DATABASE MANAGEMENT SYSTEM

MBA IT-309

DIRECTORATE OF DISTANCE EDUCATION

SWAMI VIVEKANAND
SUBHARTI UNIVERSITY
Meerut (National Capital Region Delhi)

Contents

SYLLABUS

DATABASE MANAGEMENT SYSTEM

Course Code: MBA IT-309		
Course Credit: 3	**Lecture: 2**	**Tutorial-I**
Course Type:	Skill Enhancement Course	
Lectures delivered:	**40 L+10T**	

End Semester Examination System

Maximum Marks Allotted	Minimum Pass Marks	Time Allowed
70	28	3 Hours

Continuous Comprehensive Assessment (CCA) Pattern

Tests	Assignment/ Tutorial/ Presentation/class test	Attendance	Total
15	5	10	30

Course Objective: The aim of the course is to introduce students to:

- Database, organisation of database, advantages of DBMS
- Rational database design
- Structured Query Language
- Database Utilities

UNIT	Course Content	Hours
I	**Introduction to Database:** Organisation of Database; Components of Database Management Systems; Data Models; Entity-Relationsip Model; Network Data Model; Hierarchy Data Model: Relational Data Model; Semantic Data Model; Advantagesof DBMS.	8
II	**Rational Database Design:** INtegrity Constraints; Functional Dependencies; Normalisation; Physical Database Design; Decomposition of Relation Schemes; INtroduction to Data Mining and Data Warehousing; Knowledge of Extraction through Data Mining.	8
III	**Structured Query Language, Oracle:** Creating Tables; Applying Column constraints; Inserting Rows; Views, Snapshots, Indexes and Sequences.	8
IV	**Database Utilities:** Database Utilities; Security. Object/Basic Database Administration/ Remote Database Access	6

Unit 1

An Introduction to Database Management System

Notes

Structure

1.0 LEARNING OBJECTIVES

After reading this chapter students will be able to:

- Describe data, field, record and database management system
- Compare database and file oriented approach

Notes

- Describe various features of database
- Describe the need and function of the database
- Describe the concept of different keys
- Describe functions of database management system
- Describe the role of DBA, End users and application programmes
- Describe data independence
- Describe database schema and instance
- Describe DDL and DML commands

1.1 INTRODUCTION

Data and its storage may be considered to be the heart of any information system. Data has to be up-to-date, accurate, accessible in the required form and available to one or perhaps many users at the same time. For data to be a value it must be presented in a form that supports the various operational, financial, managerial, decision-making, administrative andclerical activities withinanorganization. To meet these objectives data needs to be stored efficiently that means we wish to avoid lengthy access times and with minimal duplication that means we wish to aovid lengthy update times and the possibility of inconsistency and inaccuracy. For the data stored by a given organization to have any value at all its integrity (consistency and accuracy) must always be assured.

In this unit we consider what is database and DBMS, different needs and functions of database and concepts related to relational database management system.

1.2 INTRODUCTION TO DATABASE MANAGEMENT SYSTEM

Before understanding what DBMS is, let us first understand what a database is.

In layperson's terms, the database is a big container where data is stored in a structured format. We cannot store semi-structured or unstructured data in a database. A database is an organized collection of data that can be modified, retrieved, or updated. Data, DBMS, and applications associated with them together form the database concept. The data, stored in the database, is in the row and column format, which is called a table. Every website, which needs us to sign up, uses a database. There is no internet without databases.

For instance, a college will have to keep the information about its students, including roll number, name, age, blood group, etc. The college will also need to keep the details of the professors and infrastructure. The details, which the college has, can be stored in a database named College, or if it is just the student details, then it can be named Students. All such details should be in a structured format, such as tables, in a hierarchy.

A database management system (DBMS) is a software system that integrates data in storage and provides easy access to them. The data themselves are placed on disk in a database, which can be thought of as an integrated collection of related files.Although not all databases are organized identically, many of them are composed of files, records and fields.

For instance, the product file contains five records one each for skis, boots, poles, bindings, and wax. Finally each record consists of distinct types of data called fileds. The product file stores four fileds for each record product name, product number, supplier, and price.

Look at the following points:

- Database management softaware enables queries and reports to be prepared by extracting infor- mation from one file at a time, and, as we will shortly see, from several interrelated files concurrently.
- Database management system (DBMS) is a software product designed to integrate data and provide easy access to them.
- Database is an integrated collection of related data files and database management system is a computer programe for database management that links data in related files through common fileds.
- Database management system stores data in relevant form and give easy access to them. The data placed on disk themselves.
- Database is a collection of inter-related data which helps in efficient retrieval, insertion and deletion of data from database and organizes the data in the form of tables, views, schemas, reports etc. For Example, university database organizes the data about students, faculty, and admin staff etc. which helps in efficient retrieval, insertion and deletion of data from it.

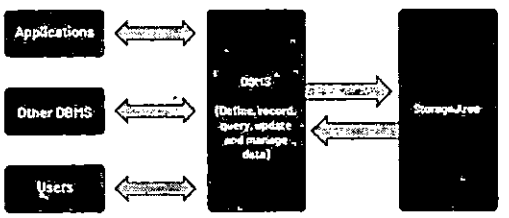
DDL is short name of Data Definition Language, which deals with database schemas and descriptions, of how the data should reside in the database.

Database Management System (DBMS) is a software application that is used to create, access, maintain, and manage databases. We can install it like any other application and use SQL for all the operations that we intend to perform on databases.

DBMS accepts the incoming data either from an application or from a user who is manually entering it. It does not matter if it is a large or small DBMS, with its help, we can store and retrieve data and make changes whenever we want to.

Some commands are predefined in DBMS, and these commands can be used to manipulate the database. These commands are also the interface between the database and end-users to establish communication.

Database Management System



Database Management System Examples

Few examples of database are:

Notes

Oracle – Oracle is a Relational Database Management System. It can be stored on-site or in the cloud. It uses enterprise-scale technology to offer a wide range of features to the users.

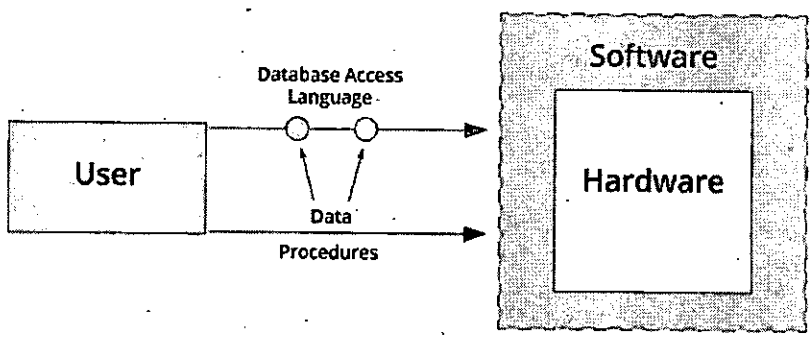
MySQL – Used by platforms like Youtube, Twitter, and Facebook, MySQL is a Relational Database Management System. It is often integrated with open-source Content Management Systems (CMS).

SQL Server – SQL Server is a Relational Database Management System developed by Microsoft. It was based on SQL, a query language that helps users in data query and database management.

1.3 COMPONENTS OF DBMS

There are five major components of DBMS:

1. Hardware – Hardware includes the actual physical computer system that is used to access the database. Mainframes to microcomputers are utilized in DBMS. Oftentimes large storage devices are used to store the huge amount of data.
2. Software – The DBMS is the Software here. It helps connect the physical database to the end users. Whenever a user wants to access the database, access has to be granted by the software.
3. Procedures – All the instructions that are given to access the DBMS, fall under Procedures. From installing DBMS, to generating reports, all of these instructions are a part of the procedures involved in DBMS.
4. Data – Data is the information that is managed by DBMS. DBMS helps in managing data and easing access to useful information. Meta data, the information stored in DBMS for better comprehension of the data, is also stored in DBMS itself.
5. Database Access Language – The language that is used to write commands on the DBMS to access, utilize, manage, update and delete data, is called the Database Access Language. Commands are written using the Database Access Language and submitted to the DBMS so they can be executed by the DBMS.

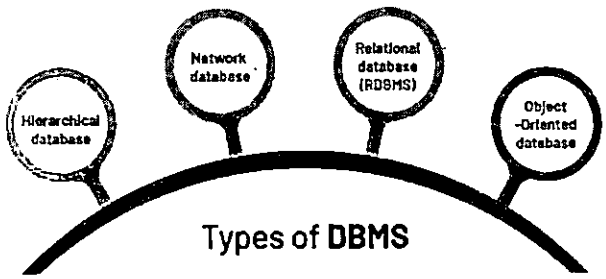


1.4 TYPES OF DBMS

A database management system is a software package for creating and managing databases. Many different types of database systems exist based on how they manage the database structure.

There are broadly four categories or types of DBMS:

- Hierarchical databases
- Network databases
- Relational databases (RDBMS)
- Object-oriented databases



Hierarchical Databases

This type of DBMS showcases a parent-child type of relationship. This relationship forms a tree-like structure where the nodes (leaves) of the tree represent records and the fields are represented by branches.

A hierarchical database is a design that uses a one-to-many relationship for data elements. Hierarchical database models use a tree structure that links a number of disparate elements to one "owner," or "parent," primary record.

The idea behind hierarchical database models is useful for a certain type of data storage, but it is not extremely versatile. Its limitations mean that it is confined to some very specific uses. For example, where each individual person in a company may report to a given department, the department can be used as a parent record and the individual employees will represent secondary records, each of which links back to that one parent record in a hierarchical structure.

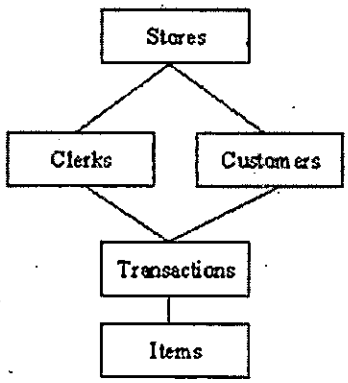
Hierarchical databases were popular in early database design, in the era of mainframe computers. While some IBM and Microsoft models are still in use, many other types of business databases use more flexible models to accommodate more sophisticated types of data management.

Hierarchical models make the most sense where the primary focus of information gathering is on a concrete hierarchy such as a list of business departments, assets or people that will all be associated with specific higher-level primary data elements.

Notes

Network Databases

This style of DBMS embraces several partnerships where it is possible to connect multiple user records at the same time in parallel.



A network database is a type of database model wherein multiple member records or files can be linked to multiple owner files and vice versa. The model can be viewed as an upside-down tree where each member information is the branch linked to the owner, which is the bottom of the tree.

Essentially, relationships are in a net-like form where a single element can point to multiple data elements and can itself be pointed to by multiple data elements.

The network database model allows each record to have multiple parent and multiple child records, which, when visualized, form a web-like structure of networked records. In contrast, a hierarchical model data member can only have a single parent record but can have many child records.

