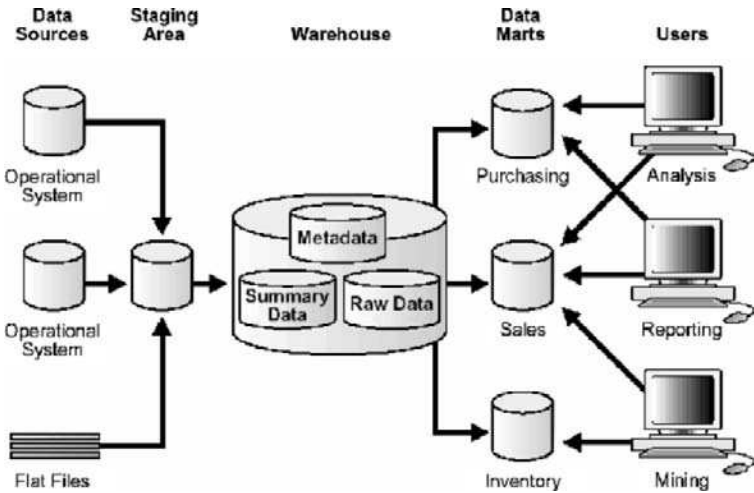


Fig. 10.8. Data warehouse architecture with staging area and data marts

an organization. This can be done by adding data marts, which are systems designed for a particular line of business. Figure 10.8 illustrates an example where purchasing, sales, and inventories are separated. In this example, a financial analyst might want to analyze historical data for purchases and sales.

Data Mart

Data marts are complete logical subsets of the complete data warehouse. Data marts should be consistent in their data representation in order to assure Data Warehouse robustness. A data mart is a set of tables that focus on a single task. This may be for a department, such as production or maintenance department, or a single task such as handling customer products.

Metadata

In general metadata are defined as “data about data” or “data describing the meaning of data.” In data warehousing, there are various types of metadata. For example information about the operational sources, the structure and semantics of the data warehouse data, the tasks performed during the construction, maintenance and access of a data warehouse, etc. A data warehouse without adequate metadata are like “a filing cabinet stuffed with papers, but without any folders or labels.” The quality of metadata and the resulting quality of information gained using a data warehouse solution are tightly linked. In a data warehouse metadata are categorized into Business and Technical metadata. Business metadata describes what is in the warehouse, its meaning in business terms. The business metadata lies above technical metadata, adding some more details to the extracted material. This type of metadata are important as it facilitates business users and increases the accessibility. In contrast, technical metadata describes the data elements as they exist in the warehouse. This type of metadata are used for data modeling initially, and once the warehouse is erected this metadata are frequently used by warehouse administrator and software tools.

Implementing a concrete Data Warehouse architecture is a complex task comprising of two major phases. In the configuration phase, a conceptual view of the warehouse is first specified according to user requirements which are often termed as data warehouse design. Then the involved data sources and the way data will be extracted and loaded into the warehouse is determined. Finally, decisions about persistent storage of the warehouse using database technology and the various ways data will be accessed during analysis are made.

10.2.4 Data Warehouse Design

Data warehouse design methods consider the read-oriented character of warehouse data and enables the efficient query processing over huge amounts of data. The core requirements and principles that guide the design of data warehouses are summarized later:

Data Warehouses Should be Organized Around Subject Areas

Subject areas are similar to the concept of functional areas like sales, project management, employees, etc. Each subject areas are associated with a conceptual schema and these can be represented using one or more entities in

the ER data model or by one or more object classes in the object oriented data model. For example: In company database the relations like employee, sales, and project management are represented as entities in ER data model or object classes in object oriented data model.

Data Warehouses Should have some Integration Capability

A common database should be designed and used so that all the different individual representations can be mapped to it. This is particularly useful if the warehouse is implemented as multidatabase or federated database.

Data should be Nonvolatile and Mass Loaded

Data in Data Warehouses should be nonvolatile. For this data extraction from current database to DW requires that a decision should be made whether to extract the data using standard relational database techniques at the row or column level or specialized techniques for mass extraction. Data cleaning techniques are required to maintain data quality, similarly data migration, data scrubbing, and data auditing. Refresh techniques propagate updates on the source data to base data and derived data in the DW. The decision of when and how to refresh is made by the DW administrator and depends on user needs (e.g., OLAP needs) and existing traffic to the DW.

Data Tends to Exist at Multiple Levels of Dimensions

Data can be defined not only by time frame but also by geographic region; type of product manufactured or sold type of store and so on. The complete size of the databases is a major problem in the design and implementation of data warehouses, especially for certain queries and updates and sequential backups. This decides whether to select relational databases or multidimensional database for the implementation of a data warehouse.

Data Warehouse Should be Flexible Enough to Meet Changing Requirements Rapidly

Insertion, updating, and retrieval of data should be very efficient and flexible to achieve good and efficient decision.

Data Warehouse Should have a Capability for Rewriting History, that is, Allowing for “what-if” Analysis

Data Warehouse should allow the administrator to update historical data temporarily for the purpose of “what-if” analysis. Once the analysis is completed, the data must be correctly rolled back. This assumes that the data must be at the proper dimension in the first place.

Good DW User Interface Should be Selected

The interface should be very user friendly for efficient use of DW. The leading choices of today are SQL.

Data Should be Either Centralized or Distributed Physically

The DW should have the capability to handle distributed data over a network. This requirement will become more critical as the use of DWs grows and sources of data expand.

10.2.5 Classification of Data Warehouse Design

The data warehouse design can be broadly classified into two categories (1) Logical design and (2) Physical design.

Logical Design

The logical design is more conceptual and abstract than physical design. In the logical design, the emphasis is on the logical relationship among the objects. One technique that can be used to model organization's logical information requirements is entity-relationship modeling. Entity-relationship modeling involves identifying the things of importance (entities), the properties of these things (attributes), and how they are related to one another (relationships). The process of logical design involves arranging data into a series of logical relationships called entities and attributes. An entity represents a chunk of information. In relational databases, an entity often maps to table. An attribute is a component of an entity that helps define the uniqueness of the entity. In relational databases, an attribute maps to a column. Whereas entity-relationship diagramming has traditionally been associated with highly normalized models such as OLTP applications, the technique is still useful for data warehouse design in the form of dimensional modeling.

In dimensional modeling, instead of seeking to discover atomic units of information and all the relationship between them, the focus is on identifying which information belongs to a central fact table and which information belongs to its associated dimension tables. In a nutshell, the logical design should result in (a) a set of entities and attributes corresponding to fact tables and dimension tables and (b) a model of operational data from the source into subject-oriented information in target data warehouse schema. Some of the logical warehouse design tools from Oracle are Oracle Warehouse Builder, Oracle Designer, which is a general purpose modeling tool.

Data Warehousing Schemas

A schema is a collection of database objects, including tables, views, indexes, and synonyms. The arrangement of schema objects in the schema models

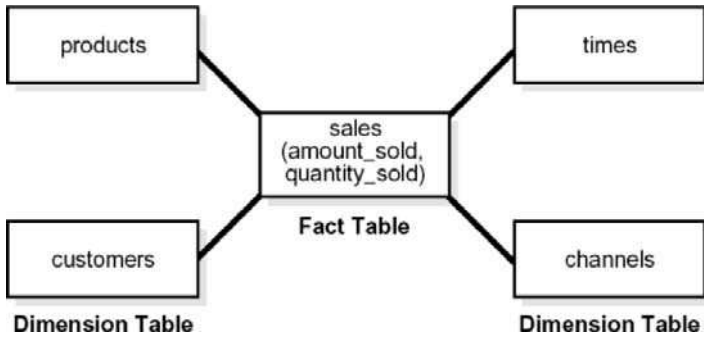


Fig. 10.9. Star schema

designed for data warehouse can be done in a variety of ways. Most data warehouses use a dimensional model.

Star Schema

The star schema is the simplest data warehouse schema. It is called a star schema because the diagram resembles a star, with points radiating from the center. The center of the star consists of one or more fact tables and the points of the star are the dimension tables as illustrated in Fig. 10.9.

A star schema optimizes performance by keeping queries simple and providing fast response time. All the information about each level is stored in one row. The most natural way to model a data warehouse is star schema, only one join establishes the relationship between the fact table and any one of the dimension tables. Another schema that is sometimes useful is the snowflake schema, which is a star schema with normalized dimensions in a tree structure.

Data Warehouse Objects

Fact and dimension tables are the two types of objects commonly used in the dimensional data warehouse schemas. Fact tables are the large tables in warehouse schema that store business measurements. Fact tables typically contain facts and foreign keys to the dimension tables. Fact tables represent data, usually numeric and additive that can be analyzed and examined. Dimension tables, also known as lookup or reference tables; contain the relatively static data in the warehouse. Dimension tables stores the information that is normally used to contain queries. Dimension tables are usually textual and descriptive.

Fact Tables A fact table typically has two types of columns: those that contain numeric facts and those that are foreign keys to dimension tables. A fact table contains either detail-level facts or facts that have been aggregated. Fact tables that contain aggregated facts are often called summary tables. A fact table usually contains facts with the same level of aggregation. Though most facts

are additive, they can also be semiadditive or nonadditive. Additive facts can be aggregated by simple arithmetical addition. Semiadditive facts can be aggregated along some of the dimensions and not along others.

Dimension Tables A dimension is a structure, often composed of one or more hierarchies, that categorizes data. Dimensional attributes help to describe the dimensional value. They are commonly descriptive, textual values. Several distinct dimensions, combined with facts, enable one to answer business questions. Dimensional data are typically collected at the lowest level of detail and then aggregated into higher level totals that are more useful for analysts. These natural rollups or aggregations within a dimension table are called hierarchies.

Hierarchies Hierarchies are logical structures that use ordered levels as a means of organizing data. A hierarchy can be used to define data aggregation. For example, in a time dimension, a hierarchy might aggregate data from the month level to the quarter level to the year level. A hierarchy can also be used to define a navigational drill path and to establish a family structure. Within a hierarchy, each level is logically connected to the levels above and below it. Data values at lower levels aggregate into the data values at higher levels. A dimension can be composed of more than one hierarchy. Hierarchies impose a family structure on dimension values. For a particular level value, a value at the next higher level is its parent, and values at the next lower level are its children. These familial relationships enable analysts to access data quickly.

Physical Design

During the physical design process the data gathered during the logical design phase is converted into a description of the physical database structure. Physical design decisions are mainly driven by query performance and database maintenance aspects. Figure 10.10 offers a graphical way of looking at the different ways of logical and physical designs.

Physical Design Structures

Some of the physical design structures that are going to be discussed in this section include (a) Table spaces (b) Tables and Partitioned Tables (c) Views (d) Integrity Constraints, and (e) Dimensions

Table Spaces

A table space consists of one or more data files, which are physical structures within the operating system. A data file is associated with only one table space. From the design perspective, table spaces are containers for physical design structures. Table spaces need to be separated by differences. For example, tables should be separated from their indexes and small tables should be separated from large tables. Table spaces should also represent logical business units.

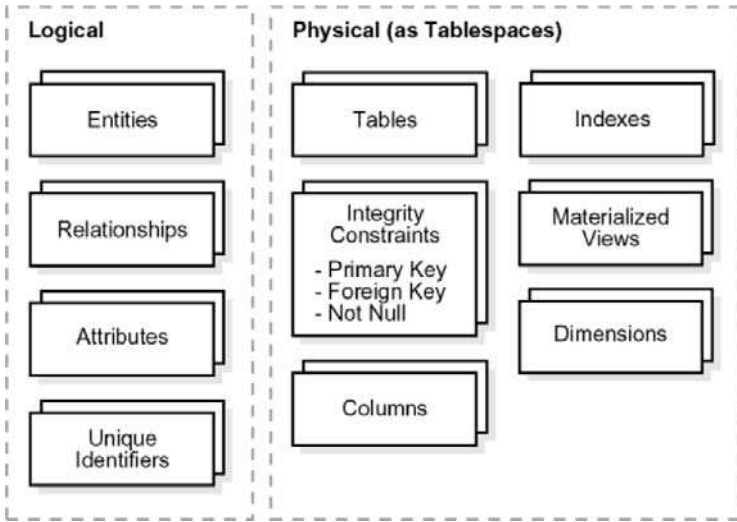


Fig. 10.10. Logical and physical design of data warehouse design

Tables and Partitioned Tables

Tables are the basic unit of data storage. They are the container for the expected amount of raw data in the data warehouse. Using partitioned tables instead of nonpartitioned ones addresses the key problem of supporting very large data volumes by allowing you to decompose them into smaller and more manageable pieces. The main design criterion for partitioning is manageability.

Data Segment Compression

Disk space can be saved by compressing heap-organized tables. A typical type of heap-organized table that one should consider for data segment compression is partitioned tables. Data segment compression can also speed up query execution. There is, however, a cost in CPU overhead. Data segment compression should be used with highly redundant data, such as tables with many foreign keys.

Views

A view is a tailored presentation of the data contained in one or more tables or other views. A view takes the output of a query and treats it as a table. Views do not require any space in the database.

Integrity Constraints

Integrity constraints are used to enforce business rules associated with the database and to prevent having invalid information in the tables. Integrity

constraints in data warehousing differ from constraints in OLTP environments. In OLTP environments, they primarily prevent the insertion of invalid data into a record, which is not a big problem in data warehousing environments because accuracy has already been guaranteed. In data warehousing environments, constraints are only used for query rewrite. NOT NULL constraints are particularly common in data warehouses. Under some specific circumstances, constraints need space in the database. These constraints are in the form of the underlying unique index.

Indexes and Partitioned Indexes

Indexes are optional structures associated with tables or clusters. In addition to the classical B-tree indexes, bitmap indexes are very common in data warehousing environments. Bitmap indexes are optimized index structures for set-oriented operations. Additionally, they are necessary for some optimized data access methods such as star transformations.

Dimensions

A dimension is a schema object that defines hierarchical relationships between columns or column sets. A hierarchical relationship is a functional dependency from one level of a hierarchy to the next one. A dimension is a container of logical relationships and does not require any space in the database. A typical dimension is city, state (or province), region, and country.

10.2.6 The User Interface

In this section, we provide a brief introduction to contemporary interfaces for data warehouses. A variety of tools are available to query and analyze data stored in data warehouses. These tools can be classified as follows:

- (a) Traditional query and reporting tools
- (b) On-line analytical processing, MOLAP, and ROLAP tools
- (c) Data-mining tools
- (d) Data-visualization tools

Traditional Query and Reporting Tools

Traditional query and reporting tools include spreadsheets, personal computer databases, and report writers and generators

OLAP Tools

On-Line Analytical Processing is the use of a set of graphical tools that provides users with multidimensional views of their data and allows them to analyze the data using simple windowing techniques. The term on-line

analytical processing is intended to contrast with the more traditional term on-line transaction processing. OLAP is a general term for several categories of data warehouse and data mart access tools. Relational OLAP (ROLAP) tools use variations of SQL and view the database as a traditional relational database, in either a star schema or other normalized or denormalized set of tables. ROLAP tools access the data warehouse or data mart directly. Multidimensional OLAP (MOLAP) loads data into an intermediate structure usually a three or higher dimensional array.

Data-Mining Tools

Data mining is knowledge discovery using a sophisticated blend of techniques from traditional statistics, artificial intelligence, and computer graphics. As the amount of data in data warehouses is growing exponentially, the users require automated techniques provided by data-mining tools to mine the knowledge in these data.

Data Visualization Tools

Data-visualization is the representation of data in graphical and multimedia formats for human analysis. Benefits of data visualization include the ability to better observe trends and patterns, and to identify correlations and clusters.

Summary

Data mining is a form of knowledge discovery that uses a sophisticated blend of techniques from traditional statistics, artificial intelligence, and computer graphics. In this chapter, a brief introduction to data mining is given which includes need for data mining, data mining functionalities, and classification of data mining systems. This chapter also discusses major issues in data mining, data mining primitives, and data mining tasks, and gives syntax for data mining query language. The data mining architecture, data mining association rules for large database, and multilevel database for transaction are discussed in depth in this chapter. This chapter also discusses the concepts of classification and prediction, which are two forms of data analysis that can be used to extract models describing important data classes or predict future data trends. The different types of classification methods are explain in detail.

The process of grouping a set of physical or abstract objects into classes of similar objects is called clustering. A cluster is a collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in other clusters. This chapter gives brief idea about cluster

analysis. This chapter also gives an idea of how to choose data mining system, applications and trends in data mining.

The purpose of data warehouse is to consolidate and integrate data from a variety of sources, and to format those data in a context for making accurate business decisions. A data warehouse is an integrated and consistent store of subject-oriented data obtained from a variety of sources and formatted into a meaningful context to support decision making in an organization. This chapter discusses the goals of data warehousing, characteristics of data in a data warehouse, and different types of data warehouse architectures. Two types of data warehouse design like logical and physical data warehouse design are discussed in depth. Finally, the user interface which gives a brief introduction to contemporary interfaces for data warehouses is discussed.

Review Questions

10.1. What is the need for data mining?

Data mining is the process of discovering interesting knowledge from large amounts of data stored in databases, data warehouses, or other information repositories. The volume of data in an organization increases day by day, in order to extract useful information from huge volume of data, data mining is necessary.

10.2. What are the functionalities of data mining system?

Functionalities of data mining are used to specify the kind of patterns to be found in data mining tasks. It can be classified into two categories such as Descriptive and Predictive. Descriptive mining task characterize the general properties of data in the database, whereas predictive mining task perform inference on the current data in order to make predictions.

10.3. Explain the “binning” method of data cleaning?

Binning method smoothen the sorted data value by consulting its “neighborhood” that is, the values around it. The sorted values are distributed into number of “buckets,” or bins:

- Smoothing by bin means each value in a bin is replace by the mean value of the bin.
- Smoothing by bin medians means each bin value replaces bin median.

10.4. What are the performance issues in data mining?

The performance issues in data mining include efficiency, scalability, and parallelization of data mining algorithms.

10.5. Explain the concept of classification and prediction with respect to data mining?

Databases are rich with hidden information that can be used for making intelligent business decisions. Classification and prediction are two forms of data analysis that can be used to extract models describing important data classes or predict future data trends. Whereas classification predicts categorical labels, prediction models continuous-valued functions.

10.6. What are the factors to be considered in designing data mining query language?

Designing a comprehensive data mining language is challenging because data mining covers wide spectrum of task, from data characterization to mining association rules, data classification and evaluation analysis.

Designing the data mining query language is specified by the following primitives:

- The kind of knowledge to be mined.
- The background knowledge to be used in the discovery process.
- The Interestingness measures and threshold or pattern evaluation.

10.7. Mention the challenges involved in mining spatial data?

A crucial challenge to spatial data mining is the exploration of efficient spatial data mining techniques due to the huge amount of spatial data and the complexity of spatial data types and spatial access methods. Spatial data mining allows the extension of traditional spatial analysis method by placing emphasis on efficiency, scalability, and co-operation with database systems, improved interaction with the user and discovery of new types of knowledge.

10.8. What is the need for Data Warehousing in an organization?

The need for Data Warehousing in most organizations is:

- A business requires an integrated, company-wide view of high-quality information.
- The information systems department must separate informational from operational systems in order to dramatically improve performance in managing company data.

10.9. Define the term “data mart”?

A data mart is a data warehouse that is limited in scope, whose data are obtained by selecting and summarizing data from a data warehouse or from separate extract, transform, and load processes from source data systems.

10.10. Distinguish between data warehouse and data mart?

Data Warehouse	Data Mart
Data warehouses are application independent	Data Mart are specific to Decision Support System application
Data warehouses are centralized	Data Mart are decentralized by user area
The data in data warehouse are historical, detailed, and summarized	In data mart some data are historical, detailed, and summarized
The data in data warehouse is lightly denormalized	The data in data mart is highly denormalized
The Data warehouse is flexible, data-oriented, and long life.	The data mart is restrictive, project-oriented, and short life.

10.11. List common tasks performed during data cleaning.

The common tasks performed during data cleaning are:

- Decoding data to make them understandable for data warehousing applications.
- Adding time stamps to distinguish values for the same attribute over time.
- Generating primary keys for each row of a table.
- Matching and merging separate extractions into one table or file and matching data to go into the same row of the generated table.
- Logging errors detected, fixing those errors, and reprocessing corrected data without creating duplicate entries.
- Finding missing data to complete the batch of data necessary for subsequent loading.

10.12. Mention the factors that one should consider in the design of Data Warehouse?

The factors that one should consider in the design of Data Warehouse are summarized later:

- Data Warehouses should be organized around subject areas
- Data Warehouses should have some integration capability
- Data should be nonvolatile and mass loaded
- Data tends to exist at multiple levels of dimensions
- Data Warehouse should be flexible enough to meet changing requirements rapidly
- Good DW user interface should be selected
- Data should be either centralized or distributed physically

Objected-Oriented and Object Relational DBMS

Learning Objectives. This chapter provides an overview of object oriented and object relational database management system. The need for object oriented concepts in DBMS, OODBMS, and ORDBMS are discussed elaborately in this chapter. The evaluation criteria and targets with respect to OODBMS and comparison of OODBMS with ORDBMS are also dealt with. After completing this chapter the reader should be familiar with the following concepts:

- Object oriented programming language
- Availability of OO Technology and applications
- Overview of OODBMS Technology
- Evaluation Criteria for OODBMS
- Evaluation targets
- Overview of ORDBMS
- ORDBMS Design
- Aggregation and Composition in UML
- Comparison of ORDBMS and OODBMS

11.1 Objected oriented DBMS

11.1.1 Introduction

This chapter provides a simple view about the Object-Oriented Database Management Systems (OODBMS). Each OODBMS will be architected based on a set of assumptions which make it more or less suited for particular application domains and usage patterns. Thus, a single OODBMS will not be the best in all situations. This chapter will also be used as an introduction to object-oriented database technology.

An evaluation of OODBMS must include analysis in four areas:

- Functionality
- Usability
- Platform
- Performance

Functionality

An analysis of functional capabilities is performed to determine if a given OODBMS provides sufficient capabilities to meet the current and future needs of a given development effort. Functional capabilities include basic database functionality, such as concurrency control and recovery as well as object-oriented database features, such as inheritance and versioning. Each evaluation will have to identify and weight a set of functional requirements to be met by the candidate OODBMS. Weighting is an important consideration since application workarounds may be possible for missing functionality.

Usability

Usability deals with the application development and maintenance process. Issues include development tools and the ease with which database applications can be developed and maintained. How a developer perceives the database and the management of persistent objects might also be considered under the category of usability. Other issues to be considered are database administration, product maturity, and vendor support. Evaluation of usability is likely to be highly subjective. Perhaps the most easily measurable evaluation criterion is platform. An OODBMS is either available or not on the application's target hardware and operating system. Heterogeneous target environments require that the OODBMS transparently interoperates within that environment.

Platform

An OODBMS is typically a multiprocessed software system communicating over a local area network. Platforms upon which database server processes, client application processes, additional administration processes (e.g., lock servers), and development tools can be hosted must be considered. Network requirements should also be evaluated. Performance may represent the most important evaluation criteria. The University of Wisconsin has performed a benchmarking of OODBMS, known as the 007 benchmark. A general purpose benchmark is only effective in predicting the performance of an OODBMS for an application which closely mirrors the behavior of that benchmark.

Performance

An effective benchmark must consider the number of interactive users, the rate of database updates and accesses, the size of the databases, the hardware and network configurations, and the general access patterns of the database applications. Thus, in order to provide useful information, a benchmark must be modeled to closely mimic the expected behavior of the application being

developed. Providing a fair and substantive evaluation of OODBMS is a difficult task. Issues regarding accuracy of marketing information and technical documentation, completeness of implementation, usability of implementation, performance, and feature interaction (regarding completeness, usability, and performance) must be considered when performing the evaluation. The objective of this chapter is to perform the first part of this evaluation process by performing an extensive analysis based on technical product documentation. In particular:

- Functional capabilities have been identified by examination of the product’s technical manuals as supplied by the vendor. Discussions with technical representatives of the vendor have been used to clarify our understandings and descriptions of the evaluated products.
- Usability has been derived by analyzing the documentation for the vendor supplied tools and by reviewing the application programming interface in order to understand how an application interacts with the database.
- Information regarding platform and heterogeneous operation has been supplied by the product vendors.
- Performance is not addressed as part of this evaluation.

Benefits of Object Oriented Programming

There are several benefits of adopting OOP. The following three benefits, although subjective, are considered by many to be major reasons for adopting OOP:

- Programs reflect reality.
- The model is more stable than functionality.
- Subclassing and virtuals improve the reusability of code.

11.1.2 Object-Oriented Programming Languages (OOPLs)

The following is a list of some popular OOPLs:

- C++ Language System
- C_Talk
- Smalltalk
- Smalltalk-80
- Actor
- Enfin
- Prokappa
- Eiffel
- KnowledgePro
- Classic-Ada with Persistence
- Objective-C
- Trellis/Owl

- Flavors
- CLOS
- Common Loops

Most OOPLs can trace their origins to Simula. The concepts of Objects and Classes are employed by most of these languages.

Comparing OOPLs

Different OOPLs are appropriate for different environments. For corporate environments Smalltalk is better than C++. Smalltalk is a higher level language and will be COBOL of the OO world in the future.

Smalltalk, Flavors, and Objective-C allows free access to inherited instance variables by descendant classes. Other OOPLs, like C++, restrict access to inherited instance variables. Where access to inherited instance variables is needed, it should be provided in the form of operations. Only CLOS permits specific instances of classes to have behavior independent of their classes.

Objective-C

Objective-C is considered by some researchers to be a cross between C and Smalltalk. It is possible to precompile Objective-C code to produce standard C as output. Objective-C incorporates the concept of an object identifier, *id*, which is a handle for referring to an object in a message expression. Objective-C is a compiled language, unlike Smalltalk and CLOS.

C++:

C++ is the most popular OOPL. It is an object-oriented extension to C, developed at AT&T Bell Laboratories. C++ supports the OOP concepts of objects, classes, inheritance, polymorphism, and parameterized types. The C++ class concept can be considered as a generalization of the C feature of a “struct.”

C++ has been evolving since it was released by AT&T as version of C with classes in 1984. The latest release is version 3.1, and it provides multiple inheritance, type-safe linkages, abstract classes, and a form of exception handling.

C++ provides an access control mechanism for the operations on objects. The operations are called member functions. Member functions can have one of the following three modes of access:

- Public
- Private
- Protected

Public member functions are accessible by all clients of the object. Private member functions are accessible only by other member functions of the class. Protected member functions are accessible only by other member functions of class derived from that class.

11.1.3 Availability of OO Technology and Applications

Some of the OO technologies that are being used to develop software application products are:

- Case tools based on OOT
- Analysis and design tools, some with OO capabilities
- Knowledge-based systems
- Hypermedia, Hypertext
- GUI front ends
- Object DBMS
- Radial application development environments

In the commercial CASE environments, vendors are employing OOT for all their products. GUI tools are mostly designed using OO concepts, and classes of GUI tools are widely available.

Transition to OOT

OOT has the reputation of requiring a learning curve. This is not only due to the necessity of learning a new language, but also due to the necessity of unlearning process oriented programming techniques. Because of the difference between the top-down structured programming and the OO techniques, the transition from a traditional structured programming environment to the object oriented environment requires a high investment of time and energy.

Containing Relationships Between Objects

Quite often, objects that contain other objects need to be represented in such a way that they are logically regarded as a single object. It is important to provide for *containing* relationships by which the composite logical object can refer to the contained objects. Typically, the contained objects are treated as private objects of enclosing object. This encapsulation might not be appropriate if such a tight coupling is not desired.

Modeling Relationships in C++

Interactions between objects can be captured during OOD by appropriate relationships. At the implementation level, C++ provides the following mechanisms for implementing object relationships:

1. Global Objects
2. Function arguments

3. Constructors
4. Base classes
5. Templates

Using Relationship Between Objects

Objects interacting in a system make use of the services offered by other objects. The *using* relationship can be used to express a subset of such interactions. *Booch* and *Vilot* have identified three roles that each object may play in using relationships:

Actor objects can operate upon other objects, but are never operated upon by other objects. Such objects make use of services offered by other objects but do not themselves provide any service to the objects they make use of.

Server objects never operate upon objects, but are operated upon by other objects.

Agent objects can both operate upon other objects and be operated upon by other objects.

Relationships Among Classes

Rumbaugh has identified three types of class relationships:

1. Generalization or “kind-of”
2. Aggregation or “part-of”
3. Association, implying some semantic connection.

Booch and Vilot have identified two more types of relationships between classes

1. Instantiation relationships
2. Metaclass relationships

Booch and Vilot suggest the following rule of thumb for identifying relationships: “If an abstraction is more than the sum of its component parts, then using relationships are more appropriate. If an abstraction is a kind of some other abstraction or if that abstraction is exactly equal to the sum of its components, then inheritance is a better approach.”

11.1.4 Overview of OODBMS Technology

This section deals with Need, Evolution, Characteristics, and Applications of Object Oriented Databases Technology.

The Need for Object-Oriented Databases

The increased emphasis on process integration is a driving force for the adoption of object-oriented database systems. For example, the Computer Integrated Manufacturing (CIM) area is focusing heavily on using object-oriented database technology as the process integration framework. Advanced office

automation systems use object-oriented database systems to handle hypermedia data. Hospital patient care tracking systems use object-oriented database technologies for ease of use. All of these applications are characterized by having to manage complex, highly interrelated information, which is strength of object-oriented database systems.

Clearly, relational database technology has failed to handle the needs of complex information systems. The problem with relational database systems is that they require the application developer to force an information model into tables where relationships between entities are defined by values. Mary Loomis, the architect of the Versant OODBMS compares relational and object-oriented databases. “Relational database design is really a process of trying to figure out how to represent real-world objects within the confines of tables in such a way that good performance results and preserving data integrity is possible. Object database design is quite different. For most parts, object database design is a fundamental part of the overall application design process. The object classes used by the programming language are the classes used by the ODBMS. Because their models are consistent, there is no need to transform the program’s object model to something unique for the database manager.

An initial area of focus by several object-oriented database vendors has been the Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), and Computer Aided Software Engineering (CASE) applications. A primary characteristic of these applications is the need to manage very complex information efficiently. Other areas where object-oriented database technology can be applied include factory and office automation. For example, the manufacture of an aircraft requires the tracking of millions of interdependent parts that may be assembled in different configurations. Object-oriented database systems hold the promise of putting solutions to these complex problems within reach of users.

Object-orientation is yet another step in the quest for expressing solutions to problems in a more natural, easier to understand way. Michael Brodie in his book *On Conceptual Modeling* states “the fundamental characteristic of the new level of system description is that it is closer to the human conceptualization of a problem domain. Descriptions at this level can enhance communication between system designers, domain experts and, ultimately, system end-users.”

The study of database history is centered on the problem of data modeling. “A data model is a collection of mathematically well defined concepts that help one to consider and express the static and dynamic properties of data intensive applications.”

A data model consists of:

- Static properties such as objects, attributes and relationships
- Integrity rules over objects and operations
- Dynamic properties such as operations or rules defining new database states based on applied state changes

Object-oriented databases have the ability to model all three of these components directly within the database supporting a complete problem/solution modeling capability. Prior to object-oriented databases, databases were capable of directly supporting points 1 and 2 above and relied on applications for defining the dynamic properties of the model. The disadvantage of delegating the dynamic properties to applications is that these dynamic properties could not be applied uniformly in all database usage scenarios since they were defined outside the database in autonomous applications. Object-oriented databases provide a unifying paradigm that allows one to integrate all three aspects of data modeling and to apply them uniformly to all users of the database.

The Evolution of Object-Oriented Databases

Object-oriented database research and practice dates back to the late 1970s and had become a significant research area by the early 1980s, with initial commercial product offerings appearing in the late 1980s. Today, there are many companies marketing commercial object-oriented databases that are second generation products. The growth in the number of object-oriented database companies has been remarkable. As both the user and vendor communities grow there will be a user pull to mature these products to provide robust data management systems.

OODBMS's have established themselves in niches such as e-commerce, engineering product data management, and special purpose databases in areas such as securities and medicine. The strength of the object model is in applications where there is an underlying need to manage complex relationships among data objects. Today, it is unlikely that OODBMS are a threat to the stranglehold that relational database vendors have in the market place. Clearly, there is a partitioning of the market into databases that are best suited for handling high volume low complexity data and databases that are suited for high complexity, reasonable volume, with OODBMS filling the need for the latter.

Object-oriented databases are following a maturation path similar to relational databases. Figure 11.1 depicts the evolution of object-oriented database technologies. On the left, we have object-oriented languages that have been extended to provide simple persistence allowing application objects to persist between user sessions. Minimal database functionality is provided in terms of concurrency control, transactions, recovery, etc. At the mid-point, we have support for many of the common database features mentioned earlier. Database products at the mid-point are sufficient for developing reasonably complex data management applications. Finally, database products with declarative semantics have the ability to greatly reduce development efforts, as well to enforce uniformity in the application of these semantics. OODBMS products today are largely in the middle with a few products exhibiting declarative semantics, such as constraints, referential integrity rules, and security

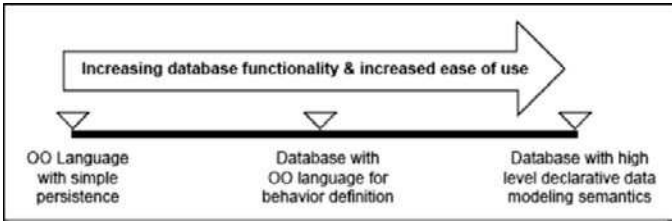


Fig. 11.1. The evolution of object-oriented databases

capabilities. In most OODBMS products, most of the database semantics are defined by programmers using low-level services provided by the database.

The next stage of evolution is more difficult. As one moves to the right the database does more for the user requiring less effort to develop applications. An example of this is that current OODBMS provide a large number of low-level interfaces for the purpose of optimizing database access. The onus is entirely on the developer for determining how to optimize his application using these features. As the OODBMS database technology evolves, OODBMS will assume a greater part of the burden for optimization allowing the user to specify high-level declarative guidance on what kinds of optimizations need to be performed. A general guideline for gauging database maturity is the degree to which functions such as database access optimization, integrity rules, schema and database migration, archive, backup and recovery operations can be tailored by the user using high-level declarative commands to the OODBMS.

Today, most object-oriented database products require the application developer to write code to handle these functions. Another sign of maturation of a new technology is the establishment of industry groups to standardize on different aspects of technology. Today we see a significant interest in the development of standards for object-oriented databases. For example, the Object Management Group (OMG) is a nonprofit industry sponsored association whose goal is to provide a set of standard interfaces for interoperable software components. Interfaces are to be defined in areas of communications (Object Request Broker), object-oriented databases, object-oriented user interfaces, etc. An OODBMS application programmers interface (API) specification is currently being developed (by ODMG, Object Database Management Group, a group of OODBMS vendors) thus allowing portability of applications across OODBMS.

Another standards body X3H7, a technical committee under X3, has been formed to define OODBMS standards in areas such as object-models and object-extensions to SQL. Today, OODBMS vendors are adding more database features to their products to provide the functionality one would expect from a mature database management system. This evolution moves us to the mid-point of the evolutionary scale shown in Fig. 11.1.

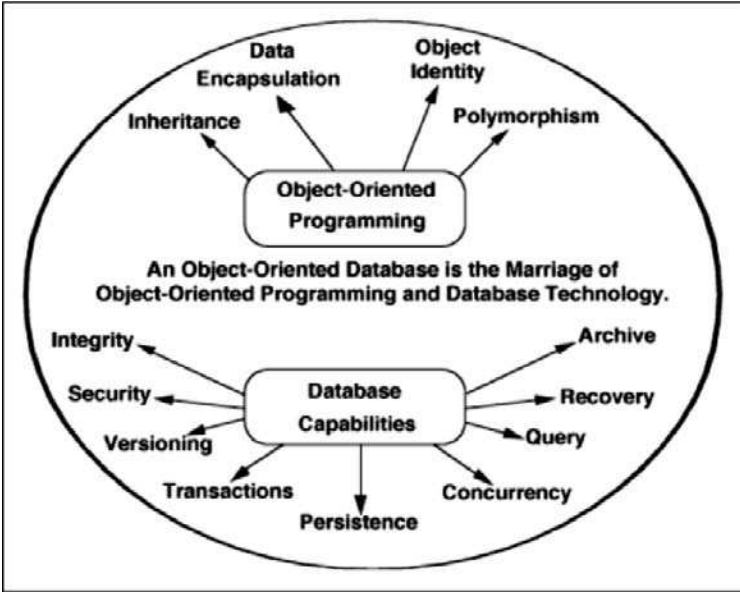


Fig. 11.2. Makeup of an object-oriented database

Characteristics of Object-Oriented Databases

Object-oriented database technology is a combination of object-oriented programming and database technologies. Figure 11.2 illustrates how these programming and database concepts have come together to provide what we now call object-oriented databases.

Perhaps the most significant characteristic of object-oriented database technology is that it combines object-oriented programming with database technology to provide an integrated application development system. There are many advantages to include the definition of operations with the definition of data. First, the defined operations apply ubiquitously and are not dependent on the particular database application running at the moment. Second, the data types can be extended to support complex data such as multimedia by defining new object classes that have operations to support the new kinds of information. Other strengths of object-oriented modeling are well known. For example, inheritance allows one to develop solutions to complex problems incrementally by defining new objects in terms of previously defined objects.

Polymorphism and dynamic binding allows one to define operations for one object and then to share the specification of the operation with other objects. These objects can further extend this operation to provide behaviors that are unique to those objects. Dynamic binding determines at runtime, which of these operations is actually executed, depending on the class of the object requested to perform the operation. Polymorphism and dynamic binding are

powerful object-oriented features that allow one to compose objects to provide solutions without having to write code that is specific to each object. All of these capabilities come together synergistically to provide significant productivity advantages to database application developers.

A significant difference between object-oriented databases and relational databases is that object-oriented databases represent relationships explicitly, supporting both navigational and associative access to information. As the complexity of interrelationships between information within the database increases, the greater is the advantages of representing relationships explicitly. Another benefit of using explicit relationships is the improvement in data access performance over relational value-based relationships.

A unique characteristic of objects is that they have an identity that is independent of the state of the object. For example, if one has a car object and we remodel the car and change its appearance – the engine, the transmission, the tires so that it looks entirely different, it would still be recognized as the same object we had originally. Within an object-oriented database, one can always ask the question, this is the same object we had previously, assuming one remembers the object's identity. Object-identity allows objects to be related as well as shared within a distributed computing network. All of these advantages point to the application of object-oriented databases to information management problems that are characterized by the need to manage:

- A large number of different data types
- A large number of relationships between the objects
- Objects with complex behaviors

An application area where this kind of complexity exists includes engineering, manufacturing, simulations, office automation, and large information systems.

11.1.5 Applications of an OODBMS

The design of an object-oriented data model is the first step in the application of object-oriented databases to a particular problem area. Developing a data model includes the following major steps:

- Object identification
- Object state definition
- Object relationships identification
- Object behavior identification
- Object classification

The following is a cursory overview of these steps.

As one begins to define an object-oriented data model, the first step is to simply observe and record the objects in the solution space. There are many techniques that aid this process. For example, one can formulate a description of the solution and identify the nouns that are candidates for being the

objects in the data model. Next, one identifies the characteristics of these objects. These characteristics become the object attributes. In a similar manner, examining the logical dependencies among objects identifies different kinds of association. For example, the parts relationship can be identified by analyzing the system decomposition into subparts. Next, one begins to enumerate the different responses that an object has to different stimuli. Finally, one classifies objects into an inheritance structure to factor out common characteristics and behaviors. All of these steps are performed iteratively until one has a complete data model.

A number of textbooks describe different variations of the earlier approach. In all cases, these methods culminate in a data model consisting of objects, attributes, relationships, behavior, and a classification structure. The methods vary in terms of targeted audience, the level of rigor, and the number and kinds of intermediate steps required arriving at a data model. Some methods are targeted to people whose background is structured analysis while other methods appeal to accomplished object-oriented developers. The practitioner has to select the methods that best match his experience and the target application.

Figure 11.3 illustrates a data model for a product and its decomposition into parts. Each part, in turn, may decompose into subparts. These associations are relationships (bidirectional relationships in this example) between objects. In an object-oriented database, relationships are maintained between objects using the object's unique identity, which means that one can change the attribute values of objects and not affect the relationships between the objects.

A significant difference between databases and object-oriented programming languages, such as C++ is that databases typically provide high-level primitives for defining relationships among objects. Typically, the implementation of relationships is managed by the OODBMS to maintain referential integrity. In addition, the OODBMS may allow one to define relationship

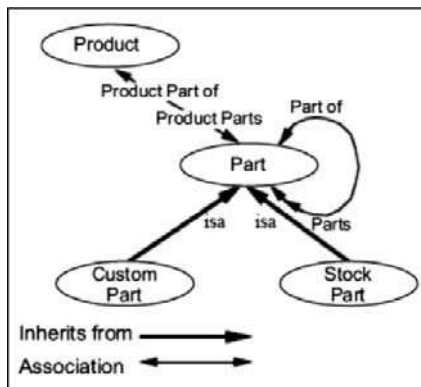


Fig. 11.3. Exploded product parts data model

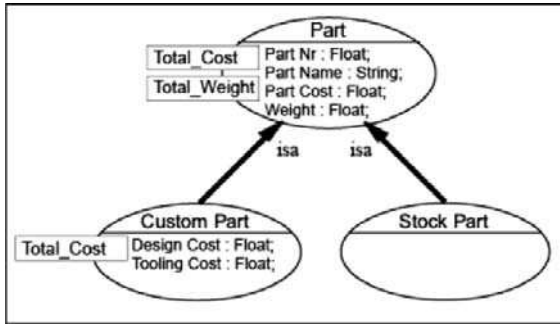


Fig. 11.4. Example of object attributes and operations

cardinality and object existence constraints. Semantic richness of relationships is well suited for the management of complex, highly interrelated information. Unfortunately, these capabilities are not provided uniformly by different object-oriented database products. An object-oriented data model also defines attributes and operations for each object as shown in Fig. 11.4.

In this example, the Custom Part and Stock Part inherit the attributes and operations from Part. The Custom Part object defines additional attributes and overrides the Total Cost operation. A significant advantage of inheritance is that it allows one to generalize objects by factoring common attributes and operations into some common object and then, using inheritance, share these common properties. As one adds more objects, relationships, and operations, inheritance helps to reduce the complexity of defining and maintaining this information.

In the real world, the data model is not static and will change as organizational information needs change and as missing information is identified. Consequently, the definition of objects must be changed periodically and existing databases migrated to conform to the new object definitions. Object-oriented databases are semantically rich introducing a number of challenges when changing object definitions and migrating databases. Object-oriented databases have a greater challenge handling schema migration because it is not sufficient to simply migrate the data representation to conform to the changes in class specifications. One must also update the behavioral code associated with each object. Improved facilities to manage this kind of change is appearing in a number of products, making it easier to maintain OODBMS-based solutions over time.

Pragmatics of Using an OODBMS

One needs to weigh a broad range of issues when considering an object-oriented database as a solution to an information management problem. These issues include: object models, data modeling tools, application design and development tools, testing and debugging tools, monitoring and tuning tools,

and database maintenance tools. A database application, like any software system, has a life cycle and requires a complete set of life cycle support tools. The following is a brief overview of the kinds of capabilities one needs to look for in these areas.

First, one needs to construct an object model of the information problem to be solved by the database. With current object-oriented databases, there are significant variations in the modeling capabilities of these products. For example, relationships in some databases are supported by high-level declarative capabilities that allow one to define self-maintaining properties of relationships. In other database products, one must program the semantics of relationships explicitly.

Some database products support a significantly richer data model providing powerful data management services that are defined as part of the data model. For example, one may define the existence of some objects to be dependent on a particular relationship. When one retracts the relationship, then the related object is also deleted. Other examples of more advanced database semantics include relationship cardinality constraints, attribute value constraints, uniqueness properties of attribute values, initial and default values, object versioning, and composite objects. All these data model issues affect how easily one can define the data model using tools provided by the database vendor. Another benefit is that the richer the data modeling facilities the less work that is required implementing the database application.

An area that many OODBMS evaluators overlook is the tools to support the development of the database applications. Since an OODBMS includes a language for specifying object behaviors, one needs to understand how such behaviors are developed and tested for a given OODBMS. Testing is particularly important since one need to integrate the compiler testing and debugging tools with the database persistent object storage manager.

During database maintenance, one often needs to introduce incremental changes into the database and the database applications. This is one of the most difficult areas since existing objects must be migrated to a new state conforming to the new schema definitions. This is a common problem with all database applications and needs to be anticipated in the design of the database and its applications. Today most OODBMS provide low level services for migrating databases, making the process of changing the schema a major challenge for database developers.

Finally, the problem of optimizing a database implementation for a particular application is a difficult one. First, there is little experience with OODBMS on which to base optimization strategies. Second, many of the optimization features are at a low level, requiring the application developer to design the application around these features. Both database instrumentation and monitoring tools are needed, as well as facilities for tuning an existing application. As with relational databases, object-oriented database systems also require extensive monitoring and tuning to extract the maximum performance for a given application.

11.1.6 Evaluation Criteria

This section is a detailed discussion of evaluation criteria that may be considered when evaluating OODBMS. These criteria are broken into three main areas:

- Functionality
- Application Development Issues
- Miscellaneous Criteria

Functionality

Functionality, defines evaluation criteria based on functional capabilities provided by the OODBMS. Subsections include Basic Object-Oriented Modeling, Advanced Object-Oriented Database Topics, Database Architecture, Database Functionality, Application Programming Interface, and Querying an OODBMS. The topics discussed in the Basic Object-Oriented Modeling subsection are those functions that are expected to be similar in all OODBMS products, such as the ability to define complex objects using classes, attributes, and behaviors. In fact, these topics are the object-oriented features found in most OO technologies (e.g., languages, design methods). The Advanced Object-Oriented Database Topics subsection describes functionality that is somewhat unique to object-oriented databases. It is expected that OODBMS products will differ significantly in these areas. The Database Functionality subsection describes the features that distinguish a database from a persistent storage manager (e.g., concurrency control and recovery).

Application Development Issues

Application Development Issues considers issues regarding the development of applications on top of an OODBMS. Miscellaneous Criteria, identifies a few nonfunctional and nondevelopmental evaluation issues. These issues deal with vendor and product maturity, vendor support, and current users of the OODBMS product.

Although the evaluation criteria identified in this chapter will be an important part of any OODBMS selection process, the issues of platform and performance will most likely dominate the selection process. Although missing functionality can often be managed at the application level, inadequate performance cannot be overcome (assuming optimal use of database facilities and features).

Miscellaneous Criteria

The list of evaluation criteria defined in this chapter is quite extensive. This list was developed as a means of covering the spectrum of issues that might

merit consideration during an OODBMS evaluation task. It is not expected that any OODBMS evaluation effort would attempt to consider all of the listed criteria. Instead, an evaluation effort must select the criteria relevant to a particular set of application requirements. Evaluation Criteria Overview is a road map of the criteria evaluation categories that are described in the remainder of this section.

Functionality

In the process of identifying functional evaluation criteria, we provide an overview of OODBMS capabilities that are typically found in commercial products. The list of such capabilities was derived from the many books, reports, and articles that have appeared in the literature over the past few years.

This section not only identifies the functionality evaluation criteria but also provides a high-level overview of DBMS and OODBMS concepts. Evaluation criteria for this section are broken into the following subsections:

- Basic Object-Oriented Modeling
- Advanced Object-Oriented Database Topics
- Database Architecture
- Database Functionality
- Application Programming Interface
- Query Capabilities

In Table 11.1, Functional Evaluation Criteria, is a road map of the categories and specific topics covered in the functional evaluation criteria.

Table 11.1. Evaluation criteria overview

Evaluation Area	Criteria Categories
Functionality	Basic Object-Oriented Modeling Advanced Object-Oriented Database Topics Database Architecture Database Functionality Application Programming Interface Querying an OODBMS
Application Development Issues	Developer's View of persistence Application Development Process Application Development Tools Class Library
Miscellaneous Criteria	Product Maturity Product Documentation Vendor Maturity Vendor Training Vendor Supporting and Consultation Vendor Participation in Standards Activities

Basic Object-Oriented Modeling

The evaluation criteria in this section distinguish database as an object-oriented database. Topics in this section cover the basic object-oriented (OO) capabilities typically supported in any OO technology (e.g., programming language, design method). These basic capabilities are expected to be supported in all commercial OODBMS. The topics are given a cursory overview here as shown in Table 11.2 for readers new to OO technology.

Complex Objects. OO systems and applications are unique that the information being maintained is organized in terms of the real-world entities being modeled. This differs from relational database applications that require a translation from the real-world information structure to the table formats used to store data in a relational database. Normalizations upon the relational database tables result in further perturbation of the data from the user's perceptual viewpoint. OO systems provide the concept of complex objects to enable modeling of real-world entities. A complex object contains an arbitrary number of fields, each storing atomic data values or references to other objects (of arbitrary types). A complex object exactly models the user perception of some real-world entity.

Object Identity. OO databases (and programming languages) provide the concept of an object identifier (OID) as a means of uniquely identifying a particular object. OIDs are system generated. A database application does not have direct access to the OID. The OID of an object never changes, even across application executions. The OID is not based on the value stored within the object. This differs from relational databases, which use the concept of primary keys to identify a particular table row (i.e., tuple). Primary keys are based upon data stored in the identified row. The concept of OIDs makes it easier to control the storage of objects (e.g., not based on value) and to build links between objects (e.g., they are based on the never changing OID). Complex objects often include references to other objects, directly or indirectly stored as OIDs.

The size of an OID can substantially affect the overall database size due to the large number of inter object references typically found within an OO application. When an object is deleted, its OID may or may not be reused. Reuse of OIDs reduces the chance of running out of unique OIDs but introduces the potential for invalid object access due to dangling references. A dangling reference occurs if an object is deleted, and some other object retains the deleted object's OID, typically as an interobject reference. This second object may later use the OID of the deleted object with unpredictable results. The OID may be marked as invalid or may have been reassigned. Typically, an OODBMS will provide mechanisms to ensure dangling references between objects are avoided.

Classes. OO modeling is based on the concept of a class. A class defines the data values stored by, and the functionality associated with, an object

Table 11.2. Functional evaluation criteria

Criteria Categories	Criteria
Basic Object-Oriented Modeling	<ul style="list-style-type: none"> – Complex Objects – Object Identity – Classes – Attributes – Behaviors – Encapsulation – Inheritance – Overriding Behaviors and Late Binding – Persistence – Naming
Advanced Object-Oriented Database Topics	<ul style="list-style-type: none"> – Relationships and Referential Integrity – Composite Objects – Location Transparency – Object Versioning – Work Group Support – Schema Evolution – Runtime Schema Access/Definition/Modification – Integration with Existing DBs and Applications – Active vs. Passive Object Mgmt. System
Database Architecture	<ul style="list-style-type: none"> – Distributed Client–Server Approach – Data Access Mechanism – Object Clustering – Heterogeneous Operaton
Database Functionality	<ul style="list-style-type: none"> – Access to Unlimited Data – Integrity – Concurrency – Recovery – Transactions – Deadlock Detection – Locking – Backup and Restore – Dump and Load – Constraints – Notification Model – Indexing – Storage Reclamation – Security
Application Programming Interface	<ul style="list-style-type: none"> – DDL/DML Language – Computational Completeness – Language Integration Style – Data Independence – Standards
Querying an OODBMS	<ul style="list-style-type: none"> – Associative Query Capability – Data Independence – Impedance Mismatch – Query Invocation – Invocation of Programmed Behaviors

of that class. One of the primary advantages of OO data modeling is this tight integration of data and behavior through the class mechanism. Each object belongs to one, and only one, class. An object is often referred to as an instance of a class. A class specification provides the external view of the instances of that class. A class has an extent (sometimes called an extension), which is the set of all instances of the class. Implementation of the extent may be transparent to an application, but minimally provides the ability to visit every instance of the class. Within an OODBMS, the class construct is normally used to define the database schema. Some OODBMS use the term type instead of class. The OODBMS schema defines what objects may be stored within the database.

Attributes. Attributes represent data components that make up the content of a class. Attributes are called data members in the C++ programming language. Instance attributes are data components that are stored by each instance of the class. Class attributes (static data members in C++) are data values stored once for all instances of the class. Attributes may or may not be visible to external users of the class. Attribute types are typically a subset of the basic data types supported by the programming language that interfaces to the OODBMS. Typically this includes enumeration types such as characters and booleans, numeric types such as integers and floats, and fixed length arrays of these types such as strings. The OODBMS may allow variable length arrays, structures (i.e., records), and classes as attribute types.

Pointers are normally not good candidates for attribute types since pointer values are not valid across application executions.

An OODBMS will provide attribute types that support interobject references. OO applications are characterized by a network of interconnected objects. Object interconnections are supported by attributes that reference other objects. Other types that might be supported by an OODBMS include text, graphic, and audio. Often these data types are referred to as Binary Large ObjectS (BLOBS). Derived attributes are attributes that are not explicitly stored but instead calculated on demand. Derived attributes require that attribute access be indistinguishable from behavior invocation.

Behaviors. Behaviors represent the functional component of a class. A behavior describes how an object operates upon its attributes and how it interacts with other related objects. Behaviors are called member functions in the C++ programming language. Behaviors hide their implementation details from users of a class.

Encapsulation. Classes are said to encapsulate the attributes and behaviors of their instances. Behavior encapsulation shields the clients of a class (i.e., applications or other classes) from seeing the internal implementation of a behavior. This shielding provides a degree of data independence so that clients need not be modified when behavior implementations are modified (they will have to be modified if behavior interfaces change).

A class's attributes may or may not be encapsulated. Attributes that are directly accessible to clients of a class are not encapsulated (public data members in C++ classes). Modifying the definition of a class's attributes that are not encapsulated requires modification of all clients that access them. Attributes that are not accessible to the clients of a class are encapsulated (private or protected data members in C++ classes). Encapsulated attributes typically have behaviors that provide clients some form of access to the attribute. Modifications to these attributes typically do not require modification to clients of the class.

Inheritance. Inheritance allows one class to incorporate the attributes and behaviors of one or more other classes. A subclass is said to inherit from one or more superclasses. The subclass is a specialization of the superclass in that it adds additional data or behaviors, or overrides behaviors of the superclass. Superclasses are generalizations of their subclasses. Inheritance is recursive. A class inherits the attributes and behaviors from its superclasses, and from its superclass's superclasses, etc. In a single inheritance model, a class may directly inherit from only a single other class. In a multiple inheritance model a class may directly inherit from more than one other class. Systems supporting multiple inheritance must specify how inheritance conflicts are handled. Inheritance conflicts are attributes or behaviors with the same name in a class and its superclass, or in two superclasses.

Inheritance is a powerful OO modeling concept that supports reuse and extensibility of existing classes. The inheritance relationships between a groups of classes define a class hierarchy. Class hierarchies improve the ability of users to understand software systems by allowing knowledge of one class (a superclass) to be applicable to other classes (its subclasses).

Overriding Behaviors and Late Binding. OO applications are typically structured to perform work on generic classes (e.g., a vehicle) and at runtime invoke behaviors appropriate for the specific vehicle being executed upon (e.g., Boeing 747). Applications constructed in such a manner are more easily maintained and extended since additional vehicle classes may be added without requiring modification of application code. Overriding behaviors is the ability for each class to define the functionality unique to itself for a given behavior. Late binding is the ability for behavior invocation to be selected at runtime based on the class of an object (instead of at compile time).

Persistence. Persistence is the characteristic that makes data available across executions. The objective of an OODBMS is to make objects persistent. Persistence may be based on an object's class, meaning that all objects of a given class are persistent. Each object of a persistent class is automatically made persistent. An alternative model is that persistence is a unique characteristic of each object (i.e., it is orthogonal to class). Under this model, an object's persistence is normally specified when it is created. A third persistence model is that any object reachable from a persistent object is also persistent. Such systems require some way of explicitly stating that a given object is persistent

(as a means of starting the network of interconnected persistent objects). Related to the concept of persistence is object existence. OODBMS may provide a means by which objects are explicitly deleted. Such systems must ensure that references to deleted objects are also removed. An alternative strategy is to maintain an object as long as references to the object exist. Once all references are removed, the object can be safely deleted.

Naming. OO applications are characterized as being composed of a network of interconnected objects. An application begins by accessing a few known objects and then traverses to additional objects via relationships from the known objects. As objects are created they are linked (i.e., related) to other existing objects. Given this scenario, the database must provide some mechanism for identifying one or more objects at application start-up without using relations from existing objects. This is typically accomplished by allowing objects to be named and providing a retrieval mechanism based upon name. An application begins by loading one or two “high-level” objects that it knows by name and then traverses to other reachable objects. Object names apply within some name scope. Within a given scope, names must be unique (i.e., the same name can not refer to two objects). The simplest scope model is for the entire database to act as a single name scope. An alternative scope model is for the application to identify name scopes. Using multiple name scopes will reduce the chance for name conflicts.

Advanced Object-Oriented Database Topics

Functional capabilities identified in this section are those that are somewhat unique to object-oriented database systems. We expect that these topics represent the most interesting evaluation topics and will provide the greatest diversity among the evaluated OODBMS.

Relationships and Referential Integrity Relationships are an essential component of the object-oriented modeling paradigm. Relationships allow objects to refer to each other and result in networks of interconnected objects. Relationships are the paths used to perform navigation-based data access typical of programmed functionality. The ability to directly and efficiently model relationships is one of the major improvements of the object-oriented data model over the relational data model.

Conceptually, relationships can be thought of as abstract entities that allow objects to reference each other. An OODBMS may choose to represent relationships as attributes of the class (from which the relationships emanate), as independent objects (in which case relationships may be extensible and allow attributes to be added to a relationship), or as hidden data structures attached to the owning object in some fashion.

Relationships are often referred to as references, associations, or links. Sometimes the term relation is used to mean the schema definition of the potential for interconnections between objects, and the term relationship is used to mean actual occurrences of an interconnection between objects.

In this document we will use the term relationship interchangeably for both the schema definition and the object level existence for connections between objects. Relationships can be characterized by a number of different independent parameters, leading to a large number of different relationship behaviors:

- *Relationships may be unidirectional or bidirectional.* A unidirectional relationship exists in only a single direction, allowing a traversal from one object to another but no traversal in the reverse direction. A bidirectional relationship allows traversal in both directions. When a relationship is established along a bidirectional relationship, that relationship is automatically created in both directions (i.e., the application explicitly creates the relationship in one direction and the OODBMS implicitly sets the relationship in the opposite direction).
- Relationships have a cardinality, typically either one-to-one, one-to-many, or many-to-many. A one-to-one relationship allows one object to be related to another object (e.g., spouse might typically be modeled as a one-to-one relationship). Setting a one-to-one relationship deletes any previously existing relationship. A one-to-many relationship allows a single object to be related to many objects in one direction, in the reverse direction an object may be related to only a single object (e.g., when modeling a house, the house might be composed of many rooms, each room is part of a single house). One-to-one and one-to-many relationships may be unidirectional or bidirectional. A many-to-many relationship, which must be bidirectional, allows each object to be related to many objects in both directions of the relationship (e.g., modeling the relationship between parents and children might use a many-to-many bidirectional relationship. A person may have many children; children may have more than one parent.).
- *Relationships may have ordering semantics.* Ordered relationships are typically considered as lists (the objects are ordered by the operations that build the relations, not by the values stored in the related objects). Unordered relationships are either sets or bags. Sets do not allow duplication; bags do.
- Relationships may support the concept of composite objects.

The existence of relationships gives rise to the need for referential integrity. Referential integrity ensures that objects do not contain references to deleted objects. Referential integrity can be automatically provided for bidirectional relationships. Given a bidirectional relationship, when an object is deleted, all related objects can be found and have their relationships to the deleted object removed. Unidirectional relationships cannot be assured of referential integrity (short of performing complete scans of the database). If an object which is the target of a unidirectional relationship is deleted, there is no efficient mechanism to identify all objects that reference the deleted object (and delete the relationships to the deleted object). Application level solutions

to this problem exist (e.g., maintenance of existence dependency lists), but may result in poor performance and are effectively duplicating much of the work done by bidirectional relationships.

Alternatively, if the OODBMS does not reuse object identifiers, then a deleted object may be tomb-stoned, meaning a mark is left denoting that the object has been deleted. When a reference to a deleted object is made it can then be trapped as an error, or simply ignored, with an appropriate update of the referencing object's relationship (to no longer relate to the deleted object). Some OODBMS products provide a similar capability by keeping reference counts to objects, and only deleting an object when all references have been removed.

Relationship implementation provides a major differentiator between OODBMS products. On disk, relationships are typically modeled using object identifiers. Once brought into memory, relationships may remain as object identifiers or be swizzled into virtual memory pointers. Swizzling is a process that converts disk-based relationship representations into memory pointers for all related objects that are in memory. This may be done on object load or on demand when a particular relationship is traversed. Swizzling trades the overhead of performing the conversion to a memory-based pointer traversal in the hopes that multiple future accesses across the relationship will result in an overall speed improvement. Systems that do not swizzle require an object identifier lookup for all relationship traversal. This lookup can be performed efficiently through the use of lookup tables. Hybrid approaches are also possible. Each application will have to consider its expected object access and relationship traversal patterns to determine if a swizzled or object identifier-based relationship approach will best suit its needs.

Composite Objects Composite objects are groupings of interrelated objects that can be viewed logically as a single object. Composite objects are typically used to model relationships that have the semantic meaning is-part-of (e.g., rooms are part of a house). Composite objects are connected by the relationship mechanisms provided by the OODBMS. Operations applied to the "root" object of such a grouping can be propagated to all objects within that group. Operations that might be applied on composite objects include:

- Copy
- Delete
- Lock

Here we are defining Identifier-Equality, Shallow-Equality, and Deep-Equality operations. These three different forms of equality checks compare object identifiers, attribute values, and attribute values of component objects, respectively. Also defined are Shallow-Copy and Deep-Copy operations. A Shallow-Copy makes a new object and copies attribute values. A Deep-Copy

makes a new object, copying nonrelationship attribute values, and then recursively creates new objects for related objects (recursively applying the Deep-Copy operation). Deep-Copy is an example of an operation being propagated across a composite object. Propagation of delete and lock operations means that if the root object is deleted or locked all of its component objects are also deleted or locked.

Location Transparency Location transparency is the concept that an object can be referenced (i.e., can use the same syntactic mechanism) regardless of what database it resides in and where on the network that database is located. Objects should be able to be moved programmatically and have all references to the object remain intact (a form of referential integrity). (The ability to move an object to a new database location will also be considered as part of database administration capabilities.)

Object Versioning Object versioning is the concept that a single object may be represented by multiple versions (i.e., instances of that object) at one time. We can define two forms of versioning, each driven by particular requirements of the applications which are driving the need for OODBMS products:

- Linear Versioning is the concept of saving prior versions of objects as an object changes. In design-type applications (e.g., CASE, CAD) prior versions of objects are essential to maintain the historical progression of a design and to allow designers to return to earlier design points after investigating and possibly discarding a given design path. Under linear versioning, only a single new version can be created from each existing version of an object.
- Branch Versioning supports concurrency policies where multiple users may update the same data concurrently. Each user's work is based upon a consistent, nonchanging base version. Each user can modify his version of an object (as he proceeds along some design path in a CAD application for example). At some future point in time, under user/application support, the multiple branch versions are merged to form a single version of the object. Branch versioning is important in applications with long transactions so that users are not prevented from access to information for long periods of time. Under branch versioning, multiple new versions may be created for an object.

Associated with the idea of versioning is that of configuration. A configuration is a set of object versions that are consistent with each other. In other words, it is a group of objects whose versions all belong together. OODBMS needs to provide support so that applications access object versions that belong to the same conceptual configuration. This may be achieved by controlling the relationships that are formed for versioned objects (i.e., they may be duplicated in the new object or replaced with relationships to other objects).

An OODBMS may provide low level facilities which application developers use to control the versioning of objects. Alternatively, the OODBMS may

implement a specific versioning policy such as automatically creating a new object version with each change. An automatic policy may result in rapid and unacceptable expansion of the database and requires some automated means of controlling this growth.

Work Group Support In addition to versioning, an OODBMS might support group applications in other manners. The ability is to designate shared and private databases with the concept of checking data in and out of these data spaces. Some databases may allow segments of the database to be taken off-line, perhaps on a portable computer, used autonomously, and then brought back on-line at a later time. Long transactions are another mechanism on top of which group applications can be built.

Schema Evolution Schema evolution is the process by which existing database objects are brought into line with changes to the class definitions of those objects (i.e., schema changes require all instances of the changed class to be modified so as to reflect the modified class structure). Schema evolution is helpful although not essential during system development (as a means of retaining test data, for example). Schema evolution is essential for maintenance and/or upgrades of fielded applications. Once an application is in the field (and users are creating large quantities of information), an upgrade or bug fix cannot require disposal of all existing user databases. Schema evolution is also essential for applications that support user-level modeling and/or extension of the application.

Here we have given a framework for schema modifications in an object-oriented database. Included in this framework are invariants which must be maintained at all times (e.g., all attributes of a class must have a distinct name), rules for performing schema modifications (e.g., changing the type of an attribute in a given class must change the type of that attribute in all classes which inherit that attribute), and a set of schema changes that should be supported by an object-oriented database. This set of schema change operations is:

1. Changes to Definition of a Class:
 - (a) Changes to an Attribute of a Class (applies to both instance and class attributes):
 - Add an attribute to a class.
 - Remove an attribute from a class.
 - Change the name of an attribute.
 - Change the type of an attribute.
 - Change the default value of an attribute.
 - Alter the relationship properties for relationship attributes.
 - (b) Changes to a Behavior of a Class:
 - Add a new behavior to the class.
 - Remove a behavior from the class.
 - Change the name of a behavior.
 - Change the implementation of a behavior.

2. Changes to the Inheritance of a Class:
 - Add a new superclass to a class.
 - Remove a superclass for a given class.
 - Change the order of superclasses for a class (it is expected that superclass ordering will be used to handle attribute and behavior inheritance conflicts).
3. Changes to the Existence of a Class:
 - Add a new class.
 - Remove an existing class.
 - Change the name of a class.

Schema changes will require modification of instances of the changed class as well as applications that referenced the changed class. Some of these changes cannot be performed automatically by the OODBMS. Deleting attributes and superclasses are examples of schema changes that could be performed automatically. Adding attributes and superclasses can only be performed if default values are acceptable for the initial state of new attributes. This is not likely, especially for relationship attributes. An OODBMS should provide tools and/or support routines for aiding programmed schema evolution.

A manual evolution approach requires instance migration to be performed off-line, probably through a dump of the database and a reload of the data through an appropriate transformation filter. Systems may perform an aggressive update by automatically adjusting each instance after each schema change. This approach may be slow due to the overhead of performing the update on all instances at a single time. This approach is the easiest for an application to implement since multiple versions of the schema need not be maintained indefinitely.

Schema changes may be performed in background mode, thus spreading the update overhead over a longer period of time. A lazy evaluation approach defers updating objects until they are accessed and found to be in an inconsistent state. Both the background and lazy approaches require extended periods where multiple versions of the schema exist and will be complicated by multiple schema modifications. Applications and stored queries will have to be updated manually as a result of schema changes. Some forms of schema changes will not require updates to applications and queries due to data independence and encapsulation of a class's data members.

It is expected that all OODBMS products will support some form of schema evolution for static schema changes. By static, we mean the schema is changed by manipulations of class definitions outside of application processing (i.e., by reprocessing database schema definitions and modifying application programs). Dynamic schema modification, meaning modification of the schema by the application, is more complex and potentially inconsistent with the basic C++ header file approach used for schema definitions in many current commercial products. Dynamic schema modification is only needed in applications that require user definable types.

Runtime Schema Access/Definition/Modification An OODBMS typically makes use of a database resident representation of the schema for the database. The existence of such a representation, and a means of accessing it, provides applications with direct access to schema information. Access to schema information might be useful in building custom tools which browse the structure and contents of a database. Modification and definition of the schema at runtime allows the development of dynamically extensible applications. Users of this modeling system define the classes, attributes, and behaviors of the information that they wish to model. Once defined, instances of these classes may be created and manipulated.