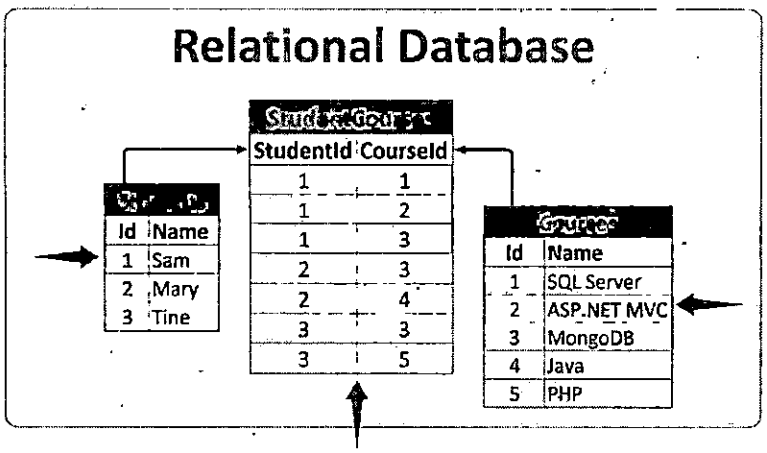
This property of having multiple links applies in two ways: the schema and the database itself can be seen as a generalized graph of record types that are connected by relationship types. The main advantage of a network database is that it allows for a more natural modeling of relationships between records or entities, as opposed to the hierarchical model. However, the relational database model has started to win over both the network and the hierarchical models because its added flexibility and productivity has become more evident as hardware technology has become faster.

Relational Databases (RDBMS)

This type of DBMS helps users to locate and manipulate data that has connections with another piece of data in the database. It uses tables for storing the data in a row-and-column format.

A relational database is a digital database based on the relational model of data, as proposed by E. F. Codd in 1970.

A system used to maintain relational databases is a relational database management system (RDBMS). Many relational database systems have an option of using the SQL (Structured Query Language) for querying and maintaining the database.



History

The term "relational database" was invented by E. F. Codd at IBM in 1970. Codd introduced the term in his research paper "A Relational Model of Data for Large Shared Data Banks". In this paper and later papers, he defined what he meant by "relational".

One well-known definition of what constitutes a relational database system is composed of Codd's 12 rules.

However, no commercial implementations of the relational model conform to all of Codd's rules, so the term has gradually come to describe a broader class of database systems, which at a minimum:

- Present the data to the user as relations (a presentation in tabular form, i.e. as a collection of tables with each table consisting of a set of rows and columns);
- Provide relational operators to manipulate the data in tabular form.

In 1974, IBM began developing System R, a research project to develop a prototype RDBMS. The first system sold as an RDBMS was Multics Relational Data Store (June 1976). Oracle was released in 1979 by Relational Software, now Oracle Corporation. Ingres and IBM BS12 followed.

Other examples of an RDBMS include IBM Db2, SAP Sybase ASE, and Informix. In 1984, the first RDBMS for Macintosh began being developed, code-named Silver Surfer, and was released in 1987 as 4th Dimension and known today as 4D.

The first systems that were relatively faithful implementations of the relational model were from:

- University of Michigan – Micro DBMS (1969)[citation needed]
- Massachusetts Institute of Technology (1971)[9]
- IBM UK Scientific Centre at Peterlee – IS1 (1970–72) and its successor, PRTV (1973–79)
- The most common definition of an RDBMS is a product that presents a view of data as

Notes

a collection of rows and columns, even if it is not based strictly upon relational theory. By this definition, RDBMS products typically implement some but not all of Codd's 12 rules.

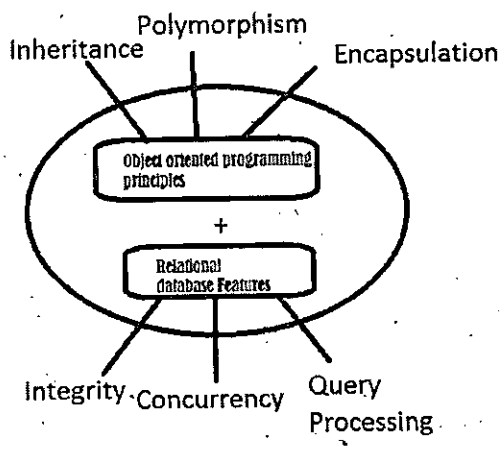
- A second school of thought argues that if a database does not implement all of Codd's rules (or the current understanding on the relational model, as expressed by Christopher J. Date, Hugh Darwen and others), it is not relational. This view, shared by many theorists and other strict adherents to Codd's principles, would disqualify most DBMSs as not relational. For clarification, they often refer to some RDBMSs as truly-relational database management systems (TRDBMS), naming others pseudo-relational database management systems (PRDBMS).
- As of 2009, most commercial relational DBMSs employ SQL as their query language. Alternative query languages have been proposed and implemented, notably the pre-1996 implementation of Ingres QUEL

Object-oriented Databases

An object-oriented database (OOD) is a database system that can work with complex data objects – that is, objects that mirror those used in object-oriented programming languages. In object-oriented programming, everything is an object, and many objects are quite complex, having different properties and method.

In this type of DBMS, data is stored in individual components called objects, where each object is a piece of data with some instructions for the tasks that should be performed on that data.

Now, we have a fair idea about what DBMS is and the kind of versatility it provides to developers. DBMS software can be used differently as there are various DBMS architectures. Choosing a DBMS completely depends on end-user goals. Choosing the right DBMS for an application will provide an edge that can make the application work seamlessly.



Application of DBMS

Sector	Application
Universities	Student information, courses, grades, etc.

Sales	Customer information, sales, etc.
Finance	Stock information, sales, bonds, etc.
Banking	Customer information, account, activities, deposits, loans, etc.
Manufacturing	Production information, suppliers, inventories, etc.
Airlines	Customer information, schedules, reservations, etc.
HR Management	Employee information, payroll, deduction, paychecks, etc.
Telecommunication	Call records, bills, usage, etc.

When Not to Use a DBMS?

Despite the earlier mentioned disadvantages, a DBMS system is still useful. However, the initial investment required to build a DBMS infrastructure is quite high. Therefore, it is not ideal to use DBMS for small projects where an organization cannot afford the hardware and training costs. But, this is only when we are setting up our own database servers. Cloud databases are cheap and come in handy, and anybody can use them.

1.5 PURPOSE OF DATABASE MANAGEMENT SYSTEM

The DBMS (Database Management System) is preferred ever the conventional file processing system due to the following advantages:

1. Controlling Data Redundancy - In the conventional file processing system, every user group maintains its own files for handling its data files. This may lead to:
   - Duplication of same data in different files.
   - Wastage of storage space, since duplicated data is stored.
   - Errors may be generated due to updation of the same data in different files.
   - Time in entering data again and again is wasted.
   - Computer Resources are needlessly used.
   - It is very difficult to combine information.
2. Elimination of Inconsistency - In the file processing system information is duplicated throughout the system. So changes made in one file may be necessary be carried over to another file. This may lead to inconsistent data. So we need to remove this duplication of data in multiple file to eliminate inconsistency.

   For example: - Let us consider an example of student's result system. Suppose that in STU- DENTfile it is indicated that Roll no= 10 has opted for 'Computer'course but in RESULT file it is indicated that 'Roll No. =l 0' has opted for 'Accounts' course. Thus, in this case the two entries for z particular student don't agree with each other. Thus, databaseis said to be in an inconsistent state. Sc to eliminate this conflicting information we need to centralize the database. On centralizing the database the duplication will be controlled and hence inconsistency will be removed.

Notes

Data inconsistency are often encountered in every day life Consider an another example, w have all come across situations when a new address is communicated to an organization that we deal it (Eg - Telecom, Gas Company, Bank).

We find that some of the communications from that organization are received at a new address while other continued tobe mailed to the old address. So combining all thedata in database would involve reduction in redundancy as well as inconsistency so it is likely to reduce the costs for collection storage and updating of Data.

Let us again consider the example of Result system. Suppose that a student having Roll no -201 changeshiscourse from 'Computer' to 'Arts'. The change ismade in the SUBJECTfile but not in RESULT'S file. This may lead to inconsistency of the data. So we need to centralize the database so that changes once made are reflected to all the tables where a particulars field is stored. Thus the update is brought automatically and is known as propagating updates.

3. **Better service to the users** - A DBMS is often used to provide better services to the users. In conventionalsystem, availability of informationisoftenpoor, since it normallydifficult toobtaininformation that the existing systems were not designed for. Once several conventional systems are combined to form one centralized database, the availability of information and its updateness is likely to improve since the data can now be shared and DBMS makes it easy to respond to anticipated information requests.

   Centralizing the data in the database also means that user can obtain new and combined information easily that would have been impossible to obtain otherwise. Also use of DBMS should allow users that don't know programming to interact with the data more easily, unlike file processing system where the programmer may need to write new programs to meet every new demand.

4. **Flexibility of the System is Improved** - Since changes are often necessary to the contents of the data stored in any system, these changes are made more easily in a centralized database than in a conventional system.Applications programs need not to be changed on changing the data in the database.

5. **Integrity can be improved** - Since data of the organization using database approach is centralized and would be used by a number of users at a time. It is essential to enforce integrity-constraints.

   In the conventional systems because the data is duplicated in multiple files so updating or changes may sometimes lead to entry of incorrect data in some files where it exists.

   For example : - The example of result system that we have already discussed. Since multiple files are to maintained, as sometimes you may enter a value forcourse which may not exist. Suppose course can have values (Computer, Accounts, Economics, and Arts) but we enter a value 'Hindi' for it, so this may lead to an inconsistent data, so lack of Integrity.

   Even if we centralized the database it may still contain incorrect data. For example: -

- Salary of full time employ may be entered as Rs. 500 rather than Rs. 5000.
- A student may be shown to have borrowed books but has no enrollment.
- A list of employee numbers for a given department may include a number of nonexistent employees.

These problems can be avoided by defining the validation procedures whenever any update operation is attempted.

6. **Standards can be enforced** - Since all access to the database must be through DBMS, so standards are easier to enforce. Standards may relate to the naming of data, format of data, structureof the data etc. Standardizing stored data formats is usually desirable for the purpose of data interchange or migration between systems.

7. **Security can be improved** - In conventional systems, applications are developed in an adhoc/ temporary manner. Often different system of an organization would access different components of the operational data, in such an environment enforcingsecurity can be quiet difficult. Setting up of a database makes it easier to enforce security restrictions since data is now centralized. It is easier to control who has access to what parts of the database. Different checks can be established for each type of access (retrieve, modify, delete etc.) to each piece of information in the database.

   Consider an Example of banking in which the employee at different levels may be given access to different types of data in the database. A clerk may be given the authority to know only the names of all the customers who have a loan in bank but not the details of each loan the customer may have. It can be accomplished by giving the privileges to each employee.

8. **Organization's requirement can be identified** - All organizations have sections and departments and each of these units often consider the work of their unit as the most important and therefore consider their need as the most important. Once a database has been setup with centralized control, it will be necessary to identify organisation's requirement and to balance the needs of the competiting units.