The idea of dynamic schema definition is foreign to the C++ programming language. In C++, class definitions are defined statically in header files. An application may not alter these class definitions at runtime. OODBMS can provide access and modification of their schemata by storing the schema as instances of predefined classes and then allowing applications to create, modify, and query the instances which model the schema. An application might wish to modify the schema in order to extend an application to store new information or to display information in alternative presentations.

Integration with Existing DBs and Applications Numerous papers have appeared in the literature describing the need for integration of object-oriented and relational database technologies. Newly developed object-oriented applications will need to access existing relational databases. Data stored in object-oriented databases must be accessible to existing Standard Query Language (SQL) applications. Some applications will require access to both relational and object-oriented databases.

The process of accessing a relational database from an object program is given as follows. Defining a mapping of the relational schema into an object model is the first task. A simple approach is to represent each relation by a class and replace foreign key fields (in the relational schema) by relationships (in the object-oriented schema). Given a mapping of tables to classes, a means of invoking SQL operations from the object program must be defined. This can be provided by defining methods on the mapped classes for creating, updating, and deleting instances. These methods are responsible for interacting with the relational database. Additional methods are required to provide an interface to query operations that translate the tuples returned from a query into a set of objects accessible by the object program. Database interface generator products, providing automated support for interfacing object programs to relational databases, are currently being developed. These products may work from a user-developed set of class definitions, or from the relational database's data definitions language (DDL). In either case, the result is a set of method specifications and implementations that provide access to a relational database from an object program.

OODBMS vendors are moving to support the need for SQL access to their databases. This need arises due to the large experience base of SQL users and the desire for existing applications to be continually supported, as OODBMS

systems become part of the information infrastructure. The basic approach to this task is to incorporate an SQL interface in the OODBMS application programming interface (API) and to provide an SQL query processing capability.

Active vs. Passive Object Management System. OODBMS may be characterized as being an active or a passive object management system. A passive OODBMS means that the database does not store the implementation of the methods defined for a class. Applications built on a passive OODBMS provide in their executable image the code for each method defined in the system. Application execution results in each object, that is accessed during that execution, being moved from the database server process to the client application process. Once the object resides in the application's address space, a message may be sent to that object resulting in the object executing one of its methods. Note that the process of moving the object from the database server to the application's address space is typically transparent to the application programmer. An active OODBMS means that the database stores the implementation of object behaviors (i.e., methods) in the database. This allows objects to execute those behaviors (i.e., respond to messages) in the database server process. Advantages of an active data model include:

- Objects may be accessed and manipulated by nonobject-oriented programs. These programs may access objects in the database through a standard programming language interface. Each such access may result in a long series of messages being sent between many different objects that are cooperating to provide some useful service on behalf of the requesting application.
- Object behaviors are stored in a single location (the database), which makes it easier to be sure that all applications have the latest version of those behaviors. This also tends to isolate those applications from changes to the object behaviors.
- Each object accessed by the application need not be transferred to the application's address space.
- Consistency checks (i.e., constraints) can be automatically maintained by the database. As an object's state is changed, the database server process can automatically execute consistency checks to ensure that the new state does not violate some constraint.

One of the significant differences between an active and a passive OODBMS becomes apparent when considering the implications of traversing 10,000 objects as part of a query or other database operation. Using a passive database requires that each of those 10,000 objects be moved from the database server to the client application prior to invoking the methods that access the objects. An active database can be programmed so that the traversal and method invocation occurs in the database server process, eliminating the need to transfer each object across the network to the application process.

Database Architecture

This section provides an overview of architectural issues relevant to an OODBMS. Many papers has been published which will describe an implementation of a persistent memory system upon which object-oriented databases may be built and describes an implementation of shared database server architecture suitable as a back-end for an object-oriented database.

Distributed Client-Server Approach Advances in local area network and workstation technology have given rise to group design type applications driving the need for OODBMS (e.g., CASE, CAD, and Electronic Offices). OODBMS typically execute in a multiple process distributed environment. Server processes provide back-end database services, such as management of secondary storage and transaction control. Client processes handle application specific activities, such as access and update of individual objects. These processes may be located on the same or different workstations. Typically a single server will be interacting with multiple clients servicing concurrent requests for data managed by that server. A client may interact with multiple servers to access data distributed throughout the network.

The evaluations and benchmarks are the three alternative workstation-server architectures that have been proposed for use with OODBMS:

- *Object server approach.* The unit of transfer from server to client is an object. Both machines cache objects and are capable of executing methods on objects. Object-level locking is easily performed. The major drawback of this approach is the overhead associated with the server interaction required to access every object and the added complexity of the server software which must provide complete OODBMS functionality (e.g., be able to execute methods). Keeping client and server caches consistent may introduce additional overheads.
- *Page server approach.* The unit of transfer from server to client is a page (of objects). Page-level transfers reduce the overhead of object access since server interaction is not always required. Architecture and implementation of the server is simplified since it needs only to perform the backend database services. A possible drawback of this approach is that methods can be evaluated only on the client, thus all objects accessed by an application must be transferred to the client. Object-level locking will be difficult to implement.
- *File server approach.* The OODBMS client processes interact with a network file service (e.g., Sun's NFS) to read and write database pages. A separate OODBMS server process is used for concurrency control and recovery. This approach further simplifies the server implementation since it need not manage secondary storage. The major drawback of this approach is that two network interactions are required for data access, one to the file service and one to the OODBMS server.

Many scientists have identified no clear winner when benchmarking the three approaches. The page server approach seemed best with large buffer pools and good clustering algorithms. The object server approach performed poorly if applications scanned lots of data, but was better than the page server approach for applications performing lots of updates and running on workstations with small buffer pools.

Data Access Mechanism. An evaluation of OODBMS products should consider the process necessary to move data from secondary storage into a client application. Typically this requires communication with a server process, possibly across a network. Objects loaded into a client's memory may require further processing, often referred to as swizzling, to resolve references to other objects which may or may not already be loaded into the client's cache. The overhead and process by which locks are released and updated objects are returned to the server should also be considered.

Object Clustering. OODBMS which transfer units larger than an object do so under the assumption that an application's access to a given object implies a high probability that other associated objects may also be accessed. By transferring groups of objects, additional server interaction may not be necessary to satisfy these additional object accesses. Object clustering is the ability for an application to provide information to the OODBMS so that objects which it will typically access together can be stored near each other and thus benefit from bulk data transfers.

Heterogeneous Operation. An OODBMS provides a mechanism for applications to cooperate, by sharing access to a common set of objects. A typical OODBMS will support multiple concurrent applications executing on multiple processors connected via a local area network. Often, the processors will be from different computer manufacturers; each having its own data representation formats. For applications to cooperate in such an environment, data must be translated to the representation format suitable for the processor upon which that data is stored (both permanently by a server and temporarily by a client wishing to access the data). To be an effective integration mechanism, an OODBMS must support data access in a heterogeneous processing environment.

Database Functionality

The primary benefit an application derives from a database is that application data persists across application executions. Additional benefits offered by a database are the ability to share data between applications, provision of concurrent access to the data by multiple applications, and providing an application the access to a data space larger than its process address space. This section reviews the issues that distinguish an OODBMS from a language with persistence (e.g., Smalltalk). A language with persistence typically provides for data to exist across executions but does not provide the additional benefits outlined earlier.

Within this section we provide a brief overview of the database topic with additional issues relevant to object-oriented databases.

Access to Unlimited Data. A database provides an application, the ability to access a virtually unlimited amount of data. In particular, the application can address more data than would fit within the application process address space. Some databases may support the notion of transient data that is maintained during program execution but not saved in the database. This is useful for providing access to very large transient data objects that do not map easily into the application's address space.

Integrity. A database is required to maintain both structural and logical integrity. Structural integrity ensures that the database contents are consistent with its schema. Logical integrity ensures that constraints specifying logical properties of the data are always true. Many papers describe new concerns for integrity in OODBMS. A major concern is that OODBMS architectures map data directly into the applications address space (unlike a typical relational database that provided direct access to the data only in a separate database server process).

Mapping data into the application's address space yields significant performance improvements over server-only data access, especially for the application areas being targeted by OODBMS. However, once data is mapped into an application's address space there is no way to guarantee it is not inadvertently or maliciously tampered with. This limits a database's ability to guarantee integrity of the data. Structural integrity mechanisms include assurances that references to deleted objects do not exist and that all instances are consistent with their class definitions. Logical integrity can be supported by encapsulating all the data members of a class and providing access to information content of an object only through behaviors defined by the class.

Additional integrity constraints can be supported if the database provides a mechanism for specifying application-level constraints and for ensuring the execution of those constraints before and/or after behavior invocation. Constraints executed before a behavior can test the consistency of the request and the input parameters. Constraints executed after a behavior can test for logical consistency of the resulting state of the object and any output parameters.

Concurrency. Databases provide concurrency control mechanisms to ensure that concurrent access to data does not yield inconsistencies in the database or in applications due to invalid assumptions made by seeing partially updated data. The problems of lost updates and uncommitted dependencies are well documented in the database literature. Relational databases solve this problem by providing a transaction mechanism that ensures atomicity and serializability. Atomicity ensures that within a given logical update to the database, either all physical updates are made or none are made. This ensures the database is always in a logically consistent state, with the DB being moved from one consistent state to the next via a transaction. Serializability

ensures that running transactions concurrently yields the same result as if they had been run in some serial (i.e., sequential) order.

Relational databases typically provide a pessimistic concurrency control mechanism. The pessimistic model allows multiple processes to read data as long as none update it. Updates must be made in isolation, with no other processes reading or updating the data. This concurrency model is sufficient for applications that have short transactions, so that applications are not delayed for long periods due to access conflicts.

For applications being targeted by OODBMS (e.g., multiperson design applications), the assumption of short transactions is no longer valid. Optimistic concurrency control mechanisms are based on the assumptions that access conflicts will rarely occur. Under this scenario, all accesses are allowed to proceed and, at transaction commit time, conflicts are resolved. OODBMS have incorporated the idea of optimistic concurrency control mechanisms for building applications that will have long transaction times.

Handling of conflicts at committed time cannot simply abort a transaction, however, since one designer may be losing days or weeks of work. OODBMS must provide techniques to allow multiple concurrent updates to the same data and support for merging these intermediate results at an appropriate time (under application control). Most systems use some form of versioning system in order to handle this situation.

An alternative policy is to allow reading and a single update to occur in parallel. Readers are made aware that the data they are reading may be in the midst of an update. Thus readers may be viewing slightly outdated information. Implementation of this approach fits well in the client-server architecture typical for an OODBMS. Each client application gets its own local copy of the data. If an update is made to the data, the server does not permanently store it until all concurrent read transactions are completed. Thus, all read transactions execute seeing a consistent data set, albeit one that is in the process of being updated.

Once all readers have completed, the write transaction is allowed to complete modifying the permanent copy of the data. Some OODBMS may, at transaction commit, inform reading clients that the data they just read is in the process of being updated.

Recovery. Recovery is the ability for a database to return to a consistent state after a software or hardware failure. Similar to concurrency, the transaction concept is used to implement recovery and to define the boundaries of recovery activity. One or more forms of database journaling, backup, check-pointing, logging, shadowing, and/or replication are used to identify what needs to be recovered and how to perform a recovery.

Databases must typically respond to application failures, system failures, and media failures. Application failures are typically trapped by the transaction mechanism and recovery is implemented by rolling back the transaction. System failures, such as loss of power, may require log and/or checkpoint supported rollback of uncommitted transactions and rollforward of transactions

that were committed but not completely flushed to disk. Media failures, such as a disk head crash, require restoration of the database from a backup version, and replaying of transactions that have been committed since the backup.

The ability of a database to recover from failures results in a heavy processing and storage overhead. In the process of evaluating an OODBMS, its ability to recover from faults, and the overhead incurred to provide that recovery capability, must be carefully considered. Applications envisioned for OODBMS (e.g., CASE tools) often do not have the same strict recovery requirements as do relational database applications (e.g., banking systems). In addition, the amount of data stored in such systems may result in unacceptable storage overheads for many forms of recovery. For these reasons, an OODBMS evaluation effort must carefully select the recovery capabilities needed based on both the functional and performance requirements of the application.

Transactions. Transactions are the mechanism used to implement concurrency and recovery. Within a transaction, data from anywhere in the (distributed) database must be accessible. A feature found in many OODBMS products is to commit a transaction but to allow the objects to remain in the client cache under the expectation that they will soon be referenced again.

Some OODBMS have incorporated the concept of long and/or nested transactions. A long transaction allows transactions to last for hours or days without the possibility of system generated aborts (due to lock conflicts for example). System generated aborts must be avoided for applications targeting OODBMS since a few hours or days of work cannot be simply discarded. Long transactions may be composed of nested transactions for purposes of recovery. Nested transactions allow a single (root) transaction to be decomposed into multiple subtransactions.

Each subtransaction can be committed or aborted independently of the other subtransactions contained in the scope of the root transaction. Each subtransactions commit is dependent upon its immediate superior committing and the root transaction committing. Nested transactions improve upon the basic transaction mechanism by providing finer-grained control over the failure properties of software. Using nested transactions, a portion of a computation can fail and be retried without affecting other parts of that same computation. Nested transactions were developed to provide concurrency controls for distributed computing systems. Again, an OODBMS evaluation must carefully consider whether the target application requires nested transactions and the performance and/or storage impacts of using this facility.

Deadlock Detection. Database systems use deadlock detection algorithms to determine when applications are deadlocked due to conflicting lock requests. Relational DBMS typically select an arbitrary transaction and abort it in an attempt to let the remaining transactions complete. The aborted transaction is normally restarted automatically by the system. This scheme works well for the class of applications supported by relational DBMS. As described in the previous section, applications being targeted by OODBMS cannot afford

a system-aborted transaction resulting in the potential loss of hours, days, or weeks of work. OODBMS provide alternative concurrency control and transaction mechanisms to reduce or avoid the possibility of deadlocks. Regardless of concurrency control protocol, an OODBMS must still detect and resolve deadlocks.

Locking. Locking of database entities is the typical approach to implementing transactions. OODBMS may provide locking at the object and/or the page level. Object-level locking may result in high overheads due to lock management. Page-level locking may reduce concurrency, especially for write locks, since not all objects on a page may be used in the transaction that holds the lock. OODBMS clustering facilities will aid in reducing the loss of concurrency due to page-level locking.

An OODBMS will implicitly acquire and release locks as data is accessed by an application. Application support may be necessary for specifying the lock mode (e.g., all locks may be acquired in read mode by default and the application must specify that write access to the object is necessary). The OODBMS may also provide an interface for explicitly locking data.

Backup and Restore. Backup is the process of copying the state of the database to some other storage medium in case of subsequent failure or simply for historical record. Ideally backups can be performed while the database is in use. Backups may be performed only on specific sections of the database. Incremental backup capabilities reduce the amount of information that must be saved by storing only changes since a prior backup. Obviously, a database must be able to be restored from a backup.

Dump and Load. A database may be dumped into a human readable ASCII format. Specific segments of the database may be dumped. A database may be recreated by loading from a dump file.

Constraints. Constraints are application-specific conditions that must always hold true for a database. Logical database integrity can be supported by providing the application developer with the ability to define constraints and for those constraints to be automatically executed at the appropriate times. Although constraints can be directly encoded in behaviors, having a separate constraint mechanism reduces duplication and ensures execution of the constraints. Some OODBMS define a model for constraint definition and invocation but require applications to explicitly invoke the constraint executions.

Constraints executed at behavior invocation can test for validity of the input parameters and the requested operation. Constraints executed at the completion of a behavior invocation can test for logical consistency of the result values and resulting database state.

Notification Model. An OODBMS may provide a passive and/or an active notification model. A notification model allows an application to be informed when an object is modified or when some other database event occurs. Passive notification systems require that the application query the database for state

changes. A passive system minimally provides the logic that determines if an object has changed state or if an event has occurred.

The application is responsible for informing the database of the objects and events in which it is interested and for periodically querying the database to see if those objects have been updated or if particular events have occurred.

An active notification system is one in which application routines are invoked automatically as a result of some object being updated or some database event was occurring. Active notification systems are similar to database triggers and constraint systems (whereby constraints are executed when one of their operands changes state). Presently there is no clear indication that such mechanisms can be efficiently supported in general purpose database management systems.

Indexing. Databases make use of indexing to provide optimized data retrieval based on some aspect of that data. In particular, query evaluations can be dramatically improved by the presence of indexes. Query optimizers must dynamically determine if an index is present and, if so, use that index to provide an efficient query execution.

Indexes in an OODBMS are typically built to provide lookup of all objects of a given class and its subclasses based on one or more data members of that class. Indexes may be segregated to some segment of a database or may span the entire database.

Different indexing implementations result in different time and space performance. Hashing and b-trees are two common index implementation techniques. Indexing add to the overhead associated with creating, deleting, and modifying objects and thus must be used judiciously.

Storage Reclamation. Data stored within a database is dynamically created and destroyed by the applications that access that database. Data that is no longer accessible, whether determined implicitly by the system or as the result of an explicit delete by an application, results in storage space that is no longer in use. Reclamation of unused space must be performed so that a database does not continuously grow. Space reclamation should be performed incrementally or as a background activity so that performance hiccups are not encountered by the applications.

Security. Secure database systems protect their data from malicious misuse. Security requirements are similar to data integrity requirements that protect data from accidental misuse. Secure databases typically provide a multilevel security model where users and data are categorized with a specific security level. Mandatory security controls ensure that users can access data only at their level and below. Discretionary security controls provide access control based on explicit authorization of a user's access to data.

Applications targeted by OODBMS often do not require strict security controls, although discretionary access controls seem desirable for work-group design type applications. Little work has been done to add security mechanisms to OODBMS.

Application Programming Interface

A major distinction between an object-oriented database application and a relational database application is the relation between the data manipulation language (DML) and the application programming language. Large scale commercial database applications are developed in a standard programming language (e.g., C, C++). In a relational database, SQL is used as the DML, providing the means to create, update, query, and delete data from the database. Thus, in a relational database application, a significant amount of effort is spent transforming data between the two different languages.

Critics claim that this impedance mismatch adds to the complexity of building relational database applications. In an object-oriented database application, the DML is typically very near to a standard programming language. For example, many OODBMS support C++ as their DML, with the result that C++ can be used for building the entire application. OODBMS proponents claim that by using a single language, the impedance mismatch is removed (or significantly reduced) and application development is simplified. The OODBMS approach is not without its critics. OODBMS applications traverse the network of interconnected objects to discover information. This hand-coded navigation of the schema cannot automatically take advantage of indexes that might be added and requires modification of the application code as the schema changes.

SQL, on the other hand, is an associative retrieval language, not requiring any information describing how information is to be found. SQL statements can be recompiled and/or optimized at runtime to find the best access path to the requested data.

It is clear that both relational and object-oriented database approaches have merit and will be used for developing particular classes of applications. It is interesting to note that relational vendors are currently adding object-oriented features to their products and those OODBMS vendors are adding SQL-like query capabilities to theirs. In addition, third party companies are developing automated integration tools that allow SQL applications to access OODBMS and OODBMS to access relational databases.

This section covers issues relevant to the application programming interface of an OODBMS. As described earlier, OODBMS aim to provide a tight integration between the data definition/manipulation language and a standard programming language in an attempt to ease the application development task.

DDL/DML Language. An OODBMS provides one or more languages in which data definitions can be specified and applications can be constructed. A common example is to use C++ header files for describing the class structures (i.e., schema) and then to implement the behaviors of those classes and the remainder of the application in C++. The data description component of the language often is an extension of some standard programming language to support specification of relationships and other object-oriented features.

Computational Completeness. Relational database query languages, such as SQL, are typically not computationally complete, meaning general purpose control and computation structures are not provided. For this reason, applications are built by embedding the query language statements in a standard programming language. OODBMS that use a standard programming language (or an extension of one) for data definition and data manipulation provide the application developer with a computationally complete language in which database manipulations and general purpose processing can be accomplished.

Language Integration Style. A number of mechanisms may be used for providing access to the database from the programming language. These include library interfaces, language extensions, or for true object-oriented languages, definition of behaviors for construction, destruction, and access via member functions. Most authors define loose language integration as one where the database operations are explicitly programmed, for example by library calls. This is common for OODBMS interfaces from the C programming language. Tight language integration makes the database operations transparent, typically through inherited behaviors in a class hierarchy. For example, C++ integration might use the standard class constructor and destructor constructs to create and delete objects from the database. Even in a tight integration, additional parameters will often be added to control database activity (e.g., clustering) and some database operations will be provided by library calls.

Data Independence. Data independence is the ability for the schema of the database to be modified without impacting the external (i.e., application) view of that schema. As described earlier, encapsulation is accomplished by the schema definition language encapsulating the internals of data members and behaviors. Applications that rely only on the public interfaces of classes will be protected from changes to the private portions and the implementations of those classes.

The concept of derived attributes also adds to data independence. A derived attribute is one that is not stored, but instead calculated on demand. An application cannot determine whether an attribute is derived or not (i.e., if it is stored or computed), thus changes to the schema do not affect the application. Typically, programming languages do not easily support the concept of derived attributes. In C++ for example, a function call is syntactically different than an attribute reference, so providing transparent derived attributes is not possible. Eiffel is an object-oriented programming language that directly supports the concept of a derived attribute.

Standards. Applications can be portable across OODBMS products if all such products agree on a standard application programming interface (API). The Object Database Management Group (ODMG) is a working group of OODBMS vendors tasked with defining a set of interface specifications aimed at ensuring application portability and interoperability. All members of OODBMS vendors have agreed to support the standard specification once it is developed (initial version expected in the fall of 1993). One of these

specifications will be a C++ binding for object definition, manipulation, and query. Currently no standard exists and all OODBMS have a custom API, thus application programs are not portable across databases. A concern is that any existing applications will have to be modified when a standard is defined and implemented by the OODBMS vendors. OODBMS typically integrate with a standard programming language such as C, C++, or Smalltalk. One issue is whether the OODBMS works with a standard version of a programming language (e.g., ANSI C, AT&T compatible C++). A second is whether the OODBMS uses a custom-built compiler or can use any third party compiler (e.g., from a compiler vendor).

Querying an OODBMS

Query languages provide access to data within a database without having to specify how that data is to be found. Thus, query languages provide a level of data independence. An application or user need not understand the structures storing the data in order to access the data. This is in strong contrast to network and hierarchical databases that require programmed navigation of the database.

Included in these requirements are a flexible-type system, inheritance, association of behaviors with data, and the use of rules or constraints. In particular, hand-coded navigational access has been shown to be less optimal than an optimized query approach, and changes to schema and indexing require changes to the navigational segments of application code. In light of this, it is clearly important for OODBMS to provide a declarative query language that is closely integrated with an object-oriented programming language.

Associative Query Capability. Standard Query Language (SQL), as defined by the ANSI X3H2 committee is being revised to incorporate object-oriented features. In the short term, OODBMS vendors are providing their own object-oriented extensions to SQL in order to provide reasonable access to their object databases from an SQL like language.

A range of query capabilities for OODBMS:

- *No query language support.* All data access must be via programmed database navigation.
- *Collection-based queries.* Queries operate on some predefined collection of objects, selecting individual objects based on some predicate, yielding a resulting collection of objects.
- *General queries.* Which can have a result of any type (e.g., value, object, or collection). An additional capability is for a query to return textual information that is suitable for report generation.

Data Independence. The basic motivation for using a declarative query language is to support data independence. Query implementations are expected to make optimal use of schema associations and indexes at runtime.

Impedance Mismatch. One of the main criticisms of relational database programming is the impedance mismatch between the data manipulation language (DML), normally SQL, and the application programming language, typically some general purpose language such as C. Relational database applications have an impedance mismatch, in that database access via the query language is table-based while application programming is individual value-based. Extra code and intellectual hurdles are required to translate between the two.

A presumed benefit of OODBMS is that the application programming language and the DML are the same. However, as noted earlier, critics claim this eliminates data independence. Declarative query capabilities, which are being added to OODBMS, will support the concept of data independence. How well these query capabilities are integrated with the application programming language will dictate the level of impedance mismatch between the application programming language and the use of associative queries.

A range of techniques for integrating queries within an OODBMS:

- Supply a Select method on all persistent classes
- Extend the object programming language to include SQL like predicates for filtering selection operations
- Embed the SQL Select statement into the object programming language, providing a preprocessor which translates these statements into an appropriate set of runtime calls

A tight integration of query invocation and query result with the selected OODBMS application language will decrease the impedance mismatch typical of database applications.

Query Invocation. As described throughout this section, the major emphasis is to provide an associative query capability from within programmed applications. An additional requirement is to provide a means for ad hoc query invocation, possibly from within a database browser tool.

Invocation of Programmed Behaviors. The query language should be able to invoke object behaviors as part of their predicates. Whether this can be done from programmed queries and/or ad hoc queries must be investigated.

Application Development Issues

Functionality directly affects application development, for example, language integration affects how developers perceive the use of the database. Another functional issue that is extremely important to the application development process is schema migration. The ability to migrate schema (i.e., to update objects in response to schema changes) affects the testing process and the ability to upgrade fielded versions of the software.

This section identifies more specific application development issues. Evaluation of application development issues is not as straightforward as that for

Table 11.3. Application development evaluation criteria

Criteria Categories	Criteria
Developer's View of Persistence	
Application Development Process	
Application Development Tools	<ul style="list-style-type: none"> – Database Administration Tools – Database Design Tools – Source Processing Tools – Database Browsing Tools – Debugging Support – Performance Tuning Tools
Class Library	

functional issues. Functional evaluations can be based on technical documentation and informally verified by reviewing interface specifications of the product. Application development issues are more abstract. Review of technical documentation will provide only a small glimpse of the application development process. Only through use of the product, on a large application, with a team of developers, will a true understanding of the application development process be derived.

Table 11.3 gives Application Development Evaluation Criteria that define application development issues to be considered in performing a review of OODBMS products.

Developer's View of Persistence

An evaluation should consider how a software developer perceives persistent objects and what coding constructs are used to access persistent objects. This issue is closely related to the language integration issue. Of particular interest is the need for explicit user code to access persistent objects, lock persistent objects, and to notify the database that a persistent object has been updated and must be stored back to the database.

Application Development Process

The application development process is the series of steps needed to define and structure databases, define database schema, process class behaviors, and to link, execute, and debug applications. An evaluation should consider the need to use vendor-supplied preprocessors and/or interactive schema development tools, should describe the integration with debuggers and program builders (e.g., UNIX make), and should consider the issues relevant to multiple developer efforts.

Application Development Tools

Both vendor and third party tools must be supplied in the areas described in the following sections.

Database Administration Tools. Database administration tools are used for creating, deleting, and reorganizing (e.g., moving) databases. Database administration tasks should be available to applications (i.e., through a programmed interface) in order to allow applications to hide database administration tasks from the user.

Database Design Tools. Database design tools are used for interactively defining the schema or classes which are to be stored within a database. Third party object-oriented modeling tools may be available which generate source code suitable for use with the OODBMS.

Source Processing Tools. Source processing tools perform the transformation of textual descriptions of applications programs into executable code. These include preprocessors specific to the OODBMS as well as standard language compilers. Also included might be tools to aid in controlling the process of application building (e.g., UNIX make facilities). Integration with application development environments should also be considered.

Database Browsing Tools. Interactive browsing tools allow database schema and contents to be viewed graphically and possibly modified.

Debugging Support. An OODBMS vendor should supply tools and or utilities that are useful during the debugging process. These facilities should be inviolable from third party debugging environments.

Performance Tuning Tools. An OODBMS should provide utilities which enable a developer to understand the performance parameters of an application and a means by which performance can be adjusted as a result of this analysis. In addition, any specific design considerations that can affect performance must be considered.

Class Library

Object-oriented development is based on building a reusable set of classes organized into a class hierarchy. The class hierarchy mechanism supports reuse of general data and behaviors in specialized classes. As described earlier, Language Integration Style, some OODBMS may provide their application interface through a class library. Inherited behaviors provide support for persistence, object creation, deletion, update, and reference traversal.

In addition to the possibility of providing database interface through the class library, an OODBMS may deliver application support classes. Such classes typically provide data abstractions such as sets, lists, dictionaries, etc. These classes should be extensible just like any other user-defined class. Ideally source code would be provided for these classes. Source is needed since documentation is often insufficient for determining the effect of invoking each method under each possible condition. Source is also useful in understanding performance characteristics and in repairing errors that may be found in the code.

Table 11.4. Miscellaneous evaluation criteria

Criteria
Product Maturity
Product Documentation
Vendor Maturity
Vendor Training
Vendor Support and Consultation
Vendor Participation in Standards Activities

Miscellaneous Criteria

A number of nontechnical criteria should also be considered when evaluating an OODBMS. This section details some of these criteria, as listed in Table 11.4, Miscellaneous Evaluation Criteria.

Product Maturity

Product maturity may be measured by several criteria including:

- Years under development
- Number of seats licensed
- Number of licensed seats actually in use
- Number of licensed seats in use for purposes other than evaluations (i.e., actual development efforts)
- Number and type of applications being built with the OODBMS product
- Number and type of shipped applications built with the OODBMS product

Product Documentation

Product documentation should be clear, consistent, and complete. The documentation should include complete examples of typical programmed capabilities (e.g., what is the sequence of calls to access data from the database and to cause updates to that data to be made permanent in the database).

Vendor Maturity

Vendor maturity may be measured by several criteria including:

- Company's size and age
- Previous experience of the company's lead technical and management personnel in the commercial database market
- Financial stability

Vendor Training

Availability and quality of vendor supplied training classes is an important consideration when selecting an OODBMS.

Vendor Support and Consultation

It is expected that significant support will be required during the OODBMS evaluation process and to overcome the initial learning curve. OODBMS vendors should provide a willing and capable support staff. Support should be available via phone and electronically. Consulting support might also be appealing where the OODBMS vendor provides expert, hands-on assistance in product use, object-oriented application design (especially in regards to database issues), and in maximizing database application performance.

Vendor Participation in Standards Activities

The vendor should be active in standards efforts in the object-oriented, language, CASE, open software, and data exchange areas. In particular:

- *Object Management Group (OMG)*. An organization funded by over 80 international information systems corporations whose charter is to develop standards for interoperation and portability of software. The OMG is focusing on object-oriented integration technologies such as Object Request Broker (ORB), OODBMS interfaces, and object interfaces for existing applications.
- *Object Database Management Group (ODMG)*. An organization of OODBMS vendors chartered to define a standard interface to OODBMS that will allow application portability and interoperability. Standards defined by the ODMG will be provided to OMG, ANSI, STEP, PCTE, etc. to aid in their respective standardization efforts.
- ANSI standardization efforts in languages (C, C++, Smalltalk), SQL, and object-oriented databases.
- Standards such as Portable Common Tool Environment (PCTE) and CASE Data Interchange.
- Format (CDIF) providing for common data representations, data exchange formats, and interoperation of tools.
- *PDES/STEP*. An effort aimed at standardizing an exchange format for product model data (product model data, such as CAD data, represents a prime application area for OODBMS).

11.1.7 Evaluation Targets

This chapter identifies the commercial OODBMS that were evaluated as part of this effort. For each evaluation target we identify:

- The platforms upon which that OODBMS is hosted
- The level of heterogeneous operation supported by the OODBMS
- The application interface languages provided by the OODBMS
- Third party products that interoperate in some way with the OODBMS

Information provided in this section was provided directly by each vendor.

Objectivity/DB

Objectivity/DB is an object-oriented database product developed and marketed by Objectivity, Inc., 800 El Camino Real, Menlo Park, CA 94025, (415) 688-8024. Evaluation information provided in this report was obtained from the technical documentation set for Objectivity/DB Version 2.0 and from discussions with technical representatives of Objectivity, Inc.

Objectivity/DB may be executed by client applications hosted in a heterogeneous network of:

- DECstation under Ultrix 4.2
- Sun4/SPARC under SunOS 4.1, Solaris 2.0 or Solaris 2.1
- VAX under Ultrix 4.2 or VMS
- Hewlett Packard 9000 series 300 under HP/UX 8.0
- Hewlett Packard 9000 series 700 or 800 under HP/UX 8.0 or HP/UX 9.0
- IBM RISC System/6000 under AIX
- Silicon Graphics Iris under IRIX 4.0
- NCR System 3300 (i386) under SVR4 Version 2.0

Applications running on any of the earlier platforms, connected via a local area network, may share access to a single database. Objectivity, Inc., has announced and released a Beta test subset version of Objectivity/DB for Windows NT. A version running on DEC/ALPHA under OSF 1.0 is also in Beta testing.

Objectivity/DB provides application interfaces for:

- AT&T compatible C++
- ANSI C

Objectivity/DB was designed as an open product and is advertised to work with any ANSI C or AT&T compatible C++ compiler. Objectivity, Inc., has partnership agreements to develop tool integrations and/or be compatible with the following products:

1. Program Development Environments:
 - SoftBench from Hewlett-Packard
 - ObjectCenter from CenterLine Software
 - FUSE from DEC
 - WorkBench 6000 from IBM
 - ObjectWorks from ParcPlace
2. RDBMS Gateways:
 - Persistence Software
3. Object-oriented GUIs:
 - UIM/X from Visual Software, Ltd.
 - XVT from XVT Software, Inc.
 - Integrated Computer Solutions, Inc.
 - Objective, Inc.
 - Micram Classify/DB.

4. Analysis and Design Tools:
 - PTech from Associative Design Technology
 - Paradigm Plus from ProtoSoft
 - ROSE from Rational
 - Softeam

ONTOS DB

ONTOS DB is an object-oriented database product developed and marketed by ONTOS, Inc., Three Burlington Woods, Burlington, MA 01803, (617) 272-7,110. Evaluation information provided in this report was obtained from the technical documentation set for ONTOS DB 2.2 and from discussions with technical representatives of ONTOS, Inc.

ONTOS DB may be hosted on the following platforms:

- IBM RISC System/6000 under AIX
- IBM PC under OS2
- Hewlett Packard 9000 series under HP/UX
- SCO 386 Unix
- Sun4/SPARC under SunOS 4.1

ONTOS DB does not support heterogeneous operation between any other target platforms. ONTOS DB provides a C++ application interface. ONTOS DB can be used with AT&T compatible C++ compilers.

ONTOS DB developers can debug using gdb or dbx UNIX debugging environments. ONTOS DB is compatible with the following products:

1. Program Development Environments:
 - ObjectCenter from CenterLine Software
 - ObjectWorks from ParcPlace
2. Analysis and Design Tools:
 - PTech from Associative Design Technology

VERSANT

VERSANT is an object-oriented database product developed and marketed by Versant Object Technology Corp., 4500 Bohannon Drive, Menlo Park, CA 94025, and (415) 329-7,500. Evaluation information provided in this report was obtained from the technical documentation for VERSANT Release 2 and from discussions with technical representatives of Versant Object Technology Corporation.

VERSANT may be hosted on the following platforms: o Sun4/SPARC under SunOS 4.0:

- IBM RISC System/6000 under AIX
- Hewlett Packard 9000 series under HP/UX

- DECstation 3100 under Ultrix
- Sequent under DYNIX/ptx
- Silicon Graphics under IRIS
- NeXT under NeXTstep
- IBM PC under OS2

VERSANT supports heterogeneous operation between their Sun4, Hewlett Packard, and IBM RISC System/6000 platforms. Versant is adding additional platforms to their heterogeneous operation as an ongoing activity. For example, the addition of OS2 platforms are expected before the end of this year.

VERSANT provides application interfaces for:

- C++
- ANSI C
- Smalltalk

C++ compilers from AT&T, Sun, Hewlett Packard, Glockenspiel are compatible with VERSANT. Versant Object Technologies has partnership agreements to develop tool integrations and/or be compatible with the following products:

1. Program Development Environments:
 - ObjectCenter from CenterLine Software
 - ObjectWorks from ParcPlace
 - SoftBench from Hewlett-Packard
 - WorkBench 6000 from IBM
2. RDBMS Gateways:
 - Persistence Software
3. Analysis and Design Tools:
 - Paradigm Plus from ProtoSoft
 - ROSE from Rational
 - ACIS Geometric Modeler from Spatial Technology

ObjectStore

ObjectStore is an object-oriented database product developed and marketed by Object Design, Inc., One New England Executive Park, Burlington, MA 01803, (617) 270-9,797. Evaluation information provided in this report was obtained from the technical documentation set for ObjectStore Release 2.0 and from discussions with technical representatives of Object Design, Inc.

ObjectStore may be hosted on the following platforms:

- Sun under Solaris 1.x and Solaris 2.x
- Hewlett Packard under HP/UX
- DEC under Ultrix
- NCR under SVR 4
- Univel under SVR 4

- Olivetti under SVR 4
- IBM RISC System/6000 under AIX
- Silicon Graphics
- IBM PC under Windows 3.1 and OS2

ObjectStore supports heterogeneous operation between their Sun, Hewlett-Packard, IBM RISC System/6000, and Silicon Graphics implementations. The next release of ObjectStore will support heterogeneous operation across all their implementations.

ObjectStore provides application interfaces for:

- AT&T compatible C++
- ANSI C

ObjectStore is advertised to work with any ANSI C or AT&T compatible C++ compiler. In addition, Object Design markets a compiler and other development tools for building ObjectStore applications.

ObjectStore has completed partnership agreements to develop tool integrations and/or be compatible with the following products:

1. Program Development Environments:
 - Borland C++ & Application Frameworks.
 - CodeCenter and ObjectCenter from CenterLine Software.
 - Energize Programming System, Lucid C++, and Lucid C from Lucid, Inc.
 - SynchroWorks from Oberon Software, Inc.
 - OpenBase from Prism Technologies, Ltd.
 - SPARCworks C++ Professional and ProWorks C++ from SunPro Marketing.
2. Object-oriented GUIs:
 - ViewCenter from CenterLine Software.
 - zApp from Inmark Development Corp.
 - Devguide from SunSoft, Inc.
 - UIM/X from Visual Software, Ltd.
3. Analysis and Design Tools:
 - Object Engineering Workbench for C++ from Innovative Software GmbH.
 - HOOPS Graphics Development System from Ithaca Software.
 - Paradigm Plus from ProtoSoft, Inc.

GemStone

GemStone is an object-oriented database product developed and marketed by Servio Corporation, 2085 Hamilton Ave., Suite 200, San Jose, CA 94125, (408) 879-6,200. Evaluation information provided in this report was obtained from the technical documentation set for GemStone Version 3.2, from GeODE

Version 2.0 (GeODE is a development environment for GemStone applications) and from discussions with representatives of Servio Corp.

GemStone may be hosted on the following platforms:

- Sun4/SPARC under SunOS 4.1
- IBM RISC System/6000 under AIX
- DEC Station under Ultrix
- Hewlett Packard 9000 under HP/UX
- Sequent under DYNIX/ptx

GeODE is available on all of the platforms listed earlier. Servio provides Macintosh, PC/Windows 3.1, and S/2 V2.0 Smalltalk application access to GemStone databases on the previously listed platforms. Applications running on any of the earlier platforms, connected via a local area network, may share access to a single database.

GemStone provides application interfaces for:

- GeODE
- C++
- C
- Smalltalk-80, Smalltalk/V
- Smalltalk DB, a multiuser extended dialect of Smalltalk, developed by Servio for building and executing GemStone applications

Programs written in any of these languages can access GemStone objects simultaneously. C++ compilers from Sun, HP, Centerline, Sequent, and IBM are compatible with the C++ interface for GemStone. Smalltalk compilers from ParcPlace and Digitalk are compatible with the Smalltalk interface for GemStone.

Servio provides GeODE (GemStone Object Development Environment) for developing GemStone applications. This environment is a complete application development framework, providing both visual and textual construction of object-oriented database programs. GeODE supports construction of Motif-based X Windows applications. GeODE includes tools for schema design, user interface construction, data browsing, reuse, visual software construction, and software debugging. Servio provides the GemStone Data-bridge, a product providing access to SYBASE relational databases from GemStone applications.

ObjectStore PSE Pro

ObjectStore PSE Pro is a Java-based object-oriented database product developed and marketed by Object Design, Inc., 25 Mall Road Burlington, MA 01803, (781) 674-5,000. Evaluation information provided in this report was obtained from the technical documentation set for ObjectStore PSE Pro Version 2.0.

ObjectStore PSE Pro may be hosted on the following platforms that support Java VMs:

- Windows 95
- Windows NT
- OS/2
- Macintosh
- Unix

ObjectStore PSE Pro supports JDK 1.1 or higher.

IBM San Francisco

IBM San Francisco is an object-oriented framework developed and marketed by IBM. Evaluation information provided in this report was obtained from the technical documentation set for IBM San Francisco Version 1.2, along with discussions with technical representatives of IBM, San Francisco division.

The IBM San Francisco framework may be used to develop client and server Java-based applications hosted in a heterogeneous network of:

- Windows 95 (client)
- Windows NT Workstation 4.0
- Windows NT Server 4.0
- AIX 3.1
- OS/400

Server applications can interface to ODBC-JDBC supported databases, or directly through object-relational mapping to:

- Microsoft SQL Server 6.5
- IBM DB2

11.1.8 Object Relational DBMS

In spite of the impact of relational databases in last decades, this kind of databases has some limitations to support data persistence required by actual applications. Due to recent hardware improvements more sophisticated applications have emerged, such as CAD/CAM (Computer-Aided Design/Computer-Aided Manufacturing), CASE (Computer-Aided Software Engineering), GIS (Geographic Information System), etc. These applications can be characterized as consisting of complex objects related by complex relationships. Representing such objects and relationships in the relational model implies that the objects must be decomposed into a large number of tuples.

Thus, a considerable number of joins is necessary to retrieve an object and, when tables are too deeply nested, performance is significantly reduced. A new generation of databases has appeared to solve these problems: the object-oriented database generation, which includes the object-relational and object databases. This new technology is well suited for storing and retrieving

complex data because it supports complex data types and relationships, multimedia data, inheritance, etc. Nonetheless, good technology is not enough to support complex objects and applications. It is necessary to define methodologies that guide designers in the object database design task, in the same way traditionally has been done with relational databases.

Unfortunately, none of these proposals can be considered as “the method,” neither for object-relational nor for object databases. On the one hand, they do not consider last versions of the representative standards for both technologies: ODMG 3.0 for object databases and SQL: 1999 for object-relational databases. And, on the other hand, some of them are based on techniques as OMT or, even, on the E/R model. So, they have to be updated considering UML, SQL: 1999, and ODMG 3.0 as their reference models. Object databases are well suited for storing and retrieving complex data by allowing the user to navigate through data. However, object-relational technology, that is, relational technology extended with new capabilities, such as triggers, methods, user defined types, etc., presents two advantages compared with object databases: it is compatible with relational technology and provides a better support for complex applications. Therefore, object-relational databases are expected to have a bigger impact in the market than object databases. For these reasons in this section we focus on object-relational databases design.

In this chapter we propose a methodology for object-relational database design. As conceptual modeling technique we have chosen the UML class diagram. UML, as a Universal Modeling Language is every day more accepted. It also presents the advantage of being able to model the full system, including the database model, in a uniform way. Besides, as UML is an extensible language, it is possible to define the required stereotypes for specific applications. The methodology provides some guidelines to translate a conceptual schema (in UML notation) into a logical schema.

As logical model we use the SQL: 1999 object-relational model so that the guidelines were not dependent of the different implementations of object-relational products. We use Oracle8i as an implementation example. In this section, we focus on aggregation and composition design. In the framework of our methodology, we propose specific guidelines to design aggregations and compositions in an object-relational model. Although the methodology, as we have explained earlier, is mainly based in SQL: 1999, in this section, we focus on the aggregation and composition implementation in Oracle8i. The reason is that this product supports a collection data type, the nested table, which is specially appropriated to implement aggregations and compositions. This collection data type is not provided neither by SQL: 1999 nor by other products, such as Informix Universal Server.

11.1.9 Object-Relational Model

In this section we summarize the object model of the current standard for object-relational databases, SQL: 1999, as well as the main characteristics

of Oracle8i object-relational model, as an example of object-relational product. SQL: 1999 data model extends the relational data model with some new constructors to support objects. Most of last versions of relational products include some object extensions. However, and because in general these products have appeared in the market before the standard approval, current versions of object-relational products do not totally adjust to the SQL: 1999 model.

Object Model of the SQL: 1999

SQL: 1999 is the current standard for object-relational databases. Its data model tries to integrate the relational model with the object model. In addition to the object extensions, SQL: 1999 provides other extensions to the SQL92, such as triggers, OLAP extensions, new data types for multimedia data storage, etc. One of the main differences between the relational and the object-relational model is that the First Normal Form (1NF), the basic rule of a relational schema, has been removed from the object-relational model. So, a column of an object table can contain a collection data type.

SQL: 1999 allows user to define new structured data types according to the required data types for each application. Structured data types provide SQL: 1999 the main characteristics of the object model. It supports the concept of strongly typed language, behavior, encapsulation, substitutability, polymorphism, and dynamic binding. Structured types can be used as the type of a table or as the type of column. A structured type used as the base type in the definition of a column, permits representing complex attributes; in this case, structured types represent value types. A structured type used as the base type in the definition of a table corresponds to the definition of an object type (or a class), being the table the extension of the type. In SQL: 1999 these kinds of tables are called typed tables. An object in SQL: 1999 is a row of a typed table.

When a typed table is defined, the system adds a new column representing the OID (Object Identifier) of each object of the table. The value of this attribute is system generated, it is unique for each object and the user cannot modify it. Figure 11.5 shows an example of a structured type defined in SQL: 1999; in (a) the structured type is used as a value type (as the type of a column of a table) whereas in (b) it is used as an object type (as the type of a table).

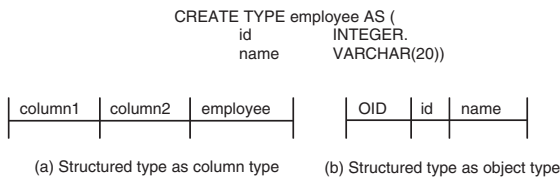


Fig. 11.5. Structured types used as value and object

A structured type can include associated methods representing its behavior. A method is an SQL function, whose signature is defined next to the definition of the structured type. The body specification is defined separately on the signature of the method.

SQL: 1999 supports simple inheritance for structured types and for typed tables. A subtype inherits the attributes and the behavior of the supertype. A subtable inherits the columns, restrictions, triggers, and methods of the supertable. A row of a typed table is an object and differs from the rest of objects by its OID. The value of the OID is generated by the system when a new object is inserted in the table. The type of this column is a reference type (REF). Therefore, each typed table has a column that contains the OID value. There are different REF types, one for each object type; that is, the REF type is a constructor of types rather than a type itself. An attribute defined as reference type holds the OID of the referred object. So, the REF type permits implementing relationships without using foreign keys. SQL: 1999 supports another structured type: the ROW type. The ROW type is a structured type defined by the user. It has neither extension nor OID. So, it cannot be used as an object type.

SQL: 1999 only supports a collection type ARRAY. The ARRAY can be used whenever another type can be placed (as the type of an attribute of a structured type, as the type of a column, etc.). The ARRAY type allows representing multivolume attributes not forcing tables to be in 1NF.

Object Model of Oracle8i

As well as SQL: 1999, Oracle8i supports structured data types that can be defined by the user (although, with a different syntax). A structured type can be used, as in SQL: 1999, as a column type or as a type of a table. A structured type used as a column type represents a value type and a structured type used as the type of a table represents an object type, being the table the extension of the type. Each row of this kind of tables is an object and, in the same way as in SQL: 1999, they have a special column of reference type (REF) that allows identifying each object (OID). It is also possible to define an attribute as a reference to an object type. Oracle8i allows associating behavior to object types, defining the signature of the methods as part of the type definition. The body of the method is defined separately.

Oracle8i supports two kinds of collections: VARRAYS, equivalent to the SQL: 1999 ARRAY and the nested table. A nested table is a table that is embedded in another table. It is possible to define a table data type and to use this type as a column type in a table. So, this column contains a table (called nested table) with a collection of values, objects, or references. Figure 11.6 shows an example of nested table (C_Table).

One of the main differences between Oracle8i and SQL: 1999 object-relational model is that Oracle8i does not support inheritance, neither of types nor of tables. There exist, however, some relational products, as for example,

column1	column2	C Table		
		<table border="1"> <tr> <td>c'</td> <td>c''</td> </tr> </table>	c'	c''
c'	c''			

Fig. 11.6. A nested table in Oracle8i

Universal Server of Informix, those support the inheritance concept in a similar way as the standard. However, another difference that makes Oracle8i more powerful than SQL: 1999 is related with the collection types. The nested table not supported by SQL: 1999, allows to represent an object collection embedded in another object, that could be a natural way to implement the UML aggregation.

11.1.10 Aggregation and Composition in UML

An aggregation is a special form of association between classes that represents the concept of “WHOLE PART.” Each object of one of the classes that belong to the aggregation (the composed class) is composed of objects of the other class of the aggregation (the component class). The composed class is often called “whole” and the component classes are often called “parts.” An intuitive example of aggregation is the relationship between a wood and its threes. The wood can be considered as the *whole* and the threes would be the *parts* that belong to the wood. Aggregation has been briefly treated in the literature and different classifications of aggregations have been proposed. However, UML distinguishes only between two kinds of aggregation: simple aggregation and composed aggregation.

Aggregation

A simple aggregation is an aggregation where each part can be part of more than one *whole*. This kind of aggregation is very common in the literature and has been often refereed as *logical or catalog aggregation*, even in the first drafts of UML. As an example of simple aggregation we can think in the catalog of dolls of the “ToysX” store that contains n Barbie models. However, the same Barbie models can appear in the catalog of dolls of different toy-stores. This is possible because there exists a logical aggregation, and the dolls do not compound physically the catalogs. Figure 11.7 shows a simple aggregation. It is represented by placing a diamond next to the whole class.

Simple aggregation does not imply any kind of restriction over the life of the parts with regard to its whole.

Composition

A composition, also called composed aggregation, is a special kind of aggregation in which the *parts* are physically included in the *whole*. Once a part has

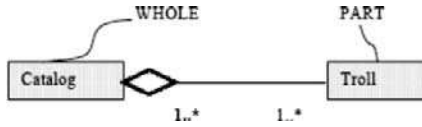


Fig. 11.7. Simple aggregation example



Fig. 11.8. Composition example

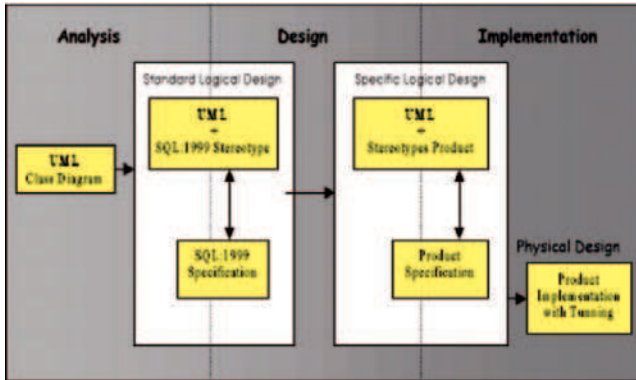


Fig. 11.9. Object-relational database design methodology

been created it lives and dies with its whole. A *part* can be explicitly removed before removing its associated *whole*. As it is a physical aggregation, a *part* can only belong to a *whole*. Figure 11.8 shows an example of composition. University is the *whole* and departments are its *parts*. The life of a department depends on the life of the university to which it belongs. If the university disappears its departments disappear as well. Besides, a department can be joined only to a university. The representation of the composition is similar to the representation of the simple aggregation. The only difference is that the diamond is fulfilled.

11.1.11 Object-Relational Database Design

The proposed methodology for object-relational database design is based on the proposal of Bertino and Marcos for object-oriented database design and on the proposal of Marcos and Cáceres. Figure 11.9 summarizes the main steps of the methodology.

The methodology proposes three phases, such as analysis, design, and implementation. Nonetheless, as it is shown in Fig. 11.9 differences between analysis, design, and implementation phases are not as strong as in structured design. At the *analysis* phase, we propose to use the UML class diagram

to design the conceptual schema instead of the Extended E/R Model (commonly used for relational databases), because UML is the standard language for object-oriented system design. Unlike E/R, UML has the advantage that it allows to design the entire system making easier the integration between different system views.

The *design* phase is divided into two steps:

- Standard design, that is, a logical design independent of any product.
- Specific design, that is, the design for a specific product (Oracle8i, Informix, etc.) Without considering tuning or optimization tasks.

Standard design is especially important in object-relational database design because each product implements a different object-relational model. This phase provides an object-relational specification independent of the product improving the database maintainability as well as making easier migration between products. As it is shown in Fig. 11.9 we propose two alternative techniques for this phase: defining the schema in SQL: 1999, because it does not depend on any specific product; and/or using a graphical notation describing a standard object-relational model (the SQL: 1999 model). This graphical notation corresponds with the relational graph that represents the logical design of a relational database. As graphical notation and due to UML can be extended, we propose to use UML extended with the required stereotypes for the SQL: 1999 object-relational model.

For the specific design (intermediate stage between design and implementation), we have to specify the schema in the SQL (language) of the chosen product. We use, as an example, Oracle8i. Besides, we also propose to use optionally a graphical technique to improve the documentation and the understandability of the generated SQL code. The graphical notation is also UML substituting the SQL: 1999 stereotypes with the specific stereotypes for the selected product. Finally, the *implementation* phase includes the physical design tasks. In this phase the schema obtained in the previous phase should be refined, making a tuning to improve the response time and storage space according to the specific needs of the application.

Relational database methodologies propose some rules to transform a conceptual schema into a standard logical schema. In the same way, we also propose a technique that allows transforming a schema from one phase to the next. This technique suggests some rules that have to be considered only as guidelines as illustrated in Table 11.5.

Class Transformation

Only persistent classes have to be transformed into a class of the database schema. A persistent class in UML is marked with the stereotype ⟨persistent⟩ (or, in Rational Rose notation with ⟨schema⟩). To transform a UML persistent class into a SQL: 1999 or Oracle8i class, it is necessary to define the object type as well as its extension. An object type in SQL: 1999 is defined as a

Table 11.5. Guidelines for object-relational database design

UML	SQL: 1999	Oracle8i
Class	Structured Type	Object Type
Class Extension	Typed Table	Table of Object Type
Attribute	Attribute	Attribute
Multivalued	ARRAY	VARRAY
Composed	ROW/Structured Type in column	Object Type in column
Calculated	Trigger/Method	Trigger/Method
Association		
One-To-One	REF/REF	REF/REF
One-To-Many	REF/ARRAY	REF/Nested Table
Many-To-Many	ARRAY/ARRAY	Nested Table/Nested Table
Aggregation	ARRAY	Nested Table
Generalization	Types/Typed Tables	Oracle cannot represent directly the generalization concept

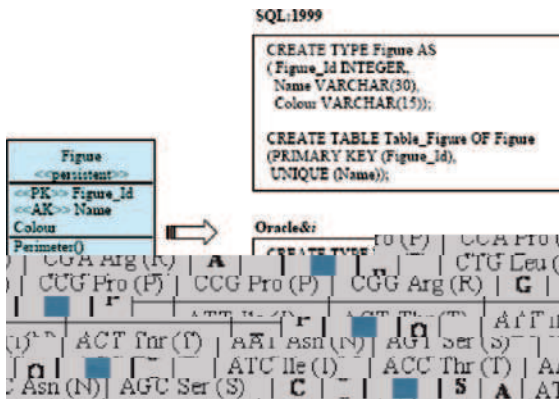


Fig. 11.10. Transformation of class

structured type, and its extension is defined as table of the aforementioned object type. A UML persistent class is translated into Oracle8i in the same way as into SQL: 1999. They only differ in the syntax of the structured type (in Oracle8i the structured type specifies “AS OBJECT”). Figure 11.10 shows an example of a UML persistent class and its corresponding specification in SQL: 1999 and Oracle8i.

Attribute and Method Transformation

Each attribute of a UML class is transformed into an attribute of the type. Nor SQL: 1999 neither Oracle8i support visibility levels, so in design and

implementation phases they disappear. Visibility levels should be implemented by defining views or privileges, etc.

Multivalued attributes are represented in SQL: 1999 and Oracle8i with a collection type. In SQL: 1999 the collection type is the ARRAY type, because it is the only collection type supported by the standard, whereas in Oracle8i it is possible to choose between the VARRAY and the nested table types. Using the VARRAY is recommended if the maximum number of elements is known; if the number of values is unknown, or very uncertain, it is recommended to use a nested table. We can notice that the possibility of defining multivalued attributes without additional tables eliminates one of the first rules in a relational database design: the first normal form is mandatory in every table.

Composed attributes can be represented in the object-relational model without creating an associated table, transforming it into a SQL: 1999 ROW type and into an Oracle8i object type without extension (that is, defining the object type and not specifying the associated table).

Derived attributes can be implemented by means of a trigger or by means of a method in both models, SQL: 1999 and Oracle8i. Each UML class method is transformed into SQL: 1999 and Oracle8i specifying the signature of the method in the definition of the object type. In this way the method is joined to the type to which it belongs. The body of the method is defined separately.

Association Transformation

UML associations can be represented in an object-relational schema either as unidirectional or as bidirectional relationships. A unidirectional association means that the association can be crossed only in one direction whereas a bidirectional association can be crossed in the two directions. If we know that queries require data in both directions of the association then it could be recommended to implement them as bidirectional relationships improving in this way the response times. However, we have to take into account that bidirectional relationships are not maintained by the system, so the consistence has to be guaranteed by means of triggers or methods. Therefore two-way relationships, despite of improving in some cases the response times, have a higher maintenance cost. The navigability (if it is represented) in a UML diagram shows the direction in which the association should be implemented.

Depending on the maximum multiplicity of the two classes involved in an association, we propose the following transformation rules (considering bidirectional associations):

- *One-to-One*. It would be implemented through a REF type attribute in each object type involved in the association. If the minimum multiplicity were one, it would be necessary to impose the NOT NULL restriction to the REF attribute in the corresponding typed table (because the restrictions have to be defined in the table rather than in the object type).

- *One-to-Many*. It would be transformed including a REF type attribute in the object type that participates in the association with multiplicity N and including an attribute of collection type in the object type that participates with multiplicity one. The collection types contain references (REF type) to the other object type involved in the relationship. In SQL: 1999 the collection type is an ARRAY (because it is the only collection type supported by the standard). However, in Oracle8i it is possible to use nested tables instead of VARRAYS because this constructor allows maintaining collections of elements without a predefined dimension. If the maximum cardinality were known (for example, suppose that a plain could not contain more than ten figures) then it would be more advisable to use a VARRAY.
- *Many-to-Many*. Following the same reasoning that in the previous case, a many-to-many association would be transformed into SQL: 1999 defining an ARRAY attribute in each object type involved in the relationship. In Oracle8i VARRAYs should be replaced by Nested Tables. If the association represents the navigability then it would be implemented as we have explained earlier, but just in one direction. Therefore the attribute of REF type (or the collection of REF type) would be defined only in a class.

Generalization Transformation

SQL: 1999 supports generalization of types and generalization of typed tables. The first one allows implementing the inheritance concept associated to a generalization; the second one allows implementing the subtype concept (every object that belongs to the subclass also belongs to the superclass) that is also associated to a generalization. The definition is made including the UNDER clause in the specification of each subtype indicating its supertype (only simple inheritance is allowed). It is also necessary to specify the corresponding hierarchy of tables by means of the UNDER clauses in the corresponding subtables. Oracle8i does not support inheritance. Therefore, generalizations are implemented using foreign keys, as in the relational model, or using REF types. Besides, it is necessary to specify restrictions (CHECK, assertions and triggers) or operations that permit to simulate their semantics.

There exist, however, some commercial products that implement inheritance such as Informix Universal Server. Although Informix does not support the entire semantics of the inheritance concept (for example, tables inherit only attributes), it allows to define it and supports some of its characteristics. It is expected that future versions of object-relational products will include inheritance. Meanwhile, when an object relational database is being designed it is important to specify the SQL: 1999 schema in order to maintain the semantics that is lost in the implementation phase due to lacks of the product models.

Aggregations and Composition Design

In this section we present the rules to transform UML simple aggregations and compositions into SQL: 1999 and Oracle8i, discussing the main differences between both of them.

Simple Aggregation Design

To represent this kind of aggregation in an object-relational model we propose to include in the definition of the *whole* type an attribute of collection type. This collection contains references to its *parts*, that is, references to the objects that compound the whole. For example, in Fig. 11.11, we can see an aggregation between a project and its plains. As we can see, it has been defined in SQL: 1999 and Oracle8i includes an attribute *Has plain* in the definition of the class *project*. This attribute is a collection of references to class *plain*. In SQL: 1999 the collection is defined by means of an ARRAY of references. In Oracle8i the collection should be a nested table. If the maximum number of components (maximum cardinality) is known (for example, suppose that in a plain there could not be more than ten figures) then it would be more advisable to use a VARRAY.

We propose to define the collection as a set of references, because a simple aggregation is an aggregation where each *part* can be part of more than one *whole*. It does not imply any kind of restriction over the life of the parts with regard to its whole.

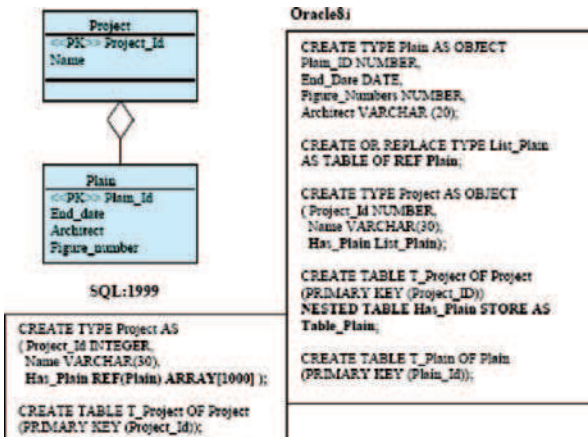


Fig. 11.11. Simple aggregation transformation

Composition

The composition is a special kind of aggregation in which the *parts* are physically linked to the *whole*. So, a composition defines three restrictions with regard the aggregation concept:

- Restriction 1.* A *part* cannot simultaneously belong to more than one *whole*.
- Restriction 2.* Once a *part* has been created it lives and dies with its *whole*.
- Restriction 3.* A *part* can be explicitly removed before to remove its associated *whole*.

Translating the UML concept of composition into an object-relational schema depends on the target model. Considering the translation to the SQL: 1999, as the standard object-relational model, there is not any difference with the aggregation implementation. To represent an aggregation or a composition in SQL: 1999 we have to introduce an attribute in the specification of the *whole*. As SQL: 1999 provides only the ARRAY collection type, this attribute has to be an ARRAY in both cases. The restrictions mentioned earlier have to be implemented by means of checks, assertions and/or triggers. This is just like some object-relational products, such as Informix that provides the set, list, and multiset collection types. However, in Oracle8i it is possible to implement directly the concept of composition maintaining the differences with regard to the aggregation concept. This is because Oracle8i besides supporting the VARRAY collection type, it also provides the nested table. A nested table is a collection type but it is also a table. Being a table, it can be defined as an object table. So, the nested table can contain the *parts* as objects rather than as references. At the same time the nested table is embedded in a column of another object table (the *whole*). Figure 11.12 shows the specification of a composition in Oracle8i.

When the composition is represented in the way defined earlier, the three composition restrictions defined previously are fulfilled. Therefore any check, assertion, or trigger has to be defined to implement the composition semantics. So, the nested table allows implementing the composition and the simple aggregation maintaining their semantics differences, in the same way as in UML.

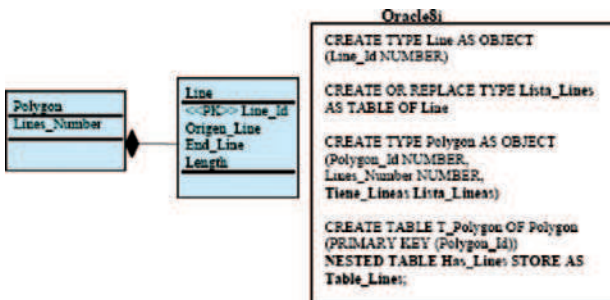


Fig. 11.12. Composition transformation

Advantages of ORDBMS

The advantages of ORDBMS are summarized below:

1. Resolves many of known weaknesses of RDBMS.
2. Reuse and sharing:
 - Reuse comes from ability to extend server to perform standard functionality centrally;
 - Gives rise to increased productivity both for developer and end-user.
3. Preserves significant body of knowledge and experience gone into developing relational applications.

Disadvantages of ORDBMS

The disadvantages of ORDBMS are summarized below:

- Complexity.
- Increased costs.
- Proponents of relational approach believe simplicity and purity of relational model are lost.
- Some believe RDBMS is being extended for what will be a minority of applications.
- OO purists not attracted by extensions either.
- SQL now extremely complex.

11.1.12 Comparison of OODBMS and ORDBMS

The comparison of OODBMS and ORDBMS with respect to data modeling, data access, and data sharing are shown in Tables 11.6, 11.7, and 11.8 respectively.

Data Modeling Comparison of ORDBMS and OODBMS

Table 11.6. Data modeling comparison of ORDBMS and OODBMS

Feature	ORDBMS	OODBMS
Object identity (OID)	Supported through REF type	Supported
Encapsulation	Supported through UDT's	Supported but broken for queries
Inheritance	Supported (separate hierarchies for UDT's and Tables)	Supported
Polymorphism	Supported (UDF invocation based on generic function)	Supported as in an object – oriented programming model language
Complex Objects	Supported through UDT's	Supported
Relationships	Strong support with user-defined referential integrity constraints	Supported (for example, using class libraries)

Data Access Comparison of ORDBMS and OODBMS**Table 11.7.** Data access comparison of ORDBMS and OODBMS

Feature	ORDBMS	OODBMS
Creating and accessing persistent data	Supported but not transparent	Supported but degree of transparency differs between products
Ad hoc query facility	Strong support	Supported through ODMG 3.0
Navigation	Supported by REF type	Strong Support
Integrity Constraints	Strong Support	No support
Object server/page server	Object server	Either
Schema evolutions	Limited support	Supported but degree of support differs between products

Data Sharing Comparison of ORDBMS and OODBMS**Table 11.8.** Data sharing comparison of ORDBMS and OODBMS

Feature	ORDBMS	OODBMS
ACID transactions	Strong support	Supported
Recovery	Strong support	Supported but degree of support differs between products
Advanced transaction models	No support	Supported but degree of support differs between products
Security, Integrity and Views	Strong Support	Limited support

Summary

The main objective of an OODBMS is to provide consistent, data independent, secure, controlled, and extensible data management services to support the object-oriented modeling paradigm. Today's OODBMS provide most of these capabilities. Many of these products are second generation OODBMS that have incorporated the lessons learned from the first generation products. Interpreting the database evolution diagram, we are about half way along the path to having features rich, powerful OODBMS in the market place. Given the high degree of interest in object-oriented technologies, there is a substantial market pull to put OODBMS products on a fast track where features and capabilities will continue to advance at a rapid rate.

A major strength of the OODBMS technology is its ability to represent complex behaviors directly. By incorporating behaviors into the database, one substantially reduces the complexity of applications that use the database. In the ideal scenario, most of the application code will deal with data entry and data display. All the functionality associated with data integrity and data management would be defined within the basic object model. The advantages of this approach are:

- All operations are defined once and reused by all applications.
- Changes to an operation affect all applications, simplifying database maintenance (although most databases require the applications to be recompiled).

The benefits of object-oriented database applications development are an increase in productivity resulting from the high degree of code reuse and an ability to cope with greater complexity resulting from incremental refinement of problems. One also gets increased design flexibility due to polymorphism and dynamic binding. Finally, both developers and users will experience benefits resulting from the naturalness and simplicity of representing data as objects.

These strengths need to be weighed against the organizational changes introduced by this new and different way of engineering solutions. Different engineering considerations contribute to performance and reliability than for relational DBMS's. Projects need to be managed differently. Clearly, one needs to approach this new technology with eyes open, recognizing that the benefits will be realized after a considerable investment has been made to learn how to use it effectively.

In this chapter we have summarized a methodology for object-relational database design focused on the aggregation and composition implementation. The methodology proposes three phases: analysis, design and implementation. As conceptual modeling technique we have chosen the UML class diagram. As logical model we have used the SQL: 1999 object-relational model, so that the guidelines are not dependent of the different implementations of each object-relational product. As a product example we have used Oracle8i. We have briefly explained the rules to transform a UML aggregation into an object-relational model considering the differences between aggregation and composition.

We have focused on the implementation in Oracle8i because this product supports a collection data type, the nested table that is specially appropriated to implement aggregations and compositions. This collection data type is not provided neither by SQL: 1999 nor by other products. In the methodology we have proposed two alternative techniques for the standard design phase: defining the schema in SQL: 1999, because it does not depend on a specific product; and/or using a graphical notation describing a standard object-relational model (the SQL: 1999 model).

This graphical notation corresponds with the relational “graph” that represents the logical design of a relational database. Although there are some proposals of UML stereotypes for database design, they are focusing on the relational model. The next step will be completing the methodology taking into account the UML use cases diagrams to design the behavior of the classes.

Review Questions

11.1. State the benefits of OOP.

There are several benefits of adopting OOP. The following three benefits, although subjective, are considered by many to be major reasons for adopting OOP:

- Programs reflect reality.
- The model is more stable than functionality.
- Subclassing and virtuals improve the reusability of code.

11.2. List some OOPLs. Compare different OOP languages.

The following is a list of some popular OOPLs:

- C++ Language System
- C.Talk
- Smalltalk
- Smalltalk-80
- Actor
- Enfin
- Prokappa
- Eiffel
- KnowledgePro
- Classic-Ada with Persistence
- Objective-C
- Trellis/Owl
- Flavors
- CLOS
- Common Loops

Most OOPLs can trace their origins to Simula. The concepts of Objects and Classes are employed by most of these languages.

11.3. Explain different modeling relationships in C++?

Interactions between objects can be captured during OOD by appropriate relationships. At the implementation level, C++ provides the following mechanisms for implementing object relationships:

1. Global Objects
2. Function arguments
3. Constructors
4. Base classes
5. Templates

11.4. Compare and contrast the different definitions of Object-oriented data models.

A data model consists of:

- Static properties such as objects, attributes, and relationships.
- Integrity rules over objects and operations.
- Dynamic properties such as operations or rules defining new database states based on applied state changes.

Object-oriented databases have the ability to model all three of these components directly within the database supporting a complete problem/solution modeling capability. Prior to object-oriented databases, databases were capable of directly supporting points 1 and 2 above and relied on applications for defining the dynamic properties of the model. The disadvantage of delegating the dynamic properties to applications is that these dynamic properties could not be applied uniformly in all database usage scenarios since they were defined outside of the database in autonomous applications. Object-oriented databases provide a unifying paradigm that allows one to integrate all three aspects of data modeling and to apply them uniformly to all users of the database.

11.5. How did the need arise for Object-oriented databases?

The increased emphasis on process integration is a driving force for the adoption of object-oriented database systems. For example, the Computer Integrated Manufacturing (CIM) area is focusing heavily on using object-oriented database technology as the process integration framework. Advanced office automation systems use object-oriented database systems to handle hypermedia data. Hospital patient care tracking systems use object-oriented database technologies for ease of use. All of these applications are characterized by having to manage complex, highly interrelated information, which is strength of object-oriented database systems.

Clearly, relational database technology has failed to handle the needs of complex information systems. The problem with relational database systems is that they require the application developer to force an information model into tables where relationships between entities are defined by values. For the most part, object database design is a fundamental part of the overall application design process. The object classes used by the programming language are the classes used by the ODBMS. Because their models are consistent, there is no

need to transform the program's object model to something unique for the database manager.

An initial area of focus by several object-oriented database vendors has been the Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), and Computer Aided Software Engineering (CASE) applications. A primary characteristic of these applications is the need to manage very complex information efficiently. Other areas where object-oriented database technology can be applied include factory and office automation. For example, the manufacture of an aircraft requires the tracking of millions of interdependent parts that may be assembled in different configurations. Object-oriented database systems hold the promise of putting solutions to these complex problems within reach of users.

11.6. Explain about the evaluation of OODBMS.

Object-oriented database research and practice dates back to the late 1970s and had become a significant research area by the early 1980s, with initial commercial product offerings appearing in the late 1980s. Today, there are many companies marketing commercial object-oriented databases that are second generation products. OODBMS's have established themselves in niches such as e-commerce, engineering product data management, and special purpose databases in areas such as securities and medicine. The strength of the object model is in applications where there is an underlying need to manage complex relationships among data objects. Today, it is unlikely that OODBMS' are a threat to the stranglehold that relational database vendors have in the market place. Clearly, there is a partitioning of the *market into* databases that are best suited for handling high volume low, complexity data and databases that are suited for high complexity, reasonable volume, with OODBMS filling the need for the latter.

Object-oriented databases are following a maturation path similar to relational databases. Figure 11.13 depicts the evolution of object-oriented database technologies. On the left, we have object-oriented languages that have been extended to provide simple persistence allowing application objects to persist between user sessions. Minimal database functionality is provided in terms of concurrency control, transactions, recovery, etc.

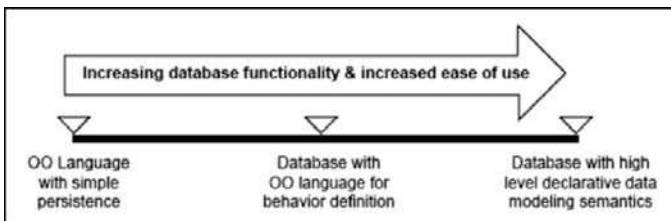


Fig. 11.13. The evolution of object-oriented databases

The next stage of evolution is more difficult. As one moves to the right the database does more for the user requiring less effort to develop applications. An example of this is that current OODBMS provide a large number of low-level interfaces for the purpose of optimizing database access. As the OODBMS database technology evolves, OODBMS will assume a greater part of the burden for optimization allowing the user to specify high-level declarative guidance on what kinds of optimizations need to be performed. A general guideline for gauging database maturity is the degree to which functions such as database access optimization, integrity rules, schema and database migration, archive, backup and recovery operations can be tailored by the user using high-level declarative commands to the OODBMS.

11.7. With a neat sketch emphasize the characteristics of Object-oriented databases.

Object-oriented database technology is a combination of object-oriented programming and database technologies. Figure 11.14 illustrates how these programming and database concepts have come together to provide what we now call object-oriented databases.

Perhaps the most significant characteristic of object-oriented database technology is that it combines object-oriented programming with database technology to provide an integrated application development system. There are many advantages to including the definition of operations with the definition of data. First, the defined operations apply ubiquitously and are not

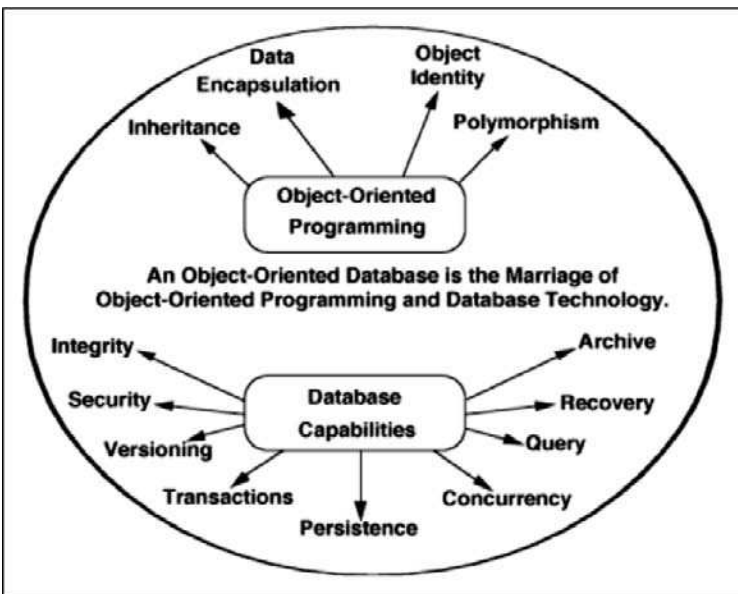


Fig. 11.14. Makeup of an object-oriented database

dependent on the particular database application running at the moment. Second, the data types can be extended to support complex data such as multimedia by defining new object classes that have operations to support the new kinds of information. Other strengths of object-oriented modeling are well known. For example, inheritance allows one to develop solutions to complex problems incrementally by defining new objects in terms of previously defined objects.

Polymorphism and dynamic binding allows one to define operations for one object and then to share the specification of the operation with other objects. These objects can further extend this operation to provide behaviors that are unique to those objects. Dynamic binding determines at runtime, which of these operations is actually executed, depending on the class of the object requested to perform the operation. Polymorphism and dynamic binding are powerful object-oriented features that allow one to compose objects to provide solutions without having to write code that is specific to each object. All of these capabilities come together synergistically to provide significant productivity advantages to database application developers.

11.8. Describe how relationships can be modeled in an OODBMS.

Using Relationship Between Objects

Objects interacting in a system make use of the services offered by other objects. The using relationship can be used to express a subset of such interactions. *Booch* and *Vilot* have identified three roles that each object may play in using relationships:

Actor objects can operate upon other objects, but are never operated upon by other objects. Such objects make use of services offered by other objects but do not themselves provide any service to the objects they make use of.

Server objects never operate upon objects, but are operated upon by other objects.

Agent objects can both operate upon other objects and be operated upon by other objects.

Relationships among Classes

Rumbaugh has identified three types of class relationships:

1. Generalization or “kind-of”
2. Aggregation or “part-of”
3. Association, implying some semantic connection
4. Booch and Vilot have identified two more types of relationships between classes
5. Instantiation relationships
6. Metaclass relationships

Booch and Vilot suggests the following rule of thumb for identifying relationships: “If an abstraction is more than the sum of its component parts, then using relationships are more appropriate. If an abstraction is a kind of some other abstraction or if that abstraction is exactly equal to the sum of its components, then inheritance is a better approach.”

11.9. What functionality would typically be provided by an ORDBMS?

Due to recent hardware improvements more sophisticated applications have emerged, such as CAD/CAM (Computer-Aided Design/Computer-Aided Manufacturing), CASE (Computer-Aided Software Engineering), GIS (Geographic Information System), etc. These applications can be characterized as consisting of complex objects related by complex relationships. Representing such objects and relationships in the relational model implies that the objects must be decomposed into a large number of tuples.

Thus, a considerable number of joins is necessary to retrieve an object and, when tables are too deeply nested, performance is significantly reduced. A new generation of databases has appeared to solve these problems: the object-oriented database generation, which includes the object-relational and object databases. This new technology is well suited for storing and retrieving complex data because it supports complex data types and relationships, multimedia data, inheritance, etc. Nonetheless, good technology is not enough to support complex objects and applications. It is necessary to define methodologies that guide designers in the object database design task, in the same way traditionally has been done with relational databases.

The object model of the current standard for object-relational databases, SQL: 1999, as well as the main characteristics of Oracle8i object-relational model, as an example of object-relational product. SQL: 1999 data model extends the relational data model with some new constructors to support objects. Most of last versions of relational products include some object extensions. However, and because in general these products have appeared in the market before the standard approval, current versions of object-relational products do not totally. Adjust to the SQL: 1999 model.

11.10. Discuss about the various criteria used to evaluate OODBMS

These criteria are broken into three main areas:

- Functionality
- Application Development Issues
- Miscellaneous Criteria

Functionality, defines evaluation criteria based on functional capabilities provided by the OODBMS. Subsections include Basic Object-Oriented Modeling, Advanced Object-Oriented Database Topics, Database Architecture, Database Functionality, Application Programming Interface, and Querying an OODBMS.

Application Development Issues considers issues regarding the development of applications on top of an OODBMS.

Miscellaneous Criteria, identifies a few nonfunctional and nondevelopmental evaluation issues. These issues deal with vendor and product maturity, vendor support, and current users of the OODBMS product.

11.11. Differentiate OODBMS and ORDBMS with a neat tabular column.

1. Data Modeling Comparison of ORDBMS and OODBMS (Table 11.9)
2. Data Access Comparison of ORDBMS and OODBMS (Table 11.10)
3. Data Sharing Comparison of ORDBMS and OODBMS (Table 11.11)

Table 11.9. Data modeling comparison of ORDBMS and OODBMS

Feature	ORDBMS	OODBMS
Object identity (OID)	Supported through REF type	Supported
Encapsulation	Supported through UDT's	Supported but broken for queries
Inheritance	Supported (separate hierarchies for UDT's and Tables)	Supported
Polymorphism	Supported (UDF invocation based on generic function)	Supported as in an object – oriented programming model language
Complex Objects	Supported through UDT's	Supported
Relationships	Strong support with user-defined referential integrity constraints	Supported (for example, using class libraries)

Table 11.10. Data Access Comparison of ORDBMS and OODBMS

Feature	ORDBMS	OODBMS
Creating and accessing persistent data	Supported but not transparent	Supported but degree of transparency differs between products
Ad hoc query facility	Strong support	Supported through ODMG 3.0
Navigation	Supported by REF type	Strong Support
Integrity Constraints	Strong Support	No support
Object server/page server	Object server	Either
Schema evolutions	Limited support	Supported but degree of support differs between products

Table 11.11. Data Sharing Comparison of ORDBMS and OODBMS

Feature	ORDBMS	OODBMS
ACID transactions	Strong support	Supported
Recovery	Strong support	Supported but degree of support differs between products
Advanced transaction models	No support	Supported but degree of support differs between products
Security, Integrity, and Views	Strong Support	Limited support

Table 11.12. Miscellaneous evaluation criteria

Criteria
Product Maturity
Product Documentation
Vendor Maturity
Vendor Training
Vendor Support and Consultation
Vendor Participation in Standards Activities

11.12. List out and briefly explain the nontechnical criteria under miscellaneous evolution category.

Miscellaneous Criteria

A number of nontechnical criteria should also be considered when evaluating an OODBMS. This section details some of these criteria, as listed in Table 11.12, Miscellaneous Evaluation Criteria.

Product Maturity

Product maturity may be measured by several criteria including:

- Years under development
- Number of seats licensed
- Number of licensed seats actually in use
- Number of licensed seats in use for purposes other than evaluations (i.e., actual development efforts)
- Number and type of applications being built with the OODBMS product
- Number and type of shipped applications built with the OODBMS product

Product Documentation

Product documentation should be clear, consistent, and complete. The documentation should include complete examples of typical programmed capabilities (e.g., what is the sequence of calls to access data from the database and to cause updates to that data to be made permanent in the database).

Vendor Maturity

Vendor maturity may be measured by several criteria including:

- Company's size and age.
- Previous experience of the company's lead technical and management personnel in the commercial database market.
- Financial stability.

Vendor Training

Availability and quality of vendor supplied training classes is an important consideration when selecting an OODBMS.

Vendor Support and Consultation

It is expected that significant support will be required during the OODBMS evaluation process and to overcome the initial learning curve. OODBMS vendors should provide a willing and capable support staff. Support should be available via phone and electronically. Consulting support might also be appealing where the OODBMS vendor provides expert, hands-on assistance in product use, object-oriented application design (especially in regards to database issues), and in maximizing database application performance.

Vendor Participation in Standards Activities

The vendor should be active in standards efforts in the object-oriented, language, CASE, open software, and data exchange areas. In particular:

- *Object Management Group (OMG)*. An organization funded by over 80 international information systems corporations whose charter is to develop standards for interoperability and portability of software. The OMG is focusing on object-oriented integration technologies such as Object Request Broker (ORB), OODBMS interfaces, and object interfaces for existing applications.
- *Object Database Management Group (ODMG)*. An organization of OODBMS vendors chartered to define a standard interface to OODBMS

that will allow application portability and interoperability. Standards defined by the ODMG will be provided to OMG, ANSI, STEP, PCTE, etc. to aid in their respective standardization efforts.

- ANSI standardization efforts in languages (C, C++, Smalltalk), SQL, and object-oriented databases.
- Standards such as Portable Common Tool Environment (PCTE) and CASE Data Interchange
- Format (CDIF) providing for common data representations, data exchange formats and interoperation of tools.
- *PDES/STEP*. An effort aimed at standardizing an exchange format for product model data (product model data, such as CAD data, represents a prime application area for OODBMS).

11.13. What are the advantages and disadvantages of extending the relational data model?

Advantages of ORDBMS

1. Resolves many of known weaknesses of RDBMS.
2. Reuse and sharing:
 - Reuse comes from ability to extend server to perform standard functionality centrally.
 - Gives rise to increased productivity both for developer and end-user.
3. Preserves significant body of knowledge and experience gone into developing relational applications.

Disadvantages of ORDBMS

- Complexity.
- Increased costs.
- Proponents of relational approach believe simplicity and purity of relational model are lost.
- Some believe RDBMS is being extended for what will be a minority of applications.
- OO purists not attracted by extensions either.
- SQL now extremely complex.

11.14. Analyze the concept OODBMS in fundamental four areas

OODBMS can be analyzed in the following four areas:

- Functionality
- Usability
- Platform
- Performance

An analysis of functional capabilities is performed to determine if a given OODBMS provides sufficient capabilities to meet the current and future needs of a given development effort. Functional capabilities include basic database functionality such as concurrency control and recovery as well as object-oriented database features such as inheritance and versioning. Each evaluation will have to identify and weight a set of functional requirements to be met by the candidate OODBMS. Weighting is an important consideration since application workarounds may be possible for missing functionality.

Usability deals with the application development and maintenance process. Issues include development tools and the ease with which database applications can be developed and maintained. How a developer perceives the database and the management of persistent objects might also be considered under the category of usability. Other issues to be considered are database administration, product maturity, and vendor support. Evaluation of usability is likely to be highly subjective. Perhaps the most easily measurable evaluation criterion is platform. An OODBMS is either available or not on the application's target hardware and operating system. Heterogeneous target environments require that the OODBMS transparently interoperates within that environment.

11.15. Explain concept of Object Versioning

Object versioning is the concept that a single object may be represented by multiple versions (i.e., instances of that object) at one time. We can define two forms of versioning, each driven by particular requirements of the applications which are driving the need for OODBMS products:

- *Linear Versioning* is the concept of saving prior versions of objects as an object changes. In design-type applications (e.g., CASE, CAD) prior versions of objects are essential to maintain the historical progression of a design and to allow designers to return to earlier design points after investigating and possibly discarding a given design path. Under linear versioning, only a single new version can be created from each existing version of an object.
- *Branch Versioning* supports concurrency policies where multiple users may update the same data concurrently. Each user's work is based upon a consistent, nonchanging base version. Each user can modify his version of an object (as he proceeds along some design path in a CAD application for example). At some future point in time, under user/application support, the multiple branch versions are merged to form a single version of the object. Branch versioning is important in applications with long transactions so that users are not prevented access to information for long periods of time. Under branch versioning, multiple new versions may be created for an object.

Associated with the idea of versioning is that of configuration. A configuration is a set of object versions that are consistent with each other. In other

words, it is a group of objects whose versions all belong together. OODBMS need to provide support so that applications access object versions that belong to the same conceptual configuration. This may be achieved by controlling the relationships that are formed for versioned objects (i.e., they may be duplicated in the new object or replaced with relationships to other objects).

An OODBMS may provide low level facilities which application developers use to control the versioning of objects. Alternatively, the OODBMS may implement a specific versioning policy such as automatically creating a new object version with each change. An automatic policy may result in rapid and unacceptable expansion of the database and requires some automated means of controlling this growth.

11.16. Explain about the concept Basic Object-Oriented modeling.

Basic Object-Oriented Modeling

The evaluation criteria in this section distinguish database as an object-oriented database. Topics in this section cover the basic object-oriented (OO) capabilities typically supported in any OO technology (e.g., programming language, design method). These basic capabilities are expected to be supported in all commercial OODBMS. The topics are given a cursory overview here for readers new to OO technology.

Complex Objects

OO systems and applications are unique that the information being maintained is organized in terms of the real-world entities being modeled. This differs from relational database applications that require a translation from the real-world information structure to the table formats used to store data in a relational database. Normalizations upon the relational database tables result in further perturbation of the data from the user's perceptual viewpoint. OO systems provide the concept of complex objects to enable modeling of real-world entities. A complex object contains an arbitrary number of fields, each storing atomic data values or references to other objects (of arbitrary types). A complex object exactly models the user perception of some real-world entity.

Object Identity

OO databases (and programming languages) provide the concept of an object identifier (OID) as a means of uniquely identifying a particular object. OIDs are system generated. A database application does not have direct access to the OID. The OID of an object never changes, even across application executions. The OID is not based on the value stored within the object. This differs from relational databases, which use the concept of primary keys to identify a particular table row (i.e., tuple). Primary keys are based upon

data stored in the identified row. The concept of OIDs makes it easier to control the storage of objects (e.g., not based on value) and to build links between objects (e.g., they are based on the never changing OID). Complex objects often include references to other objects, directly or indirectly stored as OIDs.

Classes

OO modeling is based on the concept of a class. A class defines the data values stored by, and the functionality associated with, an object of that class. One of the primary advantages of OO data modeling is this tight integration of data and behavior through the class mechanism. Each object belongs to one, and only one, class. An object is often referred to as an instance of a class. A class specification provides the external view of the instances of that class. A class has an extent (sometimes called an extension), which is the set of all instances of the class. Implementation of the extent may be transparent to an application, but minimally provides the ability to visit every instance of the class. Within an OODBMS, the class construct is normally used to define the database schema. Some OODBMS use the term *type* instead of *class*. The OODBMS schema defines what objects may be stored within the database.

Attributes

Attributes represent data components that make up the content of a class. Attributes are called data members in the C++ programming language. Instance attributes are data components that are stored by each instance of the class. Class attributes (static data members in C++) are data values stored once for all instances of the class. Attributes may or may not be visible to external users of the class. Attribute types are typically a subset of the basic data types supported by the programming language that interfaces to the OODBMS. Typically this includes enumeration types such as characters and booleans, numeric types such as integers and floats, and fixed length arrays of these types such as strings. The OODBMS may allow variable length arrays, structures (i.e., records), and classes as attribute types.

Pointers are normally not good candidates for attribute types since pointer values are not valid across application executions.

An OODBMS will provide attribute types that support interobject references. OO applications are characterized by a network of interconnected objects. Object interconnections are supported by attributes that reference other objects. Other types that might be supported by an OODBMS include text, graphic, and audio. Often these data types are referred to as binary large objects (BLOBS). Derived attributes are attributes that are not explicitly stored but instead calculated on demand. Derived attributes require that attribute access be indistinguishable from behavior invocation.

Behaviors

Behaviors represent the functional component of a class. A behavior describes how an object operates upon its attributes and how it interacts with other related objects. Behaviors are called member functions in the C++ programming language. Behaviors hide their implementation details from users of a class.

Encapsulation

Classes are said to encapsulate the attributes and behaviors of their instances. Behavior encapsulation shields the clients of a class (i.e., applications or other classes) from seeing the internal implementation of a behavior. This shielding provides a degree of data independence so that clients need not be modified when behavior implementations are modified (they will have to be modified if behavior interfaces change).

A class's attributes may or may not be encapsulated. Attributes that are directly accessible to clients of a class are not encapsulated (public data members in C++ classes). Modifying the definition of a class's attributes that are not encapsulated requires modification of all clients that access them. Attributes that are not accessible to the clients of a class are encapsulated (private or protected data members in C++ classes). Encapsulated attributes typically have behaviors that provide clients some form of access to the attribute. Modifications to these attributes typically do not require modification to clients of the class.

Inheritance

Inheritance allows one class to incorporate the attributes and behaviors of one or more other classes. A subclass is said to inherit from one or more superclasses. The subclass is a specialization of the superclass in that it adds additional data or behaviors, or overrides behaviors of the superclass. Superclasses are generalizations of their subclasses. Inheritance is recursive. A class inherits the attributes and behaviors from its superclasses, and from its superclass's superclasses, etc. In a single inheritance model, a class may directly inherit from only a single other class. In a multiple inheritance model a class may directly inherit from more than one other class. Systems supporting multiple inheritance must specify how inheritance conflicts are handled. Inheritance conflicts are attributes or behaviors with the same name in a class and its superclass, or in two superclasses.

Inheritance is a powerful OO modeling concept that supports reuse and extensibility of existing classes. The inheritance relationships between a group of classes define a class hierarchy. Class hierarchies improve the ability of users to understand software systems by allowing knowledge of one class (a superclass) to be applicable to other classes (its subclasses).

Overriding Behaviors and Late Binding

OO applications are typically structured to perform work on generic classes (e.g., a vehicle) and at runtime invoke behaviors appropriate for the specific vehicle being executed upon (e.g., Boeing 747). Applications constructed in such a manner are more easily maintained and extended since additional vehicle classes may be added without requiring modification of application code. Overriding behaviors is the ability for each class to define the functionality unique to itself for a given behavior. Late binding is the ability for behavior invocation to be selected at runtime based on the class of an object (instead of at compile time).

Persistence

Persistence is the characteristic that makes data available across executions. The objective of an OODBMS is to make objects persistent. Persistence may be based on an object's class, meaning that all objects of a given class are persistent. Each object of a persistent class is automatically made persistent. An alternative model is that persistence is a unique characteristic of each object (i.e., it is orthogonal to class). Under this model, an object's persistence is normally specified when it is created. A third persistence model is that any object reachable from a persistent object is also persistent. Such systems require some way of explicitly stating that a given object is persistent (as a means of starting the network of interconnected persistent objects). Related to the concept of persistence is object existence. OODBMS may provide a means by which objects are explicitly deleted. Such systems must ensure that references to deleted objects are also removed. An alternative strategy is to maintain an object as long as references to the object exist. Once all references are removed, the object can be safely deleted.

Naming

OO applications are characterized as being composed of a network of interconnected objects. An application begins by accessing a few known objects and then traverses to additional objects via relationships from the known objects. As objects are created they are linked (i.e., related) to other existing objects. Given this scenario, the database must provide some mechanism for identifying one or more objects at application start-up without using relations from existing objects. This is typically accomplished by allowing objects to be named and providing a retrieval mechanism based upon name. An application begins by loading one or two "high-level" objects that it knows by name and then traverses to other reachable objects. Object names apply within some name scope. Within a given scope, names must be unique (i.e., the same name can not refer to two objects). The simplest scope model is for the entire database to act as a single name scope. An alternative scope model is for the

application to identify name scopes. Using multiple name scopes will reduce the chance for name conflicts.

11.17. Describe the features of Schema Evolution.

Schema Evolution

Schema evolution is the process by which existing database objects are brought into line with changes to the class definitions of those objects (i.e., schema changes require all instances of the changed class to be modified so as to reflect the modified class structure). Schema evolution is helpful although not essential during system development (as a means of retaining test data, for example). Schema evolution is essential for maintenance and/or upgrades of fielded applications. Once an application is in the field (and users are creating large quantities of information), an upgrade or bug fix cannot require disposal of all existing user databases. Schema evolution is also essential for applications that support user-level modeling and/or extension of the application.

Here we have given a framework for schema modifications in an object-oriented database. Included in this framework are invariants which must be maintained at all times (e.g., all attributes of a class must have a distinct name), rules for performing schema modifications (e.g., changing the type of an attribute in a given class must change the type of that attribute in all classes which inherit that attribute), and a set of schema changes that should be supported by an object-oriented database. This set of schema change operations is:

1. Changes to Definition of a Class:
 - (a) Changes to an Attribute of a Class (applies to both instance and class attributes):
 - Add an attribute to a class.
 - Remove an attribute from a class.
 - Change the name of an attribute.
 - Change the type of an attribute.
 - Change the default value of an attribute.
 - Alter the relationship properties for relationship attributes.
 - (b) Changes to a Behavior of a Class:
 - Add a new behavior to the class.
 - Remove a behavior from the class.
 - Change the name of a behavior.
 - Change the implementation of a behavior.
2. Changes to the Inheritance of a Class:
 - Add a new superclass to a class.
 - Remove a superclass for a given class.
 - Change the order of superclasses for a class (it is expected that superclass ordering will be used to handle attribute and behavior inheritance conflicts).

3. Changes to the Existence of a Class:
 - Add a new class.
 - Remove an existing class.
 - Change the name of a class.

Schema changes will require modification of instances of the changed class as well as applications that referenced the changed class. Some of these changes cannot be performed automatically by the OODBMS. Deleting attributes and superclasses are examples of schema changes that could be performed automatically. Adding attributes and superclasses can only be performed if default values are acceptable for the initial state of new attributes. This is not likely, especially for relationship attributes. An OODBMS should provide tools and/or support routines for aiding programmed schema evolution.

A manual evolution approach requires instance migration to be performed off-line, probably through a dump of the database and a reload of the data through an appropriate transformation filter. Systems may perform an aggressive update by automatically adjusting each instance after each schema change. This approach may be slow due to the overhead of performing the update on all instances at a single time. This approach is the easiest for an application to implement since multiple versions of the schema need not be maintained indefinitely.

Schema changes may be performed in background mode, thus spreading the update overhead over a longer period of time. A lazy evaluation approach defers updating objects until they are accessed and found to be in an inconsistent state. Both the background and lazy approaches require extended periods where multiple versions of the schema exist and will be complicated by multiple schema modifications. Applications and stored queries will have to be updated manually as a result of schema changes. Some forms of schema changes will not require updates to applications and queries due to data independence and encapsulation of a class's data members.

It is expected that all OODBMS products will support some form of schema evolution for static schema changes. By static, we mean the schema is changed by manipulations of class definitions outside of application processing (i.e., by reprocessing database schema definitions and modifying application programs). Dynamic schema modification, meaning modification of the schema by the application, is more complex and potentially inconsistent with the basic C++ header file approach used for schema definitions in many current commercial products. Dynamic schema modification is only needed in applications that require user definable types.

11.18. What are the architecture issues relevant to the OODBMS

Database Architecture

1. *Distributed Client-Server Approach*

Advances in local area network and workstation technology have given rise to group design type applications driving the need for OODBMS

(e.g., CASE, CAD, and Electronic Offices). OODBMS typically execute in a multiple process distributed environment. Server processes provide back-end database services, such as management of secondary storage and transaction control. Client processes handle application specific activities, such as access and update of individual objects. These processes may be located on the same or different workstations. Typically a single server will be interacting with multiple clients servicing concurrent requests for data managed by that server. A client may interact with multiple servers to access data distributed throughout the network.

The evaluations and benchmarks are the three alternative workstation-server architectures that have been proposed for use with OODBMS:

- *Object server approach.* The unit of transfer from server to client is an object. Both machines cache objects and are capable of executing methods on objects. Object-level locking is easily performed. The major drawback of this approach is the overhead associated with the server interaction required to access every object and the added complexity of the server software which must provide complete OODBMS functionality (e.g., be able to execute methods). Keeping client and server caches consistent may introduce additional overheads.
- *Page server approach.* The unit of transfer from server to client is a page (of objects). Page-level transfers reduce the overhead of object access since server interaction is not always required. Architecture and implementation of the server is simplified since it needs only to perform the backend database services. A possible drawback of this approach is that methods can be evaluated only on the client, thus all objects accessed by an application must be transferred to the client. Object-level locking will be difficult to implement.
- *File server approach.* The OODBMS client processes interact with a network file service (e.g., Sun's NFS) to read and write database pages. A separate OODBMS server process is used for concurrency control and recovery. This approach further simplifies the server implementation since it need not manage secondary storage. The major drawback of this approach is that two network interactions are required for data access, one to the file service and one to the OODBMS server.

Many scientists have identified no clear winner when benchmarking the three approaches. The page server approach seemed best with large buffer pools and good clustering algorithms. The object server approach performed poorly if applications scanned lots of data, but was better than the page server approach for applications performing lots of updates and running on workstations with small buffer pools.

2. *Data Access Mechanism*

An evaluation of OODBMS products should consider the process necessary to move data from secondary storage into a client application. Typically this requires communication with a server process, possibly across a network. Objects loaded into a client's memory may require further

processing, often referred to as swizzling, to resolve references to other objects which may or may not already be loaded into the client's cache. The overhead and process by which locks are released and updated objects are returned to the server should also be considered.

3. *Object Clustering*

OODBMS which transfer units larger than an object do so under the assumption that an application's access to a given object implies a high probability that other associated objects may also be accessed. By transferring groups of objects, additional server interaction may not be necessary to satisfy these additional object accesses. Object clustering is the ability for an application to provide information to the OODBMS so that objects which it will typically access together can be stored near each other and thus benefit from bulk data transfers.

4. *Heterogeneous Operation*

An OODBMS provides a mechanism for applications to cooperate, by sharing access to a common set of objects. A typical OODBMS will support multiple concurrent applications executing on multiple processors connected via a local area network. Often, the processors will be from different computer manufacturers; each having its own data representation formats. For applications to cooperate in such an environment, data must be translated to the representation format suitable for the processor upon which that data is stored (both permanently by a server and temporarily by a client wishing to access the data). To be an effective integration mechanism, an OODBMS must support data access in a heterogeneous processing environment.

Distributed and Parallel Database Management Systems

Learning Objectives. This chapter is dedicated to distributed and parallel database management system. The distributed database design, architecture, concurrency control, and reliability concepts are discussed in this chapter. This chapter also deals with parallel database architecture; components and benefits of parallel processing. After completing this chapter the reader should be familiar with the following concepts:

1. Distributed database management system
2. Distributed DBMS architecture
3. Distributed database design
4. Semantic data control
5. Distributed concurrency control
6. Distributed DBMS reliability
7. Parallel database management system

12.1 Distributed Database

Distributed database provides a number of advantages of distributed computing to the DBMS domain. A distributed computing system consists of a number of processing elements that are interconnected by a computer network, and that cooperate in performing certain application tasks. The distributed database is a collection of multiple logically interrelated databases distributed over a computer network. Parallel processing divides a complex task into many smaller tasks, and executes the smaller tasks simultaneously in several tasks. Thus the complex task is completed with better performance and also quickly. The parallel database system makes use of the parallelism in DBMS and achieves high performance and high availability database servers at much lower price.

A distributed database is a collection of data which belong logically to the same system but are spread over the sites of a computer network. This definition emphasizes two equally important aspects of a distributed database as follows:

1. *Distribution.* The fact that the data are not resident at the same site (processor), so that we can distinguish a distributed database from a single, centralized database.
2. *Logical correlation.* The fact that the data have some properties which tie them together, so that we can distinguish a distributed database from a set of local databases or files which are resident at different sites of a computer network.

A distributed database is a collection of data which are distributed over different computers of a computer network. Each site of the network has autonomous processing capability and can perform local applications. Each site also participates in the execution of at least one global application, which requires accessing data at several sites using a communication subsystem. A simple distributed database on a local network is shown in Fig. 12.1.

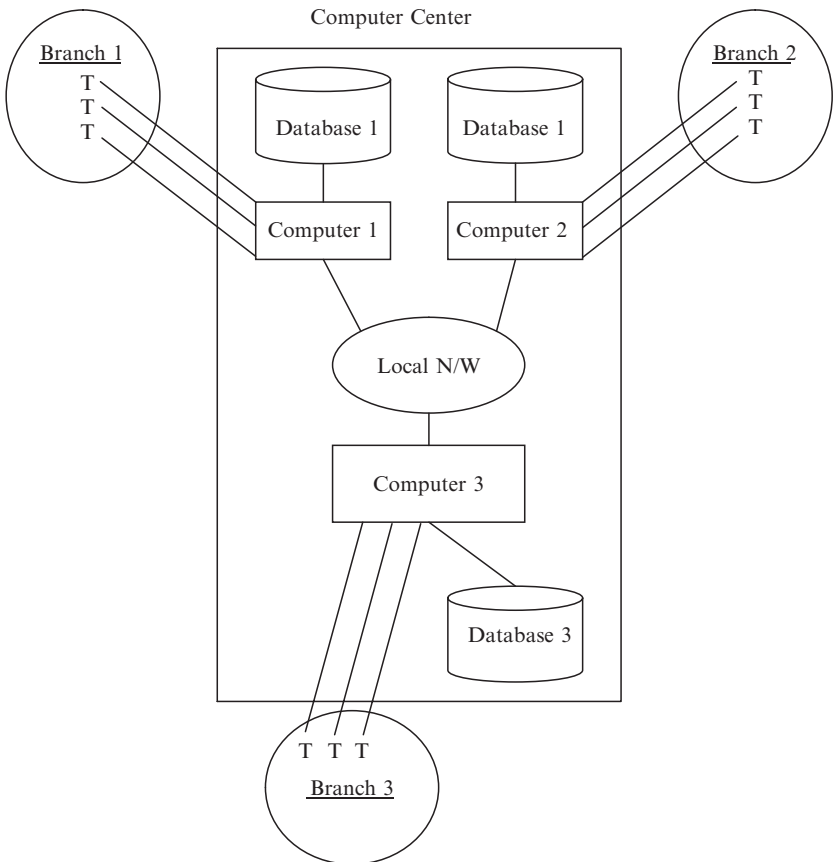


Fig. 12.1. A distributed database on a local network

12.1.1 Features of Distributed vs. Centralized Databases

The features which characterize the traditional database approach are centralized control, data independence, and reduction of redundancy, complex physical structures for efficient access, integrity, recovery, concurrency control, privacy, and security.

- *Centralized control.* The possibility of providing centralized control of the information resources of a whole enterprise or organization was considered as one of the strongest motivations for introducing databases; they were developed as the evolution of information systems in which each application had its own private files.

In general, in distributed databases it is possible to identify a hierarchical control structure based on a global database administrator, who has the central responsibility of the whole database, and on local database administrators, who have the responsibility of their respective local databases.

- *Data independence.* It means that the actual organization of data is transparent to the application programmer. Programs are written having a conceptual view of the data, the so-called *conceptual schema*. The main advantage of data independence is that programs are unaffected by changes in the physical organization of data.
- *Reduction of redundancy.* In traditional databases, redundancy was reduced as far as possible for two reasons: first, inconsistencies among several copies of the same logical data are automatically avoided by having only one copy, and second, storage space is saved by eliminating redundancy. In distributed databases, however, there are several reasons for considering data redundancy as desirable features. First, the locality of applications can be increased if the data are replicated at all sites where applications need it, and second, the availability of the system can be increased, because a site failure does not stop the execution of applications at other sites if the data are replicated.
- *Complex physical structures and efficient access.* In distributed databases, complex accessing structures are not the right tool for efficient access. Therefore, while efficient access is a main problem in distributed databases, physical structures are not a relevant technological issue. Efficient access to a distributed database cannot be provided by using intersite physical structures, because it is very difficult to build and maintain such structures and because it is not convenient to navigate at a record level in distributed databases.

The software components which are typically necessary for building a distributed database in this case are:

1. The database management component (DB)
2. The data communication component (DC)

3. The data dictionary (DD), which is extended to represent information about distribution of data in the network
4. The distributed database component (DDC)

An important property of distributed database management systems (DDBMSs) is whether they are homogeneous or heterogeneous. Homogeneous DDBMS refers to a DDBMS with the same DBMS at each site, even if the computers and/or the operating systems are not the same. A heterogeneous DDBMS uses instead at least two different DBMSs. Heterogeneous DDBMS adds the problem of translating between the different data models of the different local DBMSs to the complexity of homogeneous DDBMSs.

12.2 Distributed DBMS Architecture

The architecture of a system defines its structure. This means that the components of the system are identified, the function of each component is specified, and the interrelationships and interactions among these components are defined.

12.2.1 DBMS Standardization

The standardization efforts related to DBMSs because of the close relationship between the architecture of a system and the reference model of that system, which is developed as a precursor to any standardization activity. It is defined as “a conceptual framework whose purpose is to divide standardization work into manageable pieces and to show at a general level how these pieces are related with each other.” A reference model can be described according to three different approaches:

1. *Based on components.* The components of the system are defined together with the interrelationships between components. Thus a DBMS consists of a number of components, each of which provides some functionality. Their orderly and well-defined interaction provides total system functionality.
2. *Based on functions.* The different classes of users are identified and the functions that the system will perform for each class are defined. The system specifications within this category typically specify a hierarchical structure for user classes. This results in hierarchical system architecture with well-defined interfaces between the functionalities of different layers.
3. *Based on data.* The different types of data are identified, and an architectural framework is specified which defines the functional units that will realize or use data according to these different views. Since data are the central resource that a DBMS manages, this approach is claimed to be the preferable choice for standardization activities. The advantage of the data approach is the central importance it associates with the data resource.

12.2.2 Architectural Models for Distributed DBMS

Let us consider the possible ways in which multiple databases may be put together for sharing by multiple DBMSs. We use a classification that organizes the systems as characterized with respect to (1) the autonomy of local systems, (2) their distribution, and (3) their heterogeneity.

Autonomy

It refers to the distribution of control, not of data. It indicates the degree to which individual DBMS can operate independently. Autonomy is a function of a number of factors such as whether they can independently execute transactions, and whether one is allowed to modify them. Requirements of an autonomous system have been specified in a variety of ways.

1. The local operations of the individual DBMSs are not affected by their participation in the multidatabase system.
2. The manner in which the individual DBMSs process queries and optimize them should not be affected by the execution of global queries that access multiple databases.
3. System consistency or operation should not be compromised when individual DBMSs join or leave the multidatabase confederation.

The dimensions of autonomy are specified as follows:

1. *Design autonomy.* Individual DBMSs are free to use the data models and transaction management techniques that they prefer.
2. *Communication autonomy.* Each of the individual DBMSs is free to make its own decision as to what type of information it wants to provide to the other DBMSs or to the software that controls their global execution.
3. *Execution autonomy.* Each DBMS can execute the transactions that are submitted to it in any way that it wants to do.

Distribution

There are a number of ways DBMSs can be distributed. We abstract the alternatives into two classes: client/server distribution and peer-to-peer distribution. The client/server distribution concentrates data management duties at servers while the clients focus on providing the application environment including the user interface. The communication duties are shared between the client machines and servers. Client/server DBMSs represent the first attempt at distributing functionality. There are a variety of ways of structuring them, each providing a different level of distribution. In peer-to-peer systems, there is no distinction of client machines vs. servers. Each machine has full DBMS functionality and can communicate with other machines to execute queries and transactions.

Heterogeneity

It may occur in various forms in distributed systems, ranging from hardware heterogeneity and differences in networking protocols to variations in data managers. Heterogeneity in query languages not only involves the use of completely different data access paradigms in different data models, but also covers difference in languages even when the individual systems use the same data model. Different query languages that use the same data model often select very different methods for expressing identical requests.

12.2.3 Types of Distributed DBMS Architecture

The distributed DBMS architecture types namely client server and peer-to-peer systems are discussed below.

Client/Server Systems

The general idea of this architecture is simple and elegant: distinguish the functionality that needs to be provided and divide these functions into two classes as server functions and client functions. This provides a two level architecture which makes it easier to manage the complexity of modern DBMSs and the complexity of distribution.

As with any highly popular term, client/server has been much abused and has come to mean different things. If one takes a process-centric view, then any process that requests the services of another process is its client and vice versa. In this sense, “client/server computing” and “client/server DBMS,” as it is used in its more modern context, do not refer to processes, but to actual machines.

The architecture shown in Fig. 12.2 is quite common in relational systems where the communication between the clients and the server without trying to understand or optimize them. The server does most of the work and returns the result relation to the client. There are a number of different types of client/server architecture. The simplest is the case where there is only one server which can be accessed by multiple clients. We call this as “multiple client–single server.” From a data management perspective, this is not much different from centralized databases since the database is stored on only one machine (the server) which also hosts the software to manage it.

There are two management strategies in multiple client–multiple server: either each client manages its own connection to the appropriate server or each client knows of only its “home server” which then communicates with the other servers as required. The former approach simplifies server code, but loads the client machines with additional responsibilities. This leads to what has been called as “heavy client” systems. The latter approach, on the other hand, concentrates the data management functionality at the servers. Thus, the transparency of data access is provided at the server interface, leading to “light clients.”

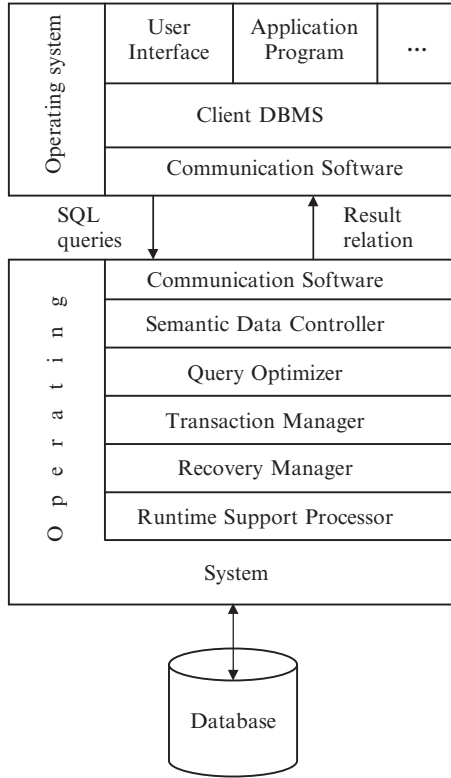


Fig. 12.2. Client/server reference architecture

Peer-to-Peer Distributed Systems

The architecture of this model shown in Fig. 12.3 provides the levels of transparency. Data independence is supported since the model is an extension of ANSI/SPARC, which provides such independence naturally. Location and replication transparencies are supported by the definition of the local and global conceptual schemas and the mapping in between.

Network transparency, on the other hand, is supported by the definition of the global conceptual schema. The user queries data irrespective of its location or of which local component of the distributed database system will service it. As mentioned before, the distributed DBMS components at different sites communicate with one another.

12.3 Distributed Database Design

Designing a distributed database is very difficult, since many technical and organizational issues, which are crucial in the design of single-site databases, become more difficult in a multiple-site system. From the technical viewpoint,

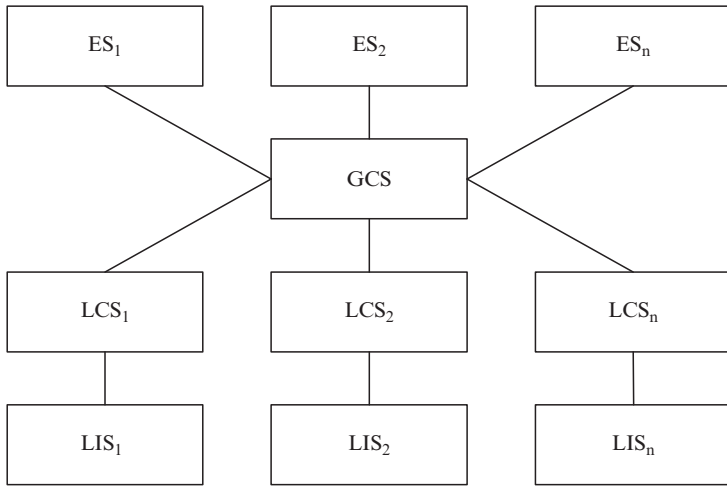


Fig. 12.3. Distributed database reference architecture

new problems arise such as the interconnection of sites by a computer network and the optimal distribution of data and applications to the sites for meeting the requirements of applications and for optimizing performances. From the optimization viewpoint, the issue of decentralization is crucial, since distributed systems typically substitute for large, centralized systems, and in thus case distributing an application has a major impact on the organization.

The mathematical problem of optimally distributing data over a computer network has been widely analyzed in the context of distributed file systems and, more recently, in the context of distributed databases. The major outcomes of this research are twofold:

1. Several design criteria have been established about how data can be conveniently distributed.
2. Mathematical foundation has been given to design aids that, in the near future, will help the designer in determining data distribution.

12.3.1 Framework for Distributed Database Design

The design of a centralized database amounts to the following factors:

- Designing the conceptual schema which describes the integrated database, i.e., all the data which are used by the database applications.
- Designing the physical database, i.e., mapping the conceptual schema storage areas and determining appropriate access methods.
- Designing the fragmentation, i.e., determining how global relations are subdivided into horizontal, vertical, or mixed fragments.
- Designing the allocation of fragments, i.e., determining how fragments are mapped to physical images; in this way, also the replication of fragment is determined.

The distinction between these two problems (related to fragmentation) is conceptually relevant, since the first one deals with the logical criteria which motivate the fragmentation of a global relation, while the second one deals with the physical placement of data at various sites. However, this distinction must be introduced with extreme care.

The application requirements include:

1. The site from which the application is issued (also called *site of origin of application*).
2. The frequency of activation of the application (i.e., the number of activation requests in the unit time); in the general case of applications which can be issued at multiple sites, we need to know the frequency of activation of each application at each site.
3. The number, type, and the statistical distribution of accesses made by each application to each required data “object.”

12.3.2 Objectives of the Design of Data Distribution

- *Processing locality.* Distributing data to maximize processing locality corresponds to the simple of placing data as close as possible to the applications with which use them. Designing data distribution for maximizing processing locality (or, conversely, for minimizing remote references) can be done by adding the number of local and remote references corresponding to each candidate fragmentation and fragment allocation, and selecting the best solution among them.
- *Availability and reliability of distributed data.* A high degree of availability for read-only application is achieved by storing multiple copies of the same information. Reliability is also achieved by storing multiple copies of the same information since it is possible to recover from crashes or from the physical destruction of one of the copies by using the other, still available copies.
- *Workload distribution.* Distributing the work load over the sites is an important feature of distributed computer systems. Workload is done in order to take advantage of the different powers or utilizations of computers at each site, and to maximize the degree of parallelism of execution of applications.
- *Storage costs and availability.* Database distribution should reflect the cost and availability of storage at the different sites. It is possible to have specialized sites in the network for data storage, or conversely to have sites which do not support mass storage at all. Typically, the cost of data storage is not relevant compared with CPU, I/O, and transmission costs of applications, but the limitation of available storage at each site must be considered.

12.3.3 Top-Down and Bottom-Up Approaches to the Design of Data Distribution

In the top-down approach, we start by designing the global schema, and we proceed by designing the fragmentation of the database, and then by allocating the fragments to the sites, creating the physical images. The approach is completed fragments by performing, at each site, the physical design of the data which are allocated to it.

When the distributed database is developed as the aggregation of existing databases, it is not easy to follow the top-down approach. In fact, in this case the global schema is often produced as a compromise between existing data descriptions. It is even possible that each pair of existing databases is independently interfaced using a different translation schema, without the notion of a global schema.

When existing databases are aggregated, a bottom-up approach to the design of data distribution can be used. This approach is based on the integration of existing schemata into a single global schema. By integration, we mean the merging of common data definitions and the resolution of conflicts among different representations given to the same data.

A heterogeneous system adds to the complexity of data integration the need for a translation between different representations. In this case, it is possible to make a one-to-one translation between each pair of different DBMSs; however, the approach which is mostly used in the prototypes of heterogeneous systems is to select a common data model, and then to translate into this unique representation all the different schemata of the involved DBMSs.

12.3.4 Design of Database Fragmentation

The design of fragmentation is the first problem that must be solved in the top-design of data distribution. The purpose of fragmentation design is to determine nonoverlapping fragments which are “logical units of allocation,” i.e., that are appropriate start points for the following data allocation problem.

Horizontal Fragmentation

Determining the horizontal fragmentation of a database amounts to determining both “logical” properties of data, such as the number of references of applications to fragments; this coordination of logical and statistical aspects is rather difficult.

Primary Horizontal Fragmentation

The primary horizontal fragments are defined using selections on global relations. The correctness of primary fragmentation requires that each tuple

of the global relation be selected in one and only one fragment. Thus, determining the primary fragmentation of a global relation requires determining a set of disjoint and complete selection prediction. The property that we require for each fragment is that the elements of them must be referenced homogeneously by all the applications. Let R be the global relation for which we want to produce a horizontal primary fragmentation.

We introduce the following definitions:

1. A *simple predicate* is a predicate of the type:

$$\text{Attribute} = \text{Value}$$

2. A *minterm predicate* y for a set P of simple predicates is the conjunction of all predicates appearing in P , either taken in natural form or negated, provided that this expression or contradiction. Thus

$$y = \bigwedge_{p_i \in P} p_i^*$$

where ($p_i^* = p_i$ or $p_i^* = \text{NOT } p_i$) and $y \neq \text{false}$.

3. A fragment is the set of all tuples for which a minterm predicate holds.

Derived Horizontal Fragmentation

The derived horizontal fragmentation of a global relation R is not based on properties of its own attributes, but it is derived from the horizontal fragmentation of another relation. Derived fragmentation is used to facilitate the join between fragments.

A *distributed join* is a join between horizontally fragmented relations. When an application requires the join between two global relations R and S , all the tuples of R and S need to be compared. Thus, in principal, it is necessary to compare all the fragments R_i of R with all the fragments S_j of S . However, sometimes it is possible to deduce that some of the partial joins $R_i \text{ JN } S_j$ are intrinsically empty. This happens when, for a given data distribution, values of the join attribute in R_i and S_j are disjoint.

A distributed join is represented efficiently using *join graphs*. The join graph G of the distributed join $R \text{ JN } S$ is a graph (N, E) , where nodes N represent fragments of R and S and nondirected edges between nodes represent joins between fragments which are not intrinsically empty. For simplicity, we do not include in N those fragments of R or S which have an empty join with all fragments of the other relation.

There are two types of reduced join graphs (as shown in Fig. 12.4) that are particularly relevant:

1. A reduced join graph is *partitioned* if the graph is composed of two or more subgraphs without edges between them as shown in Fig. 12.4.
2. A reduced join graph is *simple* if it is partitioned and each subgraph has just only one edge as shown in Fig. 12.4.

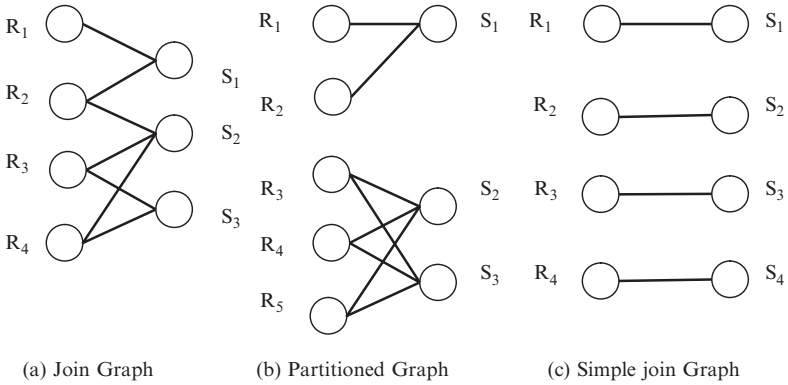


Fig. 12.4. Join graphs

Vertical Fragmentation

The purpose of vertical fragmentation is to identify fragments R_i such that many applications can be executed using just one fragment. Determining a fragmentation for a global relation R is not easy, since the number of possible partitioning grows combinatorial with the number of attributes of R , and the number of possible clusters is even larger. Thus, in the presence of a large relation, heuristic approaches are necessary to determine the partitions of clusters. We briefly indicate how such methods operate. Two alternate approaches are possible for attribute partitioning:

1. The *split approach* in which global relations are progressively split into fragments.
2. The *grouping approach* in which attributes are progressively aggregated to constitute fragments.

Vertical clustering introduces *replication* within fragments, since values of overlapping attributes are replicated. Replications have a different effect on read-only and update applications. Read-only applications take advantage of replication, because it is more likely that they can reference data locally. For update applications replications are not convenient, since they must all be same in order to preserve consistency.

Mixed Fragmentation

The simplest way of building mixed fragmentation consists of:

1. Applying horizontal fragmentation to vertical fragments.
2. Applying vertical fragmentation to horizontal fragments.

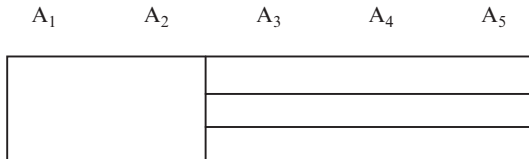
Although these operations can be recursively repeated, in generating fragmentation trees of any complexity, it seems that having more than two levels of fragmentation is not of practical interest. Horizontal fragmentation is applied just to one fragment produced by vertical fragmentation as shown in Fig. 12.5. Vertical fragmentation is applied just to one fragment produced by horizontal fragmentation as shown in Fig. 12.5.

Allocation of Fragments

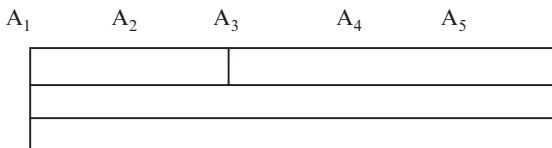
The data allocation problem is widely in the context of file allocation problem. The easiest way to apply this work to the fragment allocation problem is to consider each fragment as a separate file. However, this approach is not convenient due to the following reasons:

- Fragments are not properly modeled as individual files; since in this way we do not take into account the fact that they have the same structure or behavior.
- There are many more fragments than original global relations, and many analytic models cannot compute the solution of problems involving too many variables.
- Modeling application behavior in file systems is very simple, while in distributed databases applications can make a sophisticated use of data.

The correct approach would be to evaluate data distribution is to measure how optimized applications behave with it. This, however, requires optimizing all the important applications for each data allocation.



(a) Vertical Fragmentation followed by Horizontal Fragmentation



(b) Horizontal Fragmentation followed by Vertical Fragmentation

Fig. 12.5. Mixed fragmentation

General Criteria for Fragment Allocation

Determining a nonredundant final allocation is easier. The simplest method is “best-fit” approach; a measure is associated with each possible allocation, and the site with best measure is selected. Replication introduces further complexity in the design, because of the following reasons:

- The degree of replication of each fragment becomes a variable of the problem.
- Modeling read applications is complicated by the fact that the applications can now select among several alternatives sites for accessing fragments.

For determining the redundant allocation of fragments, either of the following two methods can be used:

- Determine the set of all sites where the benefit of allocating one copy of the fragment is higher than the cost, and allocate a copy of the fragment to each element of this set; this method selects “all beneficial sites.”
- Determine first the solution of the nonreplicated problem, and then progressively introduce replicated copies starting from the most beneficial; the process is terminated when no “additional replication” is beneficial.

Both methods have some disadvantages. In the “all beneficial sites” method, quantifying costs and benefits for each individual fragment allocation is more critical than in the nonredundant case. The “additional replication” method is a typical heuristic approach; with this method, it is possible to take into account that the increase in the degree of redundancy is progressively less beneficial.

12.4 Semantic Data Control

An important requirement of a centralized or a distributed DBMS is the ability to support data control. Data control typically includes view management, security control, and semantic integrity control. Informally, these functions must ensure that authorized users perform correct operations on the database, contributing to the maintenance of database integrity. There are several ways to store data control definitions, according to the way the directory is managed. Directory information can be stored differently according to its type; in other words, some information might be fully duplicated whereas other information might be distributed.

12.4.1 View Management

One of the main advantages of the relational model is that it provides full logical data independence. External schemas enable user groups to have their

particular view of the database. In a relational system, a view is a virtual relation, defined as the result of a query on base relations (or real relations), but not materialized like a base relation, which is stored in the database. A view is a dynamic window in the sense that it reflects all updates to the database. An external schema can be defined as a set of views and/or base relations. Besides their use in external schemas, views are useful for ensuring data security in a simple way. By selecting a subset of the database, views *hide* some data. If users may only access the database through views, they cannot see or manipulate the hidden data, which are therefore secure.

12.4.2 Views in Centralized DBMSs

A view is a relation derived from base relations as the result of a relational query. It is defined by associating the name of the view with the retrieval query that specifies it. For example, a view of system analysts (SYSAN) derived from relation EMP (ENO, ENAME, and TITLE) can be defined by the following SQL query:

```

CREATE   VIEW      SYSAN (ENO, ENAME)
AS       SELECT    ENO, ENAME
          FROM      EMP
          WHERE     TITLE = "Syst.Anal."

```

The single effect of this statement is the storage of this view definition in the catalog. No other information needs to be recorded. Therefore, the result to the query defining the view (i.e., a relation having the attributes ENO and ENAME for the SYSAN as shown in table below) is not produced. However, the view SYSAN can be manipulated as a base relation.

<u>SYSAN</u>	
<u>ENO</u>	<u>ENAME</u>
E1	M. John
E3	C. Mark
E7	R. Peter

12.4.3 Update Through Views

Views can be defined using arbitrarily complex relational queries involving selection, projection, join, aggregate functions, and so on. All views can be interrogated as base relations, but not all views can be manipulated as such. Updates through views can be handled automatically only if they can be propagated correctly to the base relations. We can classify views as being updatable and not updatable. A view is updatable only if the updates to the view can be propagated to the base relation without ambiguity. The view SYSAN in the above example is updatable. The insertion of a new system analyst

<101, John> will be mapped into the new employee <101, John, Syst.Anal>. If attributes other than TITLE were hidden by the view, they would be assigned null values.

12.4.4 Views in Distributed DBMS

The definition of a view is similar in a distributed DBMS and in a centralized system. However, a view in a distributed system may be derived from fragment relations stored at different sites. When a view is defined, its name and its retrieval query are stored in the catalog. Since views may be used as base relations by application programs, their definitions should be stored in the directory in the same way as the base relation descriptions. Depending on the degree of site autonomy offered by the system, view definitions can be centralized at one site, partially duplicated, or fully duplicated. If the view definition is not present at the site where query is issued, remote access to the view definition site is necessary.

Views derived from distributed relations may be costly to evaluate. Since in a given organization it is likely that many users access the same views, some proposals have been made to optimize view derivation. The view derivation is done by merging the view qualification with the query qualification. An alternate solution is to avoid view derivation by maintaining actual versions of the views, called *snapshots*. A snapshot represents a particular state of the database and is therefore static, meaning that it does not reflect updates to base relations. Snapshots are useful when users are not particularly interested in seeing the most recent version of the database.

12.4.5 Data Security

It is an important function of a database system that protects data against unauthorized access. Data security includes two aspects: data protection and authorization control.

Data protection is required to prevent unauthorized users from understanding the physical content of data. This function is typically provided by file systems in the context of centralized and distributed operating systems. The main data protection approach is data encryption, which is useful for both information stored on a disk and information exchanged on a network. Encrypted (encoded) data can be decrypted (decoded) only by authorized users who “know the code.”

Authorization control must guarantee that only authorized users perform operations they are allowed to perform on the database. Many different users may have access to a large collection of data under the control of a single centralized or distributed system. In relational systems, authorizations can be uniformly controlled by database administrators using high-level constructs. For example, controlled objects can be specified by predicates in the same way as a query qualification.

12.4.6 Centralized Authorization Control

Three main aspects are involved in authorization control: the *users*, who trigger the execution of the application programs; the *operations*, which are embedded in application programs; and the *database objects*, on which the operations are performed. Authorization control consists of checking whether a given triple can be allowed to proceed (i.e., the user can execute the operation of the object). An authorization can be viewed as a triple (user, operation type, and object definition) which specifies that the user has the right to perform an operation of operation type on an object. To control authorizations properly, the DBMS requires users, objects, and rights to be defined.

In a relational system, objects can be defined by their type (view, relation, tuple, and attribute) as well as by their content using selection predicates. A right expresses a relationship between a user and an object for a particular set of operations. In a SQL-based relational DBMS, an operation is a high-level statement such as SELECT, INSERT, UPDATE, or DELETE, and rights are defined (granted or revoked) using the following statements:

```
GRANT <operation type(s)> ON <object> TO <user(s)>
REVOKE <operation type(s)> FROM <object> TO <user(s)>
```

The keyword *public* can be used to mean all users. Authorization control can be characterized based on who (the grantors) can grant the rights. In its simplest form, the control is centralized: a single user or user class, the database administrators have all privileges on the database objects and are the only one allowed to use the GRANT and REVOKE statements.

A more flexible but complex form of control is decentralized; the creator of an object becomes its owner and is granted all privileges on it. In particular, there is the additional operation type GRANT. Granting the GRANT privilege means that all rights of the grantor performing the statement are given to the specified users. Therefore the person receiving the right (the grantee) may subsequently grant privileges on that object. The main difficulty with this approach is that the revoking process must be recursive.

12.4.7 Distributed Authorization Control

The additional problem of authorization control is a distributed environment system from the fact that the objects and subjects are distributed. These problems are remote user authentication, management of distributed authorization rules, as well as handling of views and of user groups. Remote user authentication is necessary since any sites of a distributed DBMS may accept programs initiated, and authorized, at remote sites. Two solutions are possible in preventing unauthorized users in remote accessing as follows:

1. The information for authenticating users (user name and password) is replicated at all sites in the catalog. Local programs, initiated at a remote site, must also indicate the user name and password.

2. All sites of the distributed DBMS identify and authenticate themselves similarly to the way users do. Intersite communication is thus protected by the use of the site password.

Distributed authorization rules are expressed in the same way as centralized ones. They can be either fully replicated at each site or stored at the sites of the reference objects. The main advantage of fully replicated approach is that authorization can be processed by query modification at compile time. However, directory management is more costly because of data duplication. The second solution is better if locality of reference is high. However, distributed authorization cannot be controlled at compile time.

12.4.8 Semantic Integrity Control

Another important and difficult problem for a database system is how to guarantee database consistency. A database state is said to be consistent if the database satisfies a set of constraints, called *semantic integrity constraints*. Maintaining a consistent database requires various mechanisms such as concurrency control, reliability, protection, and semantic integrity control. Semantic integrity control ensures database consistency by rejecting update programs which lead to inconsistent database states, or by activating specific actions on the database state, which compensate for the effects of the update programs.

Semantic integrity constraints are rules that represent the *knowledge* about the properties of an application. They define static or dynamic application properties which cannot be directly captured by the object and operation concepts of a data model. Thus the concept of an integrity rule is strongly connected with that of a data model in the sense that more semantic information about the application can be captured by means of these rules.

Two main types of integrity constraints can be distinguished: structural constraints and behavioral constraints. Structural constraints express basic semantic properties inherent to a model. Examples of such constraints are unique key constraints in the relational model, or one-to-many associations between objects in the network model. Behavioral constraints, on the other hand, regulate the application behavior. Thus they are essential in the database design process. They can express associations between objects, such as inclusion dependency in the relational model, or describe object properties and structures. The increasing variety of database applications and the recent development of database design aid tools call for powerful integrity constraints which can enrich the data model.

The main problem in supporting automatic semantic integrity control is that the cost of checking assertions can be prohibitive. Enforcing integrity assertions is costly because it generally requires access to a large amount of data which are not involved in the database updates. The problem is more difficult when assertions are defined over a distributed database.

12.4.9 Distributed Semantic Integrity Control

The method obviously works with replicated directories. The two main problems of designing an integrity subsystem for a distributed DBMS are the definition and storage of assertions, and the enforcement of these assertions.

Definition of Distributed Integrity Assertions

An integrity assertion is supposed to be expressed in tuple relational calculus. Each assertion is seen as a query qualification which is either true or false for each tuple in the Cartesian product of the relations determined by tuple variables. Since assertions can involve data stored at different sites, their storage must be decided so as to minimize the cost of integrity checking. There is a strategy based on integrity assertions that distinguishes three classes of assertions:

1. *Individual assertions.* Single-relation single-variable assertions. They refer only to tuples to be updated independently of the rest of the database.
2. *Set-oriented assertions.* Includes single-relation multivariable constraints such as functional dependency and multirelation multivariable constraints.
3. *Assertions involving aggregates.* Requires special processing because of the cost of evaluating the aggregates.

Enforcement of Distributed Integrity Assertions

Enforcing distributed integrity assertions is more complex than needed in centralized DBMSs. The main problem is to decide where to enforce the integrity assertions. The choice depends on the class of the assertion, the type of update, and the nature of the site where the update is issued:

1. *Individual assertions.* Two cases are considered. If the update is an insert statement, all the tuples to be inserted are explicitly provided by the user. In this case, all individual assertions can be enforced at the site where the update is submitted. If the update is a qualified update (delete or modify statements), it is sent to the sites storing the relation that will be updated.
2. *Assertions involving aggregates.* These assertions are among the most costly to test because they require the calculation of the aggregate functions. The aggregate functions generally manipulated are MIN, MAX, SUM, and COUNT. Each aggregate function contains a projection part and a selection part. To enforce these assertions efficiently, it is possible to produce compiled assertions that isolate redundant data which can be stored at each site storing the associated relation. This data are called *views*.

12.5 Distributed Concurrency Control

Distributed concurrency control mechanism of a distributed DBMS ensures the consistencies of the database. It is maintained in a multiuser distributed environment. If transactions are internally consistent, the simplest way of achieving this objective is to execute each transaction alone, one after another. It is obvious that such an alternate is only of theoretical interest and cannot be implemented in practical systems, since it minimizes the system throughput. The level of concurrency is probably that most important parameter in distributed systems. Therefore, the concurrency control mechanisms attempts to find a suitable trade-off between maintaining the consistency of the database and maintaining a high-level of concurrency.