9. **Overall cost of developing and maintaining systems is lower** - It is much easier to respond to unanticipated requests when data is centralized ina database than when it is stored in a conventional file system. Although the initial cost of setting up of a database can be large, one normal expects the overall cost of setting up of a database, developing and maintaining application programs to be far lower thanfor similar service using conventional systems, Since the productivity of programmers can be higher in using non-procedural languages that have been developed with DBMS than using procedural languages.

10. **Data Model must be developed** - Perhaps the most important advantage of setting up of database system is the requirement that an overalldata model for an organization be build. In conventional systems, it is more likely that files will be designed as per need of particular applications demand. The overall view isoftennotconsidered. Buildinganoverallviewofanorganization'sdataisusualcosteffectivein thelongterms.

Notes

11. **Provides backup and Recovery** - Centralizing a database provides the schemes such as recovery and backups from the failures including disk crash, power failures, software errors which may help the database to recover from the inconsistent state to the state that existed prior to the occurrence of the failure, though methods are very complex.

1.6 ELEMENTS OF DATABASE MANAGEMENT SYSTEMS

The essential elements that are found with just about every example of DBMS currently on the market are

- Schema objects
- Indexes
- Tables
- Fields and columns
- Records and rows
- Keys
- Relationships
- Data types

1.7 FUNCTIONS OF DBMS

The functions of a DBMS include concurrency, security, backup and recovery, integrity and data descriptions. Database management systems provide a number of key benefits but can be costly and time-consuming to implement

1. DBMS free the programmers from the need to worry about the organization and location of the data i.e. it shields the users from complex hardware level details.
2. DBMS can organize process and present data elements from the database. This capability enables decision makers to search and query database contents in order to extract answers that are not available in regular Reports.
3. Programming is speeded up because programmer can concentrate on logic of the application.
4. It includes special user friendly query languages which are easy to understand by non programming users of the system.

The various common examples of DBMS are Oracle, Access, SQL Server, Sybase, FoxPro, Dbase etc.

The Service Provided by the DBMS Includes

1. Authorization services like log on to the DBMS, start the database, stop the Database etc.

2. Transaction supports like Recovery, Rollback etc,
3. Import and Export of Data.
4. Maintainingdata dictionary
5. User's Monitoring

DBA, Database Designers, End Users & Application Programmers

Database Administrator (DBA)

The DBA is a person or a group of persons who is responsible for the management of the data- base. The DBA is responsible for authorizing access to the database by grant and revoke permissions to the users, for coordinating and monitoring its use, managing backups and repairing damage due to hard- ware and/or software failures and for acquiring hardware and software resources as needed. In case of small organization the role of DBA is performed by a single person and in case of large organizations there is a group of DBA's who share responsibilities.

Database Designers

They are responsible for identifying the data to be stored in the database and for choosing appropriate structure to represent and store the data. It is the responsibility of database designers to communi- cate with all prospective of the database users in order to understand their requirements so that they can create a design that meets their requirements.

End Users

End Users are the people who interact with the database through applications or utilities. The various categories of end users are:

1. **Casual End Users:** These Users occasionally access the database but may need different information each time. They use sophisticated database Query language to specify their requests. For example: High level Managers who access the data weekly or biweekly.
2. **NativeEnd Users:** These users frequently query and update the database using standard types of Queries. The operations that can be performed by this class of users are very limited and effect precise portion of the database.

   For example: Reservation clerks for airlines/hotels check availability for given request and make reservations. Also, persons using Automated Teller Machines (ATM's) fall under this category as he has access to limited portion of the database.
3. **Standalone end Users/On-line End Users :** Those end Users who interact with the database directly via on-line terminal or indirectly through Menu or graphics based Interfaces.

   For example: User of a text package, library management software that store variety of library data such as issue and return of books for fine purposes.

Notes

4. **Application Programmers:** Application Programmers are responsible for writing application programs that use the database. These programs could be written in General Purpose Programming languages such as Visual Basic, Devel- oper, C, FORTRAN, COBOL etc. to manipulate the database. These application programs operate on the data to perform various operations such as retaining information, creating new information, deleting or changingexistinginformation.

   - DBMS engine accepts logical requests from various other DBMS subsystems, converts them into physical equivalents, and actually accesses the database and data dictionary as they exist on a storage device.
   - Data definition subsystem helps the user create and maintain the data dictionary and define the structure of the files in a database.
   - Data manipulation subsystem helps the user to add, change, and delete information in a database and queryit for valuable information. Software tools within the data manipulation subsystem are mostoften the primary interface between user and the information contained in a database. It allows the user to specify its logical information requirements.
   - Application generation subsystem contains facilities to help users develop transaction-intensive applications. It usually requires that the user perform a detailed series of tasks to process a transaction. It facilitates easy-to-use data entry screens, programming languages, and interfaces.
   - Data administration subsystem helps users manage the overall database environment by providing facilities for backup and recovery, security management, query optimization, concurrency control, and changemanagement.

1.8 DBMS-ARCHITECTURE AND DATA INDEPENDENCE

Database management systems are complex softwares which were often developed and optimised over years. From the view of the user, however, most of them have a quite similar basic architecture.

Three-Schemes Architecture

The three-schema architecture divides the database into three-level used to create a separation between the physical database and the user application. In simple terms, this architecture hides the details of physical storage from the user.

The database administrator (DBA) responsible is to change the structure of database storage without affecting the user's view. It deals with the data, the relationship between them and the different access methods implemented on the database. The logical design of database is called a schema

This architecture contains three layers of database management system, which are as follows –

- External level
- Conceptual level
- Internal level

External Scheme

This is the highest level of database abstraction. It includes a number of external schemas or user views. This level provides different views of the same database for a specific user or a group of users. An external view provides a powerful and flexible security mechanism by hiding the parts of the database from a particular user.

Internal Scheme:

This is the lowest level of database abstraction. It describes how the data is stored in the database and provides the methods to access data from the database. It allows viewing the physical representation of the database on the computer system.

The interface between the conceptual and internal schema identifies how an element in the conceptual schema is stored and how it may be accessed. It is one which is closest to physical storage. The internal schema not only defines different stored record types, but also specifies what indices exist, how stored fields are represented.

Therefore, the internal scheme describes the data from a view very close to the computer or system in general. It completes the logical schemewith data technical aspects like storage methods or help functions formore efficiency.

The right hand side of the representation above is also called the three-schemes architecture: internal, logical and external scheme.

While the internal scheme describes the physical grouping of the data and the use of the storage space, the logical scheme (derived from the conceptual scheme) describes the basic construction of the data structure. The external scheme of a specific application, generally, only highlights that part of the logical scheme which is relevant for its application. Therefore, a database has exactly one internal and one logical scheme but may have several external schemes for several applications using this database.

The three level schema architecture in DBMS is given below –

The aim of the three-schemes architecture is the separation of the user applications from the physical database, the stored data. Physically the data is only existent on the internal level while other forms of representation are calculated or derived respectively if needed. The DBMS has the task to realise this representation between each of these levels.

Data Independence

With knowledge about the three-schemes architecture the term data independence can be explained as followed: Each higher level of the data architecture is immune to changes of the next lower level of the architecture.

Notes

Physical Independence

Therefore, the logical scheme may stay unchanged even though the storage space or type of some data is changed for reasons of optimisation or reorganisation.

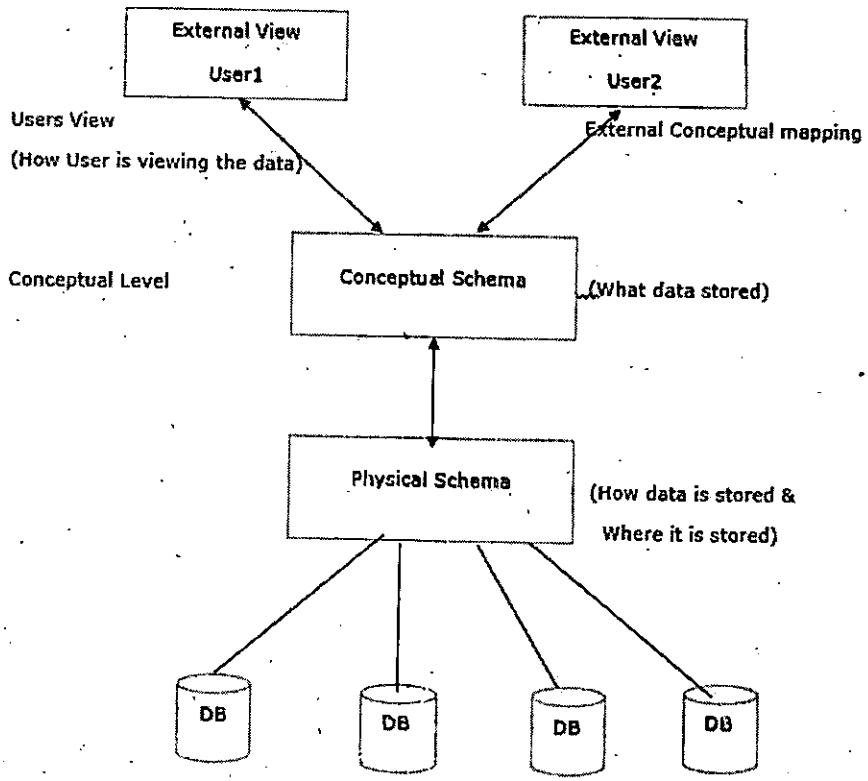


Figure 1 : Three-Schemes Architecture

Logical Independence

Also the external scheme may stay unchanged for most changes of the logical scheme. This is especially desirable as in this case the application software does not need to be modified or newly translated.

Database Schema

Definition : Overall design of data base. Schema contains 'No of records + Type of data + No of attributes'

1. External level or Sub schema
2. logical schema
3. Physical schema

Aschema is a group of related objects in a database. Within a schema, objects that are related have relationships to one another. It is overall design of the database. There is one owner of a schema, who has access to manipulate the structure of any object in the schema. Aschema does not represent a person, although the schema is associated with a user account that resides in the database.

The three models associated with a schema are as follows:

- The conceptual model, also called the logical model, is the basic database model, which deals with organizational structures that are used to define database structures such as tables and constraints.
- The internal model, also called the physical model, deals with the physical storage of the database, as well as access to the data, such as through data storage in tables and the use of indexes to expedite data access. The internal model separates the physical requirements of the hardware and the operating system from the data model.
- The external model, or application interface, deals with methods through which users may access the schema, such as through the use of a data input form. The external model allows relationships to be created between the user application and the data model. Figure 1 depicts a schema in a relational database.

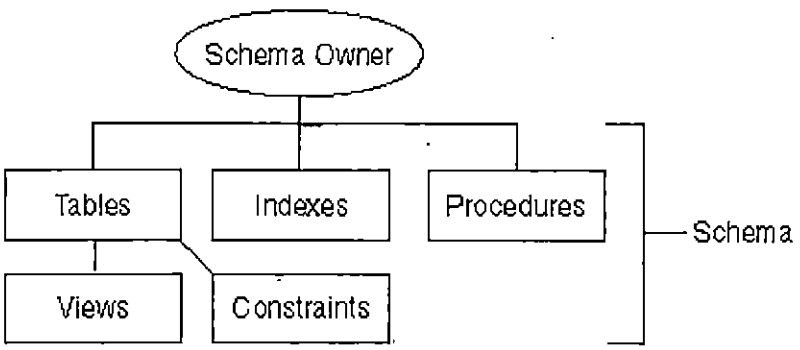


Figure 2 : Collection of objects that comprise a database schema.