The distributed system is fully reliable and does not experience any failures even though this is an entirely unrealistic assumption; there is a reason for making it. It permits as to delineate the issues related to the management of concurrency from those related to the operation of the reliable distributed system.

12.5.1 Serializability Theory

If the concurrent execution of transactions leaves the database in a state that can be achieved by their serial execution in some order, problems such as last updates will be resolved. This is exactly that the point of serializability argument. A schedule S is defined over a set of transaction $T = (T_1, T_2 \dots T_n)$ and specifies an interleaved order of execution of this transaction operations. Two operations $O_{ij}(X)$ and $O_{jk}(X)$ accessing the same database entity X are said to be in conflict if at least is a right. From this definition, first, read operations do not conflict each other. Therefore, the two types of conflicts are read-write and write-write.

12.5.2 Taxonomy of Concurrency Control Mechanism

There are number of ways that the concurrency control approaches can be classified. One obvious classification criterion is the mode of database distribution. Some algorithms that have been proposed require a fully replicated database, while others can operate on partially replicated or partitioned database. The concurrency control algorithms may also be classified according to the network topology, such as those requiring a communication subnet with broadcasting capability or those working in a star type network or circularly connected network. The most common classification criterion, however, is synchronization primitives. The corresponding breakdown of concurrency control algorithms results in two classes, algorithms that are based on mutually exclusive access to shared data and those that attempt to order the execution of transactions according to the set of rules.

The concurrency control mechanism is grouped into two broad classes namely, pessimistic control methods and optimistic control methods as shown in Fig. 12.6. Pessimistic algorithm synchronizes the concurrent execution of transaction early in their execution life cycles, whereas optimistic algorithms delay the synchronization of transactions until their termination. The pessimistic group consists of locking-based algorithm, ordering-based algorithm, and hybrid algorithm. The optimistic group can be similarly classified as locking based or timestamp ordering (TO) based.

In the locking-based approach, the synchronization of transaction is achieved by employing physical or logical locks on some portion or granule of the database. The class is subdivided further according to where the lock management activities are performed:

1. Centralized locking, one of the sites in the network, is designated as the primary site where the lock tables for the entire database are stored and charged with the responsibility of granting locks to transaction.
2. Primary copy locking is one of the copies each lock unit is designated as the primary copy, and it is a copy that has to be locked for the purpose of particular unit.
3. Decentralized blocking, the lock management, is shared by all the sides of the network. In this case, the execution of transaction involves the participation and coordination of schedulers at more than site. Each local scheduler is responsible for the lock units local to the site.

The timestamp ordering class involves organizing the execution order of transaction so that they maintain mutual and interconsistency. This ordering

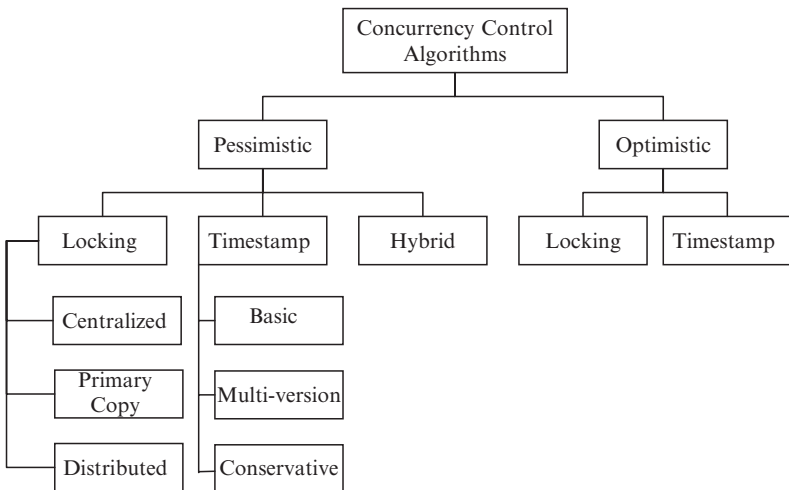


Fig. 12.6. Classification of concurrency control

is maintained by assigning timestamps to both the transactions and the data items that are stored in the database.

12.5.3 Locking-Based Concurrency Control

The main idea of this control is to ensure that the data that are shared by conflicting operations are accessed by one operation at a time. This is accomplished by associating a “lock” with each lock unit. This lock is set by a transaction before it is accessed and is reset at the end of its use. Obviously the lock unit cannot be accessed by an operation if it is locked by another. Thus, a lock request by a transaction is granted only if the associated lock is not being held by any other transaction. The distributed DBMS not only manages locks but also handles the lock management responsibilities behalf of the transaction. In other words, the users do not need to specify when data need to be locked, the distributed DBMS takes care of the every time the transaction issues read or write operations.

In a locking-based system, the scheduler is a lock manager. The transaction manager (TM) passes to the lock manager, the database operation and associated information. The lock manager then checks if the lock unit that contains the data item is already locked. If so, and if the existing lock mode is incompatible with that of the current transaction, the current operation is delayed. Otherwise, the lock is set to desired mode and database operation is passed on to the data processor for actual database access. The transaction manager is then informing of the results of the operations. The termination of transaction results in the release of its locks and initiation of another transaction that might be waiting for the access to be same data item.

Centralized 2PL Algorithm

These algorithm can be easily extended (replicated or partitioned) to the distributed database environment. One way of doing this is to delegate lock management responsibility to a single site only. This means that only of the sites has a lock manager, the transaction manager at the other sites communicates with rather than with the own lock managers. This is also known as *primary site 2PL algorithm*. The communication between the operating sites in order to execute a transaction according to a centralized 2PL is shown in Fig. 12.7. This communication between transaction manager at the site where the transaction is initiated, the lock manager at the central site, and the data processor at other participating sites are those at which the operation is to be carried out.

An important difference between the centralized TM algorithm and the TM algorithm of locking is that the distributed TM has to implement the replica control protocol if the database is replicated. The central lock manager does not send the operations to the respective data processors; that is done by the coordinating TM.

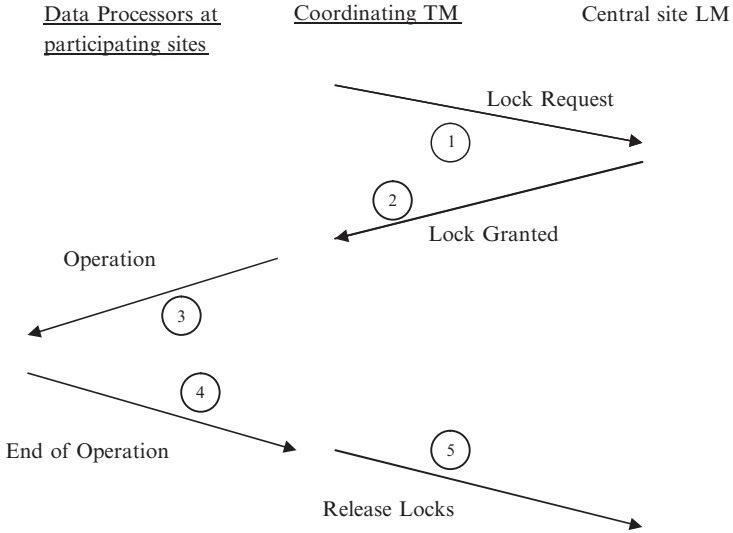


Fig. 12.7. Communication structure of distributed 2PL

Primary Copy 2PL (PC2PL)

It is a straightforward extension of centralized 2PL in an attempt to counter the latter's potential performance problems. Basically, it implements lock managers at a number of sites and makes each lock manager responsible for managing the locks for a given set of lock units. The TMs then send their lock and unlock requests to the lock managers that are responsible for specific lock unit. Thus the algorithm treats one copy of each data item as its primary copy.

Basically, the one change from centralized 2PL is that the primary copy locations have to be determined for each data item prior to sending a lock or unlock request to the local manager at the site. The load of the central site is also reduced without causing a large amount of communication among the TMs and lock managers.

Distributed 2PL (D2PL)

It expects the availability of lock managers at each site. If the database is not replicated, distributed 2PL degenerates into the primary copy 2PL algorithm. If data are replicated, the transaction implements the ROWA replica control protocol. The communication between cooperating sites that execute a transaction according to the distributed 2PL is shown in Fig. 12.8.

The D2PL transaction management algorithm is similar to the C2PL-TM with two major modifications. The messages that are sent to the central site lock manager in C2PL-TM are sent to the lock managers at all participating

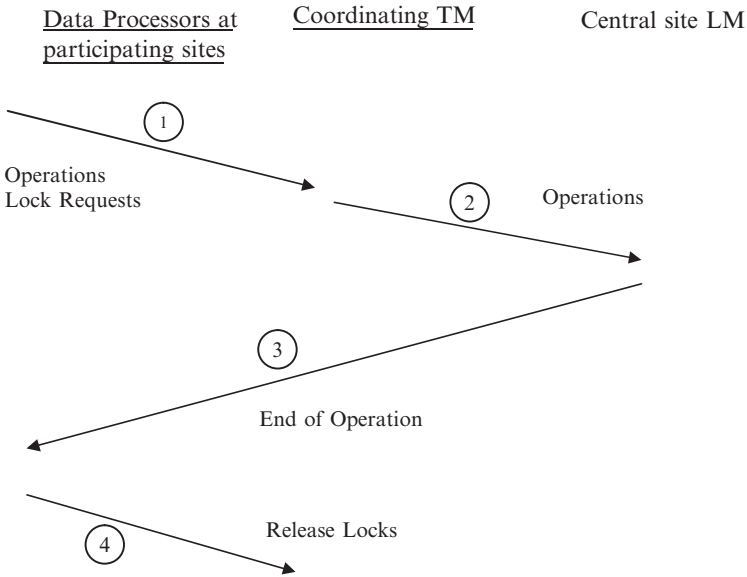


Fig. 12.8. Communication structure of distributed 2PL

sites in D2PL-TM. The second difference is that the operations are not passed to the data processors by the coordinating TM, but by the participating local managers. This means that the coordinating TM does not wait for a “lock request granted” message.

12.5.4 Timestamp-Based Concurrency Control Algorithms

A timestamp is a simple identifier that serves to identify each transaction uniquely and to permit ordering. *Uniqueness* is only one of the properties of timestamp generation. The second property is *monotonicity*. Two timestamps generated by the same transaction manager should be monotonically increasing. Thus timestamps are values derived from a totally ordered domain. It is the second property that differentiates a timestamp from a transaction identifier.

There are a number of ways that timestamps can be assigned. One method is to use a global (system wide) monotonically increasing counter. However, the maintenance of global counters is a problem in distributed systems. Therefore, it is preferable that each site autonomously assigns timestamps based on its local counter. To maintain uniqueness, each site appends its own identifier to the counter value. Thus the timestamp is a two-tuple of the form $\langle \text{local counter value, site identifier} \rangle$. Note that the site identifier is appended in the least significant position. Hence it serves only to order the timestamps of two transactions that might have been assigned the same local counter value. In

each system clock, it is possible to use system clock values instead of counter values.

12.5.5 Optimistic Concurrency Control Algorithms

The concurrency control algorithms discussed so far are pessimistic in nature. In other words, they assume that the conflicts between transactions are quite frequent and do not permit a transaction to accesses that data item. Thus the execution of any operation of a transaction follows the sequence of phases: validation (V), read (R), computation (C), and write (W) as shown in Fig. 12.9.

Optimistic algorithms, on the other hand, delay the validation phase until just before the write phase as shown in Fig. 12.10. Thus an operation submitted to an optimistic scheduler is never delayed. The read, compute, and write operations of each transactions are processed freely without updating the actual database. Each transaction initially makes its updates on local copies of data items. The validation phase consists of checking if these updates would maintain the consistency of the database. If the answer is affirmative, the changes are made global (i.e., written into the actual database). Otherwise, the transaction is aborted and has to restart. It is possible to design locking-based optimistic concurrency control algorithms.

However, the original optimistic proposals are based on timestamp ordering. Therefore, we describe only the optimistic approach using timestamps. Optimistic algorithms have not been implemented in any commercial or prototype DBMS. Therefore, the information regarding their implementation trade-offs is insufficient. As a matter of fact, the only centralized implementation of optimistic concepts (not the full algorithm) is in IBM's IMS-FASTPATH, which provides primitives that permit the programmer to access the database in an optimistic manner.

12.5.6 Deadlock Management

A deadlock can occur because transactions wait for one another. Informally, a deadlock situation is a set of requests that can never be granted by the

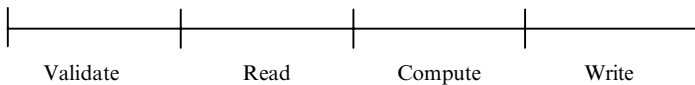


Fig. 12.9. Phases of pessimistic transaction execution



Fig. 12.10. Phases of optimistic transaction execution

concurrency control mechanism. A deadlock is a permanent phenomenon. If one exists in a system, it will not go away unless outside intervention takes place. This outside interference may come from the user, the system operator, or the software system (the operating system or the distributed DBMS).

Deadlock Prevention

This method guarantee that deadlocks cannot occur in the first place. Thus the TM checks a transaction when it is first initiated and does not permit it to proceed if it may cause a deadlock. To perform this check, it is required that all of the data items that will be accessed by a transaction be predeclared. The TM then permits a transaction to proceed if all the data items that will access are available. The fundamental problem is that it is usually difficult to know precisely which data items will be accessed by a transaction. Access to certain data items may depend on conditions that may not be resolved until run time.

Deadlock Avoidance

This scheme either employs concurrency control techniques that will never result in deadlocks or requires that schedulers detect potential deadlock situations in advance and ensures that they will not occur. We consider both of these cases. The simplest means of avoiding deadlocks is to order the resources and insist that each process requests access to these resources in that order. This solution was long ago proposed for operating systems.

Accordingly, the lock units in the distributed database are ordered and transactions always request locks in that order. This ordering of lock units may be done either globally or locally at each site. Another alternative is to make use of transaction timestamps to prioritize transactions and resolve deadlocks by aborting transactions with higher (or lower) priorities. To implement this type of prevention method, the lock method is modified as follows. If a lock request of a transaction T_i is denied, the lock manager does not automatically force T_i to wait. Instead, it applies a prevention test to the requesting transaction that currently holds the lock (say T_j). If the test is passed, T_i is permitted to wait for T_j ; otherwise, one transaction or the other is aborted.

Deadlock Detection and Resolution

Detection is done by studying GWFG for the formulation of cycles. Resolution of deadlocks is typically done by the selection of one or more victim transactions that will be preempted and aborted in order to break the cycles in GWFG. Some of the factors affecting in selection of the minimum total-cost set for the breaking the deadlock cycle are:

1. The amount of effort that has already been invested in the transaction. This effort will be lost if the transaction is aborted.
2. The cost of aborting the transaction. This cost generally depends on the number of updates that the transaction has already performed.
3. The amount of effort it will take to finish executing the transaction. The scheduler wants to avoid aborting a transaction that is almost finished. To do this, it must be able to predict the future behavior of active transactions.
4. The number of cycles that contain the transaction. Since aborting a transaction breaks all cycles that contain it, it is the best to abort transactions that are part of more than one cycle.

Centralized Deadlock Detection

In this approach, one site is designated as the deadlock detector for the entire system. Periodically, each lock manager transmits its GWFG and looks for cycles in it. Actually, the lock managers need to send only messages in their graphs (i.e., the newly created or deleted edges) to the deadlock detector. The length of intervals for transmitting this information is a system design decision; the smaller the interval, the smaller the delays due to undetected deadlocks, but the larger the communication cost.

Hierarchical Deadlock Detection

An alternative to centralized deadlock detection is the building of a hierarchy of deadlock detectors. Deadlocks that are local to a single site would be detected at that site using the local WFG. Each site also sends its local WFG to the deadlock detector at the next level. Thus, distributed deadlocks involving two or more sites would be detected by a deadlock in the next lowest level that has control over these sites.

The hierarchical deadlock detection (as shown in Fig.12.11) method reduces the dependence on the central site, thus reducing the communication cost. It is, however, considerably more complicated to implement and would involve nontrivial modifications to the lock and transaction manager algorithms.

Distributed Deadlock Detection

This algorithm delegates the responsibility of detecting deadlocks to individual sites. Thus, as in the hierarchical deadlock detection, there are deadlock detectors at each site which communicate their local WFGs with one another (in fact, only the potential deadlock cycles are transmitted). The local WFGs at each site are formed and modified as follows:

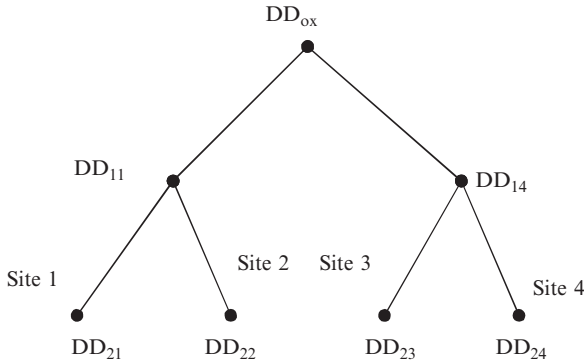


Fig. 12.11. Hierarchical deadlock detection

1. Since each site receives the potential deadlock cycles from other sites, these edges are added to the local WFGs.
2. The edges in the local WFG, which show that local transactions are waiting for transactions at other sites, are joined with edges in the local WFGs which show that remote transactions are waiting for local ones.

12.6 Distributed DBMS Reliability

A number of protocols need to be implemented within the DBMS to exploit the distribution of the database and replication of data items in order to make operations more reliable. A reliable distributed management system is one that can continue to process user request when the underlying system is unreliable. In other words, even when component of the distributed computing environment is failed, a reliable distributed DBMS should be able to continue executing user request without violating database consistency.

It is possible to discuss database reliability in isolation. However, the distributed DBMS is only one component of a distributed computing system. Its reliability is strongly dependent on the reliability of the hardware and software components that make up the distributed environment.

12.6.1 Reliability Concepts and Measures

The terms *reliability* and *availability* are used loosely in literature. Even among the researchers in the area of reliable computer systems, there is no consensus on the definitions of these terms.

System, State, and Failure

In the context of reliability, system refers to a mechanism that consists of a collection of components and interacts with its environment by responding to

stimuli from the environment with a recognizable pattern of behavior. Each component of a system is itself a system, commonly called a *subsystem*. The environment of a component is the system of which it is a part. The way the components of a system are put together is called the *design of the system*.

There are a number of ways of modeling the interaction between the software and the hardware in a computer system. One possible modeling method is to treat the program text as the design of an abstract system whose components are the hardware and software objects that are manipulated during the execution of the program. An external state of a system can be defined as the response that a system gives to an external stimulus. It is therefore possible to talk about a system changing external states according to repeated stimuli from the environment. We can define the internal state of the system similarly. It is convenient to define the internal state as the union of the external states of the components that make up the system. Again, the system changes its internal state in response to stimuli from the environment.

The behavior of the system in providing response to all the possible stimuli from the environment needs to be laid out in an authoritative specification of its behavior. The specification indicates the valid behavior of each system state. Such a specification is not only necessary for a successful system design but it is also essential to define the following reliability concepts. Any deviation of a system from the behavior in the specification is considered a failure.

Each failure obviously needs to be traced back to its cause. Failures in a system can be attributed to deficiencies either in the components that make up, or in the design, i.e., how these components are put together. Each states that reliable system goes through valid in the sense that the state fully meets its specification. However, in an unreliable system, it is possible that the system may get to an internal that may not obey its specification. Further transitions from this state would eventually cause a system failure. Such states are called *erroneous states*; the part of the state that is incorrect is called an *error in the system*. Any error in the internal states of the components of a system is called a *fault in the system*. Thus a fault that causes an error result in a system failure is shown in Fig. 12.12.

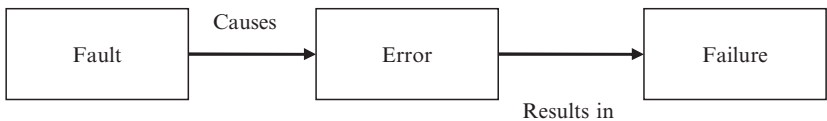


Fig. 12.12. Chain of events leading to system failure

We differentiate between errors that are permanent and those are not permanent. Permanents can apply to a failure, a fault, or an error, although we typically use the term with respect to a fault. The permanent faults also commonly called a *hard fault*, is one that reflects the irreversible change in the behavior of the system. This fault causes permanent errors that result in permanent failures. The characteristic of this failure is that the recovery from them requires intervention to “repair” the fault.

Reliability and Availability

Reliability refers to the probability that the system under consideration does not experience many failures in the given time interval. It is typically used to describe system that cannot be repaired or where the operation of the system is so critical that no down time for repair can be tolerated. Formally the reliability, $R(t)$, is defined the following conditional probability. Reliability theory, as it applies to hardware systems, has been developed significantly.

$$R(t) = \Pr\{0 \text{ failures in time } [0,t] \mid \text{no failures at } t = 0\}$$

Availability, $A(t)$, refers to the probability that the system is operational according to its specification at a given point in time t . A number of failures may have occurred prior to time t , but if they have all been repaired, the system is available at time t . It is apparent that availability refers to system that can be repaired. It can be used as some measure of “goodness” for those systems that can be repaired and which can be out of service for short periods of time during repair. Reliability and availability of a system are considered to be contradictory objectives. It is usually accepted that it is easier to develop highly available systems as oppose to highly reliable systems.

12.6.2 Failures in Distributed DBMS

Designing a reliable system that can record for failures requires identifying the types of failures with which the system has to deal. It indicates that the database recovery manager has to deal four types of failures namely, transaction failure (abort), site failure (system), media failure (disk), and communication line failure:

1. *Transaction (abort) failure*. It can fail for number of reasons. It can be due to an error in the transaction caused by incorrect input data as well as the detection of a present or potential deadlock. The frequency of this failure is not easy to measure. It is indicated that in system R, 3% of the transaction abort abnormally, in general it can be stated that:
 - Within a single application, the ratio of transaction that abort them is rather constant, being a function of the incorrect data, the available semantic data control feature and so on.
 - The number of transactions aborts by the DBMS due to concurrency.

2. *Site (system) failure.* It can be traced back to a hardware failure (processor, main memory, etc.) or to a software failure (bug in the operating system or in the DBMS code). The important point from the perspective is that a system failure is always assumed to result in the loss of main memory contents. In distributed database terminology, system failures are typically referred to as site failure, since they result in the failed site being unreachable from other sites in the distributed system.
3. *Media (disk) failures.* It refers to the failures of the secondary storage devices that store the database. Such failures may be due to operating system errors, as well as to hardware faults such as head crashes or controller failures. The important feature from the perspective of DBMS reliability is that all or part of the database that is on the secondary storage is considered to be destroyed and inaccessible. These failures are frequently treated as problems local to one site and therefore not specifically addressed in the reliability mechanisms of distributed DBMSs.
4. *Communication failures.* The communication failures are unique to distributed DBMS (not centralized DBMS). Lost or undeliverable messages are typically the consequence of communication line failures or site failures. If a communication line fails, in addition to losing messages in transit, it may also divide the network into two or more disjoint groups. This is called *network partitioning*. If the network is partitioned, the sites in each partition may continue to operate.

The detection of undeliverable messages is facilitated by the use of timers and a timeout mechanism that keeps track of how long it has been since the sender site has not received any confirmation from the destination site about the receipt of the message. The term for the failure of the communication network to deliver messages and the confirmations within this period is performance failure. It needs to be handled within the reliability protocols for distributed DBMSs.

Reasons for Failures in Distributed Systems

Soft failures make up more than 90% of all hardware system failures. It is interesting note that this percentage has not changed significantly since the early days of computing. More recent studies indicate that the occurrence of soft failures is significantly higher than that of hard failures. Most of the software failures are transient hence a dump and restart may be sufficient to recover without any need to repair the software. Software failures are most difficult to discuss because there is no agreement on a classification scheme. The software failures due to communication and database are by far the dominant causes. These are followed by operating system failures, which are then followed by failures in the application code and in the transaction management software.

When one investigates hardware causes of failures, 49% of hardware failures are disk failures, 23% are due to communication, 17% due to processor failure, 9% due to wiring, and 1% due to the failure of spares.

12.6.3 Basic Fault Tolerance Approaches and Techniques

The two fundamental approaches to constructing a reliable system are fault tolerance and fault prevention. Fault tolerance refers to a system design approach which recognizes that faults will occur; it tries to build mechanisms into the system so that the faults can be detected and removed or compensated for before they can result in a system failure. Fault prevention has two aspects. The first is fault avoidance, which refers to the techniques used to make sure that faults are not introduced into the system. These techniques involve detailed design methodologies and quality control. The second aspect is fault removal, which refers to the techniques that are employed to detect any faults that might have been remained in the system despite the application of fault avoidance and removes these faults. The fault removal techniques can be applied only during the system implementation prior to field use of the system.

Fault detection techniques are not coupled with fault tolerance features, they issue a warning when a failure occurs but do not provide any means of tolerating the failure. Therefore, it might be appropriate to separate fault detection from strictly fault tolerant approaches.

12.6.4 Distributed Reliability Protocols

Similar to local reliability protocols, the distributed versions aim to maintain the atomicity and durability of distributed transactions that execute over a number of databases. To facilitate the description of the distributed reliability protocols, we resort to a commonly used abstraction. We assume that at the originating site of a transaction there is a process that executes its operations. This process is called the *coordinator*. The coordinator communicates with the participant processes at the other sites which assist in the execution of the transaction's operations.

Components of Distributed Reliability Protocols

The reliability techniques in distributed database systems consist of commit and recovery protocols. Recall from the preceding section that the commit and recovery protocols specify how the commit and the recover commands are executed. Both of these commands need to be executed differently in a distributed DBMS than in a centralized DBMS. The primary requirement of commit protocols is that they maintain the atomicity of distributed transactions. This means that even though the execution of the distributed transaction involves multiple sites, some of which might fail while executing,

the effects of the transaction on the distributed database are all-or-nothing. This is called as *atomic commitment*. Independent recovery protocols determine how to terminate a transaction that was executing at the time of a failure without having to consult any other site. Existence of such protocols would reduce the number of messages that need to be exchanged during recovery.

Two-Phase Commit Protocol (2PC)

It is a simple and elegant protocol that ensures the atomic commitment of distributed transactions. It extends the effects of local atomic actions of distributed transactions by insisting that all sites involved in the execution of a distributed transaction agree to commit the transaction before its effects are made permanent. A brief description of the 2PC protocol that does not consider failures is as follows. Initially, the coordinator writes a begin commit record in its log, sends a “prepare” message, it checks if it could commit the transaction.

Another alternative is linear 2PC (as shown in Fig. 12.13), where participants can communicate each other. There is an ordering between the sites in the system for the purposes of communication. Let us assume that the ordering among the sites that participate in the execution of a transaction is $1, \dots, N$, where the coordinator is the first one in the order. The 2PC protocol is implemented by a forward communication from the coordinator (number 1) to N , during which the first phase is completed, and by a backward communication from N to the coordinator, during which the second phase is completed.

The coordinator sends the “prepare” message to participant 2. If participant 2 is not ready to commit transaction, it sends a “vote-abort” message (VA) to participant 3 and the transaction is aborted at this point. If, on the other hand, participant 2 agrees to commit transaction, it sends a “vote-commit” message (VC) to participant 3 and enters the READY state. This process continues until a vote-commit message reaches N . This is the end of the first phase. If N decides to commit, it sends back to $N-1$ “global-commit” (GC); otherwise, it sends a “global-abort” message (GA). Accordingly, the participants enter the appropriate state (COMMIT or ABORT) and propagate the message back to the coordinator.

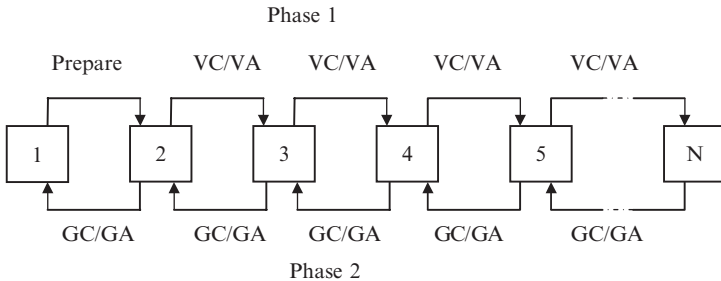


Fig. 12.13. Linear 2PC communication structure

Architectural Considerations

Here the protocols are implemented within the framework of our architectural model. This involves specification of the interface between the concurrency control algorithms and the reliability protocols. It is quite difficult to specify precisely the execution of these commands. The difficulty is twofold. First, a significantly more detailed model of the architecture than the one we have presented needs to be considered for correct implementation of these commands. Second, the overall scheme of implementation is quite dependent on the recovery procedures that the local recovery manager implements.

One possible implementation of the commit protocols within our architectural model is to perform both the coordinator and participant algorithms within the transaction managers at each site. This provides some uniformity in executing the distribution commit operations. However, it entails unnecessary communication between the participation transaction manager and its scheduler; this is because the scheduler has to decide whether a transaction can be committed or aborted.

Storing the commit protocol records in the database log maintained by the LRM and the buffer manager requires some changes to the LRM algorithms. This is the third architectural issue we address. Unfortunately, these changes are dependent on the type of the algorithm that the LRM uses. The LRM has to determine whether the failed site is the host of the coordinator or of a participant. This information can be stored together with the begin transaction record. The LRM then has to search for the last record written in the log record during execution of the commit protocol.

12.7 Parallel Database

The distribution database implies a number of computers connected by a wide area or a local area network. The increasing use of powerful personal computers, work stations, and parallel computers in distributed systems has a major impact on distributed database technology. The integration of workstations in a distributed environment enables a more efficient function distribution in which application programs run on workstations, called *application servers*, while database functions are handled by dedicated computers, called *database servers*.

A parallel computer, or multiprocessor, is itself a distributed system made of a number of nodes connected by a fast network within a cabinet. Distributed database technology can be naturally revised and extended to implement parallel database systems. It exploits the parallelism data management in order to deliver high performance and high availability. Database serves at much lower price than equivalent main frame computers.

12.7.1 Database Server and Distributed Databases

The centralized server approach enables distributed application to access a single database server efficiently. So, it is often a cost-effective alternative to distributed database whereby all the difficult problems are distributed database management that disappears at the local database server level. The addition of new application server in a local network is technically easy but may require the expansion of database server's processing power and storage capacity. Furthermore, the access to a single data server from geographically distant application servers is inefficient because communication over a wide area network is relatively slow.

The natural solution to these problems is to combine the database server, and distributed database technologies are to be termed distributed database server organization. Figure 12.14 shows the example of this organization, in which application servers and database servers are extended with distributed DBMS component. The distributed database server organization can accommodate a large variety of configurations, each being application dependent. Consider a geographically distributed database whose sites are connected by a wide area network, each site can consist of a single database server connected by a local network to a cluster of workstations. Any workstation could access the data at any database server through either the local network.

In the distributed server organization, each database server is fully dedicated to distributed and centralized database management. The first solution to improve performance is to implement the DBMS and distributed DBMS

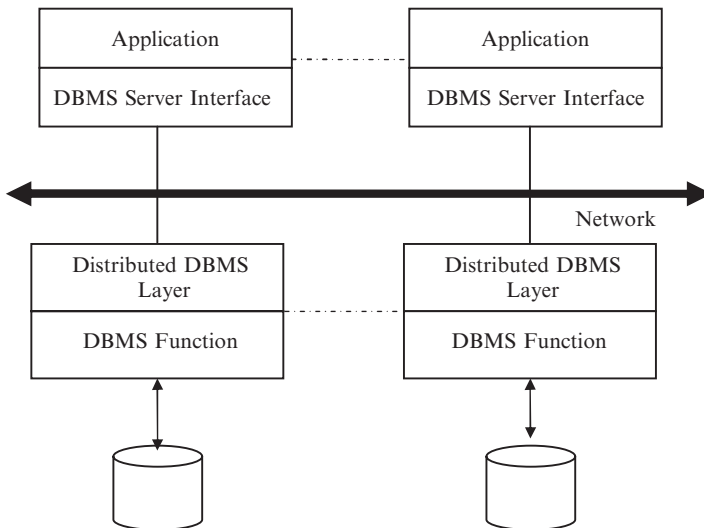


Fig. 12.14. Distributed database server

modules on top of a distributed database operating system running on a traditional computer. Another solution is to go one step further and use a parallel database system.

Parallel processing exploits multiprocessor computers to run application programs by using several processors cooperatively, in order to improve performance. Its prominent use has long been in scientific computing by improving the response time of numerical applications. Recent development in both the general purpose MIMD parallel computers using standard microprocessors and parallel programming techniques has enabled parallel processing to increase performance and availability.

The problem faced by conventional database management has long been known as "I/O bottleneck," induced by high disk access time with respect to main memory access time (typically hundreds of thousands times faster). Initially, DBM designers tackled this problem by introducing the data filtering devices within the disk. They too failed due to poor price/performance when compared to the software solution which can easily benefit from hardware progress in silicon technology.

An important result of DBM research is in the general solution of the I/O bottleneck. If we store a database of size D on a single disk with throughput T , the system throughput is bounded by T . On the contrary, if we partition the database across n disks, each with capacity D/n and throughput T' (nearly equivalent to T), we get an ideal throughput of $n * T'$ which can be better consumed by multiple processors. Note that main memory database system solution which tries to maintain the database in main memory is complimentary rather than alternative.

Parallel database system designers strived to develop software-oriented solutions in order to exploit multiprocessor hardware. The objective of parallel database system can be achieved by extending distributed database technology.

Parallel database software must effectively deploy the system's processing power to handle diverse applications, online transaction processing (OLTP) applications, decision support system (DSS) applications, as well as a mixed OLTP and DSS workload. OLTP applications are characterized by short transactions, which have low CPU and I/O usage. DSS applications are characterized by long transactions, with high CPU and I/O usage.

Parallel database software is often specialized usually to serve as query processors. Since they are designed to serve a single function, however, specialized servers do not provide a common foundation for integrated operations. These include online decision support, batch reporting, data warehousing, OLTP, distributed operations, and high availability systems. Specialized servers have been used most successfully in the area of very large databases.

Consider the versatile parallel software should offer excellent price/performance on open systems hardware, and be designed to serve a wide variety of enterprise computing needs. Features such as online backup, data replication, portability, interoperability, and support for a wide variety of client tools can

enable a parallel to support application integration, distributed operations, and mixed application workloads.

There are a number of hardware architectures allow multiple computers to share access to data, software, or peripheral devices. A parallel database is designed to take advantage of such architectures by running multiple instances, which share a single physical database. In appropriate applications, a parallel server can allow access to single database by users on multiple machines with increased performance. A parallel server processes transaction in parallel by servicing a stream of transactions using multiple CPUs on different nodes, and each CPU processes an entire transaction. This is an efficient approach because many applications consist of online insert and update transactions that tend to have short data access requirements.

12.7.2 Main Components of Parallel Processing

The main components of parallel processing are speedup and scale-up, synchronization, locking, and messaging.

Speedup and Scale-up

Speedup is the extent to which more hardware can perform the same task in less time than the original system. With added hardware, speedup holds the task constant and measures the time saved. With good speedup, additional processors reduce system response time. You can measure speedup by using the following formulae:

$$\text{Speedup} = \frac{\text{Original Processing Time}}{\text{Parallel Processing Time}}$$

The original processing time is the elapsed time spent by a small system on the given task and parallel processing time is the elapsed time spent by a larger, parallel system on the given task. For example, if the original system took 100s to perform a task, and two parallel systems took 50s, then the value of speedup would be equal to 2.

Scale-up is the ability of a system n times larger to perform a job n times larger, in the same period as the original system. With good scale-up, if transaction volumes grow, you can keep response time constant by adding hardware resources such as CPUs. We can measure the scale-up by using the formulae:

$$\text{Scale-up} = \frac{\text{Parallel Processing Volume}}{\text{Original Processing Volume}}$$

The original processing volume is the transaction volume processed in a given amount of time on a small system. The parallel processing volume is the transaction volume processed in a given amount of time on a parallel system. If the value of scale-up is 2, then it indicates the ideal of linear scale-up, i.e., twice as much as the hardware can process twice the data volume in the same amount of time.

Synchronization

Coordination of concurrent tasks is called *synchronization*. Synchronization is necessary for correctness. The key to successful parallel processing is to divide up tasks so that very little synchronization is necessary. If the synchronization necessary is less, then the speedup and scale-up are better.

In parallel processing between nodes, a high-speed communication networks are required among the parallel processors. The overhead of this synchronization can be very expensive if a great deal of internode communication is necessary. For parallel processing within a node, messaging is not required instead a shared-memory can be used. A task, in fact, may require multiple messages. If tasks must continually wait to synchronize, then several messages may be needed per task. In most database management systems, messaging and locking between nodes are handled by the distributed lock manager (DLM).

The amount of synchronization depends on the amount of resources and the number of users and tasks working on the resources. Little synchronization may be needed to coordinate a small number of concurrent tasks, but lots of synchronization may be necessary to coordinate many concurrent tasks.

A great deal of time spent in synchronization indicates high contention for resources. Too much time spent in synchronization can diminish the benefits of parallel processing. With less time spent in synchronization, better speedup and scale-up can be achieved.

Locking

Locks are fundamentally a way of synchronizing tasks. Many different locking mechanisms are necessary to enable the synchronization of tasks required by parallel processing. External locking facilities as well as mechanisms internal to the database are necessary. A DLM is the external locking facility used by many parallel databases. A DLM is an operating software, which coordinates resource sharing between nodes running a parallel server.

The instances of a parallel server use the DLM to communicate with each other and coordinate the modification of database resources. Each node operates independently of other nodes, except when contending for the same source. The DLM allows applications to synchronize access to resource such as data, software, and peripheral devices, so that concurrent requests for the same resource are coordinated between applications running on different nodes. The DLM performs the following services for applications:

1. Keeps track of the current ownership of a resource.
2. Accepts lock requests for resources from application process.
3. Notifies the requesting process when a lock on a resource is available.
4. Gets access to a resource for a process.

Messaging

Parallel processing requires fast and efficient communication between nodes. So, a system with high bandwidth and low latency, which efficiently communicates with the DLM, is used. Bandwidth is the total size of messages, which can be sent per second. Latency is the time required to locate the first bit or character in a storage location, expressed as access time minus word time. It is the time it takes place a message on the network. Latency thus indicates the number of messages, which can be put on the network per second.

A communication network with high bandwidth is like a wide highway with many lanes to accommodate heavy traffic. The number of lanes affects the speed at which the traffic can move. A network with low latency is like a highway with an entrance ramp, which permits vehicles to enter without delay. Massively parallel processing (MPP) is a parallel computing architecture that uses hundreds and thousands of processors. In clustering we use two or more systems that work together. The difference between MPP and clusters is that the MPP uses many more processors than clustering. MPP systems characteristically use networks with high bandwidth and low latency. Clusters use Ethernet connections with relatively low bandwidth and high latency.

12.7.3 Functional Aspects

A parallel database system acts as a database server for multiple application server in the common client server organization in computer network. This system supports the database functions and the client server interface and possibly general purpose functions. A parallel database system should provide the following advantages:

- *High performance.* This can be obtained through several complimentary solutions such as database-oriented operating system support, parallelism, optimization, and load balancing. Having the operating system constrained and aware of the specific database requirements simplifies the implementation of low-level database function and therefore decreasing the cost. Parallelism can increase throughput, using interquery parallelism, and decrease transaction response time, intraquery parallelism. However, decreasing the response time of complex time query through large-scale parallelism may well increase in total time and hurt throughput as a side effect. Therefore, it is crucial to optimize and parallelize queries in order to minimize the overhead of parallelism.
- *High availability.* Since the parallel database system consists of many similar components, it can exploit data replication to increase database availability. In a highly parallel system with many small disks, the probability of a disk failure at any time can be higher. Therefore, it is essential that the disk failure does not imbalance the load.

- *Extensibility.* In a parallel database, accommodating increasing database size or increasing performance demands should be easier. It is the ability of smooth expansion of the system by adding processing and storage power to the system. Basically, the parallel database system has two advantages:
 1. Linear scale-up
 2. Linear speedup

General architecture of parallel database system. The general architecture of parallel database system is shown in Fig. 12.15. Depending on the architecture the processor can support all of the subsystems.

- *Session manager.* It plays the role of a transaction monitor, providing the support for client interactions with the server. In particular, it performs the connections and disconnections between the client processes and the two other subsystems. Therefore, it initiates and closes user sessions. In case of OLTP sessions, the session manager is able to trigger the execution of preloaded transaction code within data manager modules.
- *Request manager.* It receives client requests related to query compilation and execution. It can access the database directory which holds all meta-information about data and programs. The directory itself should be managed as a database in the server. Depending on the request, it activates the various compilation places, triggers query execution, and returns the results as well as error quotes to the client application. To speedup the

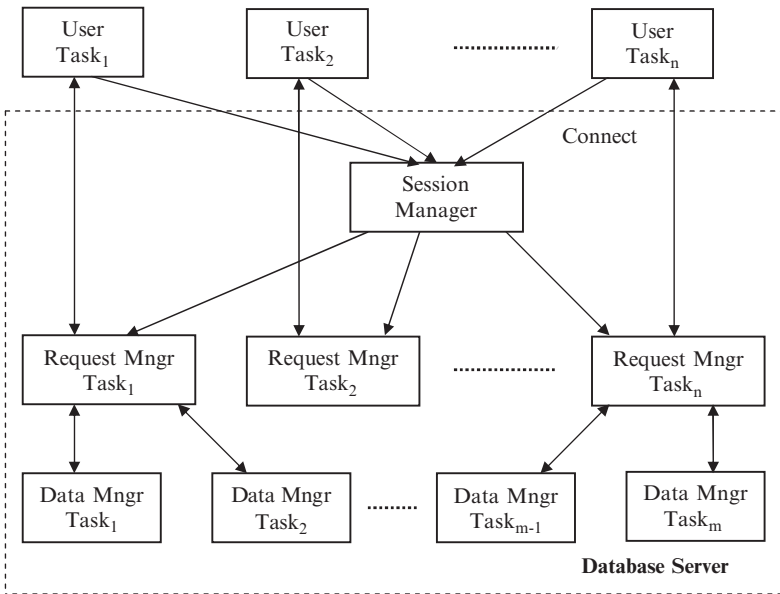


Fig. 12.15. General architecture of parallel database system

query execution, it may optimize and parallelize the query at the compile time.

- *Data manager.* It provides all the low-level functions needed to run compile queries in parallel. If the request manager is able to compile data flow control, then synchronization and communication among data manager modules are possible; otherwise, transaction control and synchronization must be done by a request manager module.

12.7.4 Various Parallel System Architectures

A parallel system represents a compromise in design choices in order to provide the advantages with better cost and performance. This architecture ranges between two extremes, a shared-memory and shared-nothing architecture, and useful intermediate point is the shared-disk architecture. More recently, hybrid architectures such as hierarchical architecture try to combine the benefits of shared-memory and shared-nothing.

Shared-Memory Architecture

In the shared-memory approach (shown in Fig. 12.16), any processor has access to any memory module or disk unit through a fast interconnect.

Most shared-memory commercial products today can exploit interquery parallelism to provide high transaction throughput and intraquery parallelism can reduce response time of decision support queries. This memory has two advantages namely, simplicity and load balancing. Since meta-information and control information can be shared by all process, writing database software is not very different than for a single processor computer and also this memory has three problems namely, cost, imitated extensibility, and low availability.

Shared-Disk Architecture

In this approach any processor has access to any disk unit through the interconnect but exclusive access to its main memory as shown in Fig. 12.17. Then, each processor can access database pages on the shared-disk and copy them

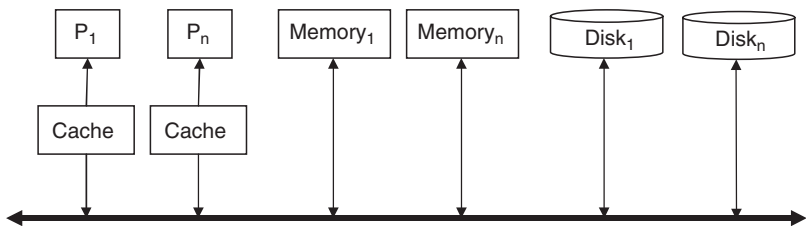


Fig. 12.16. Shared-memory architecture

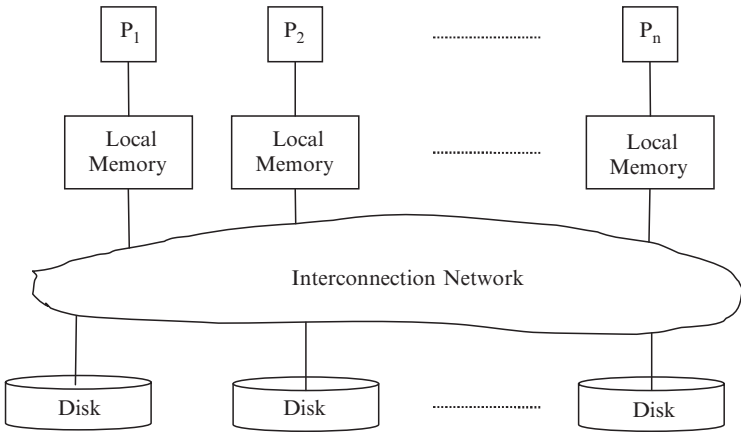


Fig. 12.17. Shared-disk architecture

into its own cache. To avoid conflicting accesses to the same pages, global locking and protocol for the maintenance of cache coherency are needed.

This disk has number of advantages namely, cost, extensibility, load balancing, availability, and easy migration from uniprocessor system. The cost of interconnect is significantly less than with shared-memory, since standard bus technology may be used. For a given processor has enough cache memory, interference on the shared-disk can be minimized. Thus extensibility can be better since memory faults can be isolated from other processor memory nodes, availability can be higher.

This architecture suffers from higher complexity and potential performance problems. It requires distributed database protocols (distributed locking and two-phase commit).

Shared-Nothing Architecture

In this approach, each processor has exclusive access to its main memory and disk unit as shown in Fig. 12.18. Then, each node can be viewed as a local site in a distributed database system. Therefore, most solution designed for distributed database, such as database fragmentation, distributed transaction management, and distributed query processing, may be reused.

This approach is also more complex than shared-memory. Higher complexity is due to the necessary implementation of distributed database function assuming large number of nodes. In addition, load balancing is more difficult to achieve because it relies on the effectiveness of database partitioning for the query workloads.

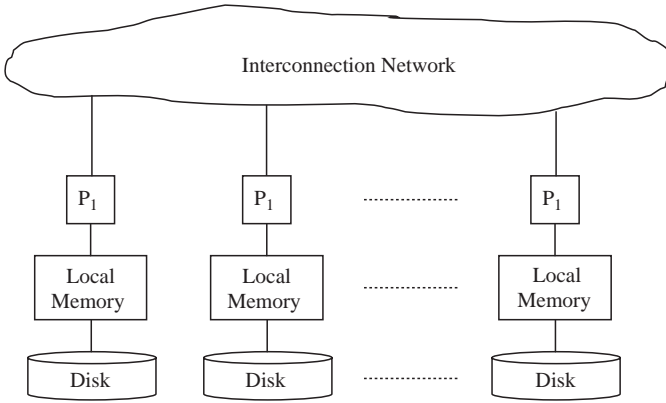


Fig. 12.18. Shared-nothing architecture

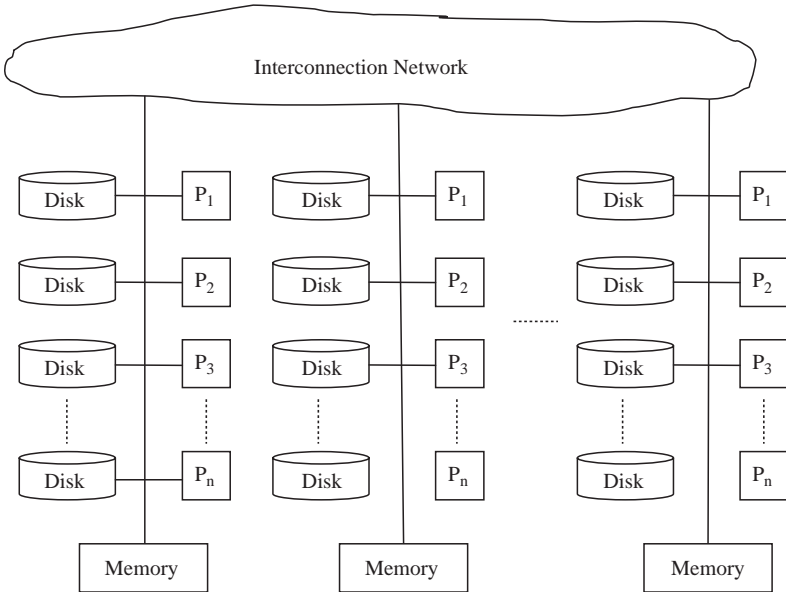


Fig. 12.19. Hierarchical architecture

Hierarchical Architecture

This architecture (or cluster architecture) is a combination of shared-nothing and shared-memory architecture. This idea is to build a shared-nothing machines whose nodes are shared-memory. This architecture is shown in Fig. 12.19.

The advantages of this architecture are evident. It combines flexibility and performance of shared-memory with high extensibility of shared-nothing. In each shared-memory node (S-M node), communication is done efficiently

using the shared-memory, thus increasing performance. Finally load balancing is eased by shared-memory component of this architecture.

12.7.5 Parallel DBMS Techniques

Implementation of parallel database systems naturally relies on distribution database techniques. The critical issues for such architectures are data placement, query parallelism, parallel data processing, and parallel query optimization.

Data Placement

Data placement in parallel database system exhibits similarities with data fragmentation in distributed databases. An obvious similarity is that fragmentation can be used to increase parallelism. There are two important differences with distributed databases from parallel database approach. First, there is no need to maximize local processing (at each node) since users are not associated with particular nodes. Second, load balancing is much more difficult to achieve in the presence of a large number of nodes. The main problem is to avoid resource contention, which may result in thrashing the entire system. Since programs are executed where the data reside, data placement is a critical performance issue.

Data placement must be done to maximize system performance, which can combine the total amount of work done by the system and the response time of individual queries. An alternative solution to data placement is full partitioning used in the DBC/1012, GAMMA, and nonstop SQL. There are three basic strategies for data partitioning: round-robin, hash, and range partitioning as shown in Fig. 12.20:

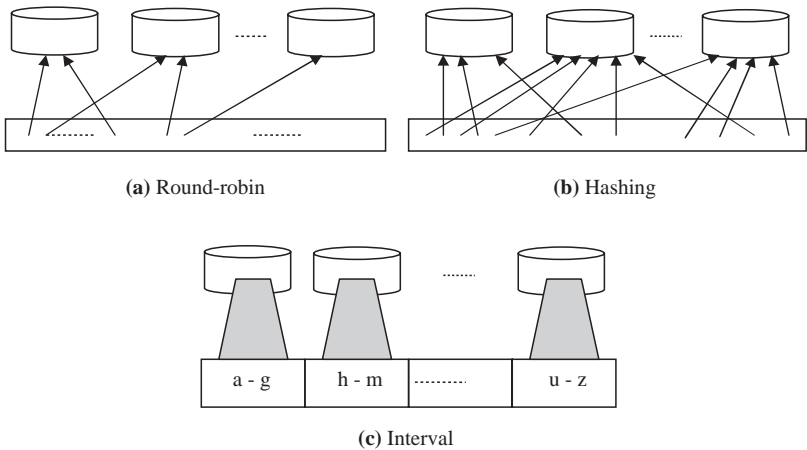


Fig. 12.20. Different partitioning schemes

1. *Round-robin partitioning.* It is the simplest strategy, it ensures uniform data distribution. With n partitions, the i th tuple in insertion order is assigned to partition $(I \bmod n)$. This strategy enables the sequential access to a relation to be done in parallel. Direct access of individual tuples requires accessing the entire relation.
2. *Hash partitioning.* It applies a hash function to some attribute which yields the partition number. This strategy allows exact match queries on the selection attribute to be processed by exactly one node and all other queries to be processed by all the nodes in parallel.
3. *Range partitioning.* It distributes tuples based on the value intervals (ranges) of some attribute. In addition to supporting exact match queries as with hashing, it is well suited for range queries. This partitioning results in high variation in partition size.

The performance of full partitioning is compared to that of the clustering the relations on a single disk. The results indicate that for a wide variety of multiuser workloads, partitioning is consistently better. However, clustering may dominate in processing the complex queries. So, the solution to data placement is variable partitioning. It is defined as the degree of partitioning, or, the number of nodes over which a relation is fragmented, is a function of the size and access frequency of the relation. This strategy is much more involved than either clustering or full partitioning because changes in data distribution may result in reorganization.

In a highly parallel system with variable partitioning, periodic reorganizations for load balancing are essential and should be frequent unless the workload is fairly static and experiences only a few updates. Such reorganizations should remain transparent to compiled programs that run on the database server.

Query Parallelism

- *Intraoperator parallelism.* Intraoperator parallelism is based on the decomposition of one operator in a set of independent suboperators, called *operator instances*. This decomposition is done using static and/or dynamic partitioning of relations. Each operator instance will then process one relation partition also called *bucket*. The operator decomposition frequently benefits from the initial partitioning of the data. The select operator can be directly decomposed into several select operators, each on a different partition and no redistribution is required. Figure 12.21 shows that if the relation is partitioned on the select attribute, partitioning properties can be used to eliminate some select instances.

The partitioning function is independent of local algorithm which is used to join operator. For instance, a hash join using a hash partitioning needs two hash functions.

partitioning properties can be used to eliminate some select instances.

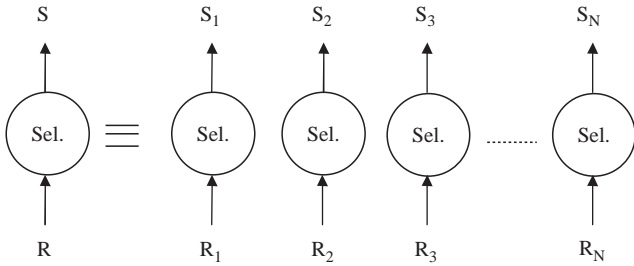


Fig. 12.21. Intraoperator parallelism

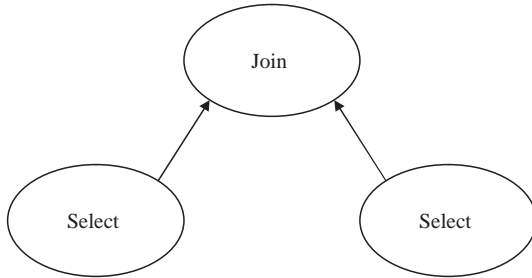


Fig. 12.22. Interoperator parallelism

- *Interoperator parallelism.* Two forms of interoperator parallelism can be exploited. With pipeline parallelism, several operators with a producer-consumer link are executed in parallel. The select operator shown in Fig. 12.22 will be executed in parallel with the subsequent operator. The advantage of such execution is that the intermediate result is not materialized, thus saving memory and disk accesses.

Independent parallelism is achieved when there is no dependency between the operators executed in parallel. This type of parallelism is very attractive because there is no interference between the processors. However, it is only possible for bushy execution and may consume more resources.

Parallel Data Processing

Partitioned data placement is the basis for the parallel execution of database queries. For efficient processing of database operators and database queries in the design of parallel algorithms, partitioned data placement is used. This task is very difficult because a good trade-off between parallelism and communication must be reached. Parallel algorithms for relational algebra operators are the building blocks necessary for parallel query processing. The parallel

processing of join is significantly more involved than that of select. The distributed join algorithms designed for high-speed networks can be applied successfully in a partitioned database context. However, the availability of a global index at run time provides more opportunities for efficient parallel execution.

Parallel Query Optimization

Parallel query optimization takes the advantage of both interoperator and intraoperator parallelism. This can be achieved by some of the techniques used for distributed DBMSs. Parallel query optimization refers to the process of producing an execution plan for a given query that minimizes an objective cost function. A query optimizer contains three components, they are search space, cost model, and search strategy.

The search space is the set of alternative execution plans to represent the input query. These plans give same result but differ on the execution order of operators and the way these operators are implemented. The cost model predicts the cost of given execution plan. To be accurate, the cost model should have good knowledge about the parallel execution environment.

- *Search space.* Execution plans are abstracted, as usual, by means of operator trees, which define the order in which the operators are executed. Operator trees are enriched with annotations, which indicate additional execution aspects, such as the algorithm of each operator. An important execution aspect to be reflected by annotations is the fact that two subsequent operators can be executed in pipeline. Pipeline and store annotations constrain the scheduling of execution of execution plans. They split an operator tree into nonoverlapping subtrees, called *phases*. Pipelined operators are executed in same phase, whereas a storing indication establishes the boundary between one phase and the subsequent phase.
- *Cost model.* The optimizer cost model is responsible for estimating the cost of a given execution plan. It is viewed by two parts: architecture dependent and architecture independent. The architecture-independent part is constituted by the cost functions for operator algorithms. If we ignore the concurrency issues, only the cost functions for data repartitioning a relation's tuples in a shared-nothing system imply transfers of data across the interconnect, whereas it reduces to hashing in shared-memory systems. Memory consumption in the shared-nothing case is complicated by interoperator parallelism.
- *Search strategy.* This does not need to be different from either centralized or distributed query optimization. However, the search space tends to be much larger because there are more alternative parallel execution plans. Thus, randomized search strategies generally outperform deterministic strategies in parallel query optimization.

Benefits of Parallel Processing and Parallel Database

Parallel processing can benefit certain kinds of application by providing enhanced throughput (scale-up) and improved response time (speedup). Improved response time can be achieved either by breaking up a larger task into smaller components or by reducing wait time. Parallel database technology can benefit in numerous applications by:

- Higher performance
- Higher availability
- Greater flexibility
- More users

Summary

In this chapter, we discussed the distributed databases and parallel databases architecture and some of the basic techniques involved. First, we have seen that distributed database is having a number of advantages over centralized systems. We then discussed the various fragmentation involved in distributed databases. In semantic data control, we discussed the view management, security control, and semantic integrity control. The two main issues for efficiently performing data control are the definition and storage of rules. Concurrency control provides the isolation and consistency properties of transaction, and then we discussed various algorithms in locking. Finally, in distributed database we discussed reliability and availability. The various failures in distributed database are listed and the different protocols used to overcome the failures are discussed. In parallel database, better performance and high availability are achieved. Various architectures and techniques of parallel database are discussed and finally the benefits are also listed.

Review Questions

12.1. What is meant by distributed databases?

A distributed database is a collection of data which belong logically to the same system but are spread over the sites of a computer network. This definition emphasizes two equally important aspects of a distributed database as follows:

- *Distribution.* The fact that the data are not resident at the same site (processor), so that we can distinguish a distributed database from a single, centralized database.

- *Logical correlation.* The fact that the data have some properties which tie them together, so that we can distinguish a distributed database from a set of local databases or files which are resident at different sites of a computer network.

12.2. What are special features of distributed database over the centralized database?

The features, which characterize the traditional database approach, are centralized control, data independence, and reduction of redundancy, complex physical structures for efficient access, integrity, recovery, concurrency control, privacy, and security.

12.3. Explain primary horizontal fragmentation?

The primary horizontal fragments are defined using selections on global relations. The correctness of primary fragmentation requires that each tuple of the global relation be selected in one and only one fragment. Thus, determining the primary fragmentation of a global relation requires determining a set of disjoint and complete selection prediction. The property that we require for each fragment is that the elements of them must be referenced homogeneously by all the applications.

12.4. What are the methods to prevent unauthorized users in remote accessing in distributed database?

Two solutions are possible in preventing unauthorized users in remote accessing as follows:

1. The information for authenticating users (user name and password) is replicated at all sites in the catalog. Local programs, initiated at a remote site, must also indicate the user name and password.
2. All sites of the distributed DBMS identify and authenticate themselves similarly to the way users do. Intersite communication is thus protected by the use of the site password.

12.5. Explain the concurrency control mechanisms?

The concurrency control mechanism is grouped into two broad classes as pessimistic control methods and optimistic control methods. Pessimistic algorithm synchronizes the concurrent execution of transaction early in their execution life cycles, whereas optimistic algorithms delay the synchronization of transactions until their termination. The pessimistic group consists of locking-based algorithm, ordering-based algorithm, and hybrid algorithm. The optimistic group can, similarly, be classified as locking based or timestamp ordering based.

12.6. Explain distributed 2PL algorithm?

It expects the availability of lock managers at each site. If the database is not replicated, distributed 2PL degenerates into the primary copy 2PL algorithm. If data are replicated, the transaction implements the ROWA replica control protocol. The communication between cooperating sites that execute a transaction according to the distributed 2PL.

12.7. Define the terms reliability and availability?

Reliability refers to the probability that the system under consideration does not experience many failures in the given time interval. It is typically used to describe system that cannot be repaired or where the operation of the system is so critical that no down time for repair can be tolerated. Availability refers to the probability that the system is operational according to its specification at a given point in time t . A number of failures may have occurred prior to time t , but if they have all been repaired, the system is available at time t . It is apparent that availability refers to system that can be repaired. It can be used as some measure of “goodness” for those systems that can be repaired and which can be out of service for short periods of time during repair.

12.8. What are the reasons for failure in distributed DBMS?

Designing a reliable system that can record for failures requires identifying the types of failures with which the system has to deal. It indicates that the database recovery manager has to deal four types of failures namely, transaction failure (abort), site failure (system), media failure (disk), and communication line failure.

12.9. What is meant by parallel processing and what are the benefits of parallel processing?

Parallel processing divides a complex task into many smaller tasks, and executes the smaller tasks simultaneously in several tasks. Thus the complex task is completed with better performance and also quickly. Parallel processing can benefit certain kinds of application by providing enhanced throughput (scale-up) and improved response time (speedup).

12.10. Explain hierarchical architecture in parallel databases?

This architecture (or cluster architecture) is a combination of shared-nothing and shared-memory architecture. This idea is to build a shared-nothing machine whose nodes are shared-memory. It combines flexibility and performance of shared-memory with high extensibility of shared-nothing. In each shared-memory node (S-M node), communication is done efficiently using the shared-memory, thus increasing performance. Finally load balancing is eased by shared memory component of this architecture.

12.11. Need for parallel databases?

Parallel processing exploits multiprocessor computers to run application programs by using several processors cooperatively, in order to improve performance. Its prominent use has long been in scientific computing by improving the response time of numerical applications. Recent development in both the general purpose MIMD parallel computers using standard microprocessors and parallel programming techniques has enabled parallel processing to increase performance and availability.

12.12. What are the main components of parallel processing? Explain speedup?

The main components of parallel processing are speedup and scale-up, synchronization, locking, and messaging. Speedup is the extent to which more hardware can perform the same task in less time than the original system. With added hardware, speedup holds the task constant and measures the time saved. With good speedup, additional processors reduce system response time. You can measure speedup by using the following formulae:

$$\text{Speedup} = \frac{\text{Original Processing Time}}{\text{Parallel Processing Time}}$$

Recent Challenges in DBMS

Learning Objectives. This chapter provides an overview of recent challenges database management system (DBMS), which includes concepts, related to genome database management system, spatial database management system, multimedia database, mobile database management system, and XML. In this chapter, the basic idea of genome, genetic code, genome directory system project, concept of mobile database, and mobile database architecture are discussed. In spatial database management system, spatial data types (SDTs), implementation of spatial database management system and in multimedia database, multimedia data model concept, issues, and architectures are described. The basic concept of XML, XML family, XML, and database applications are also discussed. After completing this chapter the reader should be familiar with the following concepts:

- Need for genome database
- Building block of deoxyribonucleic acid (DNA)
- Genetic Code, Genome Map
- Mobile Database
- Concept of Mobile Database Center
- Distributed Database for Mobile
- Spatial Database Management System
- Spatial Data Type
- Spatial Database Modeling
- Spatial DBMS Implementation
- Multimedia Data Model
- Architecture of Multimedia System
- Characterization of Multimedia Data
- Multimedia Database Management System Development
- Issues in Multimedia DBMS
- Basic concepts of XML
- XML Family
- XML and Database Applications

13.1 Genome Databases

13.1.1 Introduction

Genomic databases stores information on DNA sequences. Although other sources of information (e.g., protein sequences and proteins structures) are important in the area; tools for generating large quantities of information have only recently been developed. Therefore, databases organized around DNA had a “head start” and still represents the largest single source of information.

13.1.2 Basic Idea of Genome

The cell is the fundamental unit of life. All living organisms are made of cells. There are about 75–100 trillion cells in the human body. Nearly all cells of an organism contain the genome. An exception is the red blood cell which lacks DNA. In human beings and animals the genomes are often very long and are divided into packets called chromosomes. The number of chromosomes in human being is 46, while in mouse and dog the number of chromosomes are 40 and 78, respectively. Genomic information is encoded in the form of DNA inside the nuclei of cells. A DNA molecule is a long linear polymeric chain, composed of four types of subunits. Each subunit is called a base. The four bases in DNA are Adenine (A), Thymine (T), Guanine (G), and Cytosine (C). DNA occurs as a pair of strands. Bases pair up across the two strands. A always pairs with T and G always pairs with C. Hence, the two strands are called complementary.

A gene is a fundamental constituent of any living organism. Sequence of genes in a human body represents the signature of the person. The genes are portions of DNA. DNA consists of two strands or chains. Each of these chains is composed of phosphate and deoxyribose sugar molecules joined together by covalent bonds. A nitrogenous base is attached to each sugar molecules. There are four bases: Adenine (A), Cytosine(C), Guanine (G), and Thymine (T). In the human body there are approximately three billion such base pairs. The whole stretch of DNA is called genome of an organism. DNA is a linear chain of four different small molecules called nucleotide bases linked together. Genes are made up of linear chain of the bases, except different genes comprise of different sequence of the bases as different.

13.1.3 Building Block of DNA

The basic building blocks of DNA are sugar, phosphate, and base which together constitute nucleotide as shown in Fig. 13.1. The single-stranded and double-stranded DNA are shown in Figs. 13.2 and 13.3, respectively.

In 1953, James D. Watson and Francis Crick proposed the double helical structure of DNA through their landmark paper in the British journal

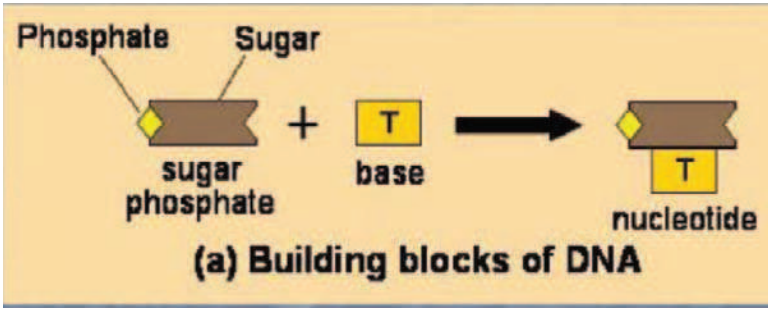


Fig. 13.1. Building blocks of DNA

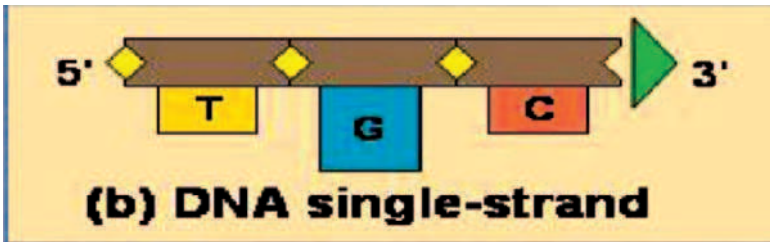


Fig. 13.2. DNA single-stranded



Fig. 13.3. Straightened out double-stranded DNA

“Nature.” For this discovery, they shared the 1962 Nobel Prize for physiology and medicine with Maurice Wilkins. The double helical structure of DNA is shown in Fig. 13.4.

Genes

Genes are regions in the genome that carry instructions to make proteins. Genes are inherited from parent to offspring, and thus are preserved across generations. Genes determine the traits of an organism.

Genes are divided into many fragments called exons. The exons are separated by noncoding regions called introns. This is shown in Fig. 13.5.