Table

A table is the primary unit of physical storage for data in a database. The table is the most funda- mental element found in a database schema. Columnsand rows are associated with tables When a user accesses the database, a table is usually referenced for the desired data. Multiple tables might comprise a database, therefore a relationship might exist between tables. Because tables store data, a table requires physical storage on the host computer for the database.

Notes

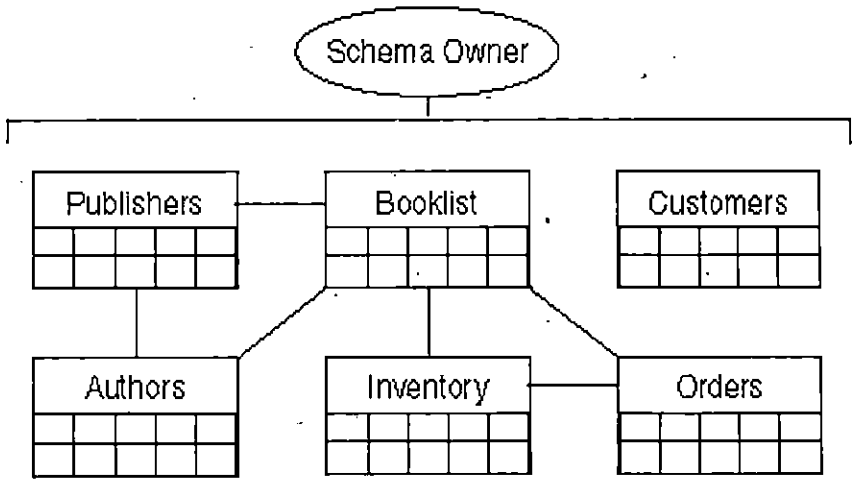


Figure 3 : Database Tables and their Relationships

Columns

A column, or field, is a specific category of information that exists in a table.Acolumn is to a table what an attribute is to an entity. In other words, when a business model is converted into a databasemodel, entities become tables and attributes become columns.Acolumn represents one related part of a table and is the smallest logical structure of storage in a database. Each column in a table is assigned a datatype. The assigned data type determines what type of values that can populate a column. When visualizing a table, a column is a vertical structure in the table that contains values for every row of data associated with a particular column. All of the data in a table associated with a field is called a column

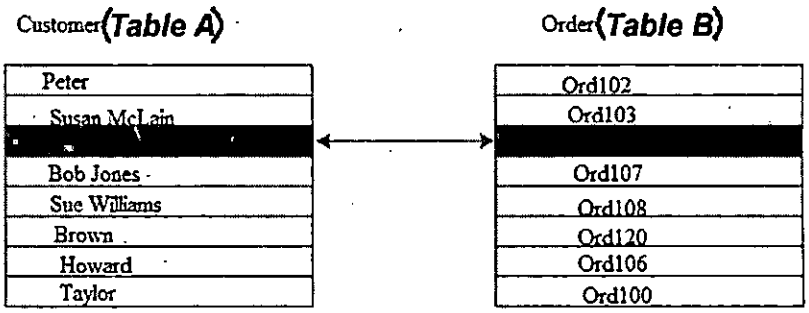


Figure 4 : Database Tables and their Relationships

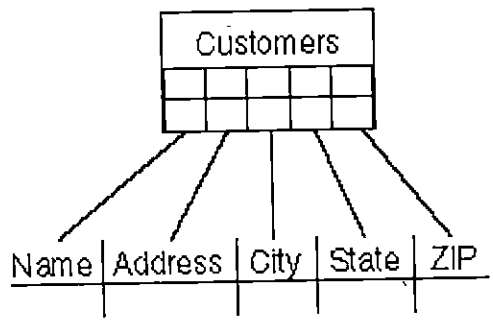


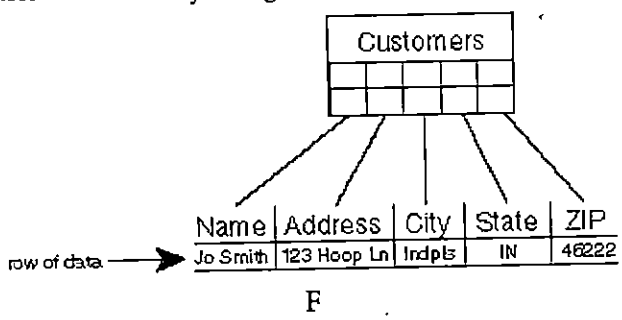
Figure 5 : Columns in a Database Table

Rows

In relational databases, a row is a data record within a table. Each row, which represents a complete record of specific item data, holds different data within the same structure. A row is occasionally referred to as a tuple

Arow of data is the collection of all the columnsin a table associated with a single occurrence:A row of data is a single record in a table.

For example, if there are 25,000 book titles with which a bookstore deals, there will be 25,000 records, or rows of data, in the book titles table once the table is populated. The number of rows within the table will obviously change as books' titles are added and removed.



F

Figure 6 : Row of data in a database table

Data Types

A data type determines the type of data that can be stored in a database column. Although many data types are available, three of the most commonly used data types are

- Alphanumeric
- Numeric
- Date and time

Alphanumeric data types are used to store characters, numbers, special characters, or nearly any combination. If a numeric value is stored in an alphanumeric field, the value is treated as a character, not a number. In other words, you should not attempt to perform

Notes

arithmetic functions on numeric values stored in alphanumeric fields. Numeric data types are usedto store only numeric values.

Date and time data types are used to store date and time values, which widely vary depending on the relational database management system (RDBMS) being used.

Keys

The integrity of the information stored in a database is controlled by keys.Akey is a column value in a table that is used to either uniquely identify a row of data in a table, or establish a relationship with another table.Akey is normally correlated with one column in table, although it might be associatedwith multiple columns. There are two types of keys: primary and foreign.

Primary Keys

A primary key is the combination of one or more column values in a table that make a row of data unique within the table. Primary keys are typically used to join related tables. Even if a table has no child table, a primary key can be used to disallow the entry of duplicate records into a table. For example, an employee's social security number is sometimes considered a primary key candidate because all SSNs are unique.

Foreign Keys

A foreign key is the combination of one or more column values in a table that reference a primary key in another table. Foreign keys are defined in child tables.A foreign key ensures that a parent record has been created before a child record. Conversely, a foreign key also ensures that the child record is deleted before the parent record.

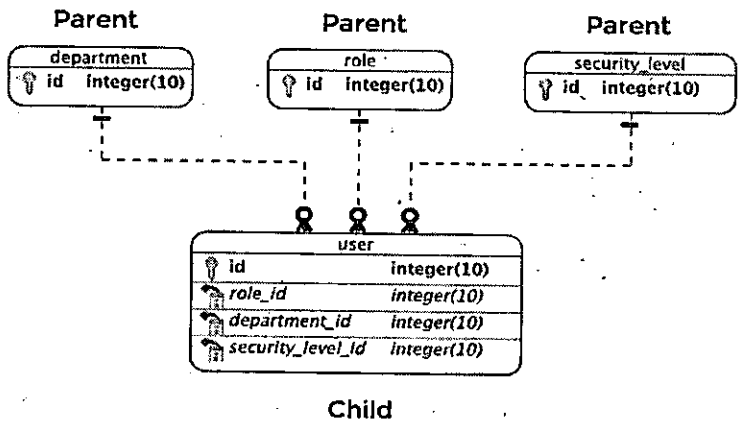


Figure 7 : Referential Integrity, or Parent/Child Relationships.

Relationship

DBMS relationships depend on the set-up and use of the DBMS. Arelationship exists between two database tables when one table has a foreign key that references the primary key of another table.

Types of Database Relationships

There are three different types of database relationships, each named according to the number of table rows that may be involved in the relationship. Each of these three relationship types exists between two tables.

- One-to-one relationships occur when each entry in the first table has one, and only one, counter- part in the second table. One-to-one relationships are rarely used because it is often more efficient to simply put all of the information in a single table.
- One record in a table is related to one record in a related table; creates equally dependent tables Ex. one student has only one PSU ID
- One-to-many relationships are the most common typeof database relationship. They occur when each record in the first table corresponds to one or more records in the second table but each record in the second table corresponds to only one record in the first table. For example, the relationship between a Teachers table and a Students table in an elementary school database would likely be a one-to-many relationship, because each student has only one teacher, but each teacher may have multiple students.
- Many-to-many relationships occur when each record in the first table corresponds to one or more records in the second table and each record in the second table corresponds to one or more records in the first table. For example, one student can be enrolled in many courses and each course can have many students enrolled

Database Management System Vs File Management System

A Database Management System (DMS) is a combination of computer software, hardware, and information designed to electronically manipulate data via computer processing. Two types of database management systems are DBMS's and FMS's. In simple terms, a File Management System (FMS) is a Database Management System that allows access to single files or tables at a time. FMS's accommodate flat files that have no relation to other files. The FMS was the predecessor for the Database Management System (DBMS), which allows access to multiple files or tables at a time.

File Management Systems

A file system is a technique of arranging the files in a storage medium like a hard disk, pen drive, DVD, etc. It helps you to organizes the data and allows easy retrieval of files when they are required. It mostly consists of different types of files like mp3, mp4, txt, doc, etc. that are grouped into directories.

A file system enables you to handle the way of reading and writing data to the storage medium. It is directly installed into the computer with the Operating systems such as Windows and Linux.

Notes

What is DBMS?

Database Management System (DBMS) is a software for storing and retrieving user's data while considering appropriate security measures. It consists of a group of programs that manipulate the database. The DBMS accepts the request for data from an application and instructs the DBMS engine to provide the specific data. In large systems, a DBMS helps users and other third-party software to store and retrieve data.

File Management Systems provide the following advantages and disadvantages:

The goals of a File Management System can be summarized as:

- Data Management. An FMS should provide data management services to the application.
- Generality with respect to storage devices. The FMS data abstractions and access methods should remain unchanged irrespective of the devices involved in data storage.
- Validity. An FMS should guarantee that at any given moment the stored data reflect the operations performed on them.
- Protection. Illegal or potentially dangerous operations on the data should be controlled by the FMS.
- Concurrency. In multiprogramming systems, concurrent access to the data should be allowed with minimaldifferences.
- Performance. Compromise data access speed and data transfer rate with functionality.

The goals of a Database Management System can be summarized as follows:

- Data storage, retrieval, and update (while hidingthe internal physical implementation details)
- A user-accessible catalog
- Transaction support
- Concurrency control services (multi-user update functionality)
- Recovery services (damaged database must be returned to a consistent state)
- Authorization services (security)
- Support for data communication Integrity services(i.e. constraints)
- Services to promote data independence
- Utility services (i.e. importing, monitoring, performance, record deletion, etc.) The components to facilitate the goals of a DBMS may include the following:
- Query processor
- Data Manipulation Language preprocessor
- Database manager (software components to include authorization control, command processor,

1.9 FEATURES COMMONLY OFFERED BY DATABASE MANAGEMENT SYSTEMS INCLUDE

Query Ability

Querying is the process of requesting attribute information from various perspectives and combinationsof factors. Example: "How many 2-door cars in Texas are green?" A database query language and report writer allow users to interactively interrogate thedatabase, analyze its data and update it accordingto the users privileges on data.

Backup and Replication

Copies of attributes need to be made regularly in case primary disks or other equipment fails. Aperiodic copy of attributes may also be created for a distant organization that cannot readily access the original. DBMS usually provide utilities to facilitate the process of extracting and disseminating attribute sets. When data is replicated between database servers, so that the information remains consistent throughout the database system and users cannot tell or even know which server in the DBMS they are using, the system is said to exhibit replication transparency.

Rule Enforcement

Often one wants to apply rules to attributes so that the attributes are clean and reliable. For example, we may have a rule that says each car can have only one engine associated with it (identified by Engine Number). If somebody tries to associate a second engine with a given car, we want the DBMS to deny such a request and display an error message. However, with changes in the model specification such as, in this example, hybrid gas-electric cars, rules may need to change. Ideally such rules should be able to be added and removed as needed without significant data layout redesign.

Security

For security reasons, it is desirable to limit who can see or change specific attributes or groups of attributes. This may be managed directly on an individual basis, or by the assignment of individuals and privileges to groups, or (in the most elaborate models) through the assignment of individuals and groupsto roles which are then granted entitlements.

Computation

Common computations requested on attributes are counting, summing, averaging, sorting, grouping, cross-referencing, and so on. Rather than have each computer application implement these from scratch, they can rely on the DBMS to supply such calculations.

Notes

Change and access logging

This describes who accessed which attributes, what was changed, and when it was changed. Logging services allow this by keeping a record of access occurrences and changes.

Automated optimization

For frequently occurring usage patterns or requests, some DBMS can adjust themselves to improve the speed of those interactions. In some cases the DBMS will merely provide tools to monitor performance, allowing a human expert to make the necessary adjustments after reviewing the statistics collected.

Features of DBMS A Typical DBMS has the Following Features

- Provides a way to structure data as records, tables, or objects
- Accepts data input from operators and stores that data for later retrieval
- Provides query languages for searching, sorting, reporting, and other "decision support" activities that help users correlate and make sense of collected data
- Provides multiuser access to data, along with security features that prevent some users from viewing and/or changing certain types of information
- Provides data integrity features that prevent morethan one user from accessing and changing the sameinformationsimultaneously
- Provides a data dictionary (metadata) that describes the structure of the database, related files, and record information

2.9 Database Instance

The term instance is typically used to describe a complete database environment, including the RDBMS software, table structure, stored procedures and other functionality. It is most commonly used when administrators describe multiple instances of the same database.

The information stored in database at the particular movement is called instance.

Also Known as: environment

Examples: An organization with an employees database might have three different instances: production (used to contain live data), pre-production (used to test new functionality prior to release into production) and development (used by database developers to create new functionality).

There are two different types of languages to make database system. They are 1. To specify the database schema, and 2.to express database queries and updates.

Data-Definition Language

A database schema is specified by a set of definitions expressed by special language called

a data-definition language (DDL). The result of compilation of DDL statements is a set of tables that is stored in a special file called Data dictionary or data directory.

A data dictionary is a file that contains metadata that is, data about data. This file is consulted before actual data are read or modified in the database system. The storage structure and access methods used by the database system are specified by a set of definitions in a special type of DDL called a data storage and definition language.

Data-Manipulation Language

The levels of abstraction apply not only to the definition or structuring of data, but also the manipulation of data. By data manipulation, it means

- The retrieval of information stored in the database
- The insertion of new information into the database
- The deletion of information from the database
- The modification of information stored in the database

A data-manipulation language (DML) is a language that enables user to access or manipulate data as organized by the appropriate data model. They are basically two types.

- Procedural DMLs require a user to specify what data are needed and how to get those data
- Nonprocedural DMLs require a user to specify what data are needed without specifying how to get those data

Non procedural DMLs are usually easier to learn and use than are procedural DMLs. A query is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a query language.

1.10 DATABASE MODELS

A database model s a collection of concepts that can be used to describe the structure of a database. By structure of a database we mean the data types, relationships and constraints that should hold on the data. A database model is the theoretical foundation of a database and fundamentally determines in which manner data can be stored, organized, and manipulated in a database system. It thereby defines the infrastructure offered by a particular database system. The most popular example of a database model is the relational model.

Data modeling is a method used to define and analyze data requirements needed to support the business processes ofanorganization and bydefining the data structuresand the relationships betweendata elements.

Data modeling techniques are used :

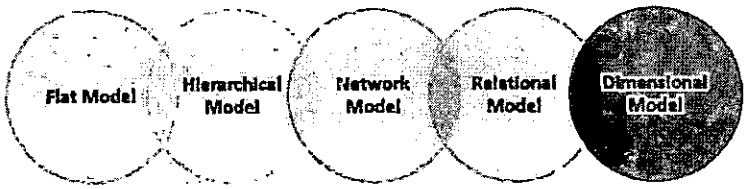
- to manage data as a resource (migrate, merge, data quality.)
- for designing computer databases.
- to better cope with change, byallowing to make changes into the model, that will automatically induce changes in the database and programs

Notes

Types of Database Models

Data models representations usually graphical of complex real-world data structures it facilitate interaction amongthedesigner, the applications programmer andthe end user. End-usersmust have different views and needs for data. Data model organizes data for various users.

Given below are the different types of database models:



1.11 FLAT MODEL

A flat database is a simple database system in which each database is represented as a single table in which all of the records are stored as single rows of data, which are separated by delimiters such as tabs or commas. The table is usually stored and physically represented as a simple text file.

Because of the limitations of flat databases, they are not unsuitable for most software applications in which there is a need to represent and store complex business relationships. However, some application developers still use flat files in order to reduce the cost and complexity of integrating a relational database.

Flat databases are also sometimes referred to as flat-file databases.

For example, there are two columns named as name and password which can be used by any security system. So each row is used to store different passwords and usernames. No two entry is the same in the flat model. In the flat model, the table format is used for storing the database. This database model has a disadvantage as it is unable to store huge chunks of data in the two-dimensional array because it is difficult to manage such a large set of entries in a flat database model.

	Route No.	Miles	Activity
Record 1	I-95	12	Overlay
Record 2	I-495	05	Patching
Record 3	SR-301	33	Crack seal

Flat model

Unlike relational databases, flat databases cannot represent complex relationships between entities. They also have no way of enforcing constraints between data. For instance

in an application used by a commercial bank, it is a good idea to ensure that, at the time of creation, a new account must be linked to an existing customer. In a relational database this is easily enforced using the concept of foreign keys to ensure that customer IDs are filled in while creating an account, and also that said customer IDs already exist in another table. This is not possible with flat databases, which means that such a constraint has to be enforced by other means, such a through application code logic.

Another limitation of flat databases vis-a-vis relational databases is the former's lack of query and indexing capability. SQL queries cannot be written in flat databases because the data is not relational, and indexes cannot be created because the data is all lumped together in one table. Data in a flat database is typically only readable by and useful to the software application associated with the database.

Flat databases are, or should only be, created for small, simple databases that will never grow large enough for the limitations outlined above to really become a problem. Some real-life examples of flat databases are contact lists in a mobile phone and the storage of a high-scores list in a simple video game. In such cases, there would be little point and no justifiable expense in integrating a complex relational database engine into the computing platform because a simple flat database will do nicely.

1.12 HIERARCHICAL MODEL

A hierarchical database model is a data model in which the data are organized into a tree-like structure. The data are stored as records which are connected to one another through links. A record is a collection of fields, with each field containing only one value.

In the hierarchical model, the data is stored in the tree-like structure in which there is a root node where the data is started to store. The sort field is used for the sibling record to maintain some order while storing data in the hierarchical model. The hierarchical database model is mostly used when there is a need for maintaining an information management system. In this database model there exist a one-to-many relationship among data. The retrieval of data has a different technique in the hierarchical model.

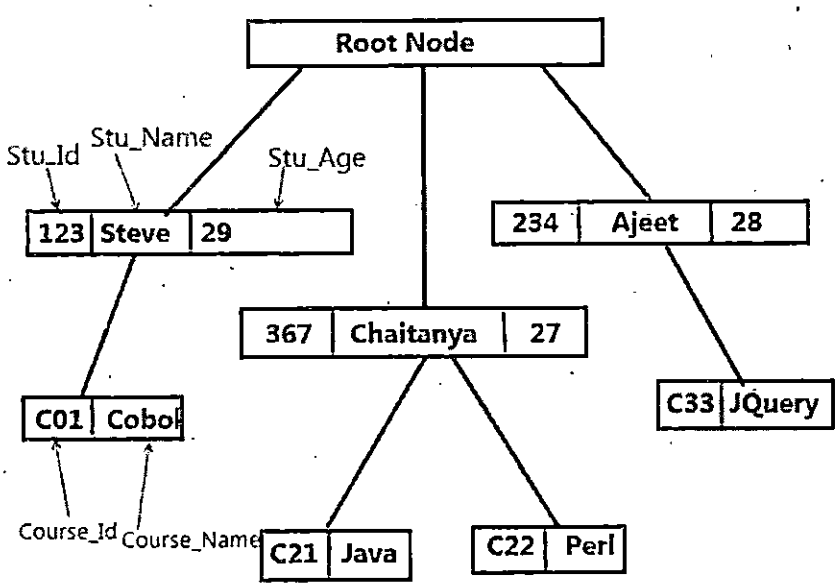
A hierarchical model represents the data in a tree-like structure in which there is a single parent for each record. To maintain order there is a sort field which keeps sibling nodes into a recorded manner. These types of models are designed basically for the early mainframe database management systems, like the Information Management System (IMS) by IBM.

This model structure allows the one-to-one and a one-to-many relationship between two/ various types of data. This structure is very helpful in describing many relationships in the real world; table of contents, any nested and sorted information.

The hierarchical structure is used as the physical order of records in storage. One can access the records by navigating down through the data structure using pointers which are combined with sequential accessing. Therefore, the hierarchical structure is not suitable for certain database operations when a full path is not also included for each record.

Data in this type of database is structured hierarchically and is typically developed as an inverted tree. The "root" in the structure is a single table in the database and other tables act as the branches flowing from the root. The diagram below shows a typical hierarchical database structure.

Notes



The technique follows navigating a downward direction using the pointers so that data can be accessed sequentially. Because of this technique, the hierarchical database model has a disadvantage as it is not capable of handling all types of database operations. The retrieval time of data is more compare to other types of database models. The records are not connected to the hierarchical model. And the model consists of an upward link that is used to represent the hierarchy of data stored in the database system.