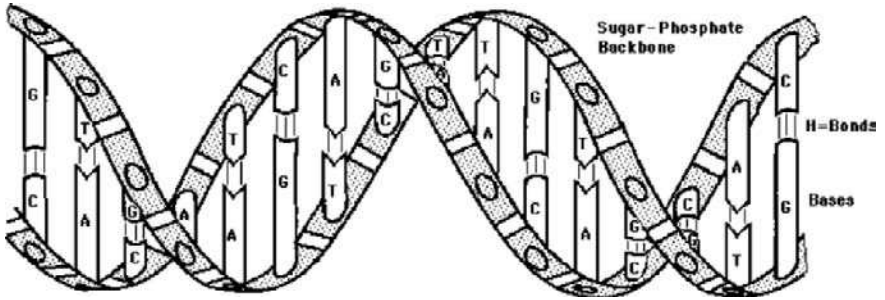


Fig. 13.4. Double helical structure of DNA

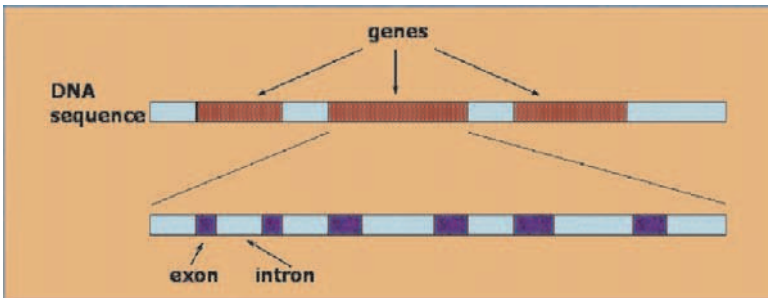


Fig. 13.5. Exons and introns in genes

13.1.4 Genetic Code

Genetic code is the rule by which genes code for proteins. There are groups of three bases called codons that code for the individual amino acids. The number of codons is greater than the number of amino acids, more than one codon can code for an amino acid. The genetic code is said to be degenerative. The genetic code is shown in Fig. 13.6.

A DNA sequence may be 40,000–100,000 base pairs long. In practice, such a long stretch of DNA is first broken up into 400–2,000 small fragments. Each such small fragment typically consists of approximately 1,000 base pairs. These fragment are sequenced experimentally and then reassembled together to reconstruct the original DNA sequence. Genes are encoded in these fragments of DNA. The huge volume of such data and their high dimensions make gene in expression data to be suitable candidates for the application of data mining functions like clustering, visualization, and string matching.

13.1.5 GDS (Genome Directory System) Project

Bioinformatics requires handling large volumes of data, involving natural interaction with information science. One needs to consider problem of data storage, analysis, and retrieval along with the computational modeling and

		SECOND POSITION OF CODON					
		T	C	A	G		
F I R S T	T	TTT Phe(F)	TCT Ser(S)	TAT Tyr(Y)	TGT Cys(C)	T C A G	T H I R D
		TTC Phe(F)	TCC Ser(S)	TAC Tyr(Y)	TGC Cys(C)		
		TTA Leu(L)	TCA Ser(S)	TAA (STOP)	TGA (STOP)		
		TTG Leu(L)	TCG Ser(S)	TAG (STOP)	TGG Trp(W)		
C	C	CTT Leu(L)	CCT Pro(P)	CCT Pro(P)	CGT Arg(R)	T C A	R D A
		CTC Leu(L)	CCC Pro(P)	CCC Pro(P)	CGC Arg(R)		
		CTA Leu(L)	CCA Pro(P)	CCA Pro(P)	CGA Arg(R)		

Fig. 13.6. Genetic code

simulation. Data mining, image processing, and visualization are the other important constituents required to help the user with a visual environment that facilitates high-dimensional data dependent on any parameter. Information theory-based loss less data compression techniques can play a vital role in management of this high volume of data.

We know that genome is the collection of all the genes within an organism. Genomic database is a database that houses the entire genomic sequence of an organism. These types of databases will typically have relatively few copies of any particular region of DNA, but their information spans the entire genome. A mutational database is a database that focuses on a narrow region of DNA by cataloging all of the known differences that have been found in that region. These databases will typically have many copies of the same region of DNA from many different individuals. They will often include comparative information for all the copies. One description of the difference between a genomic database and a mutational one is “a genomic database is a mile wide and an inch deep, while a mutational database is an inch wide and a mile deep.”

Genome Directory System is used to provide a distributed search and retrieval system for genome databases, federating and automating the finding and fetching of genome data across different formats and systems. The main objectives of the system are:

- Allow searching/retrieval of information spread across different database systems.

- Facilitate bulk retrieval of large data sets that are selected by customer needs through query, in a quick and easy manner.
- Development will be focused on life science genome databases but it will draw on and interoperate with general information technology standards and common practices.
- Include flexible adaptors for different data storage and management systems.
- Utilize existing standards, technologies, and available bioinformatics data systems while keeping the licensing as open as possible.
- Implement use of the life science identifier (LSID) for genome, gene, and related object naming and retrieval.

A large collaboration of model organism databases around the academic community is called the Generic Model Organism Database (GMOD). The purpose of GMOD is to develop reusable components suitable for creating new community databases of biology. The Genome Directory System (GDS) project has come about because of the need to query multiple organism databases without worrying about where the data reside or what format it exists in. Although GDS is being developed with model organism genome databases in mind it can also be used with other life science databases where a need for federated queries exist. This document will briefly outline the overall layout of the GDS system and describe each components purpose and technologies used at each level. An overview of the data flow for the GDS project is shown in Fig. 13.7.

Data Provider Layer

The Data Provider Layer is comprised of the components that hold the raw data and can utilize multiple systems for data access to each of the data types. The data types most commonly encountered in the life sciences include flat file, relational database management systems (RDBMS), and XML. In addition to existing in different data management systems, the format of the data itself can vary between data sources. To get around this, the Data Provider Layer also includes some external packages.

Data Directory Layer

This layer is the most important piece in the system as it bridges the disparate data sources and the client. The purpose of this layer is to accept queries from clients, manage the federation of the queries if needed, and know how and where to retrieve the information from the various data sources. The results from these queries will be returned in the form of data objects that can be further manipulated by the clients. The Data Directory Layer will be responsible for all communication going to the Data Provider Layer.

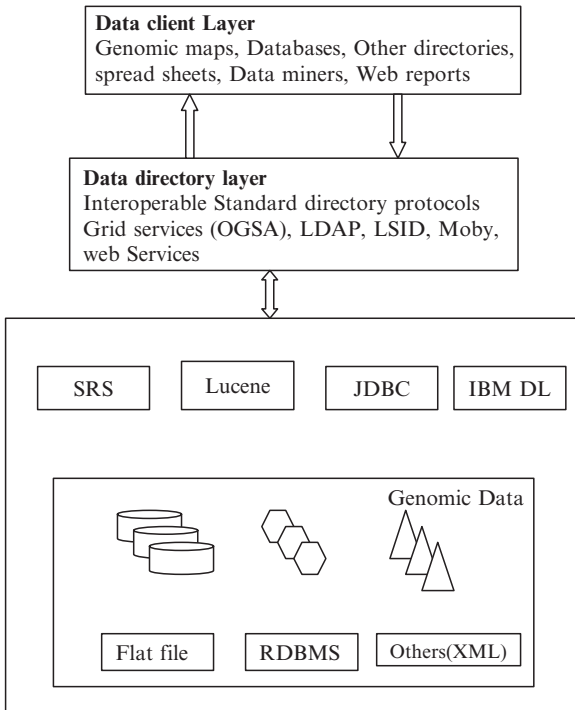


Fig. 13.7. An overview of the data flow for the GDS project

Client Layer

This layer provides the main interaction between the end user and the Genome Directory System. Its purpose is to interact with the bioinformaticians and/or bench scientists to allow them to issue queries for data retrieval. When the Data Directory Layer has processed the request it returns data objects in a form that the client can understand and process. There will be multiple points of entry into the Data Directory Layer so the type of clients should not be limited. It should work with web scripts, java clients other directory services, etc.

The molecular biology community faces an inundation of data. This implies that most of the contents of databanks will be recently determined data, and features, such as quality, will be characteristic of the newest methods of measurement. New experimental techniques will increase the amount and diversity of data; for example, the functional genomic, Proteome and expression profiles projects.

A genome is the entire genetic material of an organism, meaning the entire DNA that makes up the organism (which is replicated in every cell). The human genome is divided into 22 separate somatic (nonsex) chromosomes as

well as the X and Y chromosomes. Within the genome are genes – regions of chromosomes that serve as templates for the production of proteins. Thus genes code for proteins. A dysfunctional gene and/or gene product can have a harmful effect on the organism (e.g., Duchenne Muscular Dystrophy). However, the entire genome of higher organisms is not dedicated only to encoding for proteins. In fact, only a small portion (genes) is used as a template for protein production. Much of the intervening areas are still of use for analysis through the use of genetic markers. A genetic marker is a unique site within the genome that can be used to determine the “state” of a region of the genome, known as the “genotype.”

Genome, this is referred to as “linkage,” i.e., the disease gene is “linked” to a region (genetic marker) on a chromosome. Linkage analysis requires a set of genetically related subjects (a family), the phenotypes of the subjects with respect to a disease, and the genotypes of the subjects for a genetic marker. The likelihood that the data (family structure, phenotypes, and genotypes) correlates with a model of inheritance for the disease and with the actual inheritance patterns (observed by genotypes) provides a measure of the evidence for “linkage.” Thus, a linkage analysis calculates the likelihood that the disease-causing gene is located near a genetic marker based on the given set of subjects, phenotypes, and genotypes.

Geno Map

Geno Map is a suite of independent, yet inter-related tools primarily developed in Java. These tools manipulate data from a domain-specific networked database, allowing sharing of information among multiple distributed clients without replication and coherency problems. The main goal of Geno Map is to provide a portable, intuitive interface for managing the information associated with the gene location/discovery process.

The Geno Map system is designed to support a diverse collection of users in a wide-area network environment. The data objects being managed are of the most sensitive nature – usually identifying family relationships among members, some of who may carry stigmatizing genetic diseases. The Geno Map Database is an essential component and is used to form an administrative domain; i.e., only users belonging to that domain can access the data. Database access requires a separate level of authentication.

Geno Map is a large-scale, distributed, heterogeneous, client/server application to support the systematic exploration of the genome to narrow, and ultimately identify, the locus of a particular gene (or set of genes) involved in a disease or trait. In contrast to many applications developed in support of the Human Genome Project (HGP).

13.1.6 Conclusion

The conflicting requirements of security and heterogeneity are at the heart of the approach taken in Geno Map. The Database itself is physically partitionable across lab boundaries, and access to a database is encapsulated within well-defined APIs. Geno Map has been implemented primarily in Java with a socket-oriented, client/server design employing recent applet security features. Geno Map is currently being used in the collection of data for, and analysis of, a number of relatively small gene identification studies. It is also being used in one large genome-wide screening for the locus (loci) of the gene(s) involved in autism. The former shows its usefulness in supporting users who want to employ the analysis features of Geno Map, while not needing the large-scale data collection and management facilities, while the latter shows the usefulness in managing gene identification studies that would have been unmanageable without such a system.

13.2 Mobile Database

Recent advances in wireless networking technologies and the growing success of mobile computing devices are enabling new issues that are challenging to mobile database designers. One idea is disconnected database, where mobile hosts are strong connected, weak connected with or disconnected from fixed network. They hoard replicas before disconnection, read/write on the local replicas, and then synchronize updates when they get reconnected with the fixed network. There is no communication among mobile hosts. One idea is disconnected database, where mobile hosts are strong connected, weak connected with or disconnected from fixed network. They hoard replicas before disconnection, read/write on the local replicas, and then synchronize updates when they get reconnected with the fixed network. There is no communication among mobile hosts, where mobile hosts weak connected in pair-wise manner or disconnected with each other. They hoard replicas from their peers and synchronize updates upon connection. When disconnected, local data accesses are performed.

13.2.1 Concept of Mobile Database

A mobile database should be defined as the union of distributed database, disconnected database, ad hoc database and broadcast disks. The distributed database is treated as the home of mobile database, and the others deal with the access of mobile users. Figure 13.8 demonstrates the concept of mobile database.

Traditional database design is static and limits the flexibility of database applications, while mobility is changing the way we design databases and

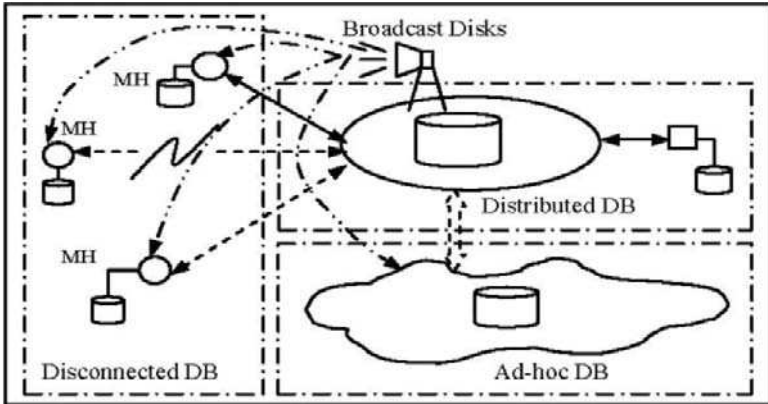


Fig. 13.8. Mobile database: a whole image

their DBMS. In mobile database everything is dynamic, varying from sporadic accesses by individual users to particular data to continuous access of a particular data by a large group of user. This is the case from disconnected database access to broadcast disks. Mobile hosts have to deal with planned or unexpected disconnections when they mobile; they are likely to have scarce resources such as low battery life, slow processor speed and limited memory; their applications are required to react to frequent changes in the environment such as new location, high variability of network bandwidth; their data interests are changing from time to time and from location to location; even data semantics in mobile hosts are varying according to data access patterns, connection duration and disconnection frequencies, etc. Data partition, location, and replication are always dynamic. All of these require a dynamic database design and reconfigure scheme.

13.2.2 General Block Diagram of Mobile Database Center

A client/server mobile database environment as illustrated in Fig. 13.9 consists of a central server database residing at a fixed location and one or more local databases on mobile clients.

The client software is used to access the database in very efficient manner. Many securities can be carried out in data connectivity. Data recovery algorithms can also be used. The very important concept in mobile database is data synchronization, in which client and server should be synchronized.

13.2.3 Mobile Database Architecture

The basic architecture for mobile database is shown in Fig. 13.10. Three important sections for mobile database are:

Fixed Hosts, perform the transaction and data management functions with the help of database servers.

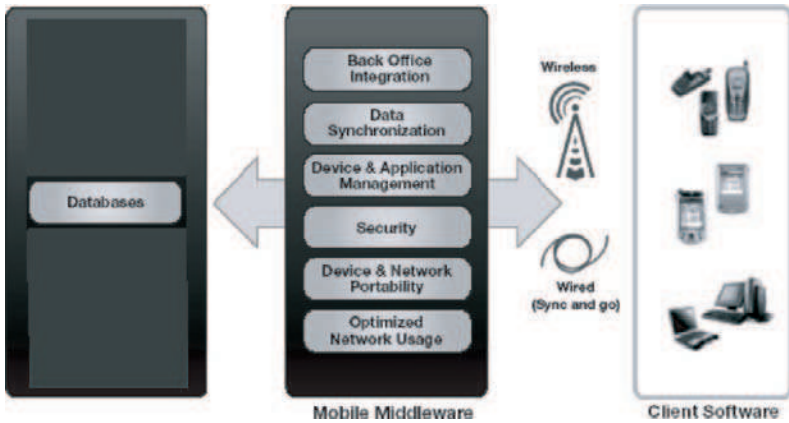


Fig. 13.9. General block diagram

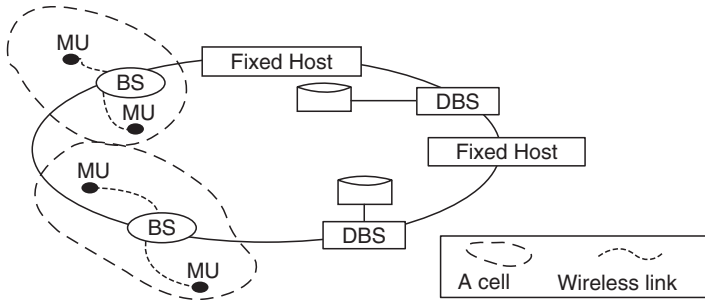


Fig. 13.10. Basic architecture of mobile database

Mobile Units, portable computers, move around a geographical region that is a collection of mobile cells:

- Mobile hosts retains network connection through the support of base stations
- Role of mobile hosts depend on the capacity

Base Stations

- Capturing mobility by hand-off processes
 - When a mobile unit leaves a mobile cell serviced by a base station, transfer the responsibility for mobile transaction and data support to the new base station.
 - Transparent processes?

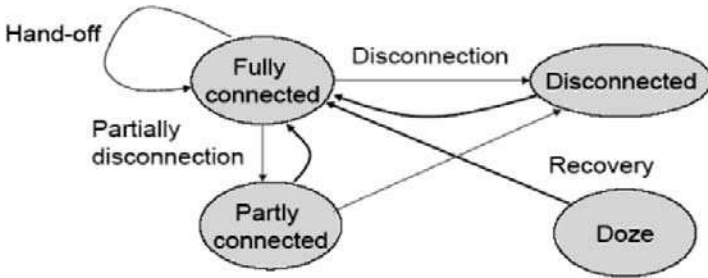


Fig. 13.11. Modes of operation

13.2.4 Modes of Operations of Mobile Database

The mobile network database can be operated in four different modes as shown in Fig. 13.11. They are:

- Fully connected
- Totally disconnected
- Partially connected
- Doze state

Several protocols are available for controlling the flow between these connections. They are:

- Disconnection
- Partially disconnection
- Recovery
- Hands-off

13.2.5 Mobile Database Management

Database management in mobile is most tedious process. The different data management available in mobile database is:

- Cache consistency
- Data replication
- Query processing

Cache Consistency

Caching of last recently used data will improve the overall performance of the system:

- Increases the level of data availability
- Cope with weak connection or disconnection

Data Replication

The data replication or data duplication will reduce redundancy and minimize the amount of memory space needed. It increases the level of data availability and performance.

Query Processing

The efficient processing of user queries is very important in data management. It is of two types:

- Involve only the context of the database
- Locate dependent query

13.2.6 Mobile Transaction Processing

A mobile transaction is a transaction where at least one mobile host is involved. Mobile database involves lot of transaction traffic. Efficient method should be employed for controlling data traffic. Some of the characteristics of mobile transaction processing are:

- Split computations among mobile hosts and stationary hosts
- Due to disconnection and mobility, mobile transactions shares their state and partial results to other transactions
- Mobile transactions require support from stationary hosts in computations and communications
- Movement of mobile hosts during the execution of mobile transactions
- Movement of transaction states and accessed data
- Long-lived transactions due to mobility and frequent disconnections
- Support concurrency, recovery, disconnection, etc.

Requirements for Mobile Transaction

The basic requirements of mobile transaction are summarized:

- Accommodate the limitations of mobile computing environments
- Minimize aborts due to disconnections
- Correctness of distributed transaction processing
- Minimize blocking of ongoing transactions
- Support local autonomy
- Disconnection processing
- Ability to distribute the transaction's processing
- Share the state and the partial results
- Capture the movement of mobile transactions
- Support long-lived transactions
- Support long disconnection periods
- Support partial failure and provide different recovery strategies

Dynamic Data/Currency Protocol for Mobile Database Design

Traditional database design is static and limits the flexibility of database applications, while mobility is changing the way we design databases and their DBMS. In mobile database everything is dynamic, varying from sporadic accesses by individual users to particular data to continuous access of a particular data by a large group of users. This is the case from disconnected database access to broadcast disks. Mobile hosts have to deal with planned or unexpected disconnections when they are mobile; they are likely to have scarce resources such as low battery life, slow processor speed, and limited memory; their applications are required to react to frequent changes in the environment such as new location, high variability of network bandwidth; their data interests are changing from time to time and from location to location; even data semantics in mobile hosts vary according to data access patterns, connection duration and disconnection frequencies, etc. Data partition, location, and replication are always dynamic. All of these require a dynamic database design and reconfigure scheme.

13.2.7 Distributed Database for Mobile

A distributed database is a single logical database that is spread physically across computers in multiple locations that are connected by a data communications network. We emphasize that a distributed database is truly a database and not a loose collection of files. The distributed database is still centrally administered, work must allow the users to share the data; thus a user or program at location A must be able to access (and perhaps update) data at location B.

The distributed database requires multiple database management systems, running at each remote site. The degree to which these different DBMSs cooperate, to work in partnership, and whether there is master site that coordinated request involving data from multiple sites distinguish different types of distributed database environments.

Distributed database framework for mobile environments is made entirely by mobile components. Mobile hosts communicate together in an ad hoc manner. Communication networks are formed on demand.

Modes of Operation of Distributed Database

There are four modes of operation of distributed database. They are:

- Sign-off mode
- Check-out mode
 - DB partition
 - Check-out with mobile read
 - Check-out with system read

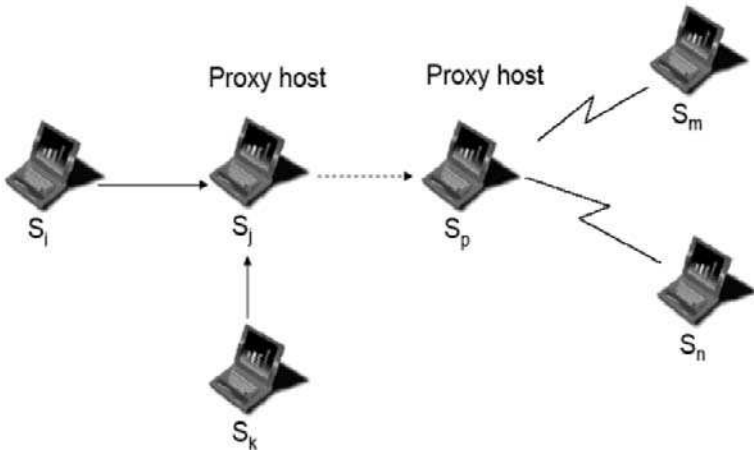


Fig. 13.12. Sign-off mode architecture

- Relaxed check-out mode
- Optimistic check-out mode

Sign-Off Mode Architecture

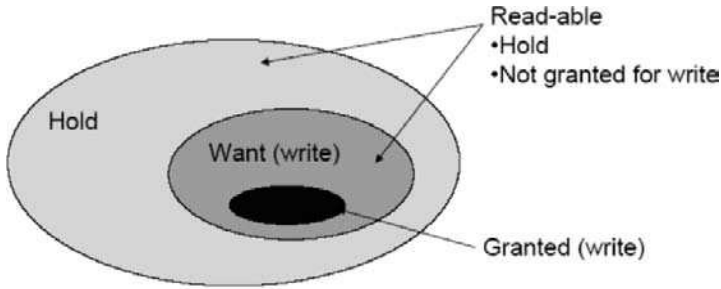
Sign-off mode architecture is shown in Fig. 13.12 which performs the following operations. They are:

- Sign-off protocol
- Sign-on protocol
- Correctness
 - Read-only transactions of the disconnected site can be serialized at the time of disconnection
- Broadcast communication
- Point-to-Point communication

Check-Out Mode Operation

The mobile database at mobile host is shown in Fig. 13.13:

- Pseudotransaction
 - Cannot be aborted in order to release lock
- DB-partition
- Check-out with mobile read
 - Not necessary to obtain read locks before disconnection
- Read version at disconnection is consistent
 - Transactions are serialized at the point in time of disconnection
- Check-out with system read
 - Pseudotransaction obtains read locks but data items are writable
- Upgrade read locks to write locks at reconnection



A mobile database at a mobile host

Fig. 13.13. Mobile database at a mobile host

Mobile Database Research Directions

As the mobile system technology matures further, more people will become mobile users communicating with one another and accessing various information resources using portable computers, personal digital assistants, wireless radio, and cellular equipment. In business environments, the ability to access critical data regardless of location is even more crucial because corporate data must be available to applications running on mobile workstations. Some of the areas are:

- Location-dependent query processing
- View maintenance in mobile computing
- Work flows in mobile environment
- Digital library services in mobile computing
- Mobile web and e-commerce
- Mobile data security

13.3 Spatial Database

Modern applications are both data and computationally intensive and require storage and manipulation of voluminous traditional (alphanumeric) and nontraditional (images, text, geometric objects, etc.). Examples of such application domains are Geographical Information Systems (GIS), Multimedia Information Systems, CAD/CAM applications, Medical Information Systems.

Spatial database management systems store data like points, lines, regions, volumes, and aim at supporting queries that involve the space characteristics of these data. In order to handle such queries, special techniques and tools enhance a spatial database system. These include new data types and models, sophisticated data structures and algorithms for efficient query processing that differ from their counterparts in a conservative alphanumeric database. When

a spatial database is enhanced by temporal characteristics we get spatiotemporal database system. In such a system, the time of insertions, deletions, and updates is of great importance, since it must be able to store and manipulate the evolution of spatial objects.

Spatial database systems are database systems for the management of spatial data. Spatial data are point objects or spatially extended objects in a 2D or 3D space or in some high-dimensional vector space. Knowledge discovery becomes more and more important in spatial databases since increasingly large amounts of data obtained from satellite images, X-ray crystallography or other automatic equipment are stored in spatial databases.

In various fields there is a need to manage geometric, geographic, or spatial data, which means data related to space. The space of interest can be, for example, the 2D abstraction of (parts of) the surface of the earth that is, geographic space, the most prominent example like the layout of a VLSI design, a volume containing a model of the human brain, or another 3D space representing the arrangement of chains of protein molecules. At least since the advent of relational database systems there have been attempts to manage such data in database systems. Characteristic for the technology emerging to address these needs is the capability to deal with large collections of relatively simple geometric objects, for example, a set of 100,000 polygons. This is somewhat different from areas like CAD databases (solid modeling, etc.) where geometric entities are composed hierarchically into complex structures, although the issues are certainly related.

A spatial database system is a database system which offers SDTs in its data model and query language. It supports SDTs in its implementation, providing at least spatial indexing and efficient algorithms for spatial join.

13.3.1 Spatial Data Types

SDTs or time series data types is mainly for 2D operations.

2D Data Types

For 2D application, the relevant data types would include the following:

- A point defined by (x, y) coordinates
- A line defined by its two end points
- A polygon defined by an ordered list of n points that form its vertices
- A path defined by a sequence (ordered list) of points
- A circle defined by its center point and radius

Given the above as data type, a function such as distance may be defined between two points, a point and a line, a line and a circle, and so on, by implementing the appropriate mathematical expressions for distance in a

programming language. Similarly, a Boolean cross function which returns true or false depending on whether two geometric object cross (or intersect) can be defined between a line and a polygon, a path and a polygon, a line and a circle, so on.

13.3.2 Spatial Database Modeling

For modeling single objects, the fundamental abstractions are point, line, and region. A point represents an object for which only its location in space. A line (meaning a curve in space, usually represented by a polyline, a sequence of line segments) is the basic abstraction for connections in space. A region is the abstraction for something having an extent in 2D space. A region may consist of several disjoint pieces. Figure 13.14 shows the three basic abstractions for single objects.

The two most important instances of spatially related collections of objects are partitions (of the plane) and networks. A partition can be viewed as a set of region objects that are required to be disjoint. A network can be viewed as a graph embedded into the plane, consisting of a set of point objects, forming its nodes, and a set of line objects describing the geometry of the edges.

13.3.3 Discrete Geometric Spaces

For geometric modeling very often Euclidean space is used which means that a point in the plane is given by a pair of real numbers. As computers can deal with only finite computations, geometric calculations become difficult. Thus methods have been suggested to introduce a discrete geometric basis for modeling as well as implementation. The approach is based on combinatorial topology. Basic concepts are those of a simplex, and a simplicial complex. For each dimension d , a d -simplex is a minimal object in that dimension, hence a 0-simplex is a point, a 1-simplex is a line segment, a 2-simplex a triangle, a 3-simplex a tetrahedron, etc. Any d -simplex is composed of $(d+1)$ simplices of dimension $d-1$. The components used in the composition of a simplex are called its faces (for a triangle its edges and vertices). A simplicial complex is

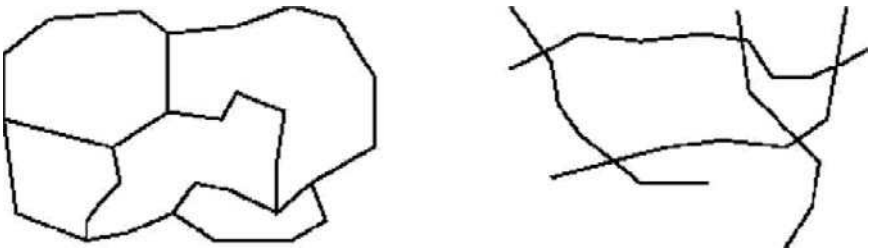


Fig. 13.14. Partitions and networks

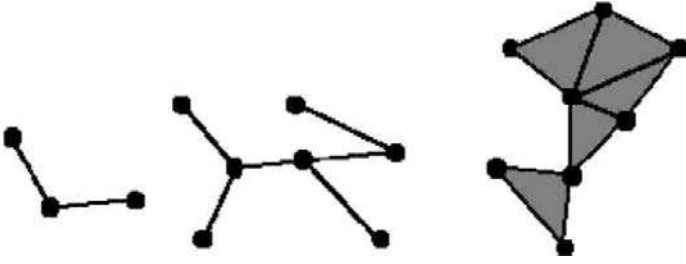


Fig. 13.15. Two simplicial complexes

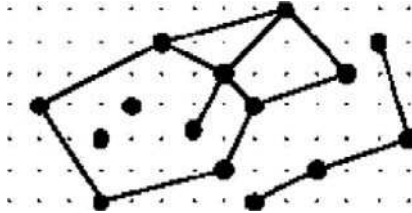


Fig. 13.16. Realm

a finite set of simplices such that the intersection of any two simplices in the set is a face. Figure 13.15 shows a 1-complex and a 2-complex.

An alternative proposal of a discrete geometric basis is the concept of a realm. Formally, a realm is a finite set of points and line segments over a discrete grid such that (a) each point or end point of a line segment is a grid point, (b) each end point of a line segment is also a point of the realm, (c) no realm point lies within a line segment (which means on it without being an end point), and (d) no two realm segments intersect except at their end points. Figure 13.16 illustrates a realm.

13.3.4 Querying

Querying is to connect the operations of a spatial algebra to the facilities of a DBMS query language. Spatial data require a graphical presentation of results as well as graphical input of queries. In the following sections, we consider the fundamental operations needed for manipulating sets of database objects.

Fundamental Operations (Algebra)

Fundamental operations can be classified as spatial selection, spatial join, spatial function application, and other set operations.

Spatial Selection

A selection is an operation that returns from a set of objects those fulfilling a predicate. Some examples are:

“Find all cities in Bavaria” (assuming Bavaria exists as a REGION value and inside is available in the spatial algebra)

cities select[center inside Bavaria]

Spatial Join

Similarly to a spatial selection, a spatial join is a join which compares any two objects through a predicate on their spatial attribute values. Some examples are:

Combine cities with their states.

cities states join[center inside area]

Spatial Function Application

In a set-oriented query a new SDT value is computed for each object in a set. Various object algebra operators allow such an embedding of a function application, for example, the filter operator of FAD, a replace operator, or the λ or extend operator.

For each river going through Bavaria, return the name, the part of its geometry lying inside Bavaria, and the length of that part.

**rivers select[route intersects Bavaria]
 extend[intersection(route, Bavaria) {part}]
 extend[length(part) {plength}]
 project[rname, part, plength]**

13.3.5 Integrating Geometry into a Query Language

Integrating geometry into a query language has the following three main aspects:

- (a) Denoting SDT values as constants in a query and graphical input of such constants.
- (b) Expressing the four classes of fundamental operations for an embedded spatial algebra.
- (c) Describing the presentation of results.

Denoting SDT values/graphical input. SDT constants may be entered through a graphical input device. In the georelational algebra atomic values are “first class citizens,” so one can introduce a named REGION value Bavaria as follows:

states extract[sname = “Bavaria”; area] {Bavaria}

Expressing the four classes of fundamental operations. Spatial function application, although not possible in classical relational algebra, is also in practice provided by query languages (in SQL by allowing expressions in the SELECT clause).

SELECT * FROM rivers WHERE route intersects Window

13.3.6 Spatial DBMS Implementation

From the point of view of the spatial algebra implementation which is done in some programming language, most likely the DBMS implementation language, the representation:

- Is a value of some programming language data type, e.g., region
- Is some arbitrary data structure which is possibly quite complex
- Supports efficient computational geometry algorithms for spatial algebra operations
- Is not geared only to one particular algorithm but is balanced to support many operations well enough

To fulfill the requirements of the DBMS, the representation must be a paged data structure compatible with the DBMS support for long fields or large attribute values. To support efficient loading and storing on disk, it should consist of a single contiguous byte block as long as it is small enough to fit into one page or implement a more complex paging strategy to access the value. For the case that a value representation happens to be large, a good strategy is to split it into a small info part. The info part might be contained in the DBMS object representation and contain a logical pointer to a separate page sequence holding the exact geometry part. The generic operations needed by the DBMS may concern, for example, transforming from/to a textual or graphic representation for input/output at the user interface, or transforming from/to an ASCII format for bulk loading or external data exchange. More specifically, for SDTs, generic approximations may be needed to interface with spatial access methods, for example, each data type must provide access to a bounding box (also called minimum bounding rectangle (MBR)).

From the spatial algebra and also the programming language point of view, the representation should be such that it is mapped by the compiler into a single or perhaps a few contiguous areas (to support the DBMS loading). The representation can support operations as follows:

Plane Sweep Sequence . Very often, algorithms on the exact geometry use a plane-sweep. The sweep needs the components of the object (e.g., the vertices) in some fixed order.

Approximations . The implementation of many operations starts with a rough test on an approximation of the object as that of the bounding box. Hence these should be part of the representation.

Stored Unary Function Values . Some operations of the spatial algebra compute properties of a spatial value, e.g., the area or perimeter of a region. They are computed after the creation of the value and then stored with it.

13.4 Multimedia Database Management System

13.4.1 Introduction

A database management system (DBMS) is a general-purpose software system that facilitates the processes of defining, constructing, and manipulating databases for various applications. DBMSs, and in particular those based on the relational data model, have been very successful at the management of administrative data. However, handling large collections of digitized multimedia data is still a major challenge. Current database systems are not equipped to represent the entire multimedia data flow, nor may it be desirable for them to support the plethora of retrieval types required to support multimedia data. Thus, continuous media data must be parsed into representable segments, which we term media objects. In order to represent the original data stream to users, synchronization constraints among media objects must be specified and maintained.

A multimedia database management system (MMDBMS) must support multimedia data types in addition to providing facilities for traditional DBMS functions like database creation, data modeling, data retrieval, data access and organization, and data independence. The area and applications have experienced tremendous growth. Especially with the rapid development of network technology, multimedia database system gets more tremendous development and multimedia information exchange becomes very important.

13.4.2 Multimedia Data

Multimedia data refers to the simultaneous use of data in different media forms, including images, audio, video, text, and numerical data. Many multimedia applications, such as recording and playback of motion video and audio, slide presentations, and video conferencing, require continuous presentation of a media data stream and the synchronized display of multiple media data streams. Such synchronization requirements are generally specified by either spatial or temporal relationships among multiple data streams. For example, a motion video and its caption must be synchronized spatially at the appropriate position in a movie, and, in a slide presentation, a sequence of images and speech fragments must be temporally combined and presented to compose one unified and meaningful data stream.

Characterization of Multimedia Data

Multimedia systems are able to deal with multimedia data like audio, still images, graphics, animation, text, etc. The most significant features of multimedia data come from the observation that its representation can be much closer to the physical or a virtual physical reality than the usual alphanumeric data which in general is used to represent symbolic information. For example, a video is a recording of the visual information over a period of time at a point in space. The classification of multimedia data can be made as time-dependent data like audio, video, and animation or time-independent data which includes data types like text, still images, and alphanumeric data types. Time-dependent data has only a meaningful interpretation with respect to a constantly progressing time scale. A timescale is needed to associate with a time-dependent data its correct interpretation at each point of time expressed by the atomic constituents of the data. Due to the closeness to physical reality a further characteristic of multimedia data is its high density of information in one datum. Due to high density of information in multimedia data the amounts of data can be huge. As long as data is static in time and size like symbols, pictures, or images no serious problems in terms of processing speed are imposed on networks, on storage devices, and on main memories of current computer technology. Also dynamic data in size can be handled efficiently by using the abstraction of files as provided by operating systems. Serious problems with such data occur only in connection with applications where extreme high numbers of data elements are involved, example, processing of satellite images for weather forecast.

When dealing with huge amount of data under real time constraints it may be convenient or even necessary to perform the processing not on the data values themselves but on the references to the values. A good example for this is video script editing. Certain applications of dynamic data may need operations which cannot be performed over references, for example, copying. In this case some form of dynamic data management has to be provided, which a kind of spreads the process over time such that at each distinct moment only a limited amount of physical resources needed. When processing dynamic data, typically parallel tasks occur. This comes from the nature of this data since in contrast to processing static data operations taken nonnegligible periods of time. Also it often is necessary to process data in parallel.

13.4.3 Multimedia Data Model

A multimedia data stream consists of a set of data upon which some time constraints are defined. The time constraints may specify discrete, continuous, or stepwise constant time flow relationships among the data. For example, some multimedia streams, such as audio and video, are continuous in nature, in that they flow across time; other data streams, such as slide presentations and animation, have discrete or stepwise time constraints. The multimedia streams

may not have convenient boundaries for data representation. To facilitate retrieval of such data in databases, we must break each media stream into a set of media objects. Each media object represents a minimum chunk of the media stream that bears some semantic meaning. Media objects in different media streams may have different internal structures. For example, a continuous video stream can be segmented into a set of media objects, each of which contains a set of video frames with specific semantic meaning. Similarly, a continuous audio stream can be segmented into a set of media objects, each of which contains a set of audio samples with specific semantic meaning. Without loss of generality, we assume that basic data elements delivered from the transportation layer are media objects.

Media objects from different data streams may need to be linked through time constraints to specify their synchronization. For example, in slide presentation applications, an audio object must be played along with a slide object. We define a *multimedia unit* O_i , to be the composition of a set of media objects O_{1i}, \dots, O_{ni} , where O_{ni} represents i th media object of the n th media stream participating in the synchronized stream. Thus, a composite data stream made up of multiple media streams consists of a set of multimedia units, where each multimedia unit unifies media objects from multiple media streams. Such multimedia units may also be considered as composite objects. Furthermore, a collection of objects from either a single data stream or a composite data stream may be conceptually modeled as a hierarchical structure. At each increasing level, a set of media objects may be considered to be a super class object which may then, in turn, be a component of another super class at a higher level.

Constraints on Media Objects

We will now address the synchronization requirements that need be placed on media data streams. There are two types of constraints that need to be specified on media objects, intra- and interstream constraints. Intrastream constraints specify the synchronization requirements to be placed on a single media stream and interstream constraints specify the synchronization requirements to be placed on more than one media stream.

Let m_i be a single media stream which consists of a set of media objects O_{i1}, \dots, O_{il} . Intrastream constraints on O_{i1}, \dots, O_{il} define time flow relationships among these objects. The intrastream constraints may be continuous, discrete, or stepwise constant. Unlike temporal constraints, media objects are not typically statically associated with independent time constraints. Owing to the sequential character of media objects in a single media stream, each media object must be associated with a relative start time and a time interval which specifies the duration of its retrieval, assuming that the first media object in the media stream starts at time zero. The actual start time of a media object is usually dynamically determined. Once a media stream is invoked, it is associated with an actual *start time*; the start time of each media

object within that stream will similarly be associated with the actual start time. Let $\langle o, t, \Delta t \rangle$ denote the intrastream constraint on object O that is presented at time t and lasts a time period Δt . Let u be a composite data stream of media streams m_1, \dots, m_n . u can then be considered as consisting of a set of multimedia units u_1, \dots, u_l , with each multimedia unit u_i . Interstream constraints on u_1, \dots, u_l define synchronization requirements among the participating component media streams. Time-related interstream constraints are defined implicitly in each media object.

13.4.4 Architecture of Multimedia System

The architecture of multimedia system is shown in Fig. 13.17. In this architecture, MTL refers to Multimedia Transaction Language, MM is the acronym for Media Manager, OODBMS stands for Object Oriented Database Management System. The multimedia transaction manager contains two main modules, a multimedia transaction language (MTL) interpreter and a media manager (MM).

The multimedia transaction language MTL interpreter allows users to specify a set of transactions associated with a multimedia transaction, including intra- and intersynchronization requirements on component transactions. A multimedia transaction specified in MTL is the MM and the underlying processed by the interpreter, and data accesses are sent to both MM and the underlying OODBMS for processing. The design strategies can be applied to

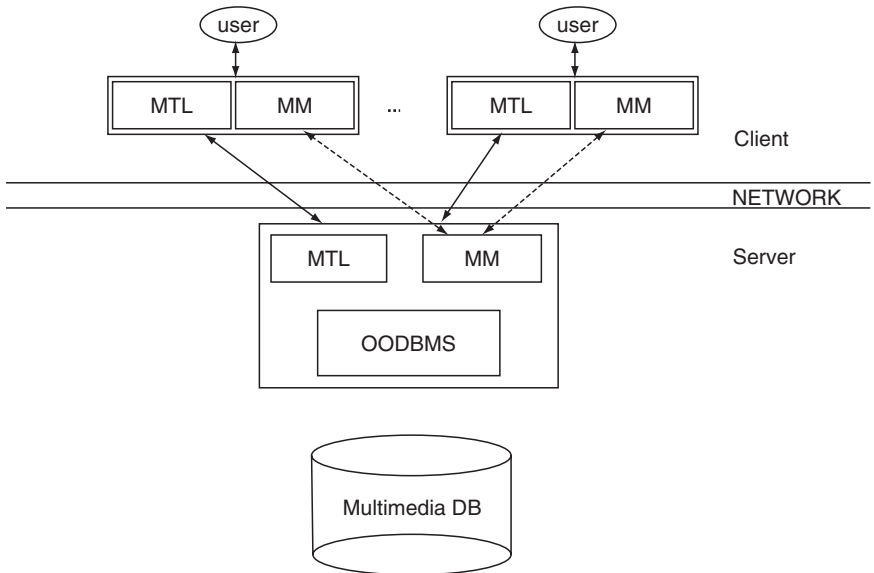


Fig. 13.17. Architecture of multimedia system

any OODBMS environment that support a C++ interface. Currently existing object-oriented database systems that fit into this category include Object-Store and ODE.

13.4.5 Multimedia Database Management System Development

The first MMDBMS rely mainly on the operating system for storing and querying files. These were ad hoc systems that served mostly as repositories. The first multimedia database system ORION was developed in 1987. In the mid 1990s, some of the commercial MMDBMS developed are Media-DB, JASMINE, and ITASCA that is the commercial successor of ORION. They were all able to handle diverse kind of data and provided mechanisms for querying, retrieving, inserting, and updating data. Most of these products disappeared from the market after some years of existence, and only some of them continued and adapted themselves successfully to the hardware and software advances as well as application changes.

From 1996 to 1998, commercial systems were proposed which handle multimedia content by providing complex object types for various kinds of media. The object-oriented style provides the facility to define new data types and operators appropriate for the new kinds of media, such as video, image, and audio. Therefore, broadly used commercial MMDBMSs are extensible Object-Relational DBMS.

The most advanced solutions are marketed by Oracle 10g, IBM DB2, and IBM Informix. They propose a similar approach for extending the basic system. DB2 Image Extender defines the distinct data type DB2IMAGE with associated user-defined functions for storing and manipulating image files. MIRROR, an acronym for Multimedia Information Retrieval Reducing Information Overload developed at the University of Twente, is a research MMDBMS that is developed to better understand the kind of data management that is required in the context of multimedia digital libraries. MARS, an acronym for Multimedia Analysis and Retrieval System is a project carried out at the University of Illinois. MARS realizes an integrated multimedia information retrieval and database management system, that supports multimedia information as first class object suited for storage and retrieval based on their semantic content. The MPEG-7 Multimedia Data Cartridge is a system extension of the Oracle 9i DBMS providing a multimedia query language, access to media, processing and optimization of queries, and indexing capacities relying on multimedia database schema derived from MPEG-7.

13.4.6 Issues in Multimedia DBMS

The key issue in a multimedia database is how to access and how to exchange multimedia information effectively. The key to retrieval process is similarity between two objects. The content of the object is analyzed and used to

evaluate specified selection predicates which are termed as content-based retrieval. In order to have an accurate representation of the multimedia objects in the database and the query object, different features like texture, shape, and color have to be combined. The results are high dimensional vectors. The efficiency of the similarity search must be supported by the use of multidimensional indexing structures and by dimension reduction methods.

In order to retrieve multimedia data from a database system, a query language must be provided. A multimedia query language must have abilities to handle complex spatial and temporal relationships. A powerful query language should have to deal with keywords, index on keywords, and semantic contents of multimedia objects. The key to efficient communication is to rely on standards for communicating metadata and associated media data. In multimedia database management system, there is a need to retrieve composite objects. In addition, the multimedia database system should use multiple representations of data for different users and profiles.

13.5 XML

13.5.1 Introduction

Extensible Markup Language (XML) is a simplified version of SGML (Standard Generalized Markup Language defined by ISO 8879) designed for Web Applications. It retains all SGML advantages of extensibility, structure, and validation in a language that is designed to be vastly easier to learn, use, and implement than full SGML. The World Wide Web Consortium, the standards body for all web technologies, describes XML as a “method for putting structured data in a text file.” XML is a standard for the definition of markup languages for the Web. XML was designed to describe data. XML uses a Document Type Definition (DTD) or an XML Schema to describe the data. It is not itself the successor to HTML as the language of the web, but successors will follow the XML standard. Like HTML, XML is built on the SGML standard, but as a refinement of the standard, not a set of tags. That is a key distinction that gives XML particular value. XML is a W3C recommendation.

13.5.2 Origin of XML

XML was developed by an XML Working Group (originally known as the SGML Editorial Review Board) formed under the auspices of the World Wide Web Consortium (W3C) in 1996. It was chaired by Jon Bosak of Sun Microsystems with the active participation of an XML Special Interest Group (previously known as the SGML Working Group) also organized by the W3C.

13.5.3 Goals of XML

The design goals for XML are:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

13.5.4 XML Family

XML is actually family of standards and technologies which are given in Table 13.1. XML 1.0 defines what elements, attributes, and entities are. Other standards include:

- *XLink* describes a standard way to add hyperlinks to an XML file.
- *XPointer* is a syntax in development for pointing to parts of an XML document. An XPointer is a bit like a URL, but instead of pointing to documents on the Web, it points to pieces of data inside an XML file.
- *CSS* (Cascade Style Sheets) describes formatting rules, same as it does for HTML.
- *XSL* is the advanced language for expressing stylesheets.
- *XML Namespaces* describes a facility for including tags from different tagsets in a single document.
- *XML Schemas* describes an alternative to the DTD for defining the elements and attributes in a document. XML Schemas help developers to precisely define the structures of their own XML-based formats.
- *DOM* (Document Object Model) is a standard set of function calls for manipulating XML files from a programming language.

13.5.5 XML and HTML

XML is not a replacement for HTML. XML and HTML were designed with different goals. XML was designed to describe data and to focus on what data is. HTML was designed to display data and to focus on how data looks. HTML is about displaying information, while XML is about describing information.

XML tags are not predefined. The user must invent their own tags. On the other hand, the tags used to mark up HTML documents and the structure of

Table 13.1. Summary of XML terminology

XML	Extensible Markup Language. A document markup language that started the following:
XSL	XSLT Stylesheet. The document that provides the {match, action} pairs and other data for XSLT to use when transforming an XML document
XPath	A sublanguage within XSLT that is used to identify parts of an XML document to be transformed. It can also be used for calculations and string manipulation.
XPointer	A standard for linking one document to another. XPath has many elements from XPointer
SAX	Simple API (Application Program Interface) for XML. An event based parser that notifies a program when the elements of an XML document have been encountered during document parsing
DOM	Document Object Model. An API that represents an XML document as a tree. Each node of the tree represents a piece of the XML document. A program can directly access and manipulate a node of the DOM representation
XQL	A standard for expressing database queries as XML documents. The structure of the query uses XPath facilities and the result of the query is represented in an XML format
XML Namespaces	A standard for allocating terminology to defined collections
XML Schema	An XML compliant language for constraining the structure of an XML document. Extends and replaces DTDs.

HTML documents is predefined. The author of HTML documents can only use tags that are defined in the HTML standard (like `<h1>`, `<p>`, etc). XML is a cross platform, software- and hardware-independent tool for transmitting information.

13.5.6 XML Document

A data object is an XML document if it is well-formed, as defined in this specification. A well-formed XML document may in addition be valid if it meets certain further constraints.

Each XML document has both a logical and a physical structure. Physically, the document is composed of units called entities. An entity may refer to other entities to cause their inclusion in the document. A document begins in a “root” or document entity. Logically, the document is composed of declarations, elements, comments, character references, and processing instructions, all of which are indicated in the document by explicit markup. The logical and physical structures must nest properly.

13.5.7 Document Type Definitions (DTD)

The purpose of DTD is to define legal building blocks of an XML document. It defines the document structure with a list of legal elements. A DTD can be declared inline in XML document or as an external reference.

A DTD is a structural specification of a type of document. DTDs are made up primarily of element definitions. Each element corresponds to a component of information in the document. Elements define the tagging of the document. They also define the content allowed for each tag.

XML provides an application independent way of sharing data. With a DTD, independent groups of people can agree to use a common DTD for interchanging data. A lot of forums are emerging to define standard DTDs for almost everything in the areas of data exchange.

Internal DTD

An example of internal DTD is:

```
<?xml version="1.0" ?>
<!DOCTYPE note(View Source for full doctype...)>
<note>
  <to>Senthil</to>
  <from>Rajan</from>
  <heading>Reminder</heading>
  <body>Don't forget viva is there in this weekend!</body>
</note>
```

The DTD can be interpreted as

!ELEMENT note (in line 2) defines the element “note” as having four elements: “to, from, heading, body.”

!ELEMENT to (in line 3) defines the “to” element to be of the type “CDATA.”

!ELEMENT from (in line 4) defines the “from” element to be of the type “CDATA”

and so on.....

13.5.8 Extensible Style Sheet Language (XSL)

XML document has no visual appearance, one approach is to write a script that loads XML document into a parser, reads the data, and writes HTML. Another approach involves loading an XML document into a parser on the browser and then, rather than creating a complete HTML stream, simply replacing portions of a web page already on display. The third approach involves loading an XML document into a XML data source object (DSO) within the

web page. The advantages of this approach are script code can access the XML data using familiar ADO recordset methods and properties. Each bound element then displays automatically the current value of the bound node.

The fourth approach is the one actually proposed by W3C using Extensible Stylesheet Language. XSL encompasses two main components, a transformation language and a formatting language. The transformation language converts one XML document to another. Typical transformations include adding, recalculating, excluding, renaming, or reordering nodes. The XSL transformation language is organized around the concept of templates. An XSL template is basically a model collections of statements interspersed with XSL tags that repeat blocks of statements for each selected node in an XML document. Additional XSL tags substitute the values of specified nodes into the model statements. An XSL style sheet is itself a valid XML document. To run the style sheet, the data are first loaded into one XML document object and the style sheet into another. The first object's transform node method is invoked to create the text of a new XML document.

13.5.9 XML Namespaces

A namespace contain a list of valid element names and has also a distinctive prefix and an identifying name.

Name Conflicts

Since element names in XML are not predefined, a name conflict will occur when two different documents use the same element names. The way to solve this name conflict is using a prefix. This prefix is the table's name creates for each XML document.

Example

This XML document carries information about fruits

```
<h:table>
  <h:tr>
    <h:td>Orange</h:td>
    <h:td>Mango</h:td>
  </h:tr>
</h:table>
```

This XML document carries information about a piece of furniture

```
<f:table>
  <f:name>Teak wood Table</f.name>
```

```

    <f:width>80</f:width>
    <f:length>120</f:length>
</f:table>

```

There will not be any conflict between these two tables because the two documents use a different name for their table element.

Instead of using only prefixes, it is also possible to add an `xmlns` attribute to the `<table>` tag to give the prefix a qualified name associated with a namespace.

Example

xmlns:namespace-prefix=namespaceURI.

When a namespace is defined in the start tag of an element, all child elements with the same prefix are associated with the same namespace.

Comments may appear anywhere in a document outside other markup. In addition, they may appear within the document type declaration at places allowed by the grammar. They are not part of the document's character data; an XML processor may, but need not, make it possible for an application to retrieve the text of comments. For compatibility, the string "--" (double-hyphen) must not occur within comments.

An example of comment,

```
<!--declarations for <head>&<body>-->
```

Document Type Declaration

XML documents may, and should, begin with an *XML declaration* which specifies the version of XML being used. For example, the following is a complete XML document, well-formed but not valid:

```

<?xml version="1.0"?>
<greeting>Hello, How are you!</greeting>
and so is this:
<greeting>Hello, How are you!</greeting>

```

The version number "1.0" should be used to indicate conformance to this version of this specification; it is an error for a document to use the value "1.0" if it does not conform to this version of this specification. It is the intent of the XML working group to give later versions of this specification numbers other than "1.0," but this intent does not indicate a commitment to produce any future versions of XML, nor if any are produced, to use any particular numbering scheme. Since future versions are not ruled out, this construct is provided as a means to allow the possibility of automatic version recognition, should it become necessary. Processors may signal an error if they receive documents labeled with versions they do not support.

The function of the markup in an XML document is to describe its storage and logical structure and to associate attribute-value pairs with its logical structures. XML provides a mechanism, the document type declaration, to define constraints on the logical structure and to support the use of predefined storage units. An XML document is *valid* if it has an associated document type declaration and if the document complies with the constraints expressed in it.

13.5.10 XML and Database Applications

XML Schema provides a standardized way of representing domains. The benefits of XML are:

Tagged Information. Structured documents allow the user to specify unique and specific content tags that makes it possible to retrieve any piece of tagged information. A format tag would not allow a user to access the structure of the document. For example, if the format tag for the abstract was italic, the format tag for a glossary entry might also be italic. There would be no way to retrieve the information because the format tags were not unique and identifiable. However, the structural tag would allow a user to retrieve the abstract or a series of abstracts.

Reusable Components. It is possible to tag the information based on the use of the individual pieces of information or components. Components are pieces of information that can be used individually or combined to form larger components. Components include: paragraphs, chapters, warnings, notes, instructions, introductions, and examples.

Structured Documentation. Structured documentation provides a great deal of power in organizing a document:

- Consistent organization and structure across documents.
- Reusability of segments (modules).
- Increased accessibility.

Separation of Content and Formatting. The separation of content from formatting makes it very easy to create multiple outputs from a single XML file. The format is defined in style sheets that can be linked to the file.

Projects in DBMS

Learning Objectives. Here we have given list of projects which we have implemented using Oracle as the back-end and Visual Basic as the front-end and the concepts from DBMS. The projects described in this chapter are generally simpler than real-life projects but complex enough to illustrate common problems that the students will encounter.

14.1 List of Projects

1. Bus Transport Management System
2. Course Administration System
3. Election Voting System
4. Hospital Management System
5. Library Management System
6. Railway Management System

14.2 Overview of the Projects

14.2.1 Front-End: Microsoft Visual Basic

Visual Basic was derived from BASIC, and is an event-driven programming language. Programming in Visual Basic is done visually, which means that as we design we will know how our application will look on execution. We can, therefore, change and experiment with the design to meet our requirement.

The Visual Basic Professional edition provides the features like:

- Allows creating powerful 32-bit applications for Microsoft Windows 9x and Windows NT.
- Includes intrinsic controls, as well as grid, tab, and data-bound controls.

- Includes Microsoft Developer Network CDs containing full online documentation.
- Active X controls, including Internet control.
- Internet Information Server Application Designer.
- Dynamic HTML Page Designer.

14.2.2 Back-End: Oracle 9i

Oracle is the first DBMS language that supported SQL in 1979. It is an object-relational database. A relational database is an extremely simple way of managing data in the form of a collection of tables. An object-relational database supports all the features of a relational database while also supporting object-oriented concepts and features. This language generally follows a cooperative approach.

Organized data can be called *information*. Oracle is also a means of easily turning data into information. Oracle will sort through and manipulate data and their relationships with each other.

A relational database management system such as Oracle basically does the following three things:

1. Acquire data
2. Store the data
3. Retrieve the data

Oracle supports this in-keep-out approach and provides tools that allow sophistication in how data are captured, edited, modified, and put in; how to maintain security; and how to manipulate. An object-relational database management system (ORDBMS) extends the capabilities of the RDBMS to support object-oriented concepts. Thus Oracle is used as an RDBMS to take advantage of its object-oriented features.

Oracle follows a familiar language used in everyday conversations. The information stored in Oracle is kept in tables. Also Oracle is a shared language.

Oracle was the first company to release a product that used the English-based Structured Query Language (SQL). Oracle's query language has structure and a set of rules and syntax that are basically the normal rules that can be easily understood.

14.2.3 Interface: ODBC

Open Database Connectivity (ODBC) is an industry standard programming interface that enables applications to access a variety of database management system residing on many different platforms. ODBC provides a large degree of database independence through a standard SQL syntax, which can be translated by database-specific drivers to the native SQL of the DBMS.

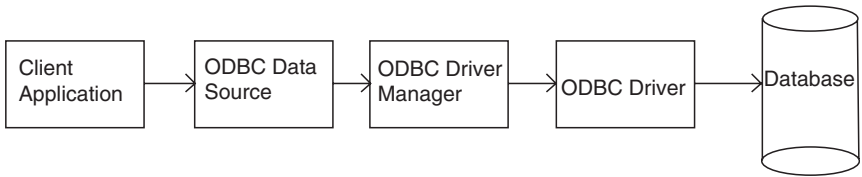


Fig. 14.1. Open database connectivity architecture

Database independence and ease of use are the primary advantages of using ODMS. Many popular development tools such as Visual Basic and Delphi support ODBC. These tools and numerous others provide their own interface to ODBC.

ODBC is a windows technology that lets a database client application connect to an external database. To use ODBC the database vendor must provide an ODBC driver for data access. Once this driver is available the client machine should be configured with the driver. The destination of the database, login ID, and password is also to be configured on every client machine. This is called a *data source*.

ODBC is composed of three parts, they are (Fig. 14.1):

1. A Driver Manager
2. One or more Driver
3. One or more Data Sources

14.3 First Project: Bus Transport Management System

14.3.1 Description

By using this project, we can reserve tickets from any part of the world, through telephone lines, via internet. This project provides and checks all sorts of constraints so that user does give only useful data and thus validation is done in an effective way (Figs. 14.2–14.8).

14.3.2 Features of the Project

- User friendliness
- Occupies less space
- Validation of data
- Can be used for other means of transports by slight modification like railways and airline reservation system
- Can be used for online transactions

ER DIAGRAM

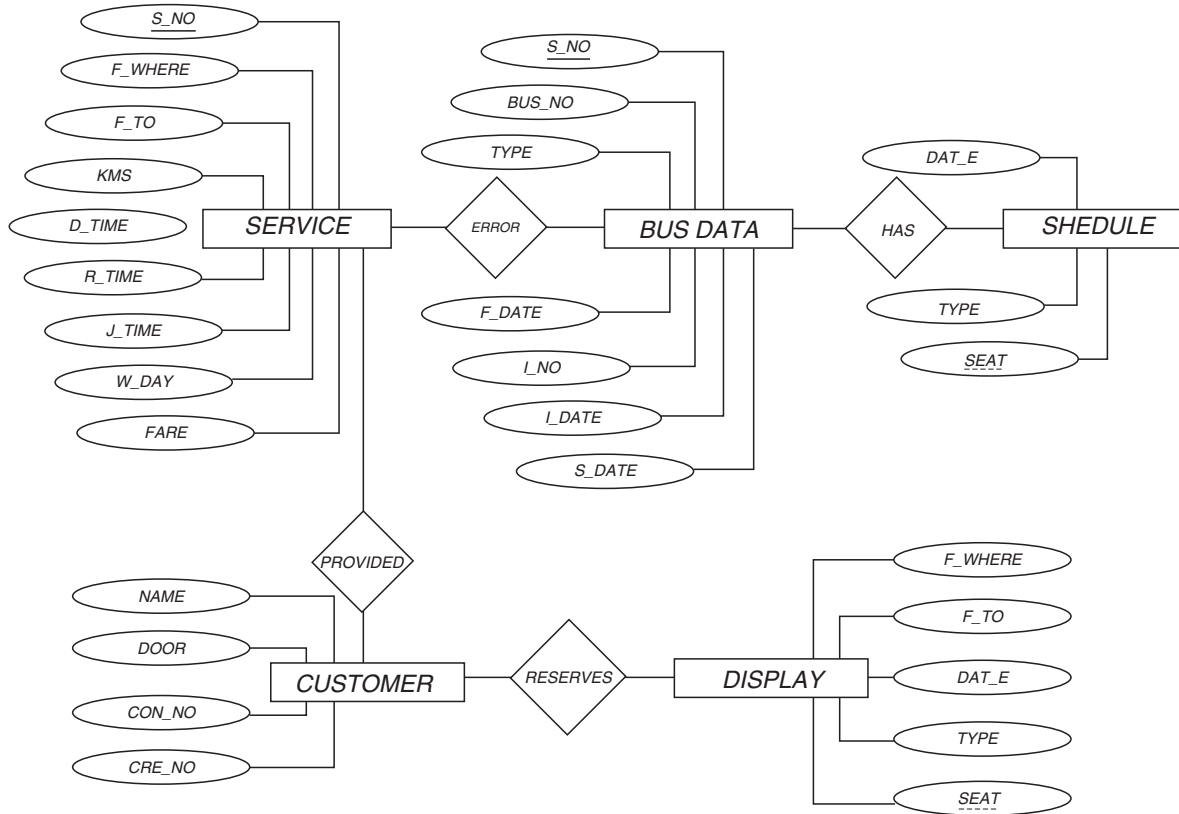


Fig. 14.2. ER diagram



Fig. 14.3. User desktop

14.3.3 Source Code

Code Sample for Integrating Main Window to Subwindows

```
Private Sub CmdAdmin_Click()
Adminwindow.Show
Mainwindow.Hide
End Sub
```

```
Private Sub CmdAbout_Click()
Aboutwindow.Show
Mainwindow.Hide
End Sub
```

```
Private Sub Timer1_Timer()
Label2.Caption = FormatDateTime(Date, vbLongDate)
Label3.Caption = FormatDateTime(Time, vbLongTime)
End Sub
```

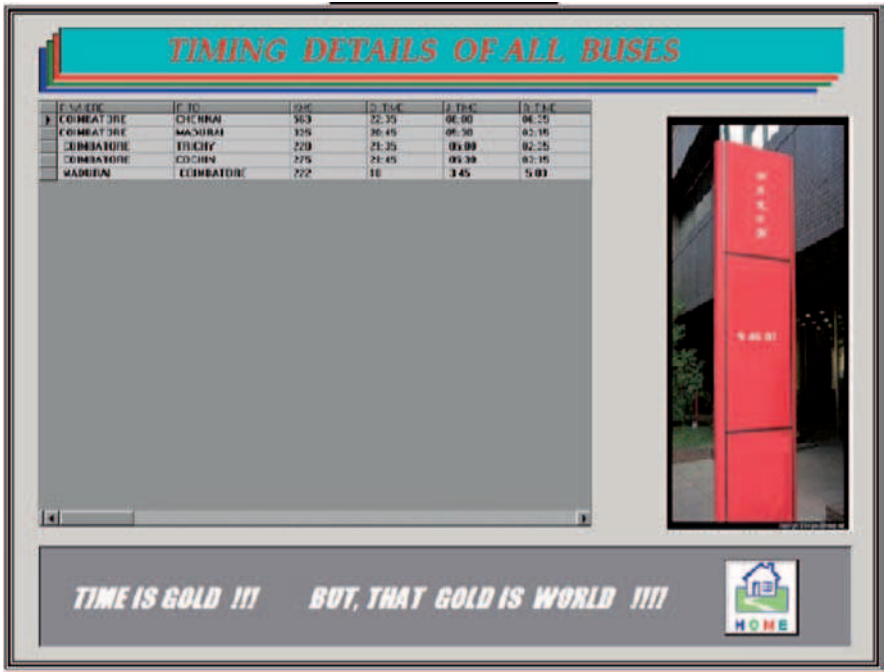


Fig. 14.4. Timing details

Code Sample for Manipulating Timing Details

```
Set DataGrid1.DataSource = Adodc1
```

```
Private Sub Form_Load()
```

```
Dim A As String
```

```
A = "SELECT F_WHERE, F_TO, KMS, D_TIME, J_TIME, R_TIME,  
W_DAY FROM SERVICE"
```

```
Set DataGrid1.DataSource = Adodc2
```

```
With Adodc2
```

```
.ConnectionString = "DSN=proect"
```

```
.UserName = "scott"
```

```
.Password = "tiger"
```

```
.CursorLocation = adUseClient
```

```
.CursorType = adOpenStatic
```

```
.CommandType = adCmdText
```

```
.RecordSource = A
```

```
.Refresh
```

```
End With
```

```
End Sub
```



Fig. 14.5. Route details

Code Sample for Manipulating Route Details

```

Private Sub Form_Load()
Dim A As String
Set DataGrid1.DataSource = Adodc1
A = "SELECT ROUTE_NO, F_WHERE, F_TO, KMS FROM SERVICE"
With Adodc1
.ConnectionString = "DSN=proect"
.UserName = "scott"
.Password = "tiger"
.CursorLocation = adUseClient
.CursorType = adOpenStatic
.CommandType = adCmdText
.RecordSource = A
.Refresh
End With
End Sub

```

Code Sample for Manipulating Bus Details

```

Private Sub Form_Load()
Dim A As String

```

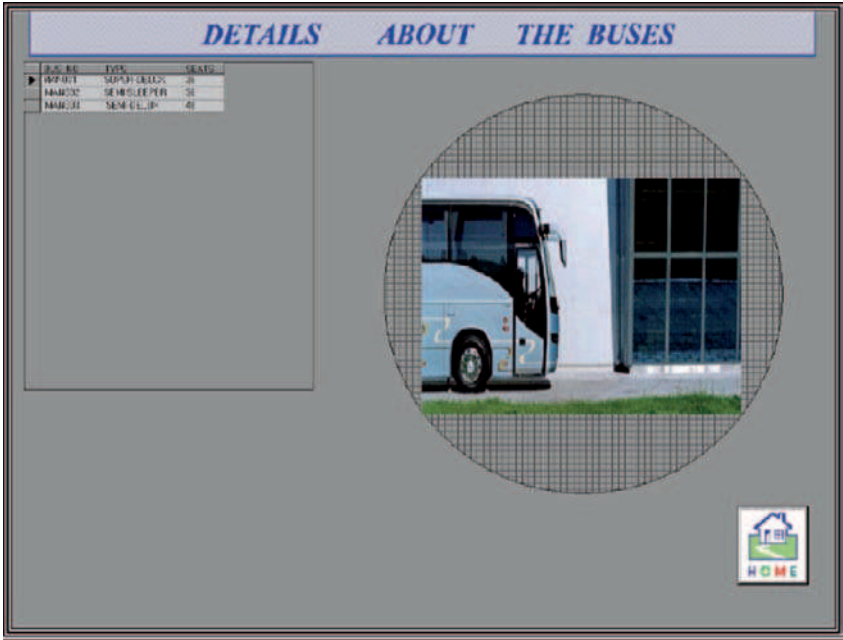


Fig. 14.6. Bus details

```
Set DataGrid1.DataSource = Adodc1
A = "SELECT bus_no, type, seats FROM busdata"
```

```
With Adodc1
    .ConnectionString = "DSN=project"
    .UserName = "scott"
    .Password = "tiger"
    .CursorLocation = adUseClient
    .CursorType = adOpenStatic
    .CommandType = adCmdText
    .RecordSource = A
    .Refresh
```

```
End With
End Sub
```

Code Sample for Manipulating Tariff Chart

```
Private Sub Form_Load()
Dim A As String
Adodc1.Visible = False
Set DataGrid1.DataSource = Adodc1
A = "SELECT F_WHERE, F_TO, KMS, ROUNDKMS*.40 AS FARE
FROM SERVICE"
```

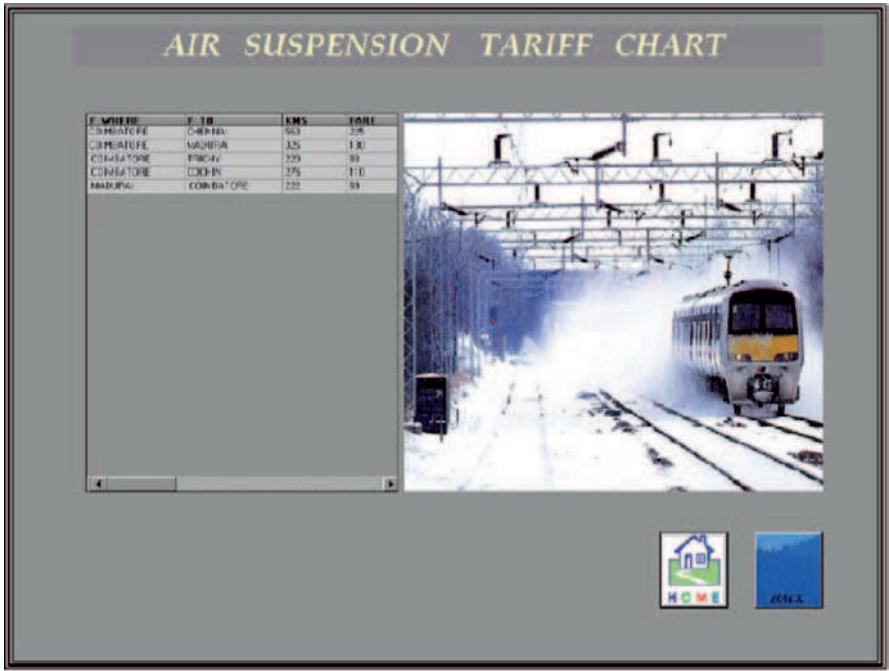



Fig. 14.7. Tariff chart

```

With Adodc1
.ConnectionString = "DSN=project"
.UserName = "scott"
.Password = "tiger"
.CursorType = adOpenStatic
.CommandType = adCmdText
.RecordSource = A
.Refresh
End With
End Sub

```

Code Sample for Manipulating Reservation Index

```

Private Sub Command1_Click()
Dim source As String
Dim dest As String

If Combo1.Text = Combo2.Text Then
MsgBox "INCORRECT DESTINATION!!", vbCritical, "ERROR
MESSAGE:"
Call Form.Activate

```

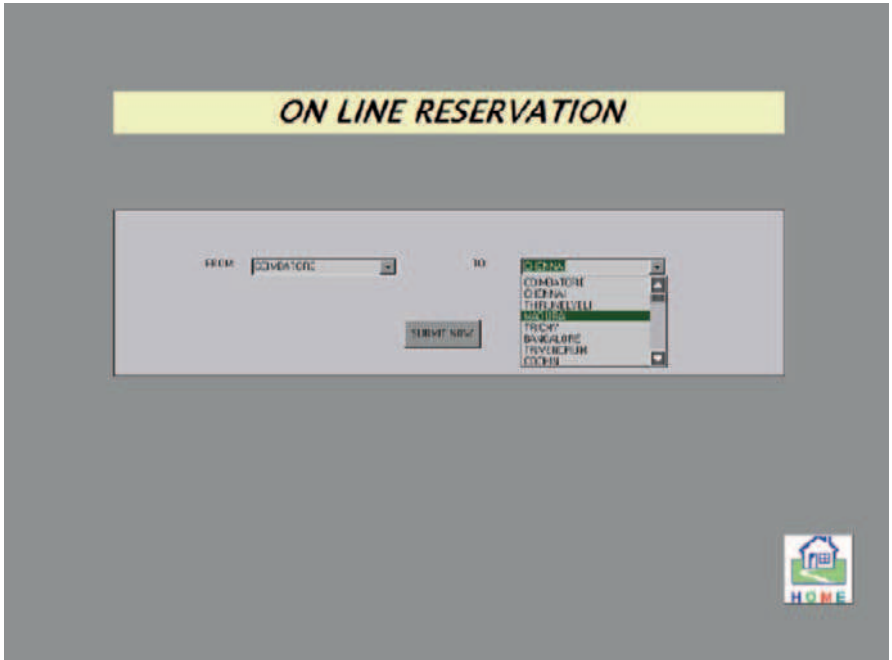


Fig. 14.8. Reservation index

Else

```
Dim A As String
DataGrid1.Visible = True
Set DataGrid1.DataSource = Adodc1
A = "SELECT SERVICE.D WHERE, SERVICE.F_ TO,
SERVICE.KMS, BUSDATA.BUS_NO FROM SERVICE,
BUSDATA WHERE (F_WHERE=" & Combo1.Text & "' AND
F_TO = " & Combo2.Text & "' AND SERVICE.S_NO
= BUSDATA.S_NO)
```

With Adodc1

```
.ConnectionString = "DSN=project"
.UserName = "scott".Password = "tiger"
.CursorLocation = adUseClient
.CursorType = adOpenStatic
.CommandType = adCmdText
.RecordSource = A
.Refresh
```

End With

If DataGrid1.ApproxCount = 0 Then

```
MsgBox "SERVICE IS NOT AVAILABLE NOW", vbInformation,
"NAREN TRAVELS"
```

```
Combo1.Visible = True
Combo2.Visible = True
Label1.Visible = True
Label2.Visible = True
Label3.Visible = True
Command1.Visible = True
Command3.Visible = False
Option1.Visible = False
Option2.Visible = False
Option3.Visible = False
Option4.Visible = False
Command4.Visible = False
Shape1.Visible = True
DataGrid1.Visible = False
Else
    source = Combo1.Text
    dest = Combo2.Text
    cn.ConnectionString = "DSN=project"
    cn.Open "DSN=project", "scott", "tiger"
    rs.Open "UPDATE DEST SET F_WHERE='& dest & '", cn,
        adOpenDynamic, adLockOptimistic, adCmdText
    rs.Open "UPDATE SOURCE SET F_TO='& source & '",
    cn, adOpenDynamic,
        adLockOptimistic, adCmdText rs.Open "COMMIT;",
    cn, adOpenDynamic, adLockOptimistic, adCmdText
    cn.Close
    Option1.Visible = True
    Option2.Visible = True
    Option3.Visible = True
    Option4.Visible = True
    Option1.Value = False
    Option2.Value = False
    Option3.Value = False
    Option4.Value = False
    Command4.Visible = True
    Command4.Enabled = False
    Combo1.Visible = False
    Combo2.Visible = False
    Label1.Visible = False
    Label2.Visible = False
    Label3.Visible = False
    Command1.Visible = False
    Command3.Visible = True
    Shape1.Visible = False
End If
```

14.4 Second Project: Course Administration System

14.4.1 Description

The primary objective of this project is to maintain the reliable data storage for “Course Administration of PSGTECH.” This project gives facility for storing the staff detail, student detail, lecture schedule detail, and updating the same.