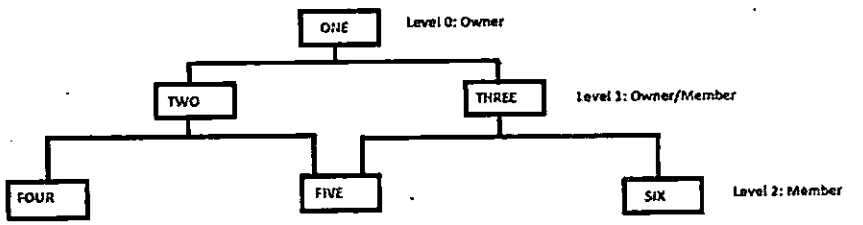
1.13 NETWORK DATA MODEL

This model was formalized by the Database Task group in the 1960s. This model is the generalization of the hierarchical model. This model can consist of multiple parent segments and these segments are grouped as levels but there exists a logical association between the segments belonging to any level. Mostly, there exists a many-to-many logical association between any of the two segments. We called graphs the logical associations between the segments. Therefore, this model replaces the hierarchical tree with a graph-like structure, and with that, there can more general connections among different nodes. It can have M: N relations i.e, many-to-many which allows a record to have more than one parent segment.

Here, a relationship is called a set, and each set is made up of at least 2 types of record which are given below:

- An owner record that is the same as of parent in the hierarchical model.
- A member record that is the same as of child in the hierarchical model.

Structure of a Network Model :



1.14 RELATIONAL DATA MODEL

The relational database model is designed to remove the dependency in the database management system. The development of the relational database model is targeted for mostly microcomputer systems. The three keys which are used in relational database models are domains, attributes, and relations. The relation is defined as a table that contains rows and columns. The columns present in the table are called attributes in the relational database model. And the domain is defined as a set of values that can be inserted in the database model. The data structure used is a table in this model and the data is stored in the form of rows and columns in the model. The rows are also known as tuples.

For example, there is a relation named employee details. It can have multiple attributes like name, age, gender. And there can be multiple tuples for one single relation named as employee details. The key which can be used to uniquely identify any row is called the primary key. And these keys can be used to join more than one table at the same time. For example, there can be a column named location in the employee table and the column location can be the primary key of the location table.

By this key, the two table location table and employee table can join and the operations can be performed on the join table. The relational database model uses the structured query language (SQL) to perform operations on the database system. The relational model contains multiple tables which look like flat database model. One of the key advantages of this model is whenever the value appears more than on time in two different records there exist some relationships among them.

1.15 ENTITY-RELATIONSHIP MODEL

ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.

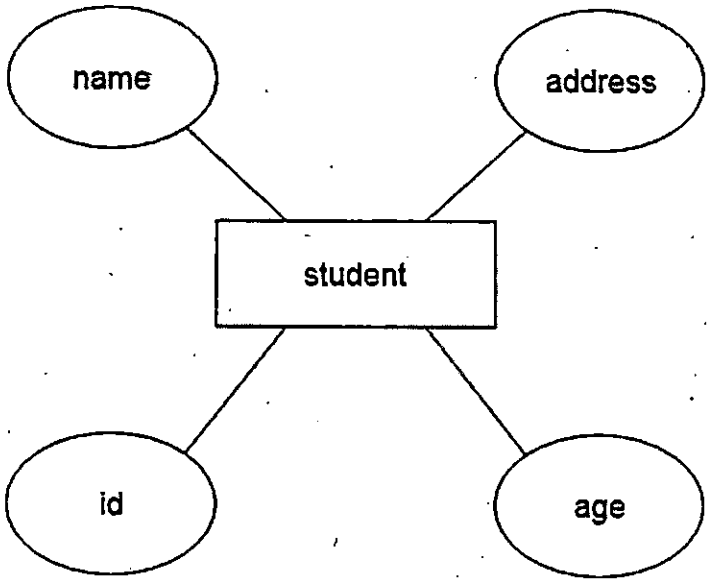
It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.

In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.

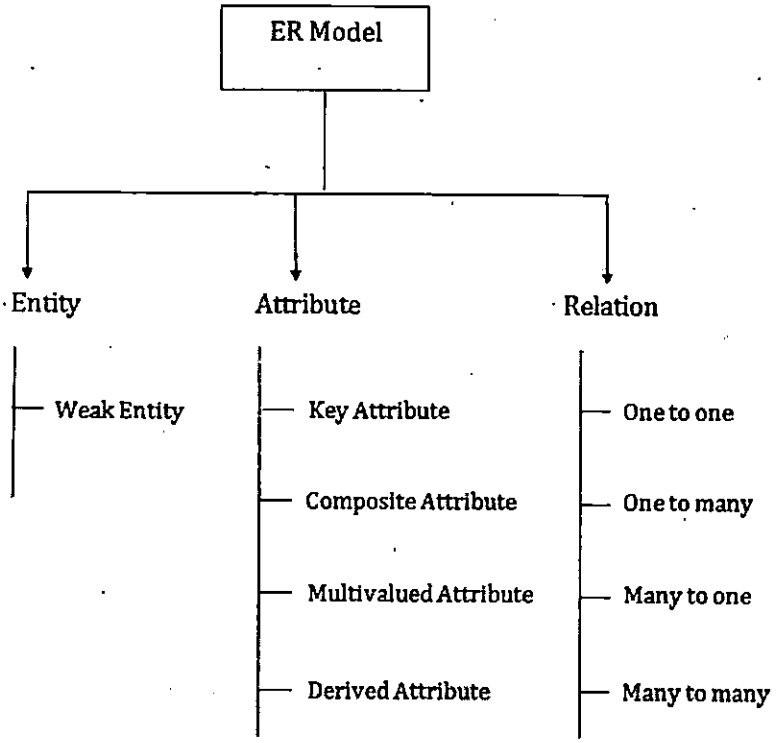
For example, Suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with

Notes

attributes like city, street name, pin code, etc and there will be a relationship between them.



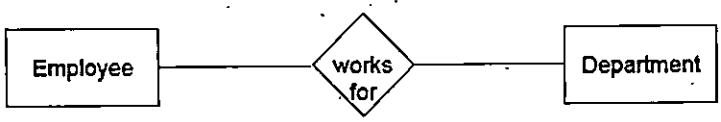
Component of ER Diagram



1. Entity:

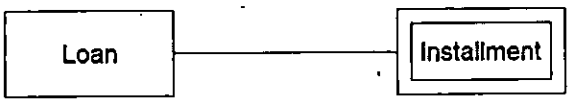
An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.

Consider an organization as an example- manager, product, employee, department etc. can be taken as an entity.



a. Weak Entity

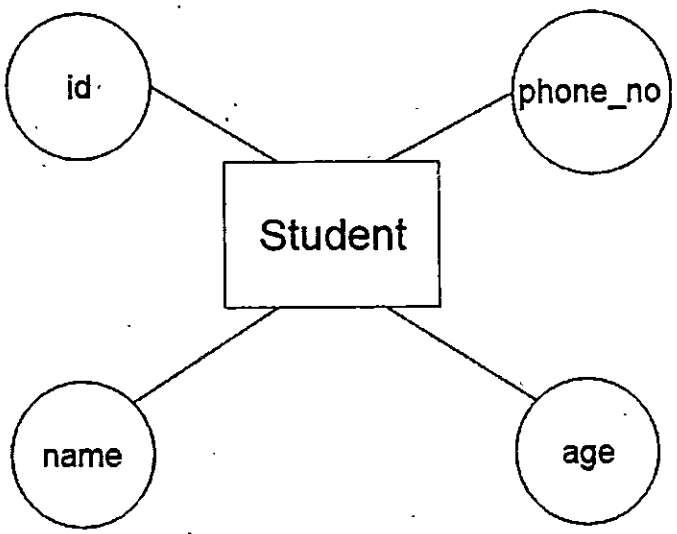
An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.



2. Attribute

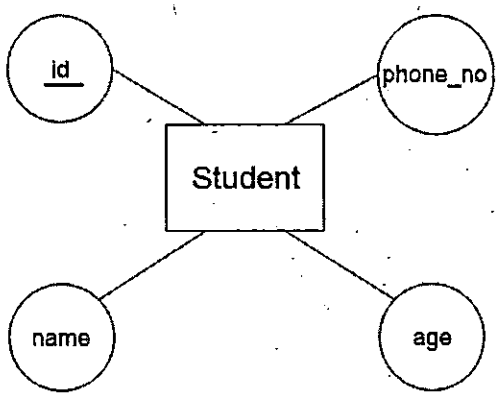
The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.

For example, id, age, contact number, name, etc. can be attributes of a student.



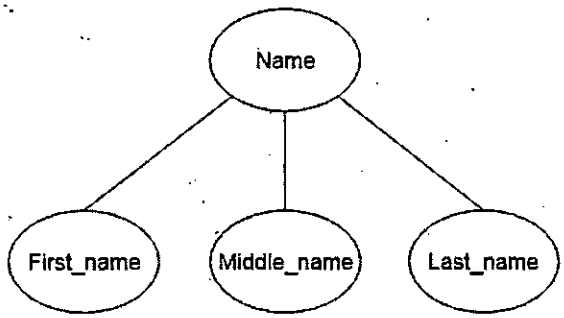
a. Key Attribute

The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined.



b. Composite Attribute

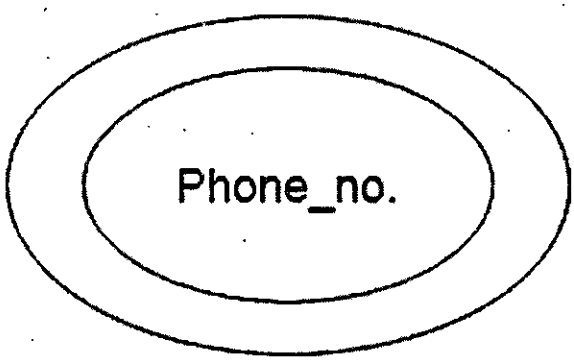
An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.



c. Multivalued Attribute

An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent multivalued attribute.

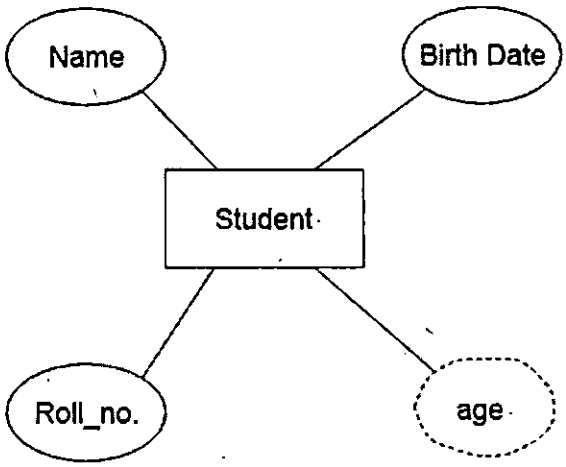
For example, a student can have more than one phone number.



d. Derived Attribute

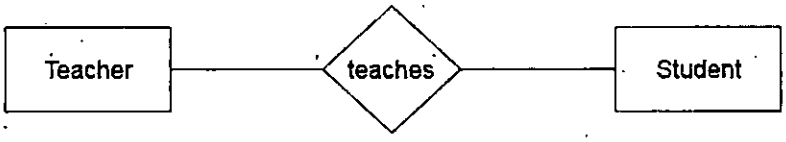
An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse.