This project uses Oracle for reliable data storage and Visual Basic 6.0 for user friendliness. Simply Oracle is used as back-end tool and Visual Basic is used as front-end tool.

Entity-relationship model is chosen for its implementation (Figs.14.9–14.15).

14.4.2 Source Code

Code Sample for Manipulating Login Details

```
Private Sub Command1_Click()
If (Trim(Text1.Text) = "EEE" Or Trim(Text1.Text) = "eee") And
Trim(Text2.Text)="eee" Then
Unload Me
Form1.Show
Else
MsgBox ("Invalid UserName/Password"), vbCritical, "INFODESK"
Text1.Text = ""
Text2.Text = ""
Text1.SetFocus
End If
End Sub
Private Sub Command2_Click()
Unload Me
End Sub
Private Sub Form_Load()
Set Skinner1.Forms = Forms
End Sub
```

Code Sample for Manipulating Academic Details

```
Dim fla As Integer
Private Sub Form_Load()
fla = 0
End Sub
Private Sub Form_Unload(Cancel As Integer)
Form3.Show
End Sub
```

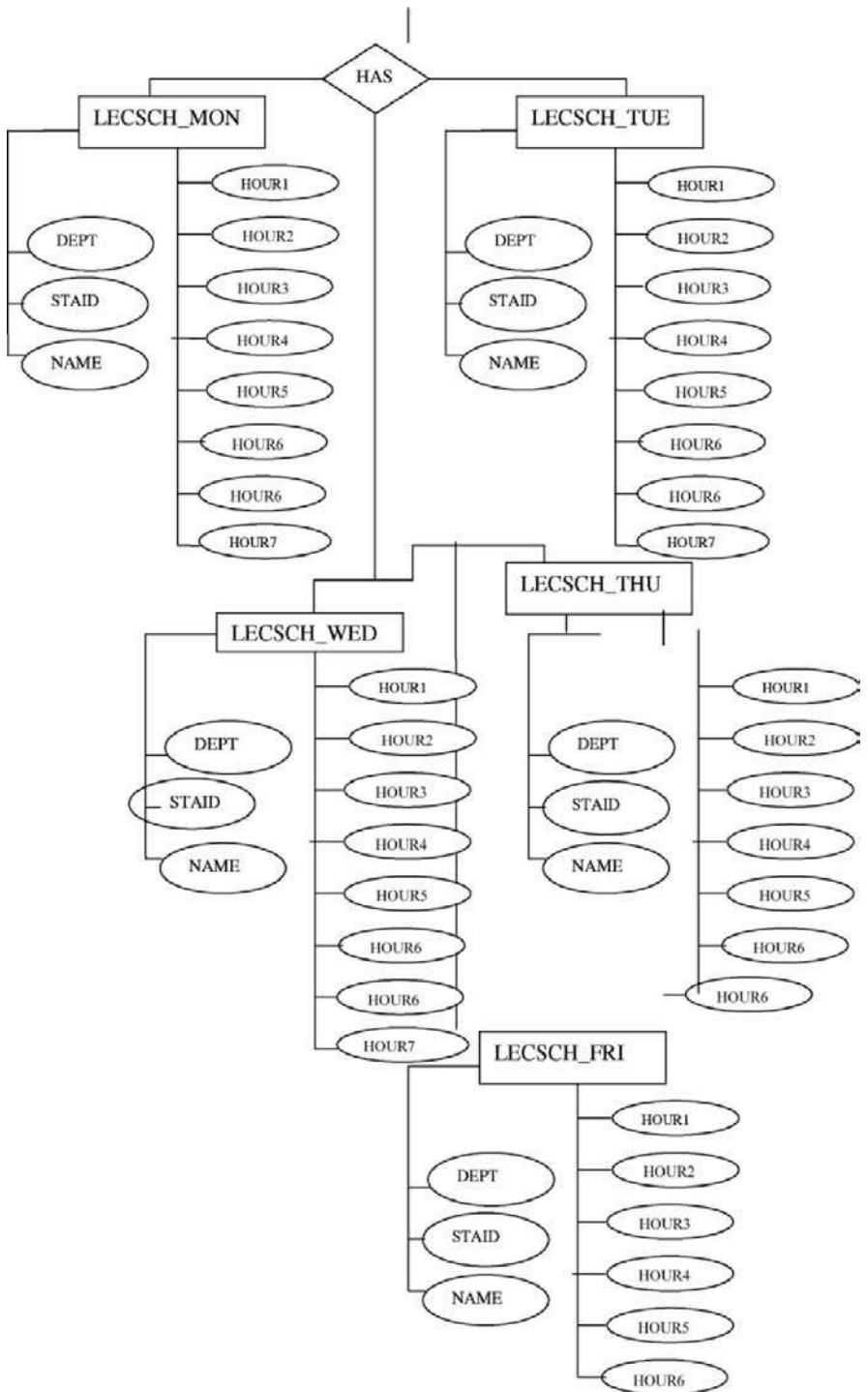


Fig. 14.9. ER diagram

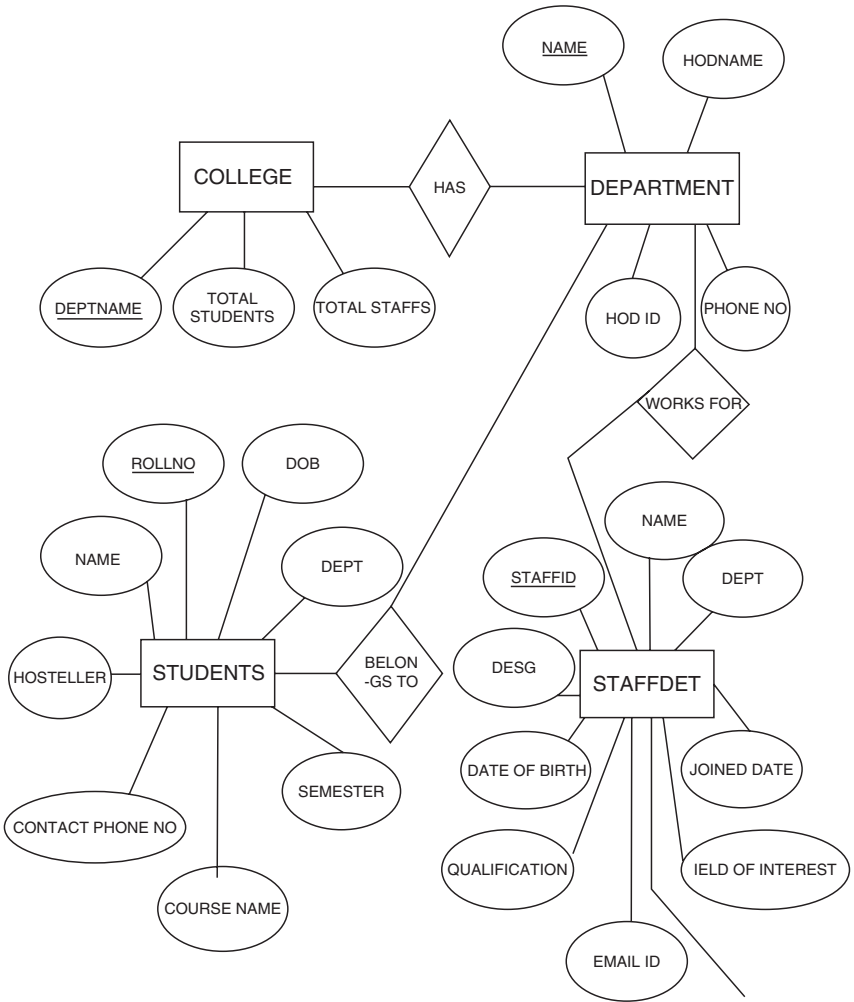


Fig. 14.9. Continued

```

Private Sub Image1_Click()
Unload Me
Form3.Show
End Sub
Private Sub Label7_Click()
Form6.Show
Me.Hide
End Sub
Private Sub Label2_Click()
MsgBox "NO DETAIL FOUND", vbInformation, "INFODESK"
End Sub
    
```

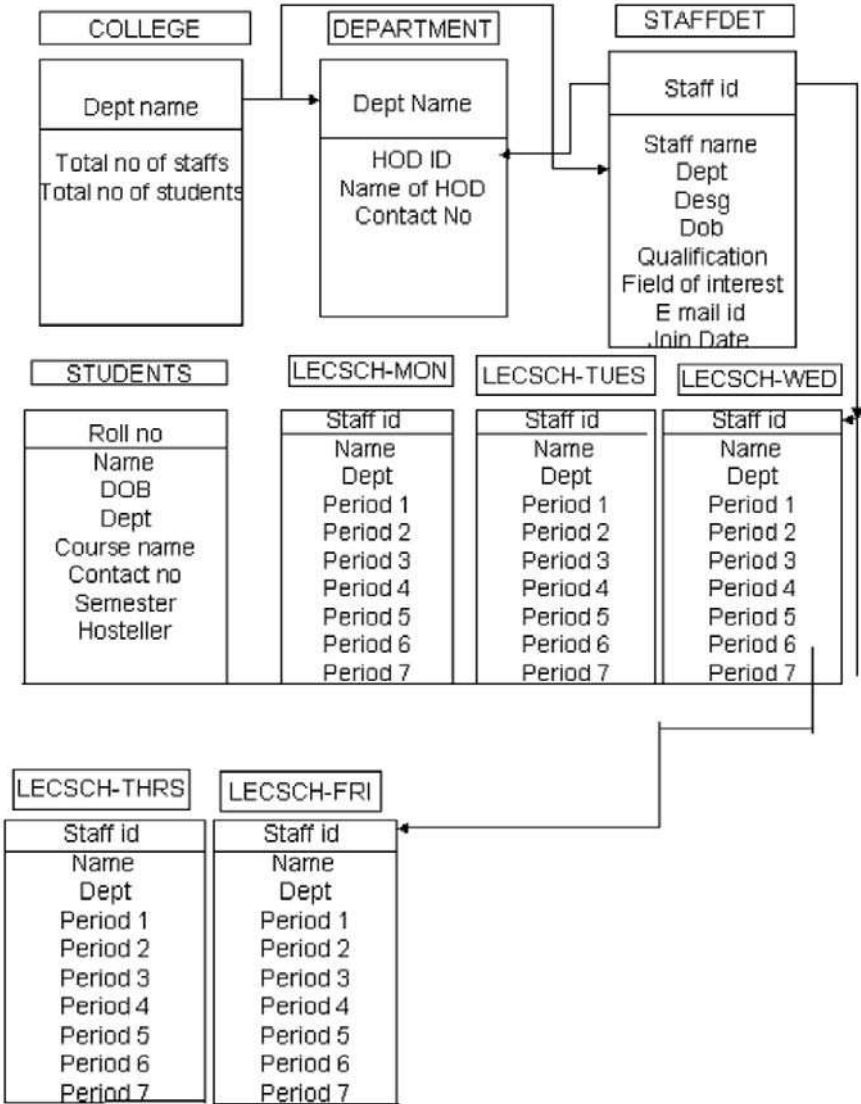


Fig. 14.10. Schema diagram

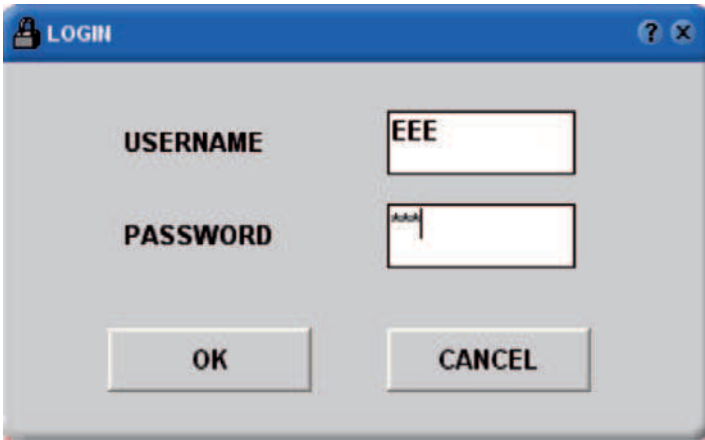


Fig. 14.11. Login details



Fig. 14.12. Academic details

The screenshot shows a web application window titled "PSG COLLEGE OF TECHNOLOGY - (EEE- STUDENT DETAIL ADDITION TO THE DATABASE)". The main heading is "STUDENT DETAIL ADDITION". The form contains the following fields and values:

ROLL NO	02E117
NAME	S KARTHKEYAN
BRANCH	EEE
COURSE NAME	DEETE1 REGULAR
SEMESTER NO	1
HOSTELLER	YES
DATE OF BIRTH	01/07/1985 (dd-mm-yyyy)
CONTACT PHONE NO	947789

Buttons at the bottom include "ADD", "CLEAR", "CANCEL", and "BACK TO XEE HOME". The footer shows "PSG COLLEGE OF TECHNOLOGY", "10:51 PM", and "3/23/2005".

Fig. 14.13. Student details

Code Sample for Manipulating Student Details

```
Dim rs1 As New ADODB.Recordset
Dim rs2 As New ADODB.Recordset
```

```
Private Sub Combo1_Click()
Command1.Enabled = True
End Sub
```

```
Private Sub Combo2_Click()
Combo1.Clear
If Combo2.ListIndex <> -1 Then
If Combo2.Text = "EEE" Then
Combo1.AddItem "BE(EEE)-REGULAR"
Combo1.AddItem "BE(EEE)-SW"
Combo1.AddItem "ME(EEE)-ELECTRICAL MACHINES"
Combo1.AddItem "ME(EEE)-POWERSYSTEM"
Combo1.AddItem "ME(EEE)-CONTROL SYSTEM"
Combo1.AddItem "ME(EEE)-APPLIED ELECTRONICS"
Else
Combo1.AddItem ("BE" & (Combo2.Text) & "-REGULAR")
```

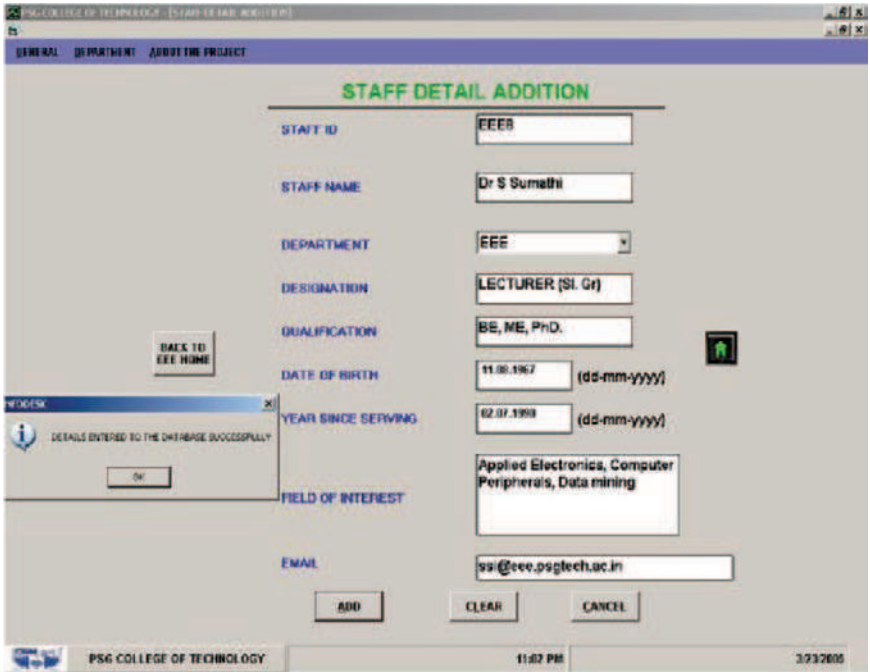


Fig. 14.14. Staff details

```
End If
End If
Combo1.SetFocus
End Sub
```

```
Private Sub Combo3_Click()
Command1.Enabled = True
End Sub
```

```
Private Sub Command1_Click()
If Trim(Text1.Text) = "" Or Trim(Text2.Text) = "" Or
Trim(Text3.Text) = "" Or
```

```
Trim(dob.Text) = "" Or
Trim(Text6.Text) = "" Or Combo1.ListIndex = -1 Or
Combo3.ListIndex = -1 Then
MsgBox "Please Enter All Details", vbInformation, "INFODESK"
Else
rs2.Open "select * from student where
rollno='&UCase(Trim(Text1.Text)) & '", db,
adOpenDynamic, adLockOptimistic
```

PSG COLLEGE OF TECHNOLOGY - (LECTURE SCHEDULE DETAIL ADDITION FOR EEE)

GENERAL DEPARTMENT ADD THE PROJECT

LECTURE SCHEDULE DETAIL ADDITION

DEPARTMENT:

STAFF ID:

STAFF NAME:

DAY SELECTION:

SAVE THE WORKING PERIOD FOR THIS STAFF ON THIS PARTICULAR DAY

PERIOD 1	PERIOD 2	PERIOD 3	PERIOD 4	PERIOD 5	PERIOD 6	PERIOD 7
E REG II YEAR	LEISURE	IV BE EEE REG	ELECTRONICS	II BE EEE SW	LEISURE	II BE EEE REG

ADD CLEAR CANCEL

BACK TO EEE HOME

PSG COLLEGE OF TECHNOLOGY 11:11 PM 3/23/2005

Fig. 14.15. Lecture schedule details

```

If rs2.EOF = True And rs2.BOF = True Then
rs1.AddNew
rs1.Fields("rollno") = UCase(Trim(Text1.Text))
rs1.Fields("name") = UCase(Trim(Text2.Text))
rs1.Fields("course_name") = UCase(Trim(Combo1.Text))
rs1.Fields("sem") = UCase(Val(Trim(Text3.Text)))
rs1.Fields("hosteller") = UCase(Trim(Combo3.Text))
rs1.Fields("dob") = UCase(Trim(dob.Text))
rs1.Fields("phone") = UCase(Val(Trim(Text6.Text)))
rs1.Fields("dept") = UCase(Trim(Combo2.Text))
rs1.Update
MsgBox "DETAILS ENTERED TO THE DATABASE SUCCESSFULLY",
vbInformation, "INFODESK"
Else
MsgBox "STUDENT ID ALREADY EXIST", vbInformation, "INFODESK"
Text1.Text = ""
Text1.SetFocus
End If
rs2.Close
End If
Command1.Enabled = False
End Sub

```

```
Private Sub Command2_Click()
Form6.Show
Unload Me
End Sub
```

Code Sample for Manipulating Staff Details

```
Dim rs1 As New ADODB.Recordset
Dim rs2 As New ADODB.Recordset
Private Sub Combo1_Click()
Command1.Enabled = True
End Sub
Private Sub Command1_Click()
If Trim(Text1.Text) = "" Or Trim(Text2.Text) = "" Or
Trim(dob.Text) = "" Or Trim(yss.Text) = "" Or
Trim(Text5.Text) = "" Or Trim(Text6.Text) = "" Or
Combo1.ListIndex = -1 Or
Trim(ID.Text) = "" Or

    Text8.Text = "" Then
MsgBox "Please Enter All Details", vbInformation, "INFODESK"
Else
    rs2.Open "select * from STAFFDET where STAFFID='" &
    UCase(Trim(ID.Text)) & "'", db, adOpenDynamic,
    adLockOptimistic
If rs2.EOF = True And rs2.BOF = True Then
rs1.AddNew
rs1.Fields(0) = UCase(Trim(ID.Text))
rs1.Fields(1) = UCase(Trim(Text8.Text))
rs1.Fields(2) = UCase(Trim(Combo1.Text))
rs1.Fields(3) = UCase(Trim(Text1.Text))
rs1.Fields(4) = UCase(Trim(Text2.Text))
rs1.Fields(5) = UCase(Trim(dob.Text))
rs1.Fields(6) = UCase(Trim(yss.Text))
rs1.Fields(7) = UCase(Trim(Text5.Text))
rs1.Fields(8) = UCase(Trim(Text6.Text))
rs1.Update
MsgBox "DETAILS ENTERED TO THE DATABASE SUCCESSFULLY",
vbInformation, "INFODESK"
Else
MsgBox "STAFF ID ALREADY EXIST", vbInformation, "INFODESK"
rs2.Close
ID.Text = ""
ID.SetFocus
```

```

End If
End If
Command1.Enabled = False
End Sub

```

Code Sample for Manipulating Lecture Schedule Details

```

Dim rs1 As New ADODB.Recordset
Dim rs2 As New ADODB.Recordset
Dim rs3 As New ADODB.Recordset
Dim rs4 As New ADODB.Recordset
Dim rs5 As New ADODB.Recordset
Dim rs6 As New ADODB.Recordset
Dim rs7 As New ADODB.Recordset
Dim rs8 As New ADODB.Recordset

```

```

Private Sub Combo1-Click()
Combo2.Clear
If Combo1.ListIndex <> -1 Then
rs1.Filter = "dept=" & Trim(Combo1.Text) & " "
If rs1.EOF = False And rs1.BOF = False Then
Do Until rs1.EOF
Combo2.AddItem UCase(rs1.Fields("staffid"))
rs1.MoveNext
Loop
Else
MsgBox "NO STAFFID EXIST", vbInformation, "INFODESK"
End If
Command1.Enabled = True
End If
End Sub

```

```

Private Sub Combo2-Click()
Combo3.Clear
Combo3.AddItem "MONDAY", 0
Combo3.AddItem "TUESDAY", 1
Combo3.AddItem "WEDNESDAY", 2
Combo3.AddItem "THURSDAY", 3
Combo3.AddItem "FRIDAY", 4
rs2.Filter = "staffid=" & Trim(Combo2.Text) & " "
If rs2.EOF = False And rs1.BOF = False Then
Text1.Text = rs2.Fields("staff_name")
End If
Command1.Enabled = True
End Sub

```

14.5 Third Project: Election Voting System

14.5.1 Description

This project provides a software for the Election Voting System to maintain the information about the voters list, candidate list, election schedule, polling process, election result, and announcement, and to access the general information about political parties, alliances, election big B's, and election cartoons. It also holds the details about eligibility of voters and election facts and figures (Figs. 14.16–14.19).

14.5.2 Source Code

Code Sample for Manipulating Candidates Details

```

Option Explicit
Dim A As New ADODB.Connection
Dim r As New ADODB.Recordset
Private Sub Command1-Click()
If Combo1 = "COIMBATORE" Then
A.Open "Provider=MSDAORA.1;Password=sathya;
User ID=SCOTT;Persist Security Info=False"
r.Open "select * from cand", A, adOpenDynamic, adLockOptimistic
    
```

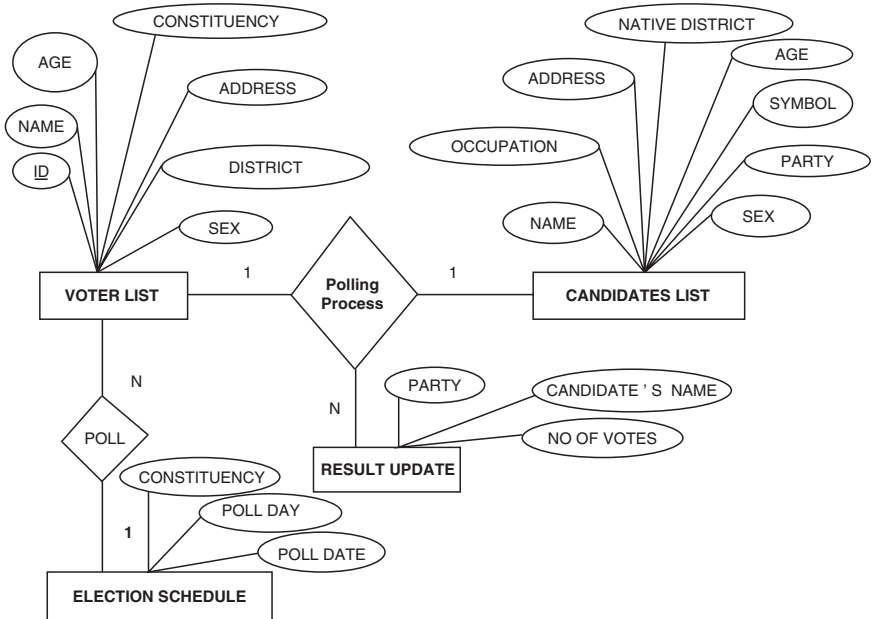


Fig. 14.16. ER diagram

CANDIDATES LIST

SELECT THE CONSTITUENCY

CANDIDATES_NAME	AGE	ADDRESS	NATI	DISTRICT	CONSTITUENCY
M. CHIDAMBARAM P	45	DGREGHTENTYKJKJKJKJKJTYJ	COIM	COIMBATORE	COIMBATORE
M. INDRA V	56	FGHGFJGFJKJKJGFNJKJRYTJYRJ	CHEP	COIMBATORE	COIMBATORE
M. INDRA V	56	VHJKKJLJKLJLMLHLJLJKLHLJL	CHEP	COIMBATORE	COIMBATORE
M. KRISNAKUMAR S	35	VGJKHKMNKLFHTHTJRYJ	COIM	COIMBATORE	COIMBATORE
M. CHANDRASEKAR L	56	GFHGFHJDTFRHKLJTYJ	MADI	CHENNAI	COIMBATORE
M. SUSHMA M	34	GHHJKLKOIFHDHGFJ	COIM	COIMBATORE	COIMBATORE
M. BALU T R	78	GHHJKJKJKJGRTGJR	COIM	COIMBATORE	COIMBATORE
M. ANTONY Y	46	DFDSGFJHKLHL	TRIC	COIMBATORE	COIMBATORE

Fig. 14.17. Candidates details

```
MSHFlexGrid1.Visible = True
Set MSHFlexGrid1.DataSource = r
MSHFlexGrid1.ColWidth(0) = 2000
MSHFlexGrid1.ColWidth(1) = 500
MSHFlexGrid1.ColWidth(2) = 4000
MSHFlexGrid1.ColWidth(3) = 500
MSHFlexGrid1.ColWidth(4) = 1500
MSHFlexGrid1.ColWidth(5) = 2000
MSHFlexGrid1.ColWidth(6) = 1700
MSHFlexGrid1.ColWidth(7) = 1750
MSHFlexGrid1.ColWidth(8) = 1500
```

```
r.Close
A.Close
```

```
ElseIf Combo1 = "MADURAI" Then
A.Open "Provider=MSDAORA.1;Password=sathya;User ID=SCOTT;
Persist Security Info=False"
r.Open "select * from candm", A, adOpenDynamic, adLockOptimistic
MSHFlexGrid1.Visible = True
Set MSHFlexGrid1.DataSource = r
MSHFlexGrid1.ColWidth(0) = 2000
MSHFlexGrid1.ColWidth(1) = 500
```

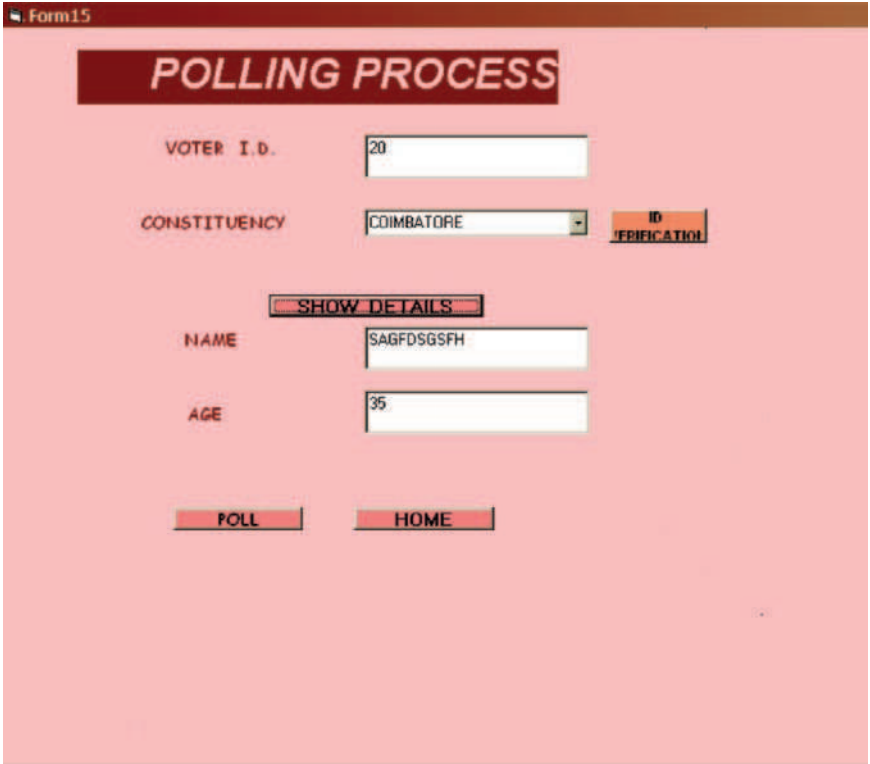


Fig. 14.18. Polling details

```
MSHFlexGrid1.ColWidth(2) = 4000
MSHFlexGrid1.ColWidth(3) = 500
MSHFlexGrid1.ColWidth(4) = 1500
MSHFlexGrid1.ColWidth(5) = 2000
MSHFlexGrid1.ColWidth(6) = 1700
MSHFlexGrid1.ColWidth(7) = 1750
MSHFlexGrid1.ColWidth(8) = 1500
```

```
r.Close
A.Close
```

```
ElseIf Combo1 = "CHENNAI" Then
A.Open "Provider=MSDAORA.1;Password=sathya;User ID=SCOTT;
Persist Security Info=False"
r.Open "select * from candm", A, adOpenDynamic, adLockOptimistic
MSHFlexGrid1.Visible = True
Set MSHFlexGrid1.DataSource = r
MSHFlexGrid1.ColWidth(0) = 2000
```


RESULT ANNOUNCEMENT		
RESULT OF COIMBATORE DISTRICT		
CANDIDATE NAME	PARTY	NO OF VOTES
Ms.INDRA.V	INDIVIDUAL	6
Mr.CHIDAMBARAM.P	JP	4
RESULT OF MADURAI DISTRICT		
CANDIDATE NAME	PARTY	NO OF VOTES
Ms.GOWRI.M	INDIVIDUAL	1
Mr.MADANKUMAR.V	BJP	0
RESULT OF CHENNAI DISTRICT		
CANDIDATE NAME	PARTY	NO OF VOTES
Mr.RAJESHKUMAR.T	NCP	1
Ms.LALITHA.M	AIADMK	1

Fig. 14.19. Results

```

MSHFlexGrid1.ColWidth(1) = 500
MSHFlexGrid1.ColWidth(2) = 4000
MSHFlexGrid1.ColWidth(3) = 500
MSHFlexGrid1.ColWidth(4) = 1500
MSHFlexGrid1.ColWidth(5) = 2000
MSHFlexGrid1.ColWidth(6) = 1700
MSHFlexGrid1.ColWidth(7) = 1750
MSHFlexGrid1.ColWidth(8) = 1500

```

```
r.Close
```

```
A.Close
```

```
End If
```

```
End Sub
```

```
Private Sub Command2_Click()
Form2.Show
End Sub
```

```
Private Sub Form_Load()
Combo1.AddItem "COIMBATORE"
Combo1.AddItem "CHENNAI"
Combo1.AddItem "MADURAI"
End Sub
```

Code Sample for Manipulating Polling Details

```
Option Explicit
Dim aw As New ADODB.Connection
Dim r2 As New ADODB.Recordset
Dim r3 As New ADODB.Recordset
Dim e As Integer
Dim i As Integer
Dim fie As Field
Private Sub Command1_Click()
aw.Open "Provider=MSDAORA.1;Password=sathya;User ID=SCOTT;
Persist Security Info=True"
r3.Open "select * from cv", aw, adOpenDynamic, adLockOptimistic
e = 0
Do While Not r3.EOF
e = e + 1
r3.MoveNext
Loop
r3.MoveFirst
For i = 0 To e - 1
If r3.Fields("SNO") = Text2.Text Then
Text1.Text = r3.Fields("NAME")
Text3.Text = r3.Fields("AGE")
Combo1.Text = r3.Fields("CONSTITUENCY")
'aw.Execute "insert into votee values(" & Text1.Text &
",," & Text2.Text & ",," & Text3.Text & ",," & Combo1.Text & ",)"
End If
If i = e Then
r3.MoveLast
Else
r3.MoveNext
End If
Next
aw.Execute "insert into votee values(" & Text1.Text & ",," &
Text2.Text & ",," & Text3.Text & ",," & Combo1.Text & ",)"
```

```
r3.Close
aw.Close
End Sub
```

```
Private Sub Command2_Click()
Form2.Show
End Sub
```

```
Private Sub Command3_Click()
If Combo1 = "COIMBATORE" Then
Form4.Show
Elseif Combo1 = "MADURAI" Then
Form3.Show
Elseif Combo1 = "CHENNAI" Then
Form14.Show
End If
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
Combo1.Text = ""
End Sub
```

```
Private Sub Form_Load()
Combo1.AddItem "COIMBATORE"
Combo1.AddItem "MADURAI"
Combo1.AddItem "CHENNAI"
End Sub
```

```
Private Sub Text2_KeyPress(KeyAscii As Integer)
If Chr(KeyAscii) = vbBack Then Exit Sub
If Not IsNumeric(Chr(KeyAscii)) Then
KeyAscii = 0
MsgBox "Enter Ur Correct ID", vbOKOnly, "Stop!"
End If
End Sub
```

Code Sample

```
Private Sub Form_Load()
a2.Open "Provider=MSDAORA.1;Password=sathya;User ID=scott;
Persist Security Info=True"
rr1.Open "select * from POLCH order by NO_OF_VOTES",
a2, adOpenDynamic, adLockOptimistic
rr1.MoveLast
Label16.Caption = rr1.Fields("CANDIDATES_NAME")
Label12.Caption = rr1.Fields("PARTY")
```

```

Label13.Caption = rr1.Fields("NO_OF_VOTES")
rr1.MovePrevious
Label11.Caption = rr1.Fields("CANDIDATES_NAME")
Label14.Caption = rr1.Fields("PARTY")
Label15.Caption = rr1.Fields("NO_OF_VOTES")
rr1.MoveLast
rr1.Close
a2.Close
a2.Open "Provider=MSDAORA.1;Password=sathya;User ID=scott;Persist
Security Info=True"
r1.Open "select * from POLMA order by NO_OF_VOTES",
a2, adOpenDynamic, adLockOptimistic
r1.MoveLast
Label17.Caption = r1.Fields("CANDIDATES_NAME")
Label19.Caption = r1.Fields("PARTY")
Label20.Caption = r1.Fields("NO_OF_VOTES")
r1.MovePrevious
Label21.Caption = r1.Fields("CANDIDATES_NAME")
Label22.Caption = r1.Fields("PARTY")
Label23.Caption = r1.Fields("NO_OF_VOTES")
r1.MoveLast
r1.Close
a2.Close
a2.Open "Provider=MSDAORA.1;Password=sathya;User ID=scott;
Persist Security Info=True"
rr1.Open "select * from res order by NO_OF_VOTES",
a2, adOpenDynamic, adLockOptimistic
rr1.MoveLast
Label3.Caption = rr1.Fields("CANDIDATES_NAME")
Label4.Caption = rr1.Fields("PARTY")
Label7.Caption = rr1.Fields("NO_OF_VOTES")
rr1.MovePrevious
Label8.Caption = rr1.Fields("CANDIDATES_NAME")
Label9.Caption = rr1.Fields("PARTY")
Label10.Caption = rr1.Fields("NO_OF_VOTES")
rr1.MoveLast
rr1.Close
a2.Close
End Sub

```

14.6 Fourth Project: Hospital Management System

14.6.1 Description

This project allows the user to enter and edit the patient's information. Since all activities are carried out online there will be less time consumption. This project has developed a design for the same. The entity-relationship diagram of this project shows common roles and responsibilities of the entities that provide the system's architecture.

The project is implemented using the Oracle 9i and Visual Basic 6.0. This software provides the entire information about the hospital and patient. It also allows us to view various details like patient's information, doctor in charge, staffs, and information about institution. The different modules like information center and enquiry center are developed in the front-end Visual Basic.

Corresponding tables are developed in the back-end and the connectivity is established. The analysis and feasibility study gives the entire information about the project (Figs. 14.20–14.24).

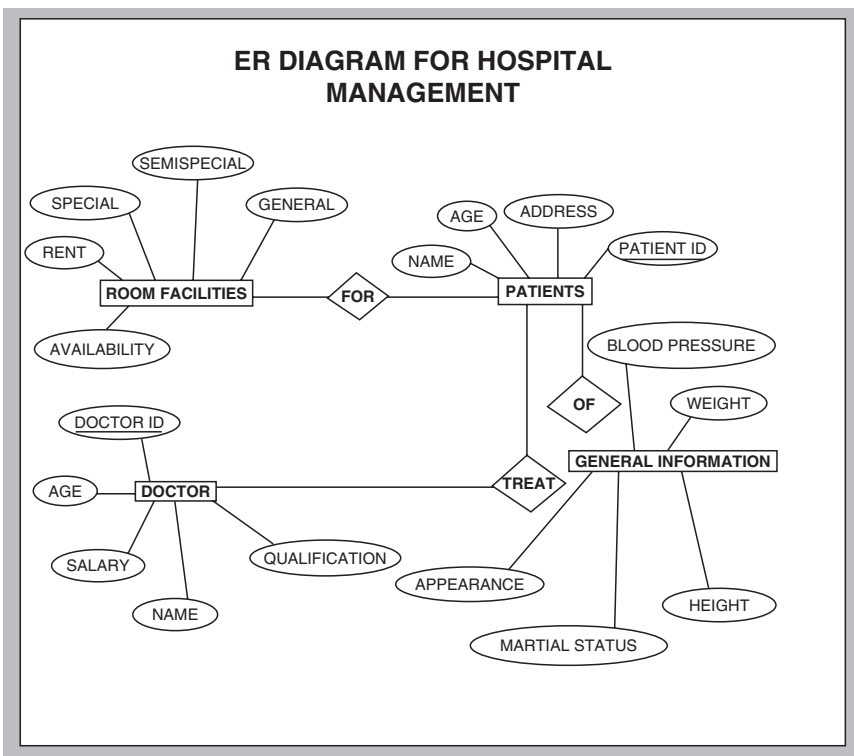


Fig. 14.20. ER diagram



Fig. 14.21. Blood donor's details

14.6.2 Source Code

Sample Code for Manipulating Blood Donor's Details

```

Private Sub Command2_Click()
Me.Hide
MDIForm1.Show
End Sub

Private Sub Command3_Click()
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
Text5.Text = ""
End Sub

Private Sub Command4_Click()
Dim st As String
If Adodc1.Recordset.RecordCount = 0 Then
Exit Sub

```

STAFF INFORMATION

STAFF ID NO: 1008 AGE: 32

NAME OF STAFF: SUDHAKAR QUALIFICATION: BDS

DEPARTMENT: DENTAL DATE OF JOINING: 12

GENDER: MALE FEMALE SALARY: 9000

VIEW NEW DELETE HOME

STAFF INFO							
ID	NAME	DEPT	AGE	QUALIFICATION	JERDATE	SALARY	GENDER
1008	SUDHAKAR	DENTAL	32	BDS	12	9000	MALE

<|>|>|>|>|>

Fig. 14.22. Staff details

```

End If
If Adodc1.Recordset.EOF = True Then
    Adodc1.Recordset.MoveFirst
End If
Adodc1.Recordset.Delete adAffectCurrent
Adodc1.Refresh
Dim ans As Integer

Private Sub Command1_Click()
If Trim(Text1.Text) = "" Or Trim(Text2.Text) =
"" Or Trim(Text3.Text) =
"" Or Trim(Text4.Text) = "" Or Trim(Text5.Text) = "" Then
    MsgBox "Please Enter All Details", vbOKOnly, "Information"
    Text1.SetFocus
End If
With Adodc1
.RecordSource = "BLGR"
.Recordset.AddNew
.Recordset.Fields(0) = Trim(Text1.Text)
.Recordset.Fields(1) = Val(Trim(Text2.Text))
.Recordset.Fields(2) = Val(Trim(Text3.Text))
.Recordset.Fields(3) = Trim(Text4.Text)

```

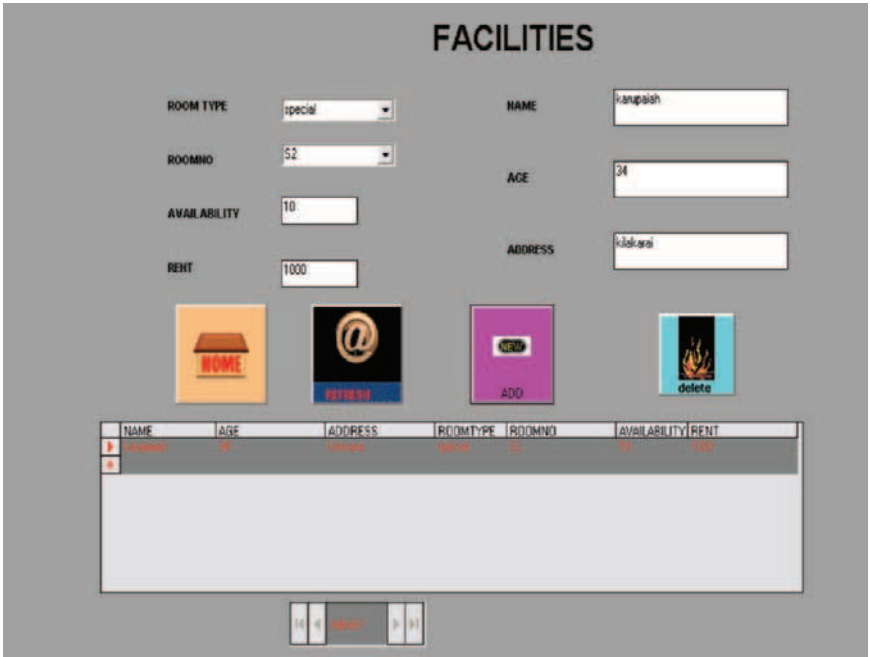


Fig. 14.23. Facilities

```
.Recordset.Fields(4) = Trim(Text5.Text)
.Recordset.Update
.Refresh
End With
End Sub
```

```
Private Sub Form_Activate()
Text1.SetFocus
End Sub
```

```
Private Sub Form_Load()
bloodbank.Enabled = True
If Button = 2 Then
    PopupMenu mnudisp, vbpopupmenurightbutton
End If
End Sub
```

Code Sample for Manipulating Staff Details

```
Private Sub Command1_Click()
With Adodc1
.Recordset.MoveFirst
End Sub
```


General Patientinfo doctors facilities window





PATIENT INFORMATION

ROOM TYPE

ROOM ALLOTTED

DATE OF ADMISSION

DOCTOR INCHARGE

ROOMTYPE	ROOMALLOT	JOINDATE	DOCTERID
special	c2	12jan2004	st102

Information

Fig. 14.24. Patient details

```

Private Sub Command2.Click()
If Trim(Text1(0).Text) = "" Or Trim(Text1(2).Text) =
"" Or Trim(Text1(3).Text) = "" Or Trim(Text1(4).Text) =
"" Or Trim(Text1(5).Text) = "" Or Trim(Text1(6).Text) = "" Then
MsgBox "Please Enter All Details", vbOKOnly, "patient"
Exit Sub
End If
With Adodc1
.RecordSource = "staff"
.Recordset.AddNew
.Recordset.Fields(0) = Text1(0).Text
.Recordset.Fields(1) = Text1(1).Text
.Recordset.Fields(2) = Text1(2).Text
.Recordset.Fields(3) = Text1(3).Text
.Recordset.Fields(4) = Text1(4).Text
.Recordset.Fields(5) = Text1(5).Text
.Recordset.Fields(6) = Text1(6).Text
If Option1.Value = True Then

```

```
.Recordset.Fields(7) = "MALE"
Else
.Recordset.Fields(7) = "FEMALE"
End If
.Recordset.Update
End With
End Sub
```

```
Private Sub Command3_Click()
With Adodc1
.RecordSource = "staff"
.Recordset.Delete adAffectCurrent
.Recordset.Update
End With
End Sub
```

```
Private Sub Command4_Click()
Me.Hide
MDIForm1.Show
End Sub
```

```
Private Sub Command5_Click()
With Adodc1
.Recordset.MovePrevious
End Sub
```

```
Private Sub Command6_Click()
With Adodc1
.Recordset.MoveNext
End Sub
```

Code Sample for Manipulating Facilities

```
Private Sub Form_Load()
    db.ConnectionString = "DSN=patient"
    db.Open "DSN=patient", "scott", "tiger"
    rs.Open "roomlist", db, adOpenDynamic, adLockOptimistic,
adCmdTable
    Combo1.Text = ""
    Combo1.AddItem "special"
    Combo1.AddItem "semi-special"
    Combo1.AddItem "general ward"
    Combo1.ListIndex = 0
End Sub
```

```
Private Sub Command1_Click()
    If Trim(Text2.Text) = "" Or Trim(Text3.Text) =
```

```

"" Or Trim(Text4.Text) = "" Then
MsgBox "Please Enter All Details", vbExclamation,
"Information"
Text2.SetFocus
Exit Sub
End If
With Adodc1
.RecordSource = "roomlist"
.Recordset.AddNew
.Recordset.Fields(0) = Trim(Text2.Text)
.Recordset.Fields(1) = Val(Trim(Text3.Text))
.Recordset.Fields(2) = Trim(Text4.Text)
.Recordset.Fields(3) = Combo1.Text
.Recordset.Fields(4) = Combo2.Text
.Recordset.Fields(5) = Val(Trim(Text1.Text))
.Recordset.Fields(6) = Trim(Text5.Text)
.Recordset.Update
.Refresh
End With
End Sub

```

```

Private Sub Form_Unload(Cancel As Integer)
rs.Close
Set rs = Nothing
db.Close
Set db = Nothing
End Sub

```

Code Sample for Manipulating Patient Details

```

Private Sub Form_Load()
cn.ConnectionString = "DSN=patient"
cn.Open "DSN=bulletin", "scott", "tiger"
rs.Open "stin", cn, adOpenDynamic, adLockOptimistic, adCmdTable
Do While Not rs.EOF
Combo2.AddItem rs.Fields(0)
rs.MoveNext
Loop
Combo1.AddItem "special"
Combo1.AddItem "general"
End Sub

Private Sub Form_Unload(Cancel As Integer)
rs.Close
cn.Close
Set cn = Nothing

```

```
Set rs = Nothing
End Sub
```

```
Private Sub Text1_LostFocus(Index As Integer)
With Adodc1
End With
End Sub
```

14.7 Fifth Project: Library Management System

14.7.1 Description

The primary objective of this project is to design a Library Database Management System to store and maintain the various details of the books, journals, and magazines available in library. It also involves additional features like staff and student databases which are important to maintain records of materials available and lent. This software is developed using Oracle as back-end and Visual Basic as front-end tool. This project is implemented by using entity-relationship model for its implementation.

This project gives the details about the library, staff, and student records. This project has been carried out with a view to provide students, staff, and all other concerned people with an easy way to access the library. As an example, we can retrieve information regarding book status, staff, or student profiles concerned (Figs. 14.25–14.31).

14.7.2 Source Code

Code Sample for Manipulating Login Details

```
Dim con As New ADODB.Connection
Dim rs As New ADODB.Recordset
Dim i As Integer
Dim k As Integer
Private Sub cmdCancel_Click()
    Timer1.Enabled = False
    ProgressBar1.Value = 0
End Sub
Private Sub cmdOK_Click()
    Timer1.Enabled = True
End Sub
```

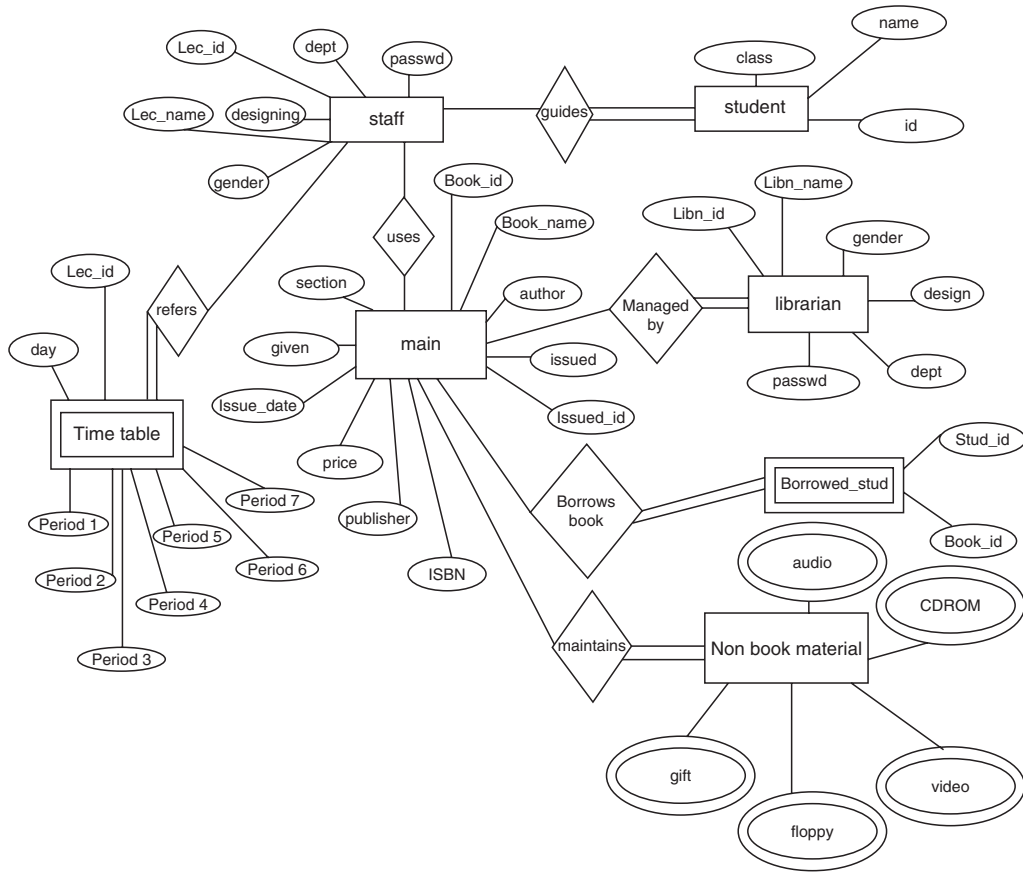


Fig. 14.25. ER diagram

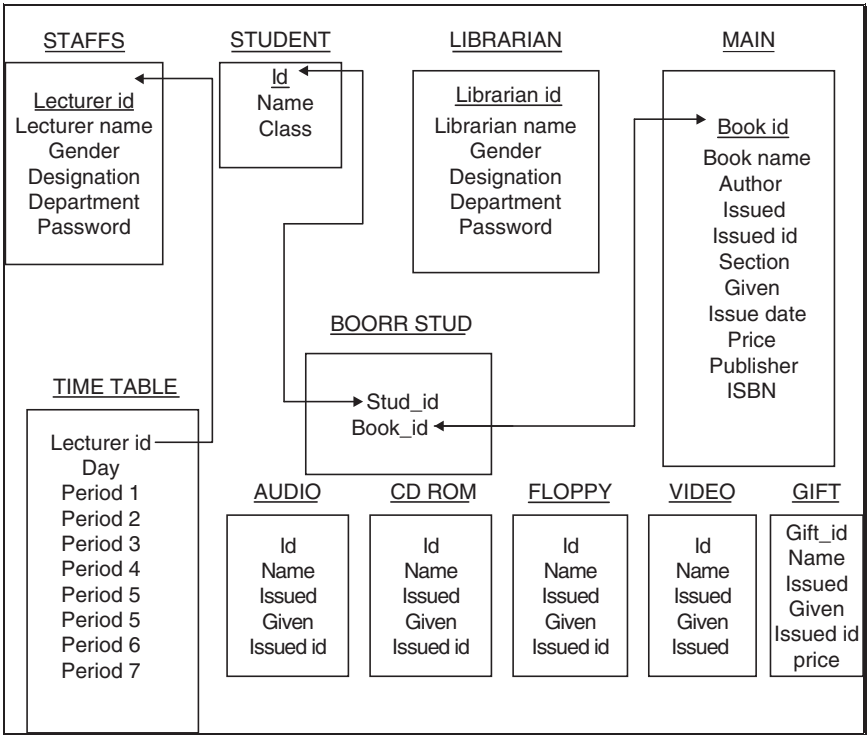


Fig. 14.26. Schema diagram



Fig. 14.27. Login details



The screenshot shows a dialog box titled "Add New Staff Details" with a close button in the top right corner. It contains five text input fields: "Enter ID" with the value "1", "Lecturer Name" with "STAFF", "Designation" with "BE", "Department" with "EEE", and "Password" with "***". To the right of these fields is a "Sex" section with two radio buttons: "Male" (unselected) and "Female" (selected). At the bottom, there are three buttons: "Cancel", "Save", and "Add New".

Fig. 14.28. Staff details: addition



The screenshot shows a dialog box titled "Staff Detail Modification Form" with a close button in the top right corner. It contains five text input fields: "Enter ID" with the value "2", "Lecturer Name" with "STAFF2", "Designation" with "BE", "Department" with "EEE", and "Password" with "***". To the right of these fields is a "Sex" section with two radio buttons: "Male" (selected) and "Female" (unselected). Below the "Sex" section is a button labeled "Click Or Press Enter". At the bottom, there are three buttons: "Save", "Delete", and "Cancel".

Fig. 14.29. Staff details: modification

Add Librarian Details

Enter ID:

Librarian Name:

Designation:

Department:

Password:

Sex:
 Male
 Female

Fig. 14.30. Librarian details

Add New Book Details

Enter The Book ID:

Enter The Book Name:

Enter The Author Name:

Enter The Section:

Enter The Price:

Enter The Publisher:

Enter The ISBN:

Fig. 14.31. Book details


```
Private Sub Command1_Click()  
Unload Me  
uid = "Administrator"  
frmLogin3.Show  
End Sub
```

```
Private Sub Form_Activate()  
txtUserName = ""  
txtPassword = ""  
txtUserName.SetFocus  
End Sub
```

```
Private Sub Form_Load()  
con.ConnectionString = Oracledsn  
con.Open Oracledsn, Oracleuser, Oraclepass  
rs.Open "staffs", con, adOpenDynamic, adLockOptimistic, adCmdTable  
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)  
con.Close  
End Sub
```

```
Private Sub Timer1_Timer()  
k = k + 3  
If k > 100 Then  
If txtUserName = "" Or txtPassword = "" Then  
MsgBox "The boxes should not be empty",  
vbExclamation, "Periods"  
txtUserName.SetFocus  
Timer1.Enabled = False  
GoTo last  
End If  
rs.Filter = "Lecturer_ID=" & txtUserName &  
" and Password=" & txtPassword & ""  
If rs.BOF = False And rs.EOF = False Then  
uid = txtUserName  
lib.Show  
Unload Me  
Timer1.Enabled = False  
k = 0  
ProgressBar1.Value = 0  
Else  
MsgBox "Username or password may be wrong or account may  
not exist contact administrator. Re-enter", vbExclamation,  
"Periods"  
Timer1.Enabled = False
```

```

        txtPassword.SetFocus
        ProgressBar1.Value = 0
        k = 0
    End If
last:
Else
ProgressBar1.Value = k
End If
End Sub

Private Sub txtPassword_KeyPress(KeyAscii As Integer)
If KeyAscii = 13 Then
If txtUserName = "" Or txtPassword = "" Then
    MsgBox "The boxes should not be empty", vbExclamation,
    "Periods"
    txtUserName.SetFocus
    GoTo last
End If
    rs.Filter = "Lecturer_ID=" & txtUserName &
    " and Password=" & txtPassword & ""
last:
End If
End Sub

```

Code Sample for Manipulating Staff Details

```

Dim db As New ADODB.Connection
Dim rs1 As New ADODB.Recordset
Dim rs2 As New ADODB.Recordset

Private Sub Command1_Click()
ProgressBar1.Value = 0
If Trim(Text1.Text) = "" Or Trim(Text2.Text) =
"" Or Trim(Text3.Text) = "" Or Trim(Text4.Text) =
"" Or Trim(Text5.Text) = "" Then
    MsgBox "Please Enter All The Data", vbInformation,
    "Information"
    Text1.SetFocus
    Exit Sub
End If

rs1.Filter = "Lecturer_ID=" & Trim(UCCase(Text1)) & ""
If rs1.EOF = True And rs1.BOF = True Then
Else
    MsgBox "ID already exist. Re-enter", vbInformation,

```

```

“Periods”
    Text1 = “”
    Text1.SetFocus
    Exit Sub
End If
ProgressBar1.Value = 50
    rs2.AddNew
    rs2.Fields(0) = Trim(UCase(Text1.Text))
    rs2.Fields(1) = Trim(UCase(Text2.Text))
If Option1.Value = True Then
    rs2.Fields(2) = “M”
Else
    rs2.Fields(2) = “F”
End If
    rs2.Fields(3) = Trim(UCase(Text3.Text))
    rs2.Fields(4) = Trim(UCase(Text4.Text))
    rs2.Fields(5) = Trim(UCase(Text5.Text))
    rs2.Update
    rs1.Close
    rs1.Open “staffs”, db, adOpenDynamic, adLockOptimistic,
adCmdTable

ProgressBar1.Value = 99
MsgBox “Details Added Successfully”, vbInformation,
“Information”
ProgressBar1.Value = 0
End Sub

Private Sub Command2.Click()
Unload Form2
End Sub

Private Sub Command3.Click()
ProgressBar1.Value = 0
Option1.Value = True
Text1.Text = “”
Text2.Text = “”
Text3.Text = “”
Text4.Text = “”
Text5.Text = “”
End Sub

```

Code Sample for Manipulating Librarian Details

```

Dim db As New ADODB.Connection
Dim rs1 As New ADODB.Recordset

```

```

Dim rs2 As New ADODB.Recordset
Private Sub Command1_Click()
ProgressBar1.Value = 0
If Trim(Text1.Text) = "" Or Trim(Text2.Text) =
"" Or Trim(Text3.Text) = "" Or Trim(Text4.Text) =
"" Or Trim(Text5.Text) = "" Then
MsgBox "Please Enter All The Data", vbInformation,
"Information"
Text1.SetFocus
Exit Sub
End If
rs1.Filter = "librarian_id=" &
Trim(UCase(Text1)) & ""
If rs1.EOF = True And rs1.BOF = True Then
Else
MsgBox "ID already exist. Re-enter", vbInformation,
"Periods"
Text1 = ""
Text1.SetFocus
Exit Sub
End If
ProgressBar1.Value = 50
rs2.AddNew
rs2.Fields(0) = Trim(UCase(Text1.Text))
rs2.Fields(1) = Trim(UCase(Text2.Text))
If Option1.Value = True Then
rs2.Fields(2) = "M"
Else
rs2.Fields(2) = "F"
End If
rs2.Fields(3) = Trim(UCase(Text3.Text))
rs2.Fields(4) = Trim(UCase(Text4.Text))
rs2.Fields(5) = Trim(UCase(Text5.Text))
rs2.Update
rs1.Close
rs1.Open "librarian", db, adOpenDynamic,
adLockOptimistic, adCmdTable
ProgressBar1.Value = 99
MsgBox "Details Added Successfully", vbInformation,
"Information"
ProgressBar1.Value = 0
End Sub

```

Code Sample for Manipulating Book Details

```

Dim db As New ADODB.Connection
Dim rs1 As New ADODB.Recordset
Dim rs2 As New ADODB.Recordset
Private Sub Command1.Click()
ProgressBar1.Value = 0
If Trim(Text1.Text) = "" Or Trim(Text2.Text) =
"" Or Trim(Text3.Text) = "" Or Trim(Text4.Text) =
"" Or Trim(Text5.Text) = "" Or Trim(Text6.Text) =
"" Or Trim(Text7.Text) = "" Then
MsgBox "Please Enter All The Data", vbInformation,
"Information"
Text1.SetFocus
Exit Sub
End If
rs1.Filter = "book_id=" &
Trim(UCase(Text1)) & ""
If rs1.EOF = True And rs1.BOF = True Then
Else
MsgBox "ID already exist. Re-enter", vbInformation,
"Periods"
Text1 = ""
Text1.SetFocus
Exit Sub
End If
ProgressBar1.Value = 50
rs2.AddNew
rs2.Fields(0) = Trim(UCase(Text1.Text))
rs2.Fields(1) = Trim(UCase(Text2.Text))
rs2.Fields(2) = Trim(UCase(Text3.Text))
rs2.Fields(3) = "NO"
rs2.Fields(4) = "NIL"
rs2.Fields(5) = Trim(UCase(Text4.Text))
rs2.Fields(6) = "NIL"
rs2.Fields(7) = "NIL"
rs2.Fields(8) = Trim(UCase(Text5.Text))
rs2.Fields(9) = Trim(UCase(Text6.Text))
rs2.Fields(10) = Trim(UCase(Text7.Text))
rs2.Update
rs1.Close
rs1.Open "main", db, adOpenDynamic, adLockOptimistic,
adCmdTable
ProgressBar1.Value = 99

```

```
MsgBox "Details Added Successfully", vbInformation,
"Information"
ProgressBar1.Value = 0
End Sub
```

14.8 Sixth Project: Railway Management System

14.8.1 Description

The main aim of this project is to allow the clients to gather information regarding railways and to book and cancel the tickets online. This project has been designed in such a way that all the activities are carried out online. This enhances the speed of the project which leads to less time consumption.