For example, A person's age changes over time and can be derived from another attribute like Date of birth.



3. Relationship

A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.



Types of relationship are as follows:

a. One-to-One Relationship

When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.

For example, A female can marry to one male, and a male can marry to one female.

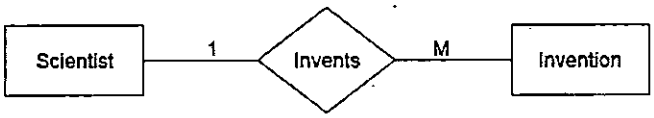
b. One-to-many relationship

When only one instance of the entity on the left, and more than one instance of an entity on

Notes

the right associates with the relationship then this is known as a one-to-many relationship.

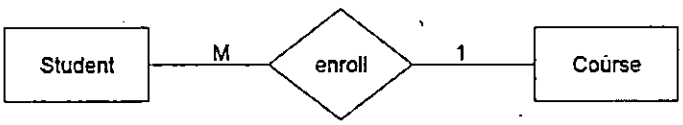
For example, Scientist can invent many inventions, but the invention is done by the only specific scientist.



c. Many-to-one relationship

When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.

For example, Student enrolls for only one course, but a course can have many students.



d. Many-to-many relationship

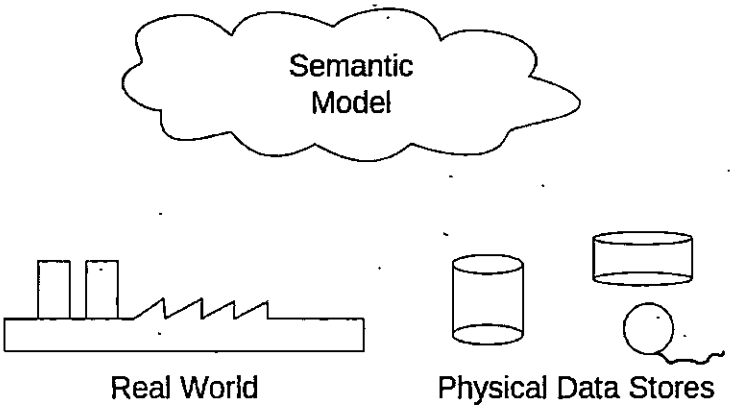
When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.

For example, Employee can assign by many projects and project can have many employees.

4. Semantic Data Model

The semantic data model is a method of structuring data in order to represent it in a specific logical way. It is a conceptual data model that includes semantic information that adds a basic meaning to the data and the relationships that lie between them. This approach to data modeling and data organization allows for the easy development of application programs and also for the easy maintenance of data consistency when data is updated.

The semantic data model is a relatively new approach that is based on semantic principles that result in a data set with inherently specified data structures. Usually, singular data or a word does not convey any meaning to humans, but paired with a context this word inherits more meaning.



In a database environment, the context of data is often defined mainly by its structure, such as its properties and relationships with other objects. So, in a relational approach, the vertical structure of the data is defined by explicit referential constraints, but in semantic modeling this structure is defined in an inherent way, which is to say that a property of the data itself may coincide with a reference to another object.

A semantic data model may be illustrated graphically through an abstraction hierarchy diagram, which shows data types as boxes and their relationships as lines. This is done hierarchically so that types that reference other types are always listed above the types that they are referencing, which makes it easier to read and understand.

Abstractions used in a semantic data model:

- Classification - "instance_of" relations
- Aggregation - "has_a" relations
- Generalization - "is_a" relations

5. Dimensional Model

The dimensional database model is defined as a specialized version of the relational database model. This model is used to represent data in the data warehouses in such a manner so that data can be summarized using OLAP queries and online analytical processing. In the dimensional database model, the database schema consists of a single table of a huge size which contains facts and dimensions. The dimension is used to define the context of fact and is in hierarchical form. The dimensional database model uses the star schema which contains a highly normalized table that has facts and dimensions.

The database models are widely used to store data in different structural forms. The goal is to maintain the structure properly so that data retrieval time is minimum as possible. And the database operations can be performed easily in different types of database models.

Notes

1.16 DATA MODELING USING THE ENTITY RELATIONSHIP MODEL

Conceptual modeling is 'animportant phaseindesigning a successfuldatabase application. Generally, the term database application refers to a particular database. These programs often provide user-friendly graphical user interfaces(GUI's) utilizing forms and menus.

Hence, part of the database application will require the design, implementation and testing of these application programs.

Traditionally the design and testing of application programs has bee considered to be more in the realm of the software Engineering domain than in the database domain.

E-R model concepts

The E-R data model gives ussubstantialflexibility in designinga database schema to model a given enterprise. Inthissectionwe consider howadatabasedesigner mayselect fromthe widerange ofalternatives. Among the decisions to be made are the following :

(a) Whether to use an attribute or an entity set to present an object.

(b) Whether a real-world concept is expressed most accurately by an entity set or by a relationship set.

(c) Whether to use a ternary relationship or a pair of binaryrelationships.

(d) Whether to use a strong or a weak entity set(section 4.8.3), a strong entity set and its dependent weak entity set may be regarded as a single 'object' in the database, since weak entities are existence dependent on a strong entity set.

(e) Whether using generalizationis appropriate , generalization is a hierarchy of a relationships contributes to modularitybyallowingcommonattributesofsimilar entitysets to bepresented in one place in an E-R diagram.

(f) Whether using aggregation is appropriate, aggregation groups a part of an E-R diagram into a single entity set, allowing us to treat the aggregate entity set as a single unit without concern for the details of its internal structure.

Notation for E-R Diagram

The overall logical structure of a database can be expressed graphically by an E-R diagram. The relative simplicityand pictorialclarity ofthis diagramming maywellaccount in large part for the widespread use of the E-R model. The choices of names for entity types, attributes, relationship types and roles is not always straightforward. One should choose names that convey as much as possible , the Such a diagram consists of the following major components. Summary of the ER diagram notation are as follows

Binary Relationship Abinary relationship R is an association between two instances of two different entity types (i.e., R . E1 × E2).

For example, ina university, a binaryrelationship exists between a student (STUDENT

entity) and an instructor (FACULTY entity) of a single class; an instructor teaches a student.

Ternary Relationship Aternaryrelationship R is anassociation between three instances of three different entitytypes (i.e., R . E1 × E2 × E3). For example, consider a student using certainequipment for a project. In this case, the STUDENT, PROJECT, and EQUIPMENT entity types relate to each other with ternary relationships: a student checks out equipment for a project.

1.17 DATA MODELING USING THE ENTITY RELATIONSHIP MODEL

Conceptual modeling is animportant phaseindesigning a successfuldatabase application. Generally, the term database application refers to a particular database. These programs often provide user-friendly graphical user interfaces(GUI's) utilizing forms and menus. Hence, part of the database application will require the design, implementation and testing of these application programs. Traditionallythe design and testing of application programs has bee considered to be more in the realm of the software Engineering domain than in the database domain.

E-R model concepts

The E-R data model gives us substantial flexibility in designinga database schema to model a given enterprise. In this section we consider how a data base designer mayselect from the wide range of alternatives. Among the decisions to be made are the following :

(a) Whether to use an attribute or an entity set to present an object.

(b) Whether a real-world concept is expressed most accurately by an entity set or by a relationship set.

(c) Whether to use a ternary relationship or a pair of binaryrelationships.

(d) Whether to use a strong or a weak entity set(section 4.8.3), a strong entity set and its dependent weak entity set may be regarded as a single 'object' in the database, since weak entities are existence dependent on a strong entity set.

(e) Whether using generalizationis appropriate , generalization is a hierarchy of a relationships contributes to modularitybyallowingcommonattributesofsimilar entitysets to bepresented in one place in an E-R diagram.

(f) Whether using aggregation is appropriate, aggregation groups a part of an E-R diagram into a single entity set, allowing us to treat the aggregate entity set as a single unit without concern for the details of its internal structure.

Notation for E-R Diagram

The overall logical structure of a database can be expressed graphically by an E-R diagram. The relative simplicityand pictorialclarity ofthis diagramming maywellaccount in large part for the widespread use of the E-R model. The choices of names for entity types, attributes,

Notes

relationship types and roles is not always straightforward. One should choose names that convey as much as possible , the Such a diagram consists of the following major components. Summary of the ER diagram notation are as follows:

Rectangles : which represent entity sets.

Ellipses : which represent attributes.

Diamonds : which represent relationship sets.

Lines : which link attributes to entitysets and entity sets to relationship sets.

Double ellipses : which represent multivalued attributes.

Dashed ellipses : which denote derived attributes.

- Double lines : which indicate total participation ofan entity in a relationship set.
- Directed line : a directed line from relationship set to the entity set specifies that relationship is either a one-to one, or manyto one relationship set.
- Undirected line : a undirected line from the relationship set relationship to the entity set specifies that relationship is either a many to many, or a one to manyrelationship set .
- Underlined ellipses : which indicate key attributes.

Entity sets

We know that a database usually contains group of entities that are similar. Anentitytype defines a collection of entities that have the same attributes. Each entity type in the database is described by its name and attributes.

The collection of all entities of a particular entity type in the database at anypoint in time is called an entity set, the entity set is usuallyreferred to using the same name as the entitytype. There are times you might wish to define an entity set even though its attributes do not formally contain a key.

Entity sets do not need to be disjoint. For example, it is possible to define the entity set of all employees of a bank and the entity set of all customers of the bank. Aperson entity maybe an employee entity, a customer entity, both or neither.

An entity set maynot have sufficient attributes to form a primarykey. Such an entity set is termed a weak entity set. Usually, this is the case only because the information represented in such an entity set is only interesting whencombined through an identifying relationship set with another entity set wecallthe identifying owner.

We will call such a set a weak entity set, and insist on the following:

- The weak entityset must exhibit akeyconstraint with respect to the identifying relationship set.
- The weak entity set must have total participation in the identifying relationship set.

Together, this assures us that we can uniquelyidentify eachentity from the weak set byconsidering the primary key of its identifying owner together with a partial key from the weak entity.

In our ER diagrams, we will represent a weak entity set byoutlining the entity and the identifying relationship set with dark lines. The required key constraint and total participation are diagrammed with our existing conventions. We underline the partial keywith a dotted line.

In the physics laboratory ER model example, the entity type student is strong because its existence does not depend on some other entity type. However, the team entity type is weak. The existence of team depends on the existence of LabSection, and we call the in identifying relationship. We draw double lines around the identifying relationship, the team entitytype, and the line connecting the two to indicate the weak entity type.

A weak entityset does not have sufficient attributes to form a primarykey(not globally unique), a discriminator is selected for the weak entity set.

- The discriminator of a weak entity set is a set of attributes that allow us to distinguish entities that are all dependent on one particular strong entity set.

Example (ofdiscriminator):

{ Section number }

- The primarykey of a weak entityset can be formed bythe primarykey of the strong entity set on whichit is existence dependent, plus its discriminator.