The project is conceptually viewed using the entity-relationship diagram which shows common roles and responsibilities of the entities that provide the system's architecture. The actual implementation of the project is done using relational model with Oracle8i as the back-end and Visual Basic 6.0 as the front-end.

The different modules like information center, enquiry center, and reservation and cancellation center are developed in the front-end and the corresponding tables are developed in the back-end. Finally the connectivity is established. The analysis and feasibility study gives the entire information about the project (Figs. 14.32–14.36).

14.8.2 Source Code

Code Sample for Viewing Train Details

```
Option Explicit
Private Sub Picture1_Click()
Form1.Show
Form3.Hide
End Sub
```

Code Sample for Manipulating Reservation Details

```
Private Sub Command1_Click()
Dim a As Integer
If Trim(Combo1.Text) = "" Then
Exit Sub
End If
If Trim(Text3.Text) = "" Or Trim(Text5.Text) =
"" Or Trim(Text6.Text) = "" Or Trim(Text10.Text) =
"" Or Trim(Text11.Text) = "" Or Trim(Combo2.Text) =
```

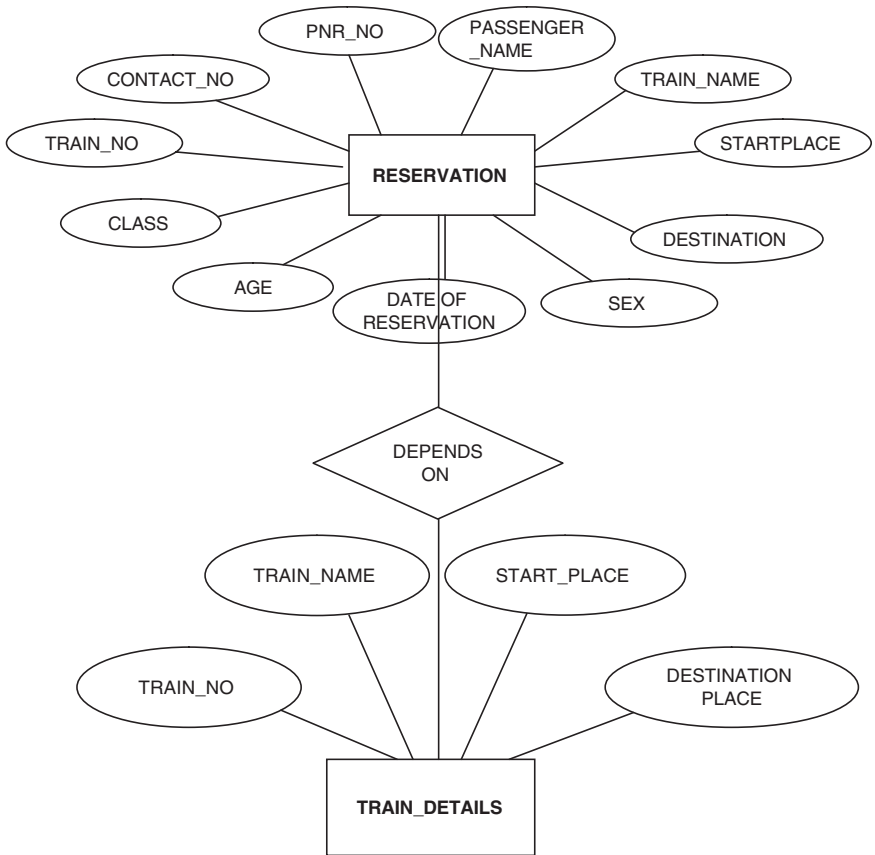


Fig. 14.32. ER diagram

"" Then

MsgBox "Please Enter All Details"

Text3.SetFocus

Exit Sub

End If

Adodc1.CommandType = adCmdUnknown

Adodc1.RecordSource = "select * from reservation"

Adodc1.Refresh

If Adodc1.Recordset.RecordCount = 0 Then

a = 1

Else

Adodc1.CommandType = adCmdUnknown

Adodc1.RecordSource = "select max(pnr) from reservation"



Fig. 14.33. Train details

```

Adodc1.Refresh
a = Adodc1.Recordset.Fields(0) + 1
End If

rs4.AddNew
rs4.Fields(0) = Trim(Combo1.Text)
rs4.Fields(1) = Trim(Text1.Text)
rs4.Fields(2) = Trim(Text2.Text)
rs4.Fields(3) = Trim(Text7.Text)
rs4.Fields(4) = Val(Trim(Text3.Text))
If Option3.Value = True Then
    rs4.Fields(5) = "M"
Else
    rs4.Fields(5) = "F"
End If

rs4.Fields(6) = Trim(Combo2.Text)
rs4.Fields(7) = Trim(Text10.Text)
rs4.Fields(8) = Trim(Text11.Text)
rs4.Fields(9) = a
rs4.Update
    
```


The screenshot shows a web-based reservation form for Southern Railways. The form is titled 'SOUTHERN RAILWAYS' and contains several input fields and buttons. The fields are: Train No. (613), Start Place (CHENNAI EGMORE), AGE (20), Train Name (TUTICORIN EXP), Destination Place (TUTICORIN), PASSENGER TYPE (Gory), SEX (male selected), ENTER THE CLASS (1), PASSENGER (1), RAC (0), ENTER THE DATE IN THE FORMAT (12-SEP-2003) (12-dec-2005), COUPON NUMBER (298765), and RESIDENTIAL ADDRESS (Tutucorin). An 'Information' dialog box is overlaid on the form, displaying a warning icon and the message 'Successfully Reserved' with an 'OK' button. At the bottom, there are buttons for 'home', 'RESERVE', and 'REBET'.

Fig. 14.34. Reservation details

```
rs4.Close
rs4.Open "reservation", db, adOpenDynamic, adLockOptimistic,
adCmdTable
MsgBox "Successfully Reserved", vbExclamation,
"Information"
End Sub
```

```
Private Sub Command2_Click()
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
Text7.Text = ""
Text11.Text = ""
Text10.Text = ""
Text5.Text = ""
Text6.Text = ""
Option2.Value = True
Option3.Value = False

End Sub
```

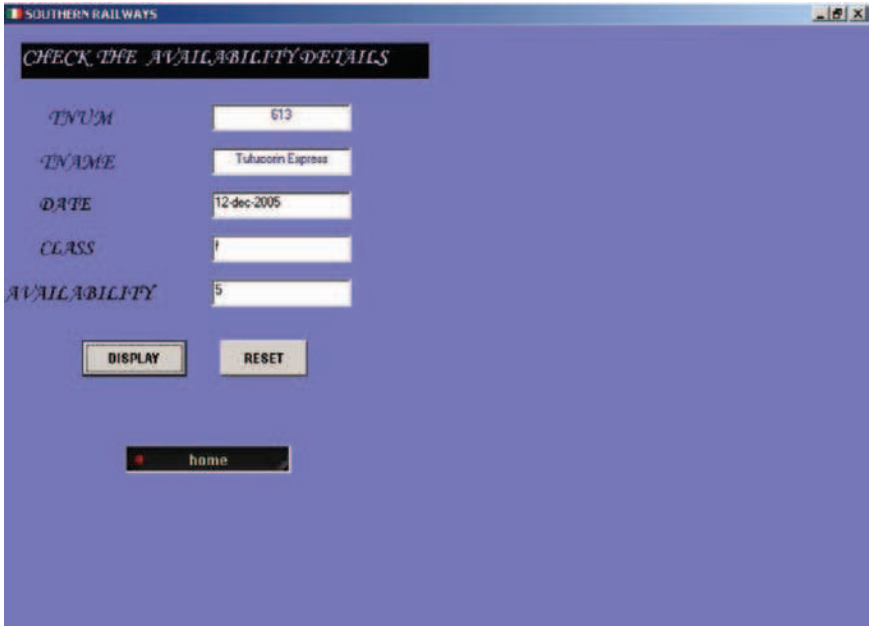


Fig. 14.35. Availability details

Code Sample for Viewing Availability Details

```
Dim con As New ADODB.Connection
Dim rs As New ADODB.Recordset
```

```
Private Sub Command1.Click()
Text1.Text = ""
Text3.Text = ""
Text2.Text = ""
Text4.Text = ""
Text5.Text = ""
End Sub
```

```
Private Sub Command2.Click()
con.Open "Provider=MSDAORA.1;Password=tiger;User
ID=system;Persist Security Info=True"
Dim str As String
str = "select count(*) from reservation where tno=" &
CInt(Trim(Text3.Text)) & " and class=" &
(Trim(Text2.Text)) & ""
Set rs = con.Execute(str)
Dim i As Integer
```

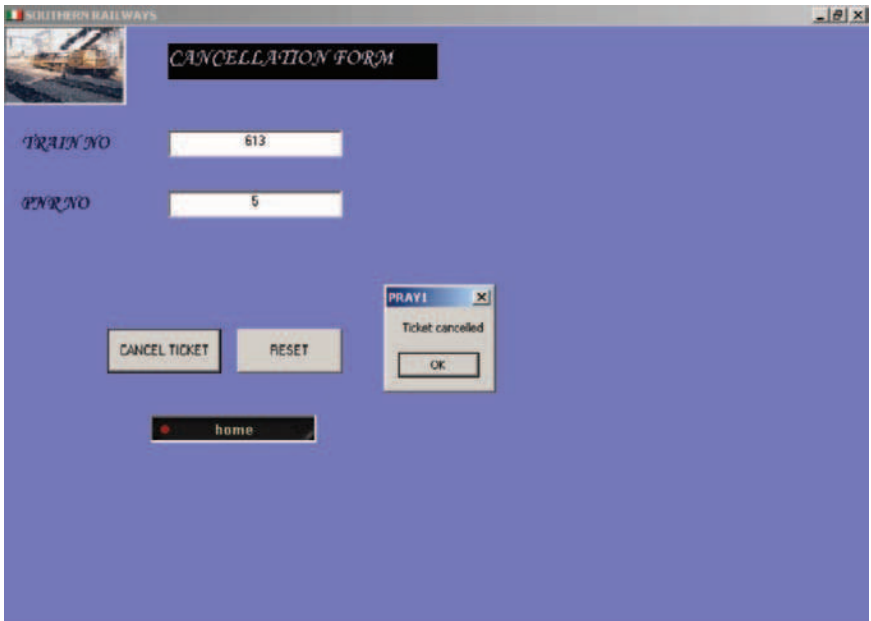


Fig. 14.36. Cancellation details

```
i = rs.Fields(0)
Text5.Text = CStr(5 - i)
con.Close
End Sub
```

```
Private Sub Picture1.Click()
Form1.Show
Form5.Hide
End Sub
```

Code Sample for Manipulating Cancellation Details

```
Option Explicit
Dim con As New ADODB.Connection
Dim rs As New ADODB.Recordset
Private Sub Picture2.Click()
End Sub
```

```
Private Sub Picture3.Click()
End Sub
```

```

Private Sub Command1_Click()
rs.Filter = "pnr = " &
Trim(Text2.Text) & ""
If rs.EOF = False And rs.BOF = False Then
rs.Delete
rs.Update
MsgBox "Ticket canceled"
End If
End Sub

```

```

Private Sub Command2_Click()
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
End Sub

```

```

Private Sub Form_Load()
con.Open "DSN=railway", "system", "tiger"
rs.Open "reservation", con, adOpenDynamic, adLockOptimistic,
adCmdTable
End Sub
Private Sub Picture1_Click(Index As Integer)
Form8.Hide
Form1.Show
End Sub

```

14.9 Some Hints to Do Successful Projects in DBMS

Class projects are slightly different from real-world applications, but they have many features in common. One of the most challenging aspects is that any project contains a level of uncertainty and ambiguity. In real-life situations, the problems are solved through experience and discussions with project manager. With class projects, the students can get some advice from the faculty members, but they need to make their own decisions and interpretations many times.

The desired steps that students should take during the initial phase of the projects are:

- Identify the goals and objectives of the proposed project.
- Additional research of the industry and similar firms will help to get an overall idea of the project.
- After collecting sufficient details develop the conceptual idea of the project by developing the ER model of the project.
- The ER model will help to analyze individual forms and reports. It is also necessary to identify the overall purpose of each form. The students should be able to describe the purpose of each form.

The following steps the students should consider during the implementation phase:

- After collecting sufficient details about the project the next step in the implementation phase is to select proper front-end, the back-end, and suitable interface.
- Some of the front-end the students can opt for is *Visual Basic* and *Power Builder*.
- As the back-end, the students can select either *SQL* or *ACCESS*. The students can also go for Oracle forms and reports.
- The students should try to develop good normalized list before creating tables using *SQL* or *ACCESS*.
- Start with an initial set of tables and keys that correct. Add columns and tables as you need them. If your initial tables are correct then you should be able to add new columns and tables without altering the existing design.
- While developing forms, take care that the forms are user-friendly. At the same time, the user should not alter important data (secret data). For this, make use of the concept of “views” wherever necessary.
- Do not forget to take backup copy of your work periodically. Always keep backup copy of your project on a different disk.

A

Dictionary of DBMS Terms

Access Plan

Access plans are generated by the optimization component to implement queries submitted by users.

ACID Properties

ACID properties are transaction properties supported by DBMSs. ACID is an acronym for atomic, consistent, isolated, and durable.

Address

A location in memory where data are stored and can be retrieved.

Aggregation

Aggregation is the process of compiling information on an object, thereby abstracting a higher-level object.

Aggregate Function

A function that produces a single result based on the contents of an entire set of table rows.

Alias

Alias refers to the process of renaming a record. It is alternative name used for an attribute.

Anomaly

The inconsistency that may result when a user attempts to update a table that contains redundant data.

ANSI

American National Standards Institute, one of the groups responsible for SQL standards.

Application Program Interface (API)

A set of functions in a particular programming language is used by a client that interfaces to a software system.

ARIES

ARIES is a recovery algorithm used by the recovery manager which is invoked after a crash.

Armstrong's Axioms

Set of inference rules based on set of axioms that permit the algebraic manipulation of dependencies. Armstrong's axioms enable the discovery of minimal cover of a set of functional dependencies.

Associative Entity Type

A weak entity type that depends on two or more entity types for its primary key.

Attribute

The differing data items within a relation. An attribute is a named column of a relation.

Authorization

The operation that verifies the permissions and access rights granted to a user.

Base Table

Base table is a named relation corresponding to an entity in the conceptual schema, whose tuples (rows) are physically stored in the database.

Bitmap Index

A compact, high speed indexing method where the key values and the conditions are compressed to a small size that can be stored and searched rapidly.

BLOB

BLOB is an acronym for Binary Large Object. BLOB is a data type for fields containing large binary data such as images.

Boyce–Codd Normal Form

A relation in third normal form in which every determinant is a candidate key.

Bucket

With reference to hash file, Bucket is the unit of a file having a particular address.

Buffer

Buffer an area in main memory containing physical database records transferred from disk.

Candidate Key

Any data item or group of data items which identify uniquely tuples in a relation.

Cardinality

The number of tuples in a relation.

Cartesian Product

All of the possible combinations of the rows from each of the tables involved in a join operation.

CASE Tool

CASE is an acronym for computer-aided software engineering. CASE tools support features for drawing, analysis, prototyping, and data dictionary. CASE tool facilitate database development.

Chasm Trap

A chasm trap exists where a model suggests the existence of relationship between entity types, but the pathway does not exist between certain entity occurrences.

Client

An individual user workstation that represents the front end of a DBMS.

Client/Server Architecture

Client/Server architecture is an arrangement of components among computers connected by a network.

Clustered Index

An index in which the logical or indexed order of the key values is the same as the physical stored order of the corresponding rows.

CODASYL

Conference on Data System Languages.

Concurrent Access

Performing two or more operations on the same data at the same time.

Concurrency Control

Concurrency control is the control on the database and transactions which are executed concurrently to ensure that each transaction completed healthy.

Composite Key

A candidate key comprising more than one attribute

Composite Index

An index that uses more than one column in a table to index data.

COMMIT

To control transactions, SQL provides this command to save recent DML changes to the database.

Condition Box

A special box used by QBE to store logical conditions that are not easily expressed in the table skeleton.

Constraints

Constraints are conditions that are used to impose rules on the table.

Conceptual View

The logical database description in ANSI/SPARC DBMS architecture.

Concurrent Access

Two or more users operating on the same rows in a database table at the same time.

Correlated Subquery

In SQL, a sub query in which processing the inner query depends on data from the outer query.

COUNT

An aggregate function that returns the number of values in a column.

Cursor

An SQL feature that specifies a set of rows, an ordering of those rows and a current row within that ordering.

Data

Data is a representation of facts, concepts or instructions in a formalized manner suitable for communication, interpretation or processing by humans or automatic means.

Data Abstraction

Data abstraction means the storage details of the data are hidden from the user and the user is provided with the conceptual view of the database.

Database

Database is the collection of interrelated data.

Data Definition Language (DDL)

The language component of a DBMS that is used to describe the logical structure of a database.

Data Manipulation Language (DML)

A language component of a DBMS that is used by a programmer to access and modify the contents of a database.

Database Instance

The actual data stored in a database at a particular moment in time.

Database State

Database state refers to the content of a database at a moment in time.

Database Management System

General purpose software used to maintain the database.

Database System

A database system means both a DBMS plus a database.

Database Administrator

A person or group of people responsible for the design and supervision of a data base.

Database Recovery

The process of restoring the database to a correct state in the event of a failure.

Database Security

Protection of the database against accidental or intentional loss, destruction, or misuse.

Data Mining

Data mining is the process of discovering implicit patterns in data stored in data warehouse and using those patterns for business advantage such as predicting future trends.

Data Model

Collection of conceptual tools for describing data and relationship between data.

Data Dictionary

Centralized store of information about database.

Data Warehouse

Data warehouse is a central repository for summarized and integrated data from operational databases and external data sources.

DB2

An IBM relational database system.

DBTG

Database Task Group.

Deadlock

The situation where each of two transactions are waiting indefinitely for the other transaction to release the resources it requests.

Degree of a Relation

The number of attributes in the relation.

Denormalization

Denormalization is the process of combining tables so that they are easier to query. Denormalization is opposite to normalization. Denormalization is done to improve query performance.

Derived Attribute

Derived attributes are the attributes whose values are derived from other related attribute.

Determinant

An attribute or set of attributes on which the value of one or more attributes depend.

Distributed Database

A database located at more than one site.

Domain

The set of all possible values for a given data item.

Domain Integrity

Data integrity that enforces valid entries for a given column

Domain Relational Calculus

Domain Relational Calculus is a calculus that was introduced by Edgar F. Codd as a declarative database query language for the relational data model.

DDL

Data Definition Language is used to define the schema of a relation.

DML

Data Manipulation Language is basically used to manipulate a relation.

Dual

A virtual table automatically created by Oracle along with the data dictionary. It has one column, DUMMY, defined to be VARCHAR2(1), and contains one row with a value of "X".

Embedded SQL

An application structure in which SQL statements are embedded within programs written in a host language like C, JAVA.

Encapsulation

Hiding the representation of an object is encapsulation.

Entity

An object that exist and is distinguishable from other objects.

Entity Class

A set of entities of the same type.

Entity Instance

Entity instance is a particular occurrence of an entity.

Entity Integrity (Table Integrity)

Integrity that defines a row as a unique entity for a particular table and ensures that the column cannot contain duplicate values.

Equijoin

A join operator where the join condition involves equality.

ER Model

ER stands for Entity-Relationship model. ER Model is based on a perception of a real world that consists of collection of basic objects called entities and relationships among these objects.

EER Model

EER stands for Enhanced ER model. EER model is the extension of original model with new modeling constructs. The new modeling constructs are supertype, subtype.

Exclusive Lock

A lock that prevents other users from accessing a database item. Exclusive locks conflict with all other kinds of locks such as shared locks.

Fantrap

A fantrap exists where a model represents a relationship between entity types but the pathway between certain entity occurrences is ambiguous.

File

A file is a collection of records of the same type.

File Organization

Methods used in organizing data for storage and retrieval such as sequential, indexed sequential, or direct.

First Normal Form

A relation is in first normal form if it contains no repeating groups.

Flat File

A file in which the fields of records are simple atomic values.

Foreign Key

Attribute or set of attributes that identifies the entity with which another entity is associated.

Fourth Normal Form

A relation is in fourth normal form if it is in BCNF and contains no multi-valued dependencies.

Function

A set of instructions that operates as a single logical unit.

Functional Dependency

A constraint between two attributes or two sets of attributes in a relation.

Generalization

In extended ER model (EER model), generalization is a structure in which one object generally describes more specialized objects.

GRANT

An SQL command for granting privileges to a user/users.

Graphical User Interface (GUI)

An interface that uses pictures and graphic symbols to represent commands and actions.

Hashing

A mathematical technique for assigning a unique number to each record in a file.

Hash Function

A function that maps a set of keys onto a set of addresses.

Hierarchical Database

A DBMS type that organizes data in hierarchies that can be rapidly searched from top to bottom.

Identifier

An attribute or collection of attributes that uniquely distinguishes an entity.

Index

A data structure used to decrease file access time.

Inheritance

Object-oriented systems have a concept of inheritance which permits class X to derive much of its code and attributes from another class Y. Class X will contain the data attributes and operations of class Y.

Intersection

A relational algebra operation performed on two union-compatible relations so as to produce a relation which contains rows that appear in both the union-compatible relations.

ISA Relationship

The relationship between each subtype and its supertype.

ISO

ISO stands for International Standards Organization. ISO in conjunction with ANSI to provide standard SQL for relational databases.

JOIN

An operation that combines data from more than one table.

JDBC

JDBC stands for Java Database Connectivity. A standard interface between Java applet or application and a database.

Key

Key is a data item that helps to identify individual occurrences of an entity type.

Leaf

In a tree structure, an element that has no subordinate elements.

Lock

A procedure used to control concurrent access to data.

Log

A file containing a record of database changes.

Logical Database Design

A part of database design that is concerned with modeling the business requirements and data.

Logical Data Independence

Application programs and terminal activities remain logically unimpaired when information preserving changes of any kind that theoretically permit unimpairment are made to the base tables.

Meta Data

Data about data is meta data. In other words, metadata is the data about the structure of the data in a database.

Mirrored Disk

Set of disks that are synchronized as follows: **each write to one disk goes to all disks in the mirrored set; reads can access any of the disk.

Mobile Database

A database that is portable and physically separate from a centralized database server but is capable of communicating with that server from remote sites.

Modification Anomaly

An unexpected side effect that occurs when changing the data in a table with excessive redundancies.

Multivalued Attribute

A multivalued attribute is an attribute to which more than one value is associated.

Multiple Tier Architecture

A client/server architecture with more than three layers a PC client, database server an intervening middleware server and application servers. The application servers perform business logic and manage specialized kinds of data such as images.

Multivalued Dependency

A type of dependency that exists when there are at least three attributes (for example X, Y, and Z) in a relation, and for each value of X there is a well-defined set of values for Y and a well-defined set of values for Z, but the set of values of Y is independent of set Z.

Natural Join

In a natural join, the matching condition is equality condition; one of the matching columns is discarded in the result table.

Normal Form

A set of conditions defined on entity specification.

Normalization

The design process for generating entity specifications to minimize both data redundancy and update anomalies.

NULL Value

A value that is either unknown or not applicable.

Object

An object is a collection of data, an identity, and a set of operations sometimes called methods.

Object-Oriented Database

An object-oriented database combines database capabilities with an object oriented analysis and design.

Object-Relational Database

Object-relational database combines RDBMS features with object-oriented features like inheritance and encapsulation.

ODBC

ODBC stands for Open Data Base Connectivity. A standard interface by which application programs can access and process SQL databases in a DBMS independent manner.

OLAP

Online Analytical Processing systems, contrary to the regular, conventional online transaction processing systems, are capable of analyzing online a large number of past transactions or large number of data records (ranging from mega bytes to gigabytes and terabytes).

OLTP

OLTP stands for Online Transaction Processing which supports large number of concurrent transactions without imposing excessive delays.

One-to-Many Relationship

A relationship between two tables in which a single row in the first table can be related to one or more rows in the second table, but a row in the second table can be related only to one row in the first table.

One-to-One Relationship

A relationship between two tables in which a single row in the first table can be related to only one row in the second table, and a row in the second table can be related to only one row in the first table.

Oracle

A relational database management system marketed by Oracle Corporation.

Outer Join

Outer join is a relational algebra operator which combines two tables. In an outer join, the matching and nonmatching rows are retained in the result.

Overflow

Overflow occurs when an insertion is attempted into a bucket or node that is full.

Partial Functional Dependency

A dependency in which one or more nonkey attributes are functionally dependent on part (but not all) of the primary key.

Physical Data Independence

Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access methods.

Polymorphism

Polymorphism is a principle of object-oriented computing in which a computing system has the ability to choose among multiple implementations of a method.

Primary Key

An attribute or set of attributes that uniquely identifies a tuple in a relation.

Procedural Language Interface

Procedural Language Interface is a method to combine a nonprocedural language such as SQL with programming language such as Visual Basic. Embedded SQL is an example for procedural language interface.

QBE

QBE stands for Query By Example. QBE uses a terminal display with attribute names as table headings for queries.

Query

Query is a request to extract useful data.

Query Plan

The plan produced by an optimizer for processing a query

Query Processing

The activities involved in retrieving data from the database are called as query processing.

Query Optimization

The activity of choosing an efficient execution strategy for processing a query is called as Query optimization.

RAID

RAID is an acronym for Redundant Array of Independent Disks. RAID is a collection of disks that operates as a single disk.

Range Query

Range query refers to selection on an interval. For example, select the name of players whose age is between thirty and thirty five.

Recursive Relationship

A relationship type where the same entity type participates more than once in different roles.

Redundant Data

Redundant data refers to the same data that is stored in more than one location in the database.

Referential Integrity

The referential integrity imposes the constraint that if a foreign key exists in a relation, either the foreign key value must match a candidate key value of some tuple in its home relation or the foreign key value must be wholly null.

Relation

A relation is a table with rows and columns.

Relationship Type

Relationship type is a set of meaningful associations among entity types.

Relational Algebra

Procedural language based on algebraic concepts. It consists of collection of operators that are defined on relations, and that produce relations as results.

Relational Calculus

A query language based on first order predicate calculus.

Relational Database

A database that organizes data in the form of tables.

Relational Database Management System (RDBMS)

Software that organizes manipulates and retrieves data stored in a relational database.

Recursive Relationship

A relationship in which one entity references itself.

Repository

A repository is a collection of resources that can be accessed to retrieve information. Repositories often consist of several databases tied together by a common search engine.

REVOKE

An SQL statement for removing privileges from a user/users.

ROLLBACK

A DBMS recovery technique that aborts active applications and attempts to reinstate the state of the database prior to initiating the applications active at the time the database failed.

Root

The top record, row, or node in a tree. A root has no parent.

Schema

Schema is the collection of named object.

Scalar Function

A function operating on a single value. Scalar functions return a single value.

Second Normal Form (2NF)

A relation schema R is in 2 NF if every nonprime attribute A in R is fully functionally dependent on the primary key of R.

Self Join

A join that merges data from a table with data in the same table, based on columns in a table that are related to one another.

Semantic Data Model

Semantic data model provides a vocabulary for expressing the meaning as well as the structure of database data.

Semijoin

A dyadic relational operator yielding the tuples of one operand that contributes to the join of both.

Sequential File Organization

The records in the file are stored in sequence according to a primary key value.

SGML

SGML stands for Standard Generalized Markup Language. A standard means for tagging and marking the format, structure, and content of documents. HTML is a subset of SGML.

Shared Lock

Lock that allows concurrent transactions to read a resource.

Sparse Index

Index in which the underlying data structure contains exactly one pointer to each data page.

Stripe

Stripping is an important concept for RAID storage. Stripping involves the allocation of physical records to different disks.

Structured Query Language (SQL)

A standard language used to manipulate data in a database.

Subquery

Query within a query.

Subtype

A subtype represents a subset or subgroup of super class entity type's instances. Subtype inherit the attributes and relationships associated with their super type.

SUM

An aggregate function that returns the sum of all values. Sum can be used with numeric columns only. NULL values are ignored.

Super Type

Super type is a generic entity type that has a relationship with one or more subtype.

Table

Table is a 2D arrangement of data. The table consists of rows and columns.

Ternary Relationship

A relationship which involves three entity types. It is a simultaneous relationship among the instances of three entity types.

Three-Tier Architecture

Three-Tier architecture is client/server architecture with three layers: a PC client, database server and an application server.

Transaction

Transaction is the execution of user program in DBMS. In other words it can be stated as the various read and write operations done by the user program on the DBMS, when it is executed in DBMS environment.

Transaction Log

File that records transactional changes occurring in a database, providing a basis for updating a master file and establishing an audit trail.

Transitive Dependency

If the attribute X is dependent on Y and the attribute Y is dependent on Z then the attribute X is transitively dependent on Z

Trigger

Action that causes a procedure to be carried out automatically when a user attempts to modify data.

Trivial Dependency

The dependency of an attribute on itself.

Tuple

A row in the tabular representation of the relation.

Tuple Relational Calculus

The tuple relational calculus is based on specifying a number of tuple variables. Each tuple variable usually ranges over a particular database relation, meaning that the variable may take as its value any individual tuple from that relation.

Two Phase Locking

A locking scheme with two distinct phases. During the first phase the DBMS may set locks, during the second it is allowed only to release locks.

Two Phase Commit

Process that ensures transactions applying to more than one server are completed on either all servers or none.

Two-Tier Architecture

Two-Tier architecture is a client/server architecture in which a PC client and a database server interact directly to request and transfer data. The PC client contains the user interface code, the server contains the data access logic, and the PC client and the server share the validation and business logic.

Union

A relational algebra operation performed on two union-compatible relations to produce a third relation which contains every row in the union-compatible relations minus any duplicate rows.

Union Compatible

Two relations are union compatible if they have same number of attributes and the attributes in the corresponding columns arise from the same domain.

Update Anomaly

An undesirable side effect caused by an insertion, deletion, or modification.

Updatable View

When the rows of an updatable view is modified then DBMS translates the view modifications into the modifications to the rows of the base tables.

Variable

A location in memory used to hold temporary values. Variables have a scope and a lifetime depending on where they are created and how they are defined.

View

A virtual table which is derived from base table using a query.

Visual Basic (VB)

A product of Microsoft that is used to develop applications for the windows environment. The professional version supports database connections.

Volatile Storage

Volatile storage loses its state when the power is disconnected.

VSAM

VSAM stands for Virtual Storage Access Method. It is IBM's implementation of the B-tree concept.

Weak Entity

An entity whose existence depends on other entity.

Write-write Conflict

The situation in which two write actions operate on the same data item.

World Wide Web (WWW)

A first attempt to set up an international database of information.

XML

A language for defining the structure and the content of documents on the World Wide Web.

B

Overview of Commands in SQL

Some of the commonly used data types, SQL*Plus commands, Aggregate functions, SQL*Plus commands summary, built-in scalar functions are given in this appendix.

Commonly Used Data Types

Data type	Description
char(n)	Fixed length character data, n characters long.
varchar2(n)	Variable length character string.
number(o,d)	Numeric data type for integers and real, where o = overall number of digits and d = number of digits to the right of decimal point.
date	Date data type for storing date and time. The default format for date is DD-MMM-YY. Example "13-oct-94."

SQL*Plus Editing Commands

Command	Abbreviation	Purpose
APPEND text	A text	Adds text at the end of a line.
CHANGE /old/new	C /old/new	Changes old to new in a line.
CHANGE /text	C /text	Deletes text from a line.
CHANGE /text	C /text	Deletes text from a line.
CLEAR BUFFER	CL BUFF	Deletes all lines.
DEL	(none)	Deletes the current line.
DEL n	(none)	Deletes line n.
DEL *	(none)	Deletes the current line.
DEL n *	(none)	Deletes line n through the current line.
DEL LAST	(none)	Deletes the last line.
DEL m n	(none)	Deletes a range of lines (m to n).

Command	Abbreviation	Purpose
DEL * n	(none)	Deletes the current line through line n.
INPUT text	I text	Adds a line consisting of text.
LIST	L	Lists all lines in the SQL buffer.
LIST n	L n or n	Lists line n.
LIST *	L *	Lists the current line.
LIST n *	L n *	Lists line n through the current line.
LIST LAST	L LAST	Lists the last line.
LIST m n	L m n	Lists a range of lines (m to n).
LIST * n	L * n	Lists the current line through line n.

Aggregate Functions

Function	Usage
AVG(expression)	Computes the average value of a column by the expression.
COUNT(expression)	Counts the rows defined by the expression.
COUNT(*)	Counts all rows in the specified table or view.
MIN(expression)	Finds the minimum value in a column by the expression.
MAX(expression)	Finds the maximum value in a column by the expression.
SUM(expression)	Computes the sum of column values by the expression.

Built-in Scalar Functions

Function	Usage
CURRENT_DATE	Identifies the current date.
CURRENT_TIME	Identifies the current time.
CURRENT_TIMESTAMP	Identifies the current date and time.
CURRENT_USER	Identifies the currently active user within the database server.
SESSION_USER	Identifies the currently active Authorization ID if it differs from the user.
SYSTEM_USER	Identifies the currently active user within the host operating system.

SQL*Plus Command Summary

Command	Description
@ (“at” sign)	Runs the SQL*Plus statements in the specified command file. The command file can be called from the local file system or from a web server.
@@ (double “at” sign)	Runs a command file. This command is identical to the @ (“at” sign) command. It is useful for running nested command files because it looks for the specified command file in the same path as the command file from which it was called.
/ (slash)	Executes the SQL command or PL/SQL block.
ACCEPT	Reads a line of input and stores it in a given user variable.
APPEND	Adds specified text to the end of the current line in the buffer.
ARCHIVE LOG	Starts or stops the automatic archiving of online redo log files manually (explicitly) archives specified redo log files or displays the information about redo log files.
ATTRIBUTE	Specifies display characteristics for a given attribute of an Object Type column and lists the current display characteristics for a single attribute or all attributes.
BREAK	Specifies where and how formatting will change in a report or lists the current break definition.
BTITLE	Places and formats a specified title at the bottom of each report page or lists the current BTITLE definition.
CHANGE	Changes text on the current line in the buffer.
CLEAR	Resets or erases the current clause or setting for the specified option such as BREAKS or COLUMNS.
COLUMN	Specifies display characteristics for a given column or lists the current display characteristics for a single column or for all columns.
COMPUTE	Calculates and prints summary lines using various standard computations on subsets of selected rows or lists all COMPUTE definitions.
CONNECT	Connects a given user to Oracle.
COPY	Copies results from a query to a table in a local or remote database.
DEFINE	Specifies a user variable and assigns it a CHAR value or lists the value and variable type of a single variable or all variables.

Command	Description
DEL	Deletes one or more lines of the buffer.
DESCRIBE	Lists the column definitions for the specified table view or synonym or the specifications for the specified function or procedure.
DISCONNECT	Commits pending changes to the database and logs the current user off Oracle but does not exit SQL*Plus.
EDIT	Invokes a host operating system text editor on the contents of the specified file or on the contents of the buffer.
EXECUTE	Executes a single PL/SQL statement.
EXIT	Terminates SQL*Plus and returns control to the operating system.
GET	Loads a host operating system file into the SQL buffer.
HELP	Accesses the SQL*Plus help system.
HOST	Executes a host operating system command without leaving SQL*Plus.
INPUT	Adds one or more new lines after the current line in the buffer.
LIST	Lists one or more lines of the SQL buffer.
PASSWORD	Allows a password to be changed without echoing the password on an input device.
PAUSE	Displays the specified text then waits for the user to press [Return].
PRINT	Displays the current value of a bind variable.
PROMPT	Sends the specified message to the user's screen.
EXIT	Terminates SQL*Plus and returns control to the operating system. QUIT is identical to EXIT.
RECOVER	Performs media recovery on one or more tablespaces one or more datafiles or the entire database.
REMARK	Begins a comment in a command file.
REPFOOTER	Places and formats a specified report footer at the bottom of each report or lists the current REPFOOTER.
REPHEADER	Places and formats a specified report header at the top of each report or lists the current REPHEADER definition.
RUN	Lists and executes the SQL command or PL/SQL block currently stored in the SQL buffer.
SAVE	Saves the contents of the SQL buffer in a host operating system file (a command file).
SET	Sets a system variable to alter the SQL*Plus environment for your current session.

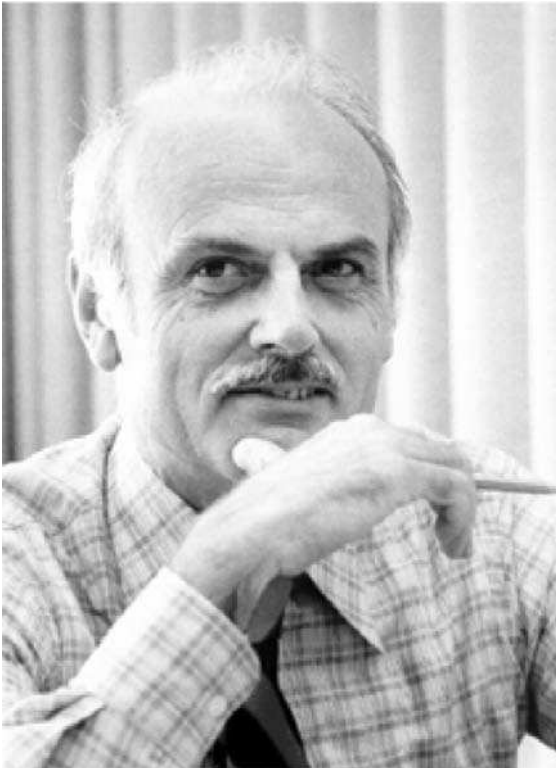
Command	Description
SHOW	Shows the value of a SQL*Plus system variable or the current SQL*Plus environment.
SHUTDOWN	Shuts down a currently running Oracle instance.
SPOOL	Stores query results in an operating system file and optionally sends the file to a printer.
START	Executes the contents of the specified command file.
STARTUP	Starts an Oracle instance and optionally mounts and opens a database.
STORE	Saves attributes of the current SQL*Plus environment in a host operating system file (a command file).
TIMING	Records timing data for an elapsed period of time lists the current timer's title and timing data or lists the number of active timers.
TITLE	Places and formats a specified title at the top of each report page or lists the current TITLE definition.
UNDEFINE	Deletes one or more user variables that is defined either explicitly (with the DEFINE command) or implicitly (with an argument to the START command).
VARIABLE	Declares a bind variable that can be referenced in PL/SQL.
WHENEVER OSERROR	Exits SQL*Plus if an operating system command generates an error.
WHENEVER SQLERROR	Exits SQL*Plus if a SQL command or PL/SQL block generates an error.

C

Pioneers in DBMS

This appendix looks at the pioneers in field of database management system. Even though many great people have contributed for the development of database management system, we consider here the work of Dr. Edgar F. Codd, Peter Chen, and Ronald Fagin. The pioneers' biography would certainly motivate the readers to work in the database management system development.

Author: Dr. Edgar F. Codd (1923–2003)



C.1 About Dr. Edgar F. Codd

Ted Codd was a genuine computing pioneer. He was also an inspiration to all of us who had the fortune to know him and work with him. He began his career in 1949 as a programming mathematician for IBM on the Selective Sequence Electronic Calculator. He subsequently participated in the development of several important IBM products, including its first commercial electronic computer (IBM 701) and the STRETCH machine, which led to IBM's 7090 mainframe technology. Then, in the 1960s, he turned his attention to the problem of managing large commercial databases – and over the next few years he created, single handed, the invention with which his name will forever be associated: the relational model of data.

The relational model is widely recognized as one of the great technical innovations of the twentieth century. Codd described it and explored its implications in a series of research papers – staggering in their originality – which he published throughout the period 1969–1979. The effect of those papers was twofold: they changed for good the way the Information Technology (IT) world (including the academic component of that world in particular) perceived the database management problem; and they laid the foundation for an entire new industry, the relational database industry, now worth many billions of dollars a year. In fact, not only did Codd's relational model set the entire discipline of database management on a solid scientific footing, but it also formed the basis for a technology that has had, and continues to have, a major impact on the very fabric of our society. It is no exaggeration to say that Ted Codd is the intellectual father of the modern database field.

Codd's supreme achievement with the relational model should not be allowed to eclipse the fact that he made major original contributions in several other important areas as well, including multiprogramming, natural language processing, and more recently Enterprise Delta (a relational approach to business rules management), for which he and his wife were granted a US patent. The depth and breadth of his contributions were recognized by the long list of honors and elected positions that were conferred on him during his lifetime: including IBM Fellow, elected ACM Fellow, elected Fellow of the Britain Computer Society, elected member of the National Academy of Engineering, and elected member of the American Academy of Arts and Sciences. In 1981 he received the ACM Turing Award, the most prestigious award in the field of Computer Science. He also received an outstanding recognition award from IEEE: the very first annual achievement award from the international DB2 Users Group, and another annual achievement award from DAMA in 2001. Computerworld, in celebration of the 25th anniversary of its publication, selected him as one of 25 individuals in or related to the field of computing who have had the most effect on our society. And Forbes magazine, which in December 2002 published a list of the most important innovations and contributions for each of the 85 years of its existence, was selected for the year 1970 the relational model of data by E.F. Codd.

Ted Codd was a native of England and a Royal Air Force veteran of World War II. He moved to the United States in 1946 and became a naturalized US citizen. He held MA degrees in Mathematics and Chemistry from Oxford University and MS and Ph.D. degrees in Communication Sciences from the University of Michigan. He is survived by his wife Sharon and her parents, Sol and Nora Boroff, of Williams Island, FL; a brother David Codd and his wife, Barbara and a sister, Katherine Codd, all of England; and a second sister Lucy Pickard of Hamilton, Ontario. He also leaves four children and their families; Katherine Codd Clark, her husband Lawrence, and their daughters, Shannon and Allison, of Palo Alto, CA; Ronald E.F. Codd, his wife Susie, and their son Ryan and daughter Alexis of Alamo, CA; Frank Codd and his wife Aydes of Castro Valley, CA; and David Codd, his wife Ileana, and their daughter Melissa and son Andrew of Boca Raton, FL. He also leaves nieces and nephews in England, Canada, and Australia, as well as many, many friends and colleagues worldwide.

Prof. Peter Chen is the originator of the entity-relationship model (ER model), which serves as the foundation of many system analysis and

Author: Dr. Peter Chen



design methodologies, computer-aided software engineering (CASE) tools, and repository systems including IBM's Repository Manager/MVS and DEC's CDD/Plus. After years of efforts of many people in developing and implementing the ideas, now "entity-relationship model (ER model)," "entity-relationship diagram (ER diagram)," and "Peter Chen" have become commonly used terms in "online" dictionaries, books, articles, web pages, course syllabi, and commercial product brochures.

Dr. Peter Chen's original paper on the ER model is one of the most cited papers in the computer software field. Prof. Peter Chen was honored by the selection of his original ER model paper as one of the 38 most influential papers in Computer Science. Based on one particular citation database, Chen's paper is the 35th most cited article in Computer Science. It is the fourth most downloaded paper from the ACM Digital Library in January 2005 (Communications of ACM, March 2005).

The ER model was adopted as the metamodel for the American National Standards Institute (ANSI) Standard in Information Resource Directory System (IRDS), and the ER approach has been ranked as the top methodology for database design and one of the top methodologies in systems development by several surveys of FORTUNE 500 companies.

Dr. Chen's work is a cornerstone of software engineering, in particular CASE. In the late 1980s and early 1990s, IBM's Application Development Cycle (AD/Cycle) framework and DB2 repository (RM/MVS) were based on the ER model. Other vendors' repository systems such as Digital's CDD+ were also based on the ER model. Prof. Chen has made significant impact on the CASE industry by his research work and by his lecturing around the world on structured system development methodologies. Most of the major CASE tools including Computer Associates' ERWIN, Oracle's Designer/2000, and Sybase's PowerDesigner (and even a general drawing tool like Microsoft's VISIO) are influenced by the ER model.

The ER model also serves as the foundation of some of the recent work on object-oriented analysis and design methodologies and Semantic Web. The UML modeling language has its roots in the ER model.

The hypertext concept, which makes the World Wide Web extremely popular, is very similar to the main concept in the ER model. Dr. Peter Chen is currently investigating this linkage as an invited expert of several XML working groups of the World Wide Web Consortium (W3C).

Prof. Peter Chen's work is cited heavily in a book published in 1993 for general public called *Software Challenges* published by Time-Life Books as a part of the series on "Understanding Computers."

Dr. Chen is a Fellow of the IEEE, the ACM, and the AAAS. He is a member of the European Academy of Sciences. He has been listed in Who's Who in America and Who's Who in the World for more than 15 years. He is the recipient of prestigious awards in several fields of IT: data management, information management, software engineering, and general information science/technology:

- The Data Resource Management Technology Award from the Data Administration Management Association (NYC) in 1990.
- The Achievement Award in Information Management in 2000 from DAMA International, an international professional organization of data management professionals, managers, and Chief Information Officers (CIOs). Dr. E.F. Codd (the inventor of the relational data model) is the winner of the same award in 2001.
- Inductee, the Data Management Hall of Fame in 2000.
- The Stevens Award in Software Method Innovation in 2001, and the award was presented at IEEE International Conference on Software Maintenance in Florence, Italy on 8 November 2001.
- The IEEE Harry Goode Award at the IEEE-CS Board of Governors Meeting in San Diego, February 2003. The previous winners of the Harry Goode Award include the inventors of computers, core memory, and semiconductors.
- The ACM/AAAI Allen Newell Award at the ACM Award Banquet in San Diego, June 2003. He was introduced at the opening ceremony in the 2003 International Joint Conference on Artificial Intelligence (IJACI-03) on 11 August 2003 in Acapulco, Mexico. The previous seven winners of the Allen Newell Award include a Nobel Prize and National Medal of Science winner, two National Medal of Technology winners (one of them is also an ACM Turing Award winner), and other very distinguished scientists who either have made significant contributions to several disciplines in computer science or have bridged computer science with other disciplines.
- The Pan Wen-Yuan Outstanding Research Award in 2004. Starting 1997, the awards have been given to usually three individuals each year (one in Taiwan, one in Mainland China, and one in “overseas” – outside of Taiwan and Mainland China) in the high-tech fields (including electronics, semiconductors, telecommunications, computer science, computer hardware/software, IT, and IS). In 2003, the overseas winner was Prof. Andrew C.C. Yao of Princeton University, who is also a winner of the ACM Turing Award.

Dr. Peter Chen was recognized as a “software pioneer” in the “Software Pioneers” Conference, Bonn, Germany, 27–28 June 2001, together with a group of very distinguished scientists including winners of President’s Medals of Technology, ACM Turing Awards, ACM/AAAI Allen Newell Awards, or IEEE distinguished awards such as Harry Goode Awards. The streamed video and slides of the talks in the “Pioneers” Conference may be available at the conference website. All the speeches in the conference are documented in a book (with four DVDs) published by Springer, and how to order the book can be found in the section on Papers Online.

Prof. Peter Chen is a member of the Advisory Committee of the Computer and Information Science and Engineering (CISE) Directorate of the National

Science Foundation (NSF). He was a member of the Airlie Software Council, which consists of software visionaries/gurus and very-high-level software organization executives, organized by US Department of Defense (DoD). He was an advisor to the President of Taiwan's largest R&D organization, Industrial Technology Research Institute (ITRI), with over 6,000 employees, which has been the driving force of Taiwan's high-tech growth in the past three decades.

Dr. Peter Chen was one of five main US delegates to participate in the first IEEE USA–China International Conference, which was held in Beijing, in 1984 and to meet with PRC leaders and government officers in the Science and Technology fields and the Education area. Since 1984, he has been an Honorary Professor of Huazhong University of Science and Technology in Wuhan, China.

Dr. Peter Chen is also the Editor-in-Chief of Data & Knowledge Engineering, the Associate Editor for the Journal of Intelligent Robotic Systems, Electronic Government, and other journals. In the past, he was the Associate Editor for IEEE Computer, Information Sciences, and other journals.

At MIT, UCLA, and Harvard, Prof. Peter Chen taught various courses in Information Systems and Computer Science. At LSU, he has been doing research and teaching on Information Modeling, Software Engineering, Data/Knowledge Engineering, Object-Oriented Programming, Internet/Web, Java, XML, Data Warehousing, E-commerce (B2B and B2C), Homeland Security, Identity Theft, System Architecture, Digital Library, and Intelligent Systems for Networking (Sensors Networks, Wi-Fi, and Cellular).

Prof. Peter Chen is the Principal Investigator of a large NSF-funded multidisciplinary project on profiling of terrorists and malicious cyber transactions for counter terrorisms and crimes. Dr. Peter Chen is also the Executive Director of the China–US Million Book Project (funded by NSF through CMU and the Ministry of Education of PRC), which is in the process of creating a large digital library of over one million books in English and Chinese. He has been the Principal Investigator of various research projects in system architecture, information/knowledge management, software engineering, and performance analysis sponsored by many government agencies and commercial companies.

Dr. Peter Chen holds the position of M.J. Foster Distinguished Chair Professor of Computer Science in Louisiana State University since 1983.

Charles W. Bachman attended Michigan State College and graduated in 1948 with a Bachelor's degree in Mechanical Engineering (Tau Beta Phi). He graduated in 1950 with a Master's degree in Mechanical Engineering from University of Pennsylvania. He attended Wharton School of Business in the University of Pennsylvania at the same time and completed three quarters of the requirements for an MBA.

He worked for the Dow Chemical Company in Midland Michigan. He started in Engineering Department working on engineering economics problems (operation research). In 1962 he transferred to the Finance Department

Author: Charles W. Bachman



to establish a decision support project to assist in the evaluation of the return on capital of new and old production plants and product profitability. In 1955 he transferred to the Plastics Product Division as a process engineer and later as an assistant plant manager. In 1957 he started the first Computer Department for business data processing for Dow. As Chairman of the SHARE Data Processing Committee, that launched the SHARE 9PAC project for the IBM 709 computer in 1958. The tape-oriented File Maintenance and Report Generation System created was an early version of what is now called a 4GL with a “WYSIWYG” user interface. At the same time, Bachman pioneered the introduction of probability into the CPM/PERT scheduling that was used for Dow new plant construction.

He worked for the General Electric Company. First assignment (1961–1964) for GE’s Manufacturing Services (New York City) was to design and build a generic manufacturing information and control system. The MIACS application system that came from this project contained many elements, which underlay most, current day, manufacturing control systems. It did manufacturing planning, parts explosion, factory dispatching, handled factory feedback, and replanning as required to handle new orders and correct for changing factory circumstances.

The MIACS system contained the first version of the Integrated Data Store (IDS) database management system which was the basis for General

Electrics IDS and IDS II, Cullinet's IDMS, and a host of other DBMS based on Bachman's Network Data Model. IDS was the first disk-based database management system used in production. It seized a number of new opportunities available at that time and created a unique product. It was built upon a "virtual memory" system that was being applied to the storage and retrieval of dynamic and permanent data. It provided a page-turning buffer management system that provided almost instantaneous access to the data most recently accessed. It provided for the declaration and processing data organized in application-specific network structures. It fully integrated its, record-at-a-time, STORE, RETRIEVE, MODIFY, and DELETE language statements into the GE GECOM programming language. IDS created a new paradigm for the application programmers. It changed their I/O vantage point from data flowing "IN and OUT of the program" to the program moving "IN and OUT of the database." Once a record was stored, it remained available in the database, forever, unless it was explicitly deleted. IDS was characterized as a "network model" database management system, because it provided for the direct construction and navigation of the semantic graphs that underlie most business applications systems.

The MIACS system also contained a transaction-oriented operating system that accepted the input of new "problem control cards," with their associated data cards, and stored them until they could be dispatched. It dispatched each such problem in priority sequence, following the completion of the prior problem. It loaded the required program blocks into the buffer area, allocated all unneeded buffer blocks to the IDS page-turning system, and then dispatched the computer to the program. The solving of one problem might engender the creation of one or more new problem statements with their associated data records. The storage and retrieval of problem statements and their associated data were handled by the IDS database management system, along with all of the application requirements.

Bachman developed data structure diagrams (ER diagrams), commonly known as *Bachman diagrams*, as a graphical representation of semantic structures within the data.

In 1964, Bachman transferred to GE's Computer Department in Phoenix, Arizona with assignment to convert the GE 225 version of IDS to a commercial product for GE's 400 and 600 computer lines. At the same time, Bachman worked with the ANSI SPARC Study Group on DBMS, in creating their report of Network Databases. This task group was responsible for creating the specification for the integration of IDS into the COBOL programming language. This report formed the basis for GE's IDS II and many other DBMS based on the specification.

Later Bachman started the GE-Weyerhaeuser project team that created first "nonstop" operating system (WEYCOS) for the GE 600 computer. This team also created the first multiprogramming version of IDS, which allowed many programs to access to a common database with transparent locking, deadlock (interference) detection, recovery, and restart.

Bachman developed a database-oriented version (dataBASIC) of the BASIC programming language. Its integrated database facility was based on the “universal relation” concept (before the concept was formerly described). The product was shipped for both the GE 400 and 600 product lines. The City of Tulsa, OK used dataBASIC to construct their public safety and police system.

Honeywell Information Systems, Inc. acquired the General Electric’s Computer Division. Bachman’s first assignment was to manage a group to specify and implement a version of IDS for Honeywell’s advanced product line, to be built by the newly merged company. In 1973 Bachman transferred to Honeywell’s Advanced System Project as Chief Staff Engineer.

He has given the Association of Computer Machinery’s Alan M. Turing Award in 1973 for pioneering work in database management systems. The Turing Award is the software industry’s equivalent of the Nobel Prize. The 1973 Turing Lecture by Bachman was entitled “The Programmer as Navigator.” He published the “extended network” data model in 1973.

He served as Vice Chairman with the ANSI SPARC’s Study Group on DBMS, to explore the possible standardization database management languages. Group report spelled out the first architectural statement about the various interfaces and protocols required to support the data independence concept and established what is now broadly known as the “three schema approach.” He elected a “Distinguished Fellow” of the British Computer Society in 1978 for database research. Only 22 people have been so honored today. He published the “role” data model in 1978.

He began work in 1976 as leader of Honeywell’s Distributed System Architecture Project. This work served as the prototype of the later ANSI SPARC Study Group – Distributed System Architecture and the International Standard Organization’s (ISO) Open System Interconnection Project. He became Chairman of the ANSI Study Group in 1978 and Chairman of the ISO Open Systems Interconnection Subcommittee in 1979.

In 1980 he began working on concepts more recently called *computer-aided software engineering*. He was awarded 16 US patents while at Honeywell for database inventions and one British patent for pioneering work on model-driven development (executable functional specifications).

In Cullinane (Cullinet) Database Systems, he joined Cullinet as Vice President of Product Management, while retaining responsibility as Chairman for the ISO Open Systems Interconnection Subcommittee. He also continued work on prototype CASE systems. Cullinet’s IDMS system is a direct copy of Bachman’s original IDS DBMS. During the 2 years with Cullinet, the role data model, which had been developed at Honeywell, was enhanced to facilitate its integration with the existing Cullinet IDMS software. The result was the “Partnership” Data Model which was published in 1983 and which was awarded a software patent in the US.

Bachman Information Systems, Inc. was created on 1 April 1983 to productize the CASE concepts, which had been developed while at Honeywell

and Cullinet. Key concepts' use included the establishment of a clear separation between the specification of the business level (logical) rules characterized as the business model and the specification of the physical level rules characterized by existing database languages, communication languages, and programming languages.

This distinction between logical and physical levels became very important as the implementation rules from existing COBOL, PL/I, IDMS, IMS, and Relational DBMS could be "reverse engineered" into an enhanced data model based on the Partnership Data Model, extended with some object-oriented concepts.

Bachman Information Systems received its first round of venture capital funding in 1986, and after several additional rounds went public in 1990. Bachman Information Systems did business on a worldwide basis and was highly respected for its products supporting data modeling and database administrator professionals. In this period, a number of patents were awarded to Bachman Information Systems dealing with aspects of the CASE products. Mr. Bachman was a co-inventor on six of these.

Bachman Information Systems, Inc. of Boston, MA and Cadre Technology, Inc. of Providence, RI merged to form a new company, named "Cayenne Software, Inc." Bachman and Cadre developed and marketed similar products, i.e., CAD/CAM products to help the software professionals in carrying out their tasks. The largest difference in the two former companies is that Bachman marketed its products to the commercial market and Cadre marketed theirs to the engineering/scientific market.

In June 1996, Charlie was given a Life Achievement Award by the Massachusetts Software Council. In August 1996, he and his wife, Connie, moved to Tucson, Arizona. In the fall of 1997, Charlie was showcased as one of the "wizards" in the Association of Computer Machinery and The Computer Museums exhibition, "The Wizards and Their Wonders." This was a photographic exhibit and its contents were published in a book of the same name. That same fall, Mr. Bachman retired as an employee and the Chairman of the Board of Cayenne Software (formerly Bachman Information Systems) after 14 years service.

Mr. Bachman lives with his wife of 52 years, Connie Hadley, and continues his consulting work. He has worked on metamodeling and software engineering projects with Constellar Corp. and The Webvan Group. He is currently working on the story of the development of IDS.

C.2 Ronald Fagin

Ronald Fagin's article: "A normal form for relational databases that is based on domains and keys" published in ACM Transactions on Database Systems (volume 6, issue 3, September 1981).

C.2.1 Abstract of Ronald Fagin's Article

A new normal form for relational databases, called “domain–key normal form (DK/NF),” is defined. Also, formal definitions of insertion anomaly and deletion anomaly are presented. It is shown that a schema is in DK/NF if and only if it has no insertion or deletion anomalies. Unlike previously defined normal forms, DK/NF is not defined in terms of traditional dependencies (functional, multivalued, or join). Instead, it is defined in terms of the more primitive concepts of domain and key, along with the general concept of a “constraint.” We also consider how the definitions of traditional normal forms might be modified by taking into consideration, for the first time, the combinatorial consequences of bounded domain sizes. It is shown that after this modification, these traditional normal forms are all implied by DK/NF. In particular, if all domains are infinite, then these traditional normal forms are all implied by DK/NF.

D

Popular Commercial DBMS

Some of the popular commercial DBMS like System R, DB2, and Informix, their features and applications are given in this appendix.

D.1 System R

D.1.1 Introduction to System R

SYSTEM R is a Database Management System which implements the concept of Relational Data Architecture. It is introduced by Codd in 1970 as an approach toward providing solution to various problems in database management systems. The system provides a high-level data independence by isolating the end user as much as possible from underlying storage structures. The system permits definition of a variety of relational views on common underlying data. Data control features are provided including authorization, integrity assertions, triggered transactions, a logging and recovery subsystem, and facilities for maintaining data consistency in a shared-update environment.

D.1.2 Keywords Used

Database

Database is an ordered collection of useful information in such a way that storing and retrieval of information is more easy, accurate, and much efficient.

Data Model

A data model is a collection of high-level data description constructs that hide many low-level storage details.

Relational Model

In this model a database is a collection of one or more relations, where each relation is a table with rows and columns.

The main construct for representing data in the relational model is a relation. A relation consists of a relation schema and a relation instance. The relation instance is a table, and the relation schema describes the column heads for the table.

RSI

It is the abbreviation of Relational Storage Interface. It is the external interface which handles access to single tuples of base relations.

RSS

It is the abbreviation of Relational Storage System. It is a complete storage subsystem of RSI. It manages devices, space allocation, storage buffers, transaction consistency and locking, deadlock detection, back out, transaction recovery, system recovery and it maintains indexes on selected fields of base relations, and pointer chains across relations.

RDI

It is the abbreviation of Relational Data Interface. It is the external interface which can be called directly from a programming language, or used to support various emulators and other interfaces.

RDS

It is the abbreviation of Relational Data System. It supports RDI, provides authorization, integrity enforcement, and support for alternative views of data.

D.1.3 Architecture and System Structure

Architecture and system structure includes major interfaces and components as illustrated in Fig. D.1. They are:

1. RSI (Relational Storage Interface)
2. RDI (Relational Data Interface)
3. SEQUEL
4. VM (Virtual Machines)

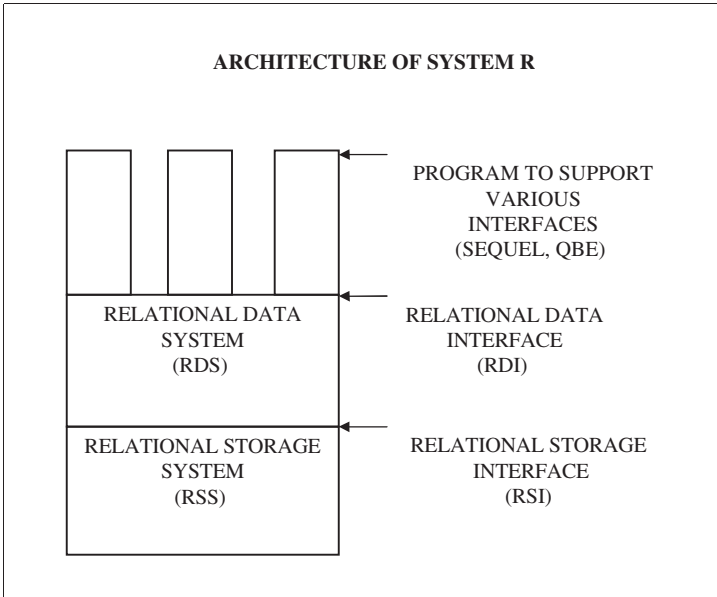


Fig. D.1. Architecture of system R

Relational storage interface takes care about the devices, space allocation, storage buffers, transaction consistency and locking, deadlock detection and backout, transaction recovery and system recovery with the help of RSS.

Relational data interface takes care about the authorization, integrity enforcement, and supports for alternative data views with the help of RDS.

SEQUEL is the high-level Language which is embedded within the RDI, and is used as the basis for all data definition and manipulation.

Virtual machines concept is successfully implemented in SYSTEM R. The main goal of this implementation is to effectively support for Concurrent Transactions on shared data and support the Multiuser Environment. Each VM is dedicated to particular user who is logged on to the computer. RDS and RSS on that particular VM will take care about all accesses and authorizations.

The provision for many database machines, each executing shared, re-entrant code and sharing control information, means that the database system need not provide its own multitasking to handle concurrent transactions. Rather, one can use the host operating system to multithread at the level of VM. Furthermore, the operating system can take advantage of multiprocessors allocated to several VM, since each machine is capable of providing all data management services.

D.1.4 Relational Data Interface

Query Facilities in RDI

Similar to other Database Sublanguages SEQUEL also provides most of the data manipulation facilities as described earlier.

EXAMPLE 1:

```
Consider the following block of query.
SELECT NAME
FROM EMP
WHERE ID= '1234';
```

Explanation

This is the simple query which will give the Names of the employees who have the ID as 1234. This query has no problem in execution. It is efficient too. But consider the following Nested Query:

Example 2:

```
SELECT NAME
FROM EMP
WHERE SAL >
SELECT SAL
FROM EMP
WHERE EMPNO = B1.MGR
```

Explanation

This query is formed by combining two simple queries. Experience has shown that this block label notation has three disadvantages:

- It is not possible to select quantities from the inner block, such as: “For all employee who earn more than their manager, list the employee’s name and his manager’s name.”
- Since the query is asymmetrically expressed, the optimizer is biased toward making an outer loop for the first block and an inner loop for the second block. Since this may not be the optimum method for interpreting the query, the optimization process is made difficult.
- Human factors studies have shown that the block label notation is hard for nonprogrammers to learn.

Because of these disadvantages, the block label notation has been replaced by the following more symmetrical notation, which allows several tables to be listed in the FROM clause and optionally referred to by variable names.

EXAMPLE 3:

```

SELECT DNO
FROM EMP
WHERE JOB = 'CLERK'
GROUP BY DNO
HAVING COUNT (*) > 10

```

Explanation

In the above block of statements three new terms are used they are GROUP BY, HAVING, and COUNT().

GROUP BY is used to grouping the selected tuples according to particular field value.

HAVING is used to select the tuples which satisfy the give condition form the grouped tuples.

COUNT will provide number of tuples in each group.

D.1.5 Data Manipulation Facilities in SEQUEL

The RDI facilities for insertion, deletion, and update of tuples are also provided via the SEQUEL data sublanguage. SEQUEL can be used to manipulate either one tuple at a time or a set of tuples with a single command. The current tuple of a particular cursor may be selected for some operation by means of the special predicate CURRENT TUPLE OF CURSOR. The values of a tuple may be set equal to constants, or to new values computed from their old values, or to the contents of a program variable suitably identified by a BIND command. These facilities will be illustrated by a series of examples. Since no result is returned to the calling program in these examples, no cursor name is included in the calls to SEQUEL.

EXAMPLE 4:

```

CALL SEQUEL ('UPDATE EMP SET SAL = SAL * 1.1
            WHERE DNO = 50');

```

Explanation

This command will update the salary value of the employees who are having ID as 50 to 1.1 times of his salary. This type of update is called as ORIENTED UPDATE.

Example 5:

```

CALL BIND ('PVSAL', ADDR (PVSAL));
CALL SEQUEL ('UPDATE EMP SET SAL = PVSAL WHERE
            CURRENT TUPLE OF CURSOR C3');

```


Explanation

This will update the tuple which is pointed by the cursor. This will update only one tuple. This type of update is called as INDIVIDUAL UPDATE.

Example 6:

```
CALL BIND ('PVEMPNO', ADDR (PVEMPNO));
CALL BIND ('PVNAME', ADDR (PVNAME));
CALL BIND ('PVMGR', ADDR (PVMGR));
CALL SEQUEL ('INSERT INTO EMP:
             < PVEMPNO, PVNAME, 50, "TRAINEE", 8500,
             PVMGR>');
```

Explanation

This example inserts a new employee tuple into EMP. The new tuple is constructed partly from constants and partly from the contents of program variables.

This type of insertion is called INDIVIDUAL INSERTION.

Example 7:

```
CALL SEQUEL ('DELETE EMP
             WHERE DNO =
             SELECT DNO
             FROM DEPT
             WHERE LOC = "EVANSTON");
```

Explanation

The SEQUEL assignment statement allows the result of a query to be copied into a new permanent or temporary relation in the database. This has the same effect as a query followed by the RDI operator KEEP. This type of deletion is called as set ORIENTED DELETION.

Example 8:

```
CALL SEQUEL ('UNDERPAID (NAME, SAL)
             SELECT NAME, SAL
             FROM EMP
             WHERE JOB = "PROGRAMMER"
             AND SAL < 10000');
```

Explanation

The new table UNDERPAID represents a snapshot taken from EMP at the moment the assignment was executed. UNDERPAID then becomes an independent relation and does not reflect any later changes to EMP.

D.1.6 Data Definition Facilities

System R takes a unified approach to data manipulation, definition, and control. Like queries and set oriented updates, the data definition facilities are invoked by means of the RDI operator `SEQUEL`.

The `SEQUEL` statement `CREATE TABLE` is used to create a new base relation. For each field of the new relation, the field name and datatype are specified. If desired, it may be specified at creation time that null values are not permitted in one or more fields of the new relation. A query executed on the relation will deliver its results in system-determined order (which depends upon the access path which the optimizer has chosen), unless the query has an `ORDER BY` clause. When a base relation is no longer useful, it may be deleted by issuing a `DROP TABLE` statement.

System R currently relies on the user to specify not only the base tables to be stored but also the RSS access paths to be maintained on them. Access paths include images and binary links. They may be specified by means of the `SEQUEL` verbs `CREATE` and `DROP`. Briefly, images are value ordering maintained on base relation by the RSS, using multilevel index structures. The index structures associate a value with one or more Tuple Identifiers (TID). A TID is an internal address which allows rapid access to a tuple. Images provide associative and sequential access on one or more fields which are called the sort fields of the image. An image may be declared to be `UNIQUE`, which forces each combination of sort field values to be unique in the relation. At most one image per relation may have the clustering property, which causes tuples whose sort field values are close to be physically stored near each other.

Binary links are access paths in the RSS which link tuples of one relation to related tuples of another relation through pointer chains. In System R, binary links are always employed in a value dependent manner: the user specifies that each tuple of relation 1 is to be linked to the tuples in relation 2 which have matching values in some field(s), and that the tuples on the link are to be ordered in some value-dependent way.

Example 9:

A user may specify a link from `DEPT` to `EMP` by matching `DNO`, and that `EMP` tuples on the link are to be ordered by `JOB` and `SAL`. This link is maintained automatically by the system. By declaring a link from `DEPT` to `EMP` on matching `DNO`, the user implicitly declares this to be a one-to-many relationship. Any attempts to define links or to insert or update tuples in violation of this rule will be refused. Like an image, a link may be declared to have the clustering property, which causes each tuple to be physically stored near its neighbor in the link.

It should be clearly noted that none of the access paths (images and binary links) contain any logical information other than that derivable from the data values themselves.

The query power of SEQUEL may be used to define a view as a relation derived from one or more other relations. This view may then be used in the same ways as a base table: queries may be written against it, other views may be defined on it, and in certain circumstances described below, it may be updated. Any SEQUEL query may be used as a view definition by means of a DEFINE VIEW statement.

Views are dynamically windows on the database, in that updates made to base tables become immediately visible via the views defined on these base tables. Where updates to views are supported, they are implemented in terms of updates to the underlying base tables. The SEQUEL statement which defines a view is recorded in a system-maintained catalog where it may be examined by authorized users. When an authorized user issues a DROP VIEW statement, the indicated view and all the other views defined in terms of it disappear from the system for this user and all other users.

If a modification is issued against a view, it can be supported only if the tuples of the view are associated one-to-one with tuples of an underlying base relation. In general, this means that the view must involve a single base relation and contain a key of that relation; otherwise, the modification statement is rejected. If the view satisfies the one-to-one rule, the WHERE clause of the SEQUEL modification statement is merged into the view definition; the result is optimized and the indicated update is made on the relevant tuples of the base relation.

Two final SEQUEL commands complete the discussion of the data definition facility. The first is KEEP TABLE, which causes a temporary table created, for example, by assignment0 to become permanent. (Temporary tables are destroyed when the user who created them logs off.). The second command is EXPAND TABLE, which adds new fields to an existing tuples, and are interpreted as having null values in the expanded fields until they are explicitly updated.

D.1.7 Data Control Facilities

Data control facilities at the RDI have four aspects:

1. Transaction
2. Authorization
3. Integrity assertions
4. Triggers

Transaction

A Transaction is a series of RDI calls which the user wishes to be processed as an atomic act. The meaning of “atomic” depends on the level of consis-

tency specified by the user. The highest level of consistency, Level 3, requires that a user's transactions appear to be serialized with the transactions of other concurrent users. The user controls transactions by the RDI operators `BEGIN_TRANS` and `END_TRANS`. The user may specify save points within a transaction by the RDI operator `SAVE`. As long as a transaction is active, the user may back up to the beginning of the transaction or to any internal save point by the operator `RESTORE`. This operator restores all changes made to the data transaction. No cursors may remain active (open) beyond the end of a transaction. The RDI transactions are implemented directly by RSI transactions, so the TDI commands `BEGIN_TRANS`, `END_TRANS`, `SAVE`, and `RESTORE` are passed through to the RSI with some RDS bookkeeping to permit the restoration of its internal state.

System R does not require a particular individual to be the database administrator, but allows each user to create his own data objects by executing the `SEQUEL` statements `CREATE TABLE` and `DEFINE VIEW`. The creator of a new object receives full authorization to perform all operations on the object (subject, of course, to his authorization for the underlying tables, if it is a view). The user may then grant selected capabilities may be independently granted for each table or view: `READ`, `INSERT`, `DELETE`, `UPDATE`, `DROP`, `EXPAND`, `IMAGE` specification, `LINK` specification, and `CONTROL`.

For each capability which a user possesses for a given table, he may optionally have `GRANT` authority (the authority to further grant or revoke the capability to/from other users).

Authorization

System R relies primarily on its view mechanism for read authorization. If it is desired to allow a user to read only tuples of employees in department 50, and not to see their salaries, then this portion of the `EMP` table can be defined as a view and granted to the user. No special statistical access is distinguished, since the same effect can be achieved by defining a view. To make the view mechanism more useful for authorization purposes, the reserved word `USER` is always interpreted as the user-id of the current user. Thus the following `SEQUEL` statement defines a view of all those employees in the same department as the current user:

Example 10: To view all Employees in the same Department.

```

DEFINE VIEW VEMP AS:
  SELECT *
  FROM EMP
  WHERE DNO =
        SELECT DNO
        FROM EMP
        WHERE NAME=USER

```

Integrity Assertions

The third important aspect of data control is that of integrity assertions. Any SEQUEL predicate may be stated as an assertion about the integrity of data in a base table or view. At the time the assertion is made (by an ASSERT statement in SEQUEL), its truth is checked; if true, the assertion is automatically enforced until it is explicitly dropped by a DROP ASSERTION statement. Any data modification, by any user, which violates an active integrity assertion is rejected. Assertion may apply to individual tuples (e.g., “No employee’s salary exceeds \$5000”) or to sets of tuples (e.g., “The average salary of each department is less than \$2000”). Assertions may describe permissible states of the database (as in the examples above) or permissible transitions in the database. For this latter purpose the keywords OLD and NEW are used in SEQUEL to denote data values before and after modification.

Example 11:

Consider the situation that, each employee’s salary must be nondecreasing.

```
ASSERT ON UPDATE TO EMP::NEW SAL ≥ OLD SAL
```

Explanation

Unless otherwise specified, integrity assertions are checked and enforced at the end of each transaction. Transaction assertions compare the state before the transaction began with the state after the transaction concluded. If some assertion is not satisfied, the transaction is backed out to its beginning point. This permits complex updates to be done in several steps (several calls to SEQUEL, bracketed by BEGIN_TRANS and END_TRANS), which may cause the database to pass through intermediate states which temporarily violate one or more assertions. However, if an assertion is specified as IMMEDIATE, it cannot be suspended within a transaction, but is enforced after each data modification. In addition, “Integrity points” within a transaction may be established by the SEQUEL command ENFORCE INTEGRITY. This command allows user to guard against having a long transaction is backed out its most recent integrity point.

Triggers

The fourth aspect of data control, triggers, is a generalization of the concept of assertions. A trigger causes a prespecified sequence of SEQUEL statements to be executed whenever some triggering event occurs. The triggering event may be retrieval, insertion, deletion, or update of a particular base table or view. For example, suppose that in our example database, the NEMPS field of the DEPT table denotes the number of employees in each department. This

value might be kept up to date automatically by the following three triggers (as in assertions, the keywords OLD and NEW denote data values before and after the change which invoked the trigger):

Example 12:

```

DEFINE TRIGGER EMPINS
    ON INSERTION OF EMP:
        (UPDATE DEPT
         SET NEMPS = NEMPS +1
         WHERE DNO = NEW EMP.DNO)
DELETE TRIGGER EMPDEL
    ON DELETION OF EMP:
        (UPDATE DEPT
         SET NEMPS = NEMPS -1
         WHERE DNO = OLD EMP.DNO)
DEFINE TRIGGER EMPUPD
    ON UPDATE OF EMP:
        (UPDATE DEPT
         SET NEMPS = NEMPS -1
         WHERE DNO = OLD EMP.DNO;
         UPDATE DEPT
         SET NEMPS = NEMPS +1
         WHERE DNO = NEW EMP.DNO)

```

Explanation

The RDS automatically maintains a set of catalog relations which describe the other relations, views, images, links, assertions, and triggers known to the system. Each user may access a set of views of the system catalogs which contain information pertinent to him. Access to catalog relations is made in exactly the same way as other relations are accessed (i.e., by SEQUEL queries). Of course, no user is authorized to modify the contents of a catalog directly, but any authorized user may modify a catalog indirectly by actions such as creating a table. In addition, a user may enter comments into his various catalog entries by means of the COMMENT statement.

D.2 Relational Data System

RDI is the principal external interface of the System R. It provides high level, data independence facilities for data retrieval, manipulation, definition, and control. The data definition facilities of the RDI allow a variety of alternative relational views to be defined on common underlying data. The Relational Data System (RDS) is the subsystem which implements the RDI. The RDS

contains an optimizer which plans the execution of each RDI command, choosing a low cost access path to data from among those provided by the RSS. The RDI consists of a set of operators which may be called from PL/I or other host programming languages. All the facilities of the SEQUEL data sublanguage are available at the RDI by means of the RDI operator called SEQUEL. The SEQUEL language can be supported as a stand-alone interface by a simple program, written on top of the RDI, which handles terminal communications. In addition, programs may be written on top of the RDI to support other relational interfaces, such as Query By Example (QBE) or to simulate nonrelational interfaces.

The facilities of the RDI are basically those of the SEQUEL data sublanguage. Several changes have been made to SEQUEL since the earlier publication of the language; they are described below.

Example 13:

Consider the following database of employees and their departments:
 EMP (EMPNO, NAME, DNO, JOB, SAL, MGR)
 DEPT (DNO, DNAME, LOC, NEMPS)

Explanation

The RDI interface SEQUEL to a host programming language by means of a concept called a cursor. A cursor is a name which is used at the RDI to identify a set of tuples called its active set (e.g., the result of a query) and furthermore to maintain a position on the tuple of the set. The cursor is associated with a set of tuples by means of the RDI operator FETCH.

Consider the following commands:

Example 14:

```
CALL BIND ('X', ADDR(X));
CALL BIND ('Y', ADDR(Y));
CALL SEQUEL (C1, 'SELECT NAME: X, SAL: Y FROM EMP
WHERE JOP =
          "PROGRAMMER" ');
```

Explanation

The SEQUEL call has the effect of associating the cursor C1 with the set of tuples which satisfy the query and positioning it just before the first such tuple. The optimizer is invoked to choose an access path whereby the tuples may be materialized. However, no tuples are actually materialized in response to the SEQUEL call. The materialization of tuples is done as they are called for, one at a time, by the FETCH operator. Each call to FETCH delivers the next tuple of the active set into program variables X and Y.

```
CALL FETCH (C1);
```

A program may wish to write a SEQUEL predicate based on the contents of a program variable.

Example 15:

To find the programmers whose department number matches the contents of program variable Z. This facility is also provided by the RDI BIND operator, as follows:

```
CALL BIND ('X', ADDR (X));
CALL BIND ('Y', ADDR (Y));
CALL BIND ('Z', ADDR (Z));
CALL SEQUEL (C1, 'SELECT NAME: X FROM EMP WHERE JOB
               = "PROGRAMMER" AND DNO = Z');
```

```
CALL FETCH (C1);
```

Explanation

Some programs may not know in advance the degree and datatypes of the tuples to be returned by a query. An example of such a program is one which supports an interactive user by allowing him to type in queries and display the results. This type of program need not specify in its SEQUEL call the variable into which the result is to be delivered. The program may issue a SEQUEL query, followed by the DESCRIBE operator which returns the degree and datatypes. The program then specifies the destination of the tuples in its FETCH commands. The following example illustrates these techniques:

Example 16:

```
CALL SEQUEL (C1, 'SELECT * FORM EMP WHERE DNO = 50');
```

Explanation

This statement invokes the optimizer to choose an access path for the given query and associates cursor C1 with its active set.

Example 17:

```
CALL DESCRIBE (C1, DEGREE, P);
```

Explanation

P is a pointer to an array in which the description of the active set of C1 is to be returned. The RDI returns the degree of the active set in DEGREE, and the datatypes and lengths of the tuple components in the elements of the array. If the array (which contains an entry describing its own length) is too short to hold the description of a topic, the calling program must allocate a larger array and make another call to DESCRIBE. Having obtained a description

of the tuples to be returned, the calling program may proceed to allocate a structure to hold the tuples and may specify the location of this structure in its FETCH command:

Example 18:
CALL FETCH (C1, Q);

Explanation

Q is a pointer to an array of pointers which specify where the individual components of the tuple are to be delivered. If this “destination” parameter is present in a FETCH command, it overrides any destination which may have been specified in the SEQUEL command which defined the active set of C1.

A special RDI operator Open is provided as a shorthand method to associate a cursor with an entire relation. For example, the command:

Example 19:
CALL OPEN (C1, ‘EMP’);
is exactly equivalent to
CALL SEQUEL (C1, ‘SELECT * FROM EMP’);

Explanation

The use of OPEN is slightly preferable to the use of SEQUEL to open a cursor on a relation, since OPEN avoids the use of the SEQUEL parser.

D.3 DB2

D.3.1 Introduction to DB2

DB2 is a strategic product from IBM. It is available on all of IBM’s key platforms. IBM’s Information Warehouse architecture employs DB2 as a key component. DB2 is a relational database management system. The relational model is founded on the mathematics of set theory, thereby providing a solid theoretical base for the management of data. Relational databases are typically easier to use and maintain than databases based on nonrelational technology. An IBM relational database management system that is available as a licensed program on several operating systems. Programmers and users of DB2 can create, access, modify, and delete data in relational tables using a variety of interfaces.

DB2’s foundation in the relational model also provides it with improved data availability, data integrity, and data security because the relational model rigorously defines as part of the database. Programmers and users of DB2 can create, access, modify, and delete data in relational tables using a variety of interfaces. Because DB2 is a relational database management system, it is

more easily lends itself to a distributed implementation. Tables can be located at desperate locations across a network and application can seamlessly access information in those tables from within a single program using DB2. DB2 uses SQL, which is the standard language for maintaining and querying relational databases. DB2 was one of the first databases to uses SQL exclusively to access data. SQL provides the benefits of quick data retrieval, modification, definition, and control. It is also transportable from environment to environment.

DB2 Universal Database Enterprise – Extended Edition (DB2 UDB EEE) was designed to support the very large databases that business intelligence applications often require. IBM DB2 can work with Windows, Linux, AIX, and Solaris.

D.3.2 Definition of DB2 Data Structures

DB2 data structures are referred to as objects. We can use SQL to define DB2 data structure. Each DB2 object is used to support the structure of the data being stored. A description of each type of DB2 object follows:

These objects are created with the DCL verbs of SQL and must be created in a specific order. The hierarchy of DB2 objects is listed in Fig. D.2.

D.3.3 DB2 Stored Procedure

Stored procedures are specialized programs that are stored in relational database management system instead of an external code library. Stored procedure must be directly and explicitly invoked before it can execute.

DB2 equips user to perform a variety of tasks on existing stored procedures, such as:

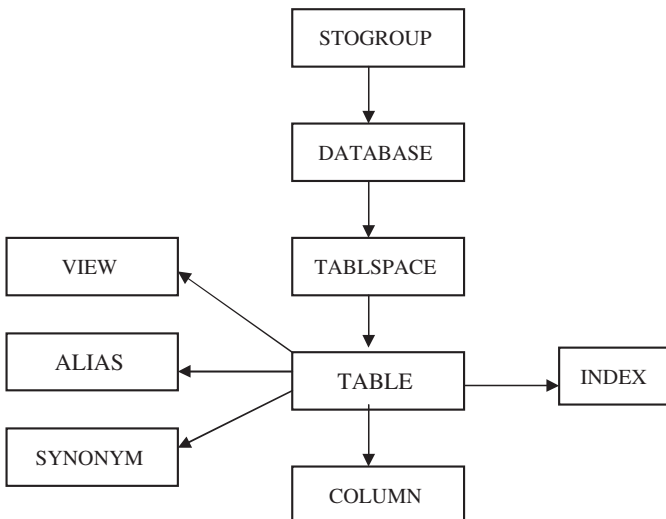


Fig. D.2. The DB2 object hierarchy

ALIAS	A locally defined name for a table or view in the same local DB2 subsystems or in a remote DB2 subsystem.
COLUMN	A single, nondecomposable data element in a DB2 table.
DATABASE	A logical grouping of DB2 objects related by common characteristics such as logical functionality, relation to an application system or subsystem, or type of data.
INDEX	A DB2 object that consist of one or more VSAM data sets.
STOGROUP	A series of DASD volumes assigned a unique name and used to allocate VSAM data sets for DB2 objects.
TABLE	A DB2b object that consists of columns and rows that define the physical characteristics of the data to be stored.
TABLE SPACE	A DB2 object that defines the physical structure of the data sets used to house the DB2 table data.
VIEW	A virtual table consisting of a SQL SELECT statement that accesses data from one or more tables or views.

- Viewing
- Modifying
- Running and testing
- Copying and pasting stored procedures across connections
- Building, in one step, stored procedures on target databases
- Customizing settings to enable remote debugging of installed procedures.

Stored procedures run in a separate DB2 address space known as the stored procedure address space. To execute a stored procedure, a program must issue the SQL call statement. When the call is issued, the name of the stored procedure and its list of parameters are send to DB2. DB2 searches SYSIBM.SYS PROCEDURES for the appropriate row that 1 defines the stored procedure to be executed.

DB2 Stored Procedure Builder provides a single development environment that supports multiple languages – including Java and SQL procedure language – and the entire DB2 Universal Database™. DB2 Stored Procedure Builder can launch from the DB2 Program Group or from add-in menus on IBM VisualAge® for Java, Microsoft® Visual C++, and Microsoft Visual Basic. After start-up, the wizards in DB2 Stored Procedure Builder take user through each task, one step at a time. The first step is to define user project. Simply follow the wizards, which will ask user to provide a project name and decide how user want to connect to the database. User also will be asked for a logon name and password. Once user project is defined, users are ready to create a new stored procedure or work on an existing one. Launching a new procedure, The Stored Procedure Builder Project View window, gives user a picture of all users existing stored procedures and their connections. This is the window where user can select existing procedures for modification or, using the menu or toolbar command, create a new stored procedure.

D.3.4 DB2 Processing Environment

When accessing DB2 data an application program is not limited to a specific technological platform. The different environments are Time Sharing Option (TSO), Customer Information Control System (CICS), IMS/VIS, Call Attach Facility (CAF), and RRSAF as shown in Fig. D.3. Each of this environment acts as a door that provides access to DB2 data. Each DB2 program must be connected to DB2 by an attachment facility, which is the mechanism by which an environment is connected to a DB2 subsystem. Additionally, a thread must be established for each embedded SQL program that is executing. A thread is control structure used by DB2 to communicate with an application program. The thread is used to send requests to DB2, to send data from DB2 to the program, and to communicate the states of each SQL statement after it is executed.

Time Sharing Option

TSO is one of the five basic environments from which DB2 data can be accessed. TSO enables users to interact with Multiple Virtual Storage (MVS) using an online interface. The Interactive System Productivity facility (ISPF), provides the mechanism for communicating by panels, which is the common method for interaction between TSO application and users. The TSO Attachment Facility provides access to DB2 resources in two ways.

- Online in the TSO foreground, driven by application programs, CLISTs, or REXX EXECs coded to communicate with DB2 and TSO, possibly using ISPF panels.

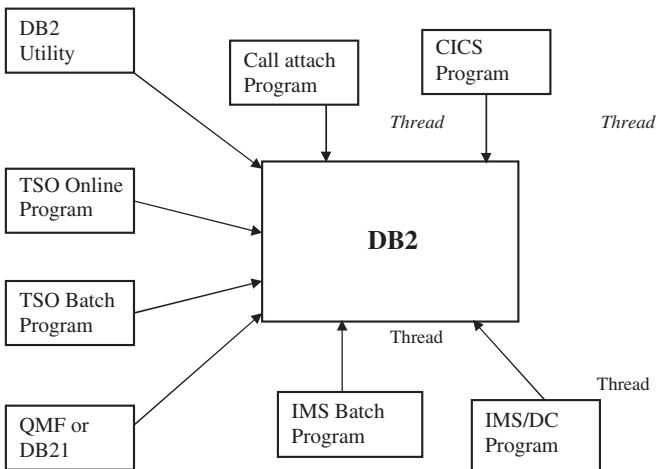


Fig. D.3. DB2 processing environment

- In batch mode using the TSO Terminal Monitor Program, IKJEFT01 (or IKJEFT1B), to invoke the DSN command and run a DB2 application program.

Customer Information Control System

CICS is a teleprocessing monitor that enables programmers to develop online, translation-based programs. By means of Basic Mapping Support (BMS) and the data communications facilities of CICS, programs can display formatted data on screens and receive formatted data from users. When DB2 data are accessed using CICS, multiple threads can be active simultaneously, giving multiple users concurrent access to a DB2 subsystems of a single CICS region.

Information Management System

Information Management System (IMS) is IBM's prerelational database management system offering. It is based on the structuring of related data items in inverted tree or hierarchies. IMS is combination of two components:

- IMS/DB the database management systems
- IMS/TM, the transaction management environment, also known as IMS/DC.

IMS programs are categorized, based on the environment in which they run and the types of databases they can access. The four types of IMS programs are batch programs, batch message processors, message processing programs, and fast path programs.

Query Management Facility

IBM's Query Management Facility (QMF) is an interactive query tool used to produce formatted query output. QMF forms enable user to perform the following:

- Code a different column heading
- Specify control breaks
- Code control-break heading and footing text
- Specify edit codes to transform column data
- Compute averages, percentages, standard deviations, and totals for specific columns.
- Display summary results across a row, suppressing the supporting detail rows
- Omit columns in the query from the report.

Call Attach Facility

CAF is used to manage connections between DB2 and batch and online TSO application programs. CAF programs can be executed as one of the following:

- An MVS batch job
- A started task
- A TSO batch job
- An online TSO application

CAF is used to control a program's connection to DB2. The DB2 program communicates to DB2 through the CAF language interface, DSNALI. Five CAF calls are used to control the connections.

CONNECT	Establishes a connection between the programs MVS address space and DB2
DISCONNECT	Eliminates the connection between the programs MVS address space and DB2
OPEN	Establishes a thread for the program to communicate with DB2
CLOSE	Terminates the thread
TRANSLATE	Provides the program with DB2 error message information, placing it in the SQLCA

D.3.5 DB2 Commands

DB2 commands are operator issued request that administer DB2 resources and environments. There are six categories of DB2 commands, which are delineated by the environment from which they are issued. These are:

- DB2 environment command
- DSN commands
- IMS commands
- CICS commands
- TSO commands
- IRLM commands

DB2 Environment Command

There are three types of environment commands:

- *Information gathering command.* It is used to monitor DB2 objects and resources.

- *Administrative commands.* These are provided to assist the user with the active administration, resources specification, and environment modification of DB2 sub systems.
- *Environment control commands.* These commands affect the status of the DB2 subsystem and the distributed data facility.

All DB2 environment commands have a common structure as follows:
cp command operand

DSN Commands

DSN commands are actually the subcommands of the DSN command processor. DSN is a control program that enables users to issue DB2 environment commands, plan management commands, and commands to develop and run application development programs.

IMS Commands

IMS commands affect the operation of DB2 and IMS/TM. IMS commands must be issued from a valid terminal connected to IMS/TM and the issuer must have the appropriate IMS authority.

CISS Command

The CICS commands affect the operation of DB2 and CICS. CICS commands must be issued from a valid terminal connected to CICS and the issuer must have the appropriate CICS authority.

TSO Command

The DB2 TSO commands are CLISTS that can be used to help compile and run DB2 programs or build utility JCL. There are two TSO commands:

DSNH Can be used to precompiled, translate, compile, link, bind, and run DB2 application programs.

DSNU Can be used to generate JCL for any online DB2 utility.

IRLM Commands

The IRLM commands affect the operation of the IRLM defined to a DB2 subsystem. IRLM commands must originate from an MVS console, and the issuer must have the appropriate security.

D.3.6 Data Sharing in DB2

DB2 data sharing allows applications running on multiple DB2 subsystems to concurrently read and write to the same data set. Data sharing enables multiple DB2 subsystems to behave as one. DB2 data sharing provides many benefits. The primary benefit of data sharing is to provide increased availability to data. An additional benefit is expanded capacity. Each data-sharing group may consist of multiple members, application programs are provided with enhanced data availability. Data sharing increases the flexibility of configuring DB2.

DB2 and the INTERNET

There are two main reasons for DB2 professionals to use the Internet:

- To develop applications that allow for Web-based access to DB2 data
- To search for DB2 product, technical, and training information

IBM provides two options for accessing DB2 data over the web: DB2WWW and Net.Data.

DB2 WWW

DB2 WWW is an IBM product for connecting DB2 databases to the Web. Using a Web browser and DB2 WWW, companies can use the Internet as a front end to DB2 databases. Using DB2 WWW, data stored in DB2 tables is presented to users in style of a Web page. DB2WWW provides two-tier and three-tier client/server environment.

Net. Data

Net. Data, another IBM product, is an upwardly compatible follow-on version of DB2 WWW. DB2 WWW applications are compatible with Net. Data.

Data Warehousing with DB2

A data warehouse is best defined by the type and the manner of data stored in it and the people who use the data. Data warehousing enable the end users to have the access to corporate operational data to follow and respond to business trends. Data warehousing enables an organization to make information available for analytical processing and decision making.

A data warehouse is a collection of data that are

- Separate from operational systems
- Accessible and available for queries
- Subject oriented by business

- Integrated and consistently named and defined
- Associated with defined period of time
- Static, or nonvolatile, such that updates are not made

The data warehouse defines the manner in which data

- Are systematically constructed and cleansed
- Are transformed in to a consistent view
- Are distributed wherever it is needed
- Are made easily accessible
- Are manipulated for optimal access by disparate processes

DB2's hardware-based data compression techniques are optimal for the data-warehousing environment.

D.3.7 Conclusion

Today's competitive business climate dictates that companies derive more information out of their databases. Analysts looking for business trends in their company's database pose increasingly complex queries, often through query generator front-end tools. Businesses must extract as much useful information as possible from the large volumes of data that they keep, making parallel database technology a key component of such business intelligence applications. Enterprises and independent software vendors continue to require support for more application productivity and capability. And many growing enterprises have data stored in many systems, often both file systems and database systems from a variety of vendors. All of these areas contribute to high performance at low cost. Being able to access and manage these data with high performance, fast response time and low total cost of ownership is a compelling advantage in business today.

D.4 Informix

D.4.1 Introduction to Informix

In 1980, Roger Sippl and Laura King founded Relational Database Systems (RDS) in Sunnyvale, California. In February 1988, RDS merged with Innovative Software of Overland Park, Kansas, which had been founded by Mike Brown and Mark Callegari in 1979. The 1988 merger, which was the first major acquisition by Informix, was an effort to broaden platform coverage for the Informix DBMS and add needed end-user tools. The tools (initially Macintosh-based) never did exactly meet the executives' expectations, but the acquisition could be interpreted as a welcome gesture of support for the end user.

Roger Sippl and Laura King founded Relational Database Systems at a time when both relational database management and the UNIX operating system were just beginning to be encountered on mini- and micro-computers: Rather than tailoring the DBMS for mainframe hardware and proprietary operating systems, RDS built a product that used an open operating system, ran on small, general-purpose hardware, used a standard programming interface (SQL), and supplied a number of end-user tools and utilities. RDS was among the first companies to bring enterprise-level database management out of the computer room and onto the desktop.

Informix based its relational database management products on open systems and standards such as industry-standard Structured Query Language (SQL) and the UNIX operating system. Two notable innovations have propelled Informix to an industry-leading position in database management technology: the parallel processing capabilities of Informix Dynamic Scalable Architecture (DSA) and the ability to extend relational database management to new, complex datatypes using the object-relational powers of INFORMIX-Universal Server. Informix introduced its first RDBMSs – INFORMIX-Standard Engine and INFORMIX-OnLine.

There are four major types of Informix RDBMS product users. These users include the database administrator or DBA, the system administrator or SA, the application developer, and the application user. The DBA is the person generally responsible for keeping the Informix RDBMS running. The SA is responsible for the operating system and the machine on which the RDBMS is running. An application developer builds the applications that access the Informix RDBMS. Finally, the application user is the person who runs the application to access the data in the Informix RDBMS and performs specific tasks on that data.

All user applications that access the Informix RDBMS are considered clients, and the actual Informix RDBMS is considered the server. The client/server process is natural in the RDBMS world because the RDBMS is its own software process, running throughout the day and waiting for tasks to perform. A client can have the Informix RDBMS server to perform one of four basic tasks. These tasks are select, insert, update, or delete. A select is considered a query because it looks at a specific set of data. An insert actually adds new information, usually an entire row, into the database. An update task changes existing data. A delete actually removes an entire row of data; consider it the opposite of an insert.

D.4.2 Informix SQL and ANSI SQL

The SQL version that Informix products support is compatible with standard SQL (it is also compatible with the IBM version of the language). However, it does contain *extensions* to the standard; that is, extra options or features for certain statements, and looser rules for others. Most of the differences occur

in the statements that are not in everyday use. For example, few differences occur in the SELECT statement, which accounts for 90% of the SQL use for a typical person. However, the extensions do exist and create a conflict. Thousands of Informix customers have embedded Informix-style SQL in programs and stored procedures. They rely on Informix to keep its language the same. Other customers require the ability to use databases in a way that conforms exactly to the ANSI standard. They rely on Informix to change its language to conform.

- Informix resolves the conflict with the following compromise: The Informix version of SQL, with its extensions to the standard, is available by default.
- User can ask any Informix SQL language processor to check the use of SQL and post a warning flag whenever user use an Informix extension.

D.4.3 Software Dependencies

IBM Informix® Dynamic Server TM 9.30 (IDS) delivers a first-in-class database that combines the robustness, high performance, and scalability of the IBM Informix flagship relational database management system (RDBMS) with advanced object-relational technology to store, retrieve, and manage rich data intelligently and efficiently. IBM IDS is built on the IBM Informix Dynamic Scalable Architecture TM (DSA) – the goal of which is to provide the most effective parallel database architecture available – to help manage increasingly large and complex databases while substantially improving overall system performance and scalability. IBM IDS delivers proven technology that efficiently integrates new and complex data directly into the server. It handles time-series, geospatial, geodetic, XML, video, image, and other user-defined data – side by side with traditional legacy data – to meet the most rigorous data and business demands. IBM IDS allows user to lower the total-cost-of-ownership by leveraging existing standards for development tools, systems infrastructure, and customer skill sets as well as its development-neutral environment and comprehensive array of application development tools for rapid deployment of applications under Linux, Windows, and UNIX (Fig. D.4).

The dynamic scalable architecture of IBM IDS provides the ability to fully exploit the processing power available in SMP environments by performing database activities in parallel (such as I/O, complex queries, index builds, log recovery, inserts, and backups and restores). It was designed from the ground up to provide built-in multithreading and parallel processing capabilities, thus providing the most efficient use of all available system resources.

Virtual processors and multithreading. IBM IDS gives user the unique ability to scale user database system by employing a dynamically configurable pool of database server processes (virtual processors) and dividing large tasks into subtasks for rapid processing. The virtual processors schedule and manage user requests and parallel subtasks using multiple concurrent threads.

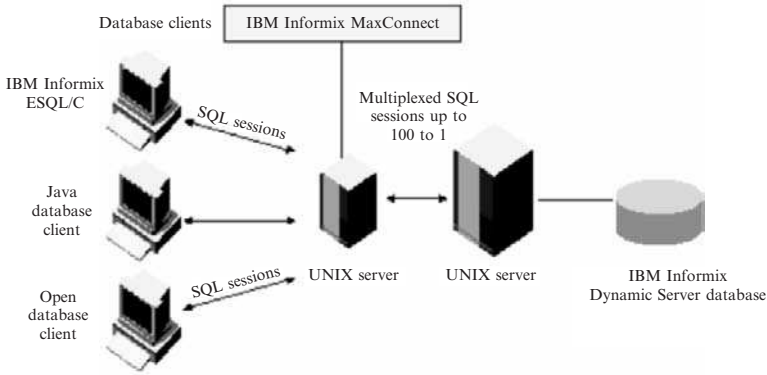


Fig. D.4. IBM Informix Max Connect multiplexes a number of SQL sessions into a much smaller number of communication sessions at the IBM Informix database level maximizing scalability and performance

A thread represents a discrete task within a database server process and many threads may execute in parallel across the pool of virtual processors. When a thread is waiting for a resource, a virtual processor can work on behalf of another thread. Not only can one virtual processor respond to a large number of user requests, but one user request can also be distributed across multiple virtual processors. For example, for a processing-intensive request, such as a multitable join, the database server divides the task into multiple subtasks and then spreads these subtasks across all the available virtual processors.

D.4.4 New Features in Version 7.3

Most of the new features for Version 7.3 of Informix Dynamic Server fall into five major areas:

- Reliability, availability, and serviceability
- Performance
- Windows NT-specific features
- Application migration
- Manageability

Several additional features affect connectivity, replication, and the optical subsystem. The features are:

- *Performance:* Enhancements to the SELECT statement to allow selection of the first n rows.
- *Application migration:*

1. New functions for case-insensitive search (UPPER, LOWER, INITCAP)
2. New functions for string manipulations (REPLACE, SUBSTR, LPAD, RPAD)
3. New CASE expression
4. New NVL and DECODE functions
5. New date-conversion functions (TO_CHAR and TO_DATE)
6. New options for the DBINFO function
7. Enhancements to the CREATE VIEW and EXECUTE PROCEDURE statements

New Features in Version 8.2

Following are new features that have been implemented in Version 8.2 of Dynamic Server with AD and XP Options:

- Global Language Support (GLS)
- New aggregates: STDEV, RANGE, and VARIANCE
- New TABLE lock mode for the LOCK MODE clause of ALTER TABLE and CREATE TABLE statement
- Support for specifying a lock on one or more rows for the Cursor Stability isolation level

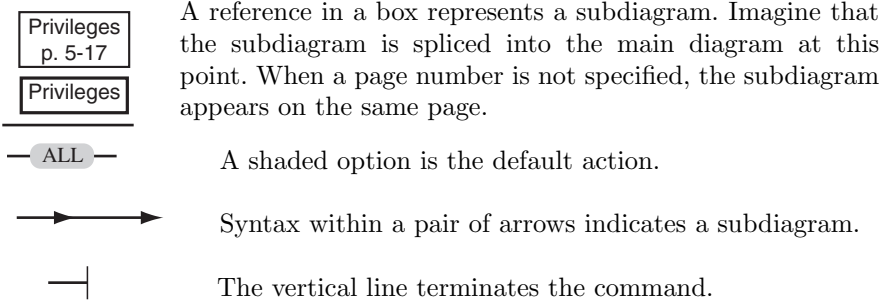
Following *features*, which were introduced in Version 8.1 of Dynamic Server with AD and XP Options:

- The CASE expression in certain Structured Query Language (SQL) statements
- New join methods for use across multiple computers
- Nonlogging tables
- External tables for high-performance loading and unloading

Command-Line Conventions

This section defines the format of commands that are available in Informix products. These commands have their own conventions, which might include alternative forms of a command, required and optional parts of the command, and so forth. Each diagram displays the sequences of required and optional elements that are valid in a command. A diagram begins at the upper-left corner with a command. It ends at the upper-right corner with a vertical line. Between these points, user can trace any path that does not stop or back up. Each path describes a valid form of the command. User must supply a value for words that are in italics.

Element	Description
command	This required element is usually the product name or other short word that invokes the product or calls the compiler or preprocessor script for a compiled Informix product. It might appear alone or precede one or more options. User must spell a command exactly as shown and use lowercase letters.
<i>Variable</i>	A word in italics represents a value that user must supply, such as a database, file, or program name. A table following the diagram explains the value.
-flag	A flag is usually an abbreviation for a function, menu, or option name or for a compiler or preprocessor argument. User must enter a flag exactly as shown, including the preceding hyphen.
.ext	A filename extension, such as .sql or .cob , might follow a variable that represents a filename. Type this extension exactly as shown, immediately after the name of the file. The extension might be optional in certain products.
(. , ; + * - /)	Punctuation and mathematical notations are literal symbols that user must enter exactly as shown.
' '	Single quotes are literal symbols that user must enter as shown.



How to Read a Command-Line Diagram

Figure D.5 shows a command-line diagram. To construct a command correctly, start at the top left with the command. Then follow the diagram to the right, including the elements that user want. The elements in the diagram are case sensitive.

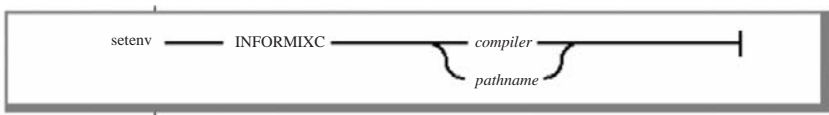


Fig. D.5. Example of a command line diagram

To construct a command correctly, start at the top left with the command. Then follow the diagram to the right, including the elements that user want. The elements in the diagram are case sensitive.

These are the steps to be followed:

1. Type the word `setenv`.
2. Type the word `INFORMIXC`.
3. Supply either a compiler name or pathname. After user choose *compiler* or *pathname*, user come to the terminator. User command is complete.
4. Press RETURN to execute the command.

Informix's current application development products, are INFORMIX-NewEra and INFORMIX-4GL, have been incorporated into the Universal Tools Strategy announced in March of 1997. The Universal Tools Strategy gives application developers a wide choice of application development tools for Informix database servers, permitting developers to take a modular, component-based, open tools approach. The INFORMIX-Data Director family of plug-in modules lets developers extend, manage, and deploy applications for INFORMIX-Universal Server using their choice of Informix and other industry-standard tools.

The following products are included under the Universal Tools Strategy:

- INFORMIX-Data Director for Visual Basic
- INFORMIX-Data Director for Java (formerly J works)
- INFORMIX-New Era
- INFORMIX-4GL
- INFORMIX-Java Object Interface (JOI) (formerly Java API)
- INFORMIX-JDBC
- INFORMIX-C++ Object Interface (COI)
- INFORMIX-CLI
- INFORMIX-ESQL/C
- INFORMIX-Developer SDK

D.4.5 Conclusion

The powerful and extensible IBM Informix Database Server is designed to deliver breakthrough scalability, manageability, and performance. IBM IDS enables user to manage business logic, create and access rich data, and define complex database functions in an integrated, intelligent information management system. With IBM IDS, user benefit from the performance and scalability offered by the proven Dynamic Server Architecture, while gaining all the advantages of object-oriented technology and unlimited extensibility – resulting in an immense capacity to grow and adapt to ever-changing needs.

Bibliography

1. Abiteboul, S., Hull, R., and Vianu, V., *Foundations of Databases*, Addison-Wesley, Reading, MA, 1995
2. Aho, A., Beeri, C., and Ullman, J., The Theory of Joins in Relational Databases, *ACM Transactions on Database Systems*, 4:3, 1979
3. Aho, A., Sagiv, Y., and Ullman, J., Efficient Optimization of a Class of Relational Expressions, *ACM Transactions on Database Systems*, 4:4, 1979
4. Aho, A., and Ullman, J., Universality of Data Retrieval Languages, *Proceedings of the POPL Conference*, San Antonio, TX, ACM, 1979
5. Albano, A., De Antonellis, V., and Di Leva, A. (Eds.), *Computer-Aided Database design: The DATAID Project*, North-Holland, Amsterdam, 1985
6. Atzeni, P., and De Antonellis, V., *Relational Database Theory*, Benjamin-Cummings, Menlo Park, CA, 1993
7. Atzeni, P., Mecca, G., and Merialdo, P., To Weave the Web, *Proceedings of 23rd International Conference on Very Large Data Bases*, Athens, Greece, Morgan Kaufmann, San Francisco, 1997
8. Atzeni, P., Mecca, G., and Merialdo, P., Design and Maintenance of Data-Intensive Web Sites, *Proceedings of 6th International Conference on Extending Database Technology*, Valencia, Spain, Lecture Notes in Computer Science, Vol. 1377, pp. 436–450, Springer, Berlin Heidelberg New York, 1998
9. Astrahan, M., et al., System R: A Relational Approach to Data Base Management, *ACM Transactions on Database Systems*, 1:2, 1976
10. Armstrong, W., Dependency Structures of Data Base Relationships, *Proceedings of the IFIP Congress*, 1974
11. Arkinson, M., and Buneman, P., Types and Persistence in Database Programming Languages, *ACM Computing Surveys*, 19:2, 1987
12. Atzeni, P., and De Antonellis, V., *Relational Database Theory*, Benjamin/Cummings, Menlo Park, CA, 1993
13. ANSI, *American National Standards Institute: The Database Language SQL*, Document ANSI X3.135, 1986
14. Bachman, C., The Data Structure Set Model. In: Rustin (Ed.), *Proceedings of 1974 ACM-SIGMOD Workshop on Data Description, Access and Control*, Ann Arbor, MI, May 1–3, 1974
15. Baeza-Yates, R., and Larson, P.A., Performance of B1-trees with Partial Expansions, *IEEE Transactions on Knowledge and Data Engineering*, 1:2, 1989

16. Baeza-Yates, R., and Ribero-Neto, B., *Modern Information Retrieval*, Addison-Wesley, Reading, MA, 1999
17. Bancilhon, F., Delobel, C., and Kanellakis, P. (Eds.), *Building an Object-Oriented Database System: The Story of O2*, Morgan Kaufmann, San Mateo, CA, 1992
18. Batini, C., Ceri, S., and Navathe, S.B., *Conceptual Database Design, an Entity-Relationship Approach*, Benjamin-Cummings, Menlo Park, CA, 1992
19. Bernstein, P.A., Hadzilacos, V., and Goodman, N., *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, Reading, MA, 1987
20. Bernstein, P.A., Middleware: A Model for Distributed System Services. *Communications of the ACM*, 39:2, 89–98, 1996
21. Bertino, E., and Martino, L., *Object-oriented Database Systems: Concepts and Architectures*, Addison-Wesley, Reading, MA, 1993
22. Brodie, M.L., and Stonebraker, M., *Legacy Systems: Gateways, Interfaces & the Incremental Approach*, Morgan Kaufmann, San Mateo, CA, 1995
23. Bachman, C., Data Structure Diagrams, Data Base (Bulletin of ACM SIG-FIDET), 1:2, 1969
24. Bachman, C., The Programmer as a Navigator, The California Association of Community Managers, 16:1, 1973
25. Bachman, C., The Data Structure Set Model. In: Rustin (Ed.), *Proceedings of 1974 ACM-SIGMOD Workshop on Data Description, Access and Control*, Ann Arbor, MI, 1974
26. Bachman, C., and Williams, S., A General Purpose Programming System for Random Access Memories, *Proceedings of the Fall Joint Computer Conference, AFIPS*, 26, 1964
27. Cannan, S.J., and Otten. G.A.M., *SQL-The Standard Handbook*, McGraw-Hill, New York, 1992
28. Cattell, R.G.G., *Object Data Management – Object Oriented and Extended Relational Database Systems*, revised edition, Addison-Wesley, Reading, MA, 1994
29. Ceri, S. (Ed.), *Methodology and Tools for Database Design*, North-Holland, Amsterdam, 1983
30. Ceri, S., and Fraternali, P., *Designing Database Applications with Objects and Rules: The IDEA Methodology*, Addison-Wesley, Reading, MA, 1997
31. Ceri, S., Fraternali, P., and Paraboschi, S., Design Principles for Data-Intensive Web Sites, *ACM SIGMOD Record*, 28:1, 1999
32. Ceri, S., Gottlob, G., and Tanaka, L., *Logic Programming and Data Bases*, Springer, Berlin Heidelberg New York, 1989
33. Ceri, S., and Pelagatti, G., *Distributed Databases: Principles and Systems*, McGraw-Hill, New York, 1984
34. Ceri, S., and Widom, J., Deriving Production Rules for Constraint Maintenance, *Proceedings of the International Conference on Very Large Data Bases*, Brisbane, Australia, pp. 566–577, Morgan Kaufmann, San Francisco, 1990
35. Chamberlin, D.D., *A Complete Guide to DB2 Universal Database*, Morgan Kaufmann, San Francisco, CA, 1998
36. Chamberlin, D.D., Astrahan, M.M., Eswaran, P.P., Lorie, R.A., Mehl, J.W., Reisner, P., and Wade, B.W., SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control, *IBM Journal of Research and Development*, 20:6, 97–137, 1976
37. Chamberlin, D.D., and Boyce, R.F., SEQUEL: A Structured English Query Language, *Proceedings of ACM Sigmoid Workshop*, 1, 249–264, 1974

38. Chakravarthi, S., *Active Database Management Systems: Requirements, State-of-the-Art, and an Evaluation*. In: Entity-Relationship Approach: The Core of Conceptual Modeling 1991
39. Chakravarthi, S., Divide and Conquer: A Basis for Augmenting a Conventional Query Optimizer with Multiple Query Processing Capabilities, *Proceedings of the Seventh International Conference on Data Engineering*, 1991
40. Chakravarthi, S., Anwar, E., Maugis, L., and Mishra, D., Design of Sentinel: An Object-oriented DBMS with Event-based Rules, *Information and Software Technology*, 36:9, 1994
41. Chakravarthi, U., Grant, J., and Minker, J., Logic-Based Approach to Semantic Query Optimization, *ACM Transactions on Database Systems*, 15:2, 1990
42. Chen, P.P., The Entity-Relationship Model: Toward a Unified View of Data, *ACM Transactions on Database Systems*, 1:1, 9–36, 1976
43. Cheng, J., and Malaika, S. (Eds.), *Web Gateway Tools: Connecting IBM and Lotus Applications to the Web*, Wiley, New York, 1997
44. Cochran, R., Pirahesh, H., and Mattos, N., Integrating Triggers and Declarative Constraints in SQL Database Systems, *Proceedings of the International Conference on Very Large Data Bases*, Mumbai (Bombay), pp. 567–578, Morgan Kaufmann, San Francisco, 1996
45. Codd, E.F., A Relational Model for Large Shared Data Banks, *Communications of the ACM*, 13:6, 377–387, 1970
46. Codd, E.F., Further Normalization of the Data Base Relational Model. In: Rustin, R. (Ed.), *Database Systems*, pp. 33–64, Prentice Hall, Eaglewood Cliffs, NJ, 1972
47. Codd, E.F., Relational Competencies of Database Sublanguages. In: Rustin, R. (Ed.), *Database Systems*, pp. 65–98, Prentice Hall, Eaglewood Cliffs, NJ, 1972
48. Codd, E.F., Extending the Database Relational Model to Capture More Meaning, *ACM Transactions on Database Systems*, 4:4, 397–434, 1979
49. Codd, E.F., Relational Database: A practical Foundation for Productivity, *Communications of the ACM*, 25:2, 109–117, 1982
50. Codd, E.F., Twelve Rules for On-Line Analytical Processing, *Computerworld*, April 1995
51. Comer, D.E., *Internetworking with TCP/IP, Volume 1: Principles, Protocols, and Architecture*, 3rd edition, Prentice Hall, Eaglewood Cliffs, NJ, 1995
52. Date, C.J., *An Introduction to Database Systems*, 6th edition, Addison-Wesley, Reading, MA, 1995
53. Date, C.J., and Darwen, H., *A Guide to the SQL Standard*, 3rd edition, Addison-Wesley, Reading, MA, 1993
54. Date, C., A Critique of the SQL Database Language, *ACM SIGMOD Record*, 14:3, 1984
55. Date, C., and White, C., *A Guide to DB2*, 3rd edition, Addison-Wesley, Reading, MA, 1989
56. Date, C., and White, C., *A Guide to SQL/DS*, Addison-Wesley, Reading, MA, 1988
57. Davies, C., Recovery Semantics for a DB/DC System, *Proceedings of the ACM National Conference*, 1973
58. Davis, W., *System Analysis and Design*, Addison-Wesley, Reading, MA, 1983
59. Dhavan, C., *Mobile Computing*, McGraw-Hill, New York, 1997

60. Dittrich, K., Object-Oriented Database Systems: The Notion and the Issues. In: Dittrich and Dayal (Eds.), *Proceedings of the International Workshop on Object-Oriented Database Systems*, 1986
61. Dittrich, K., and Dayal, U., (Eds.), *Proceedings of the International Workshop on Object-Oriented Database Systems*, IEEE CS, Pacific Grove, CA, September 1986
62. Dittrich, K., Kotz, A., and Mulle, J., An Event/Trigger Mechanism to Enforce Complex Consistency Constraints in Design Databases. In: *SIGMOD Record*, 15:3, 1986
63. Eisenberg, A., and Melton, J., Standards in Practice, *ACM SIGMOD Record*, 27:3, 53–58, 1998
64. Elmagarmid, A.K. (Ed.), *Database Transaction Models for Advanced Applications*, Morgan Kaufmann, San Mateo, CA, 1992
65. Elmagarmid, A., Leu, Y., Litwin, W., and Rusinkiewicz, M., A Multidatabase Transaction Model for Interbase. In: *International Conference on Very Large Data Bases*, 1990
66. Elmasri, R., James, S., and Kouramajian, V., Automatic Class and Method Generation for Object-Oriented Databases, *Proceedings of the Third International Conference on Database and Object-Oriented Databases (DOOD-93)*, Phoenix, AZ, December 1993
67. Elmasri, R., Kouramajian, V., and Fernando, S., Temporal Database Modeling: An Object-Oriented Approach, *International Conference on Information and Knowledge Management*, November 1993
68. Elmasri, R., Larson, J., and Navathe, S., *Schema Integration Algorithms for Federated Databases and Logical Database Design*, Honeywell CSDD, Technical Report CSC-86-9:8212, January 1986
69. Elmasri, R.A., and Navathe, S.B., *Fundamentals of Database Systems*, 2nd edition, Benjamin-Cummings, Menlo Park, CA, 1994
70. Fairly, R., *Software Engineering Concepts*, McGraw-Hill, New York, 1985
71. Fagin, R., Multivalued Dependencies and a New Normal Form for Relational Databases, *ACM Transactions on Database Systems*, 2:3, 1977
72. Fagin, R., Normal Forms and Relational Database Operators, *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, 1979
73. Fagin, R., A Normal Form for Relational Databases That is Based on Domains and Keys, *ACM Transactions on Database Systems*, 6:3, 1981
74. Fagin, R., Nievergelt, J., Pippenger, N., and Strong, H., Extendible Hashing—A Fast Access Method for Dynamic Files, *ACM Transactions on Database Systems*, 4:3, 1979
75. Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R. (Eds.), *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT, Cambridge, MA, 1996
76. Fleming, C.C., and von Halle, B., *Handbook of Relational Database Design*, Addison-Wesley, Reading, MA, 1989
77. Florescu, D., Levy, A., and Mendelzon, A., Database Techniques for the World-Wide Web: A Survey. *ACM SIGMOD Record*, 27:3, 59–74, 1998
78. Gogolla, M., and Hohenstein, U., Towards a Semantic View of an Extended Entity-Relationship Model, *ACM Transactions on Database Systems*, 16:3, 1991

79. Goldberg, A., and Robson, D., *Smalltalk-80: The Language and Its Implementation*, Addison-Wesley, Reading, MA, 1983
80. Goldfine, A., and Konig, P., *A Technical Overview of the Information Resource Dictionary System (IRDS)*, 2nd edition, NBS IR 88-3700, National Bureau of Standards, 1988
81. Gotlieb, L., Computing Joins of Relations. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1975
82. Graham, I.S., *HTML Sourcebook*, 2nd edition, Wiley, New York, 1996
83. Gray, J., and Anderton, M., Distributed Computer Systems: Four Case Studies, *IEEE Proceedings*, 75:5, 719–726, 1987
84. Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Fellow, F., and Pirahesh, H., Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub Totals, *Data Mining and Knowledge Discovery*, 1:1, 29–53, 1997
85. Gray, J., and Reuter, A., *Transaction Processing Concepts and Techniques*, Morgan Kaufmann, San Mateo, CA, 1994
86. Greenspun, P., *Philip & Alex's Guide to Web Publishing*, Morgan Kaufmann, San Mateo, CA, 1999
87. Hamilton, G., Catteli, R., and Fisher, M., *JDBC Database Access with Java-A Tutorial and Annotated Reference*, Addison Wesley, Reading, MA, 1997
88. Hammer, M., and McLeod, D., Semantic Integrity in a Relational Database System, *Proceedings of 23rd International Conference on Very Large Data Bases* 1975
89. Hammer, M., and McLeod, D., Database Descriptions with SDM: A Semantic Data Model, *ACM Transactions on Database Systems*, 6:3, 1981
90. Hammer, M., and Sarin, S., Efficient Monitoring of Database Assertions. In: *Proceedings of the 1978 ACM SIGMOD International Conference on Management of Data*, 1978
91. Harald Kosh, *Distributed Multimedia Database Technologies supported by MPEG-7 and MPEG-21*, CRC, West Palm Beach, FL, 2003
92. Hull, R., and King, R., Semantic Database Modeling: Survey, Applications and Research Issues, *ACM Computing Surveys*, 19:3, 201–260, 1987
93. Inmon, B., *Building the Data Warehouse*, Wiley, New York, 1996
94. Ioannidis, Y., and Kang, Y., Randomized Algorithms for Optimizing Large Join Queries. In: *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data* 1990
95. Ioannidis, Y., and Kang, Y., Left-Deep vs. Bushy Trees: An Analysis of Strategy Spaces and Its Implications for Query Optimization. In: *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, 1991
96. Ioannidis, Y., and Wong, E., Transforming Nonlinear Recursion to Linear Recursion. In: *International Conference on Expert Database Systems*, 1988
97. Irani, K., Purkayastha, S., and Teorey, T., A Designer for DBMS-Processable Logical Database Structures, *Proceedings of 23rd International Conference on Very Large Data Bases*, 1979
98. Isakowitz, T., Bieber, and M., Vitali, F. (Guest Eds.), Web Information Systems, *Communications of the ACM*, 41:7, 78–117, 1998
99. Ju, P., *Databases on the Web: Designing and Programming for Network Access*, IDG Books Worldwide, Foster City, CA, 1997
100. Kim, W., Object-Oriented Databases: Definition and Research Directions, *IEEE Transactions on Knowledge and Data Engineering*, 2:3, September 1990

101. Kim, W. (Ed.), *Modern Database Systems: the Object Model, Interoperability, and Beyond*, ACM and Addison-Wesley, New York, 1995
102. Kimball, R., *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*, Wiley, New York, 1996
103. Kumar, A., and Segev, A., Cost and Availability Tradeoffs in Replicated Concurrency Control, *ACM Transactions on Database Systems*, 18:1, 1993
104. Kumar, A., and Stonebraker, M., Semantics Based Transaction Management Techniques for Replicated Data, in *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, 1987
105. Kumar, V., and Hsu, M., *Database Recovery*, Kluwer Academic, Dordrecht 1998
106. Kung, H., and Robinson, J., Optimistic Concurrency Control, *ACM Transactions on Database Systems*, 6:2, 1981
107. Lacroix, M., and Pirotte, A., Domain-Oriented Relational Languages, *Proceedings of 23rd International Conference on Very Large Data Bases*, 1977
108. Lacroix, M., and Pirotte, A., ILL: An English Structured Query Language for Relational Data Bases. In: Nijssen (Ed.), *Proceedings of the IFIP TC-2 Working Conference on Modelling in Data Base Management Systems*, 1977
109. Lamport, L., Time, Clocks and the Ordering of Events in a Distributed System, *Communications of the ACM*, 21:7, 558–565, 1978
110. Liu, C., Peek, J., Jones, R., Buus, B., and Nye, A., *Managing Internet Information Services*, O'Reilly & Associates, Sebastopol, CA, 1994
111. Loomis, M.E.S., *Object Databases: the Essentials*, Addison-Wesley, Reading, MA, 1995
112. Lucyk, B., *Advanced Topics in DB2*, Addison-Wesley, Reading, MA, 1993
113. Lum, V.Y., Ghosh, S.P., Schkolnik, M., Taylor, R.W., Jefferson, D., Su. S., Fry, J.P., Teorey, T.J., Yao, B., Rund, D.S., Kahn, B., Navathe, S.B., Smith, D., Aguilar, L., Barr, W.J., and Jones, P.E., 1978 New Orleans Data Base Design Workshop Report, *Proceedings of the International Conference on Very Large Data Bases*, Rio de Janeiro, Brazil, 328–339, 1979
114. Maier, D., Stein, J., Otis, A., and Purdy, A., Development of an Object-Oriented DBMS, *Object-Oriented Programming, Systems, Languages, and Applications*, 1986
115. Maier, D., *The Theory of Relational Databases*, Computer Science Press, Potomac, MD, 1983
116. Mannila, H., and Raiha, K.J., *The Design of Relational Databases*, Addison-Wesley, Reading, MA, 1992
117. McFadden, F.R., and Hoffer, J.A., *Modern Database Management*, 4th edition, Benjamin Cummings, Menlo Park, CA, 1994
118. Melton, J., SQL3 Update, *Proceedings of the IEEE International Conference on Data Engineering 1996*, 566–672, 1996
119. Melton, J., and Simon, A.R., *Understanding the New SQL*, Morgan Kaufmann, San Mateo, CA, 1993
120. Mohan, C., and Narang, I., Algorithms for Creating Indexes for Very Large Tables without Quiescing Updates, in *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, 1992
121. Mohan, C. et al., ARIEL: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging, *ACM Transactions on Database Systems*, 17:1, March, 1992

122. Moss, J., Nested Transactions and Reliable Distributed Computing, *Proceedings of the Symposium on Reliability in Distributed Software and Database Systems*, IEEE CS, July 1982
123. Mylopoulos, J., Bernstein, P., and Wong, H., A Language Facility for Designing Database-Intensive Applications, *ACM Transactions on Database Systems*, 5:2, 1980
124. Ng, P., Further Analysis of the Entity-Relationship Approach to Database Design, *IEEE Transactions on Software Engineering*, 7:1, 1981
125. Nievergelt, J., Binary Search Trees and File Organization, *ACM Computing Surveys*, 6:3, 1974
126. Nijssen, G. (Ed.), *Modelling in Data Base Management Systems*, North-Holland, Amsterdam, 1976
127. Nijssen, G. (Ed.), *Architecture and Models in Data Base Management Systems*, North-Holland, Amsterdam, 1977
128. Obermarck, R., Distributed Deadlock Detection Algorithm, *ACM Transactions on Database Systems*, 7:2, 1982
129. Olle, T., *The CODASYL Approach to Data Base Management*, Wiley, 1978
130. O'Neil, P., *Database Principles, Programming, Performance*, Morgan Kaufmann, Sun Mateo, CA, 1994
131. Oracle, *RDBMS Database Administrator's Guide*, Oracle, 1992
132. Oracle, *Performing Tuning Guide*, Version 7.0, Oracle, 1992
133. Oracle, *Oracle 8 Server Concepts*, Vols. 1 and 2, Release 8.0, Oracle Corporation, 1997
134. Oracle Corporation, *Oracle 8 Server: Concepts Manual*, Redwood City, CA, 1998
135. Oracle Corporation, *Oracle 8 Server: SQL Language Reference Manual*, Redwood City, CA, 1998
136. Ozsu, M.T., and Valduriez, P., *Principles of Distributed Database Systems*, 2nd edition, Prentice Hall, Eaglewood Cliffs, NJ, 1999
137. Papadimitriou, C., The Serializability of Concurrent Database Updates, *Journal of the ACM*, 26:4, 1979
138. Papadimitriou, C., *The Theory of Database Concurrency Control*, Computer Science Press, New York, 1986
139. Papazoglou, M., and Valder, W., *Relational Database Management: A Systems Programming Approach*, Prentice-Hall, Englewood Cliffs, NJ, 1989
140. Paredaens, J., De Bra, P., Gysses, M., and Van Gucht, D., *The Structure of the Relational Database Model*, Springer, Berlin Heidelberg New York, 1989
141. Pazandak, P., and Srivatsava, J., Evaluating Object DBMSs for Multimedia, *IEEE Multimedia*, 4:3, 34-49, 1997
142. Pressman, R.S., *Software Engineering, a Practitioner's Approach*, 3rd edition, McGraw-Hill, New York, 1992
143. Ramakrishnan, R. (Ed.), *Applications of Logic Databases*, Kluwer Academic, Dordrecht, 1995
144. Ramakrishnan, R., *Database Management Systems*, McGraw-Hill, New York, 1997
145. Reisner, P., Human Factors Studies of Database Query Languages: A Survey and Assessment, *ACM Computing Surveys*, 13:1, 1981
146. Rosenfeld, L., and Morville, P., *Information Architecture for the World Wide Web*, O'Reilly and Associates, Sebastopol, CA, 1998

147. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W., *Object Oriented Modelling and Design*, Prentice Hall, Eaglewood Cliffs, NJ, 1991
148. Rustin, R. (Ed.), *Data Base Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1972
149. Rustin, R. (Ed.), *Proceedings of the BJNAV2*, 1974
150. Samaras, G., Britton, K., Citton, A., and Mohan, C., Two-Phase Optimizations in a Commercial Distributed Environment, *Journal of Distributed and Parallel Databases*, 3:4, 325–360, 1995
151. Samet, H., *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1989
152. Selinger, P. et al., Access Path Selection in a Relational Database Management System. In: *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, 1979
153. Senko, M., Specification of Stored Data Structures and Desired Output in DIAM II with FORAL, *Proceedings of 23rd International Conference on Very Large Data Bases*, 1975
154. Senko, M., A Query Maintenance Language for the Data Independent Accessing Model II, *Information Systems*, 5:4, 1980
155. Senn, J.A., *Analysis & Design of Information Systems*, 2nd edition, McGraw-Hill, New York, 1989
156. Shasha, D., *Database Tuning: A Principled Approach*, Morgan Kaufmann, San Mateo, CA, 1994
157. Shasha, D., and Goodman, N., Concurrent Search Structure Algorithms, *ACM Transactions on Database Systems*, 13:1, 1988
158. Sheth, A.P., and Larson, J.A., Federated Database Systems for Managing Distributed, Heterogenous, and Autonomous Databases, *ACM Computing Surveys*, 22:3, 183–236, 1990
159. Siegel, J. (Ed.), *CORBA: Fundamentals and Programming*, Wiley, New York, 1996
160. Silberschatz, A., Korth, H.F., and Sudarshan, S., *Database System Concepts*, McGraw-Hill, New York, 1996
161. Stonebraker, M., *Object-Relational DBMSs – The Next Great Wave*, Morgan Kauffmann, San Mateo, CA, 1994
162. Stonebraker, M. (Ed.), *Readings in Database Systems*, 2nd edition, Morgan Kauffmann, San Mateo, CA, 1994
163. Stonebraker, M., Rowe, L.A., Lindsay, B.G., Gray, J., Carey, M.J., Brodie, M.L., Bernstein, P.A., and Beech, D., Third-Generation Database System Manifesto, *ACM SIGMOD Record*, 19:3, 31–44, 1990
164. Smith, J.M., and Smith, D.C.P., Database Abstractions: Aggregation and Generalization, *Proceedings of the 1977 ACM Transactions on Database Systems*, 2:1, 105–133, 1977
165. Subrahmanian, V.S., *Principles of Multimedia Database Systems*, Morgan Kaufmann, San Mateo, CA, 1998
166. Tansel, A., et al., (Eds.) *Temporal Databases: Theory, Design, and Implementation*, Benjamin Cummings, Menlo Park, CA, 1993
167. Teorey, T., *Database Modeling and Design: The Fundamental Principles*, 2nd edition, Morgan Kauffmann, Los Altos, CA, 1994
168. Teorey, T., Yang, D., and Fry, J., A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model, *ACM Computing Surveys*, 18:2, 1986

169. Teorey, T.J., *Database Modeling and Design: the E-R Approach*, Morgan Kaufmann, San Mateo, CA, 1990
170. Teorey, T.J., and Fry, J.P., *Design of Database Structures*, Prentice Hall, Eaglewood Cliffs, NJ, 1982
171. Teorey, T.J., Yang, D., and Fry, J.P., A Logical Design Methodology for Relational Databases Using the Extended Entity-Relational Approach, *ACM Computing Surveys*, 18:2, 201–260, 1986
172. Tsichritzis, D., and Lochovsky, F.H., *Data Models*, Prentice Hall, Eaglewood Cliffs, NJ, 1982
173. Tsou, D.M., and Fischer, P.C., Decomposition of a Relation Scheme into Boyce Codd Normal Form, *SIGACT News*, 14:3, 23–29, 1982
174. Ullman, J., Implementation of Logical Query Languages for Databases, *ACM Transactions on Database Systems*, 10:3, 1985
175. Ullman, J.D., *Principles of Database and Knowledge Base Systems*, Vol. 1, Computer Science Press, Potomac, MD, 1988
176. Ullman, J.D., *Principles of Database and Knowledge Base Systems*, Vol. 2, Computer Science Press, Potomac, MD, 1989
177. Ullman, J.D., and Widom, J., *A First Course in Database Systems*, Prentice Hall, Upper Saddle River, NJ, 1997
178. Valduriez, P., and Gardarin, G., *Analysis and Comparison of Relational Database Systems*, Addison-Wesley, Reading, MA, 1989
179. Vassiliou, Y., Functional Dependencies and Incomplete Information, *Proceedings of 23rd International Conference on Very Large Data Bases*, 1980
180. Verheijen, G., and VanBekum, J., NIAM: An Information Analysis Method. In: Olle et al. (Eds.), *Proceedings of the CRIS Conference*, 1982
181. Verhofstadt, J., Recovery Techniques for Database Systems, *ACM Computing Surveys*, 10:2, 1978
182. Vielle, L., Recursive Axioms in Deductive Databases: The Query-Subquery Approach. In: *Proceedings International Conference on Expert Database Systems*, 1986
183. Vossen, G., *Data Models, Database Languages, and Database Management Systems*, Addison-Wesley, Reading, MA, 1990
184. Wang, Y., and Madnick, S., The Inter-Database Instance Identity Problem in Integrating Autonomous Systems. In: *Proceedings of the Fifth IEEE International Conference on Data Engineering*, 1989
185. Wang, Y., and Rowe, L., Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture. In: *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, 1991
186. Wallace, D., William Allan Award Address: Mitochondrial DNA Variation in Human Evolution, Degenerative Disease, and Aging, *American Journal of Human Genetics*, 1994
187. Weddell, G., Reasoning About Functional Dependencies Generalized for Semantic Data Models, *ACM Transactions on Database Systems*, 17:1, 1992
188. Weikum, G., Principles and Realization Strategies of Multilevel Transaction Management, *ACM Transactions on Database Systems*, 16:1, 1991
189. Widom, J., Research Problems in Data Warehousing, *Proceedings of the 4th International Conference on Information and Knowledge Management*, November 1995
190. Widom, J., and Ceri, S., *Active Database Systems*, Morgan Kauffmann, San Mateo, CA, 1996

191. Wiederhold, G., *Database Design*, McGraw-Hill, New York, 1983
192. Wiorkowski, G., and Kull, D., *DB2-Design and Development Guide*, 3rd edition, Addison-Wesley, Reading, MA, 1992
193. Wong, E., and Youssefi, K., Decomposition-A Strategy for Query Processing, *ACM Transactions on Database Systems*, 1:3, 1976
194. Yannakakis, Y., Serializability by Locking, *Journal of the ACM*, 31:2, 1984
195. Yao, S., Optimization of Query Evaluation Algorithms, *ACM Transactions on Database Systems*, 4:2, 1979
196. Yao, S. (Ed.), *Principles of Database Design, Vol. 1: Logical Organizations*, Prentice-Hall, Englewood Cliffs, NJ, 1985
197. Youssefi, K., and Wong, E., Query Processing in a Relational Database Management System, *Proceedings of 23rd International Conference on Very Large Data Bases*, 1979
198. Zaniolo, C., *Analysis and Design of Relational Schemata for Database Systems*, Ph.D. dissertation, University of California, LA, 1976
199. Zaniolo, C., *Design and Implementation of a Logic Based Language for Data Intensive Applications*, MCC Technical Report#ACA-ST-199-88, June, 1988
200. Zaniolo, C., Database Relations with Null Values, *Journal of Computer and System Science*, 28:1, 142-166, 1984
201. Zaniolo, C., Ceri, S., Faloutsos, C., Snodgrass, R.T., Subrahmanian, V.S., and Zicari, R., *Introduction to Advanced Database Systems*, Morgan Kaufmann, San Mateo, CA, 1997
202. Zaniolo, C. et al., *Advanced Database Systems*, Morgan Kaufmann, San Mateo, CA, 1997
203. Zloof, M., Query by Example, *Proceedings of the National Computer Conference, American Federation of Information Processing Societies*, 44, 1975
204. Zobel, J., Moffat, A., and Sacks-Davis, R., An Efficient Indexing Technique for Full-Text Database Systems, *Proceedings of 23rd International Conference on Very Large Data Bases*, 1992