Example:

{Course name, section number }

or, {Course number, section number }

Note: An entity type is strong if its existence does not depend on some other entity type. Otherwise, the entity type is weak.

Keys

An attribute or set of attributes that uniquely identifies a particular entity is a key. For example, Emanuel Vagas'Student Identification Number uniquelyidentifies him. It is important to be able to specify how entities within a given entityset and relationships withina given relationship set are distinguished..

Concepts of Super Key

A super key of an entity set is a set of one or more attributes whose values uniquely determine each entity. The combination of primary keys of the participating entity sets forms a super key of a relationship set.

(customer-id, account-number) is the super key of depositor

Note : Thismeans a pair of entity sets can haveat most one relationship in a particular relationship set. E.g. if we wish to track all access-dates to each account byeachcustomer, we cannot assume a relationship for each access.

Super key: a set of one or more attributes which, takencollectively, allow us to

Notes

identifying an entity instance in anentity set uniquely.

Example:

{ student ID, name}

{ social insurance number, address }

{ row number, column number }

Concepts of Candidate Key

A minimalset of attributes in a table that uniquely identifies a record. When there is more than one attribute in the candidate key, it is called composite key. Acandidate key that is chosen to represent a record uniquely. Each entity type must have an attribute or set of attributes that distinguishes one instance from other instances of the same type. Attribute (or combination of attributes) that uniquelyidentifies each instance of an entity type .

Candidate key : a smallest possible superkey. i.e. a superkey for which no proper subset is a superkey.

Example :

{ student ID }

{ social insurance number }

{ row number, column number }

Concepts of Primary Key

A table can have at most one primary key, but more than one unique key. A primary key is a combination of columns which uniquelyspecify a row. It is a specialcase of unique keys. One difference is that primary keys have an implicit NOT NULL constraint while unique keys do not. Thus, the values in unique keycolumns may or maynot be NULL, and in fact such a column maycontain at most one NULL fields.

Another difference is that primary keys must be defined using another syntax. That is, a table may consist of many candidate keys, but ONLY ONE can be selected as a primary key. attribute we say that the primary key is a composite primarykey. One of the common mistakes students have made is to refer to one of the attributes in the composite key as primary key. Please note that when there is more than one attribute inthe primarykeyALLATTRIBUTES TOGETHER are called the primarykeysince theytogether define the record uniquely.

Primary key: a candidate key chosen by database designer as the principal means of identifying entities withinan entityset.

Example :

{ student ID }

Primarykeyselection criteria:

{ not change over time}

{ no null value}

Extended E-R Features

The processes of specialization, generalization and Aggregation are used to find such opportunities. These processes serve as conceptual models for the development of superclass/ subclass relationships.

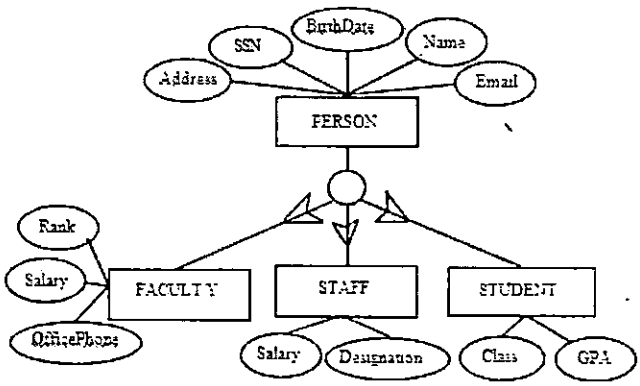
Generalization

Generalization is the process of defining general entity types from a set of specialized entity types by identifying their common characteristics. In other words, this process minimizes the differences between entities by identifying a general entity type that features the common attributes of specialized entities. Generalization is a bottom-up approach as it starts with the specialized entity types (subclasses) and forms a generalized entity type (superclass).

For example, suppose that someone has given us the specialized entitytypes FACULTY, STAFF, and STUDENT, and we want to represent these entity types separately in the E-R model as depicted in Figure 3.25 (a). However, if we examine them closely, we can observe that a number of attributes are common to all entity types, while others are specific to a particular entity.

For example, FACULTY, STAFF, and STUDENT all share the attributes Name, SSN, Birth Date, Address, and Email. On the other hand, attributes such as GPA, Class, and MajorDept are specific to the STUDENTS; OfficePhone is specific to FACULTY, and Designation is specific to STAFF. Common attributes suggest that each of these three entity typesis a form of a more generalentitytype. This general entitytype is simplya PERSON superclass entity with common attributes of three subclasses .

Thus, in the generalization process, we group specialized entity types to form one general entity type andidentifycommon attributes of specialized entities as attributes of a generalentitytype. The general entity type is a superclass of specialized entitytypes or subclasses.



Specialization

Specialization is the process of defining one or more subclasses of a superclass byidentifying its distinguishing characteristics. Unlike generalization, specialization is thus a top-down

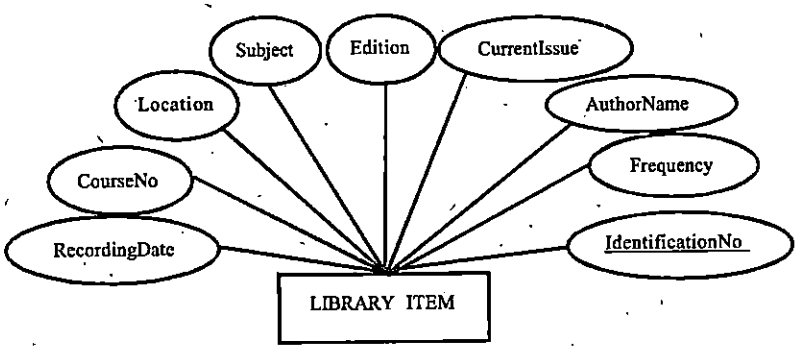
Notes

approach. It starts with the general entity (superclass) and forms specialized entity types (subclasses) based on specialized attributes or relationships specific to a subclass.

For example, consider LIBRARY ITEM is an entity type with several attributes such as IdentificationNo, Recording Date, Frequency, and Edition. After careful review of these items, it should become clear that some items such as books do not have values for attributes such as Frequency, Recording Date, and Course No, while Video CDs do not have an Author or an Edition.

In addition, all items have common attributes such as IdentificationNo, Location, and Subject. Someone creating a library database, then, could use the specialization process to identify superclass and subclass relationships.

In this case, the original entity LIBRARY ITEM forms a superclass entity type made up of attributes shared by all items, while specialized items with distinguishing attributes, such as BOOK, JOURNALS, and VIDEOCD, form subclasses.



Aggregation

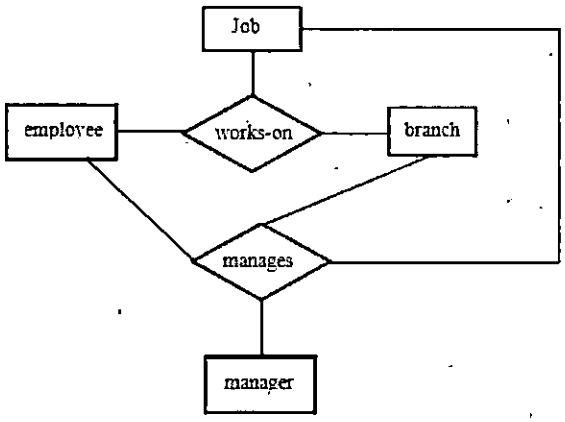
One limitation ofthe E-Rmodelis that it isnot possible to expressrelationships amongrelationships. To illustrate the need for such a construct, we consider again a database describing information about employee and branch. Suppose that each employee-branchpair may work with the managers.

Using our basic E-R modeling constructs, we obtain the E-R diagram. It appears that the relationship set works-on and manages can be combined into one single relationship set. We should not combine them, because doing so would obscure the logical structure of this schema.

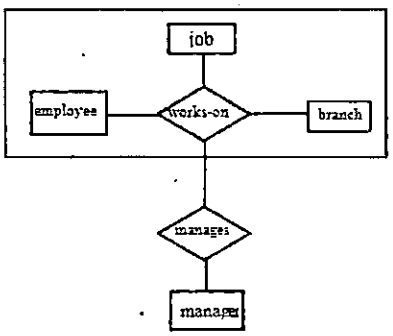
For example if we consider the works-on and managesrelationshipsets, then this combinationspecifies that a managesmust be assigned to everyemployee- branch pair, which is not true.

There is redundant information in the resultant figure, however , since everyemployee-branchpair in manages is also in works-on. The best way to modela situation such as the one just described is to use aggregation.

Aggregation is an abstraction through which relationships are treated as higher-level entities.



Thus we regard the relationship set work-on and the entity sets employee and branch as a higher-level entity set called works-on. Such en entity set is treated in the same manner as in any other entity set.



1.18 ADVANTAGES AND DISADVANGES OF DBMS MODEL

A Database Management System (DBMS) is defined as the software system that allows users to define, create, maintain and control access to the database. DBMS makes it possible for end users to create, read, update and delete data in database. It is a layer between programs and data.

Advantages of DBMS

Compared to the File Based Data Management System, Database Management System has many advantages. Some of these advantages are given below –

Reducing Data Redundancy

The file based data management systems contained multiple files that were stored in many

Notes

different locations in a system or even across multiple systems. Because of this, there were sometimes multiple copies of the same file which lead to data redundancy.

This is prevented in a database as there is a single database and any change in it is reflected immediately. Because of this, there is no chance of encountering duplicate data.

Sharing of Data

In a database, the users of the database can share the data among themselves. There are various levels of authorisation to access the data, and consequently the data can only be shared based on the correct authorisation protocols being followed.

Many remote users can also access the database simultaneously and share the data between themselves.

Data Integrity

Data integrity means that the data is accurate and consistent in the database. Data Integrity is very important as there are multiple databases in a DBMS. All of these databases contain data that is visible to multiple users. So it is necessary to ensure that the data is correct and consistent in all the databases and for all the users.

Data Security

Data Security is vital concept in a database. Only authorised users should be allowed to access the database and their identity should be authenticated using a username and password. Unauthorised users should not be allowed to access the database under any circumstances as it violates the integrity constraints.

Privacy

The privacy rule in a database means only the authorized users can access a database according to its privacy constraints. There are levels of database access and a user can only view the data he is allowed to. For example - In social networking sites, access constraints are different for different accounts a user may want to access.

Backup and Recovery

Database Management System automatically takes care of backup and recovery. The users don't need to backup data periodically because this is taken care of by the DBMS. Moreover, it also restores the database after a crash or system failure to its previous condition.

Data Consistency

Data consistency is ensured in a database because there is no data redundancy. All data appears consistently across the database and the data is same for all the users viewing the database. Moreover, any changes made to the database are immediately reflected to all the users and there is no data inconsistency.

Advantages of DBMS

- Various formats of data can be stored, and data can be retrieved by a range of methods in the Structured Query Language.
- As most databases are usually centralized in nature, they can be accessed quickly and managed easily.
- We can set authorized users who can view, share, and access data. This ensures security for data.
- DBMS facilitates the smooth incorporation of programming languages, such as C++, Python, and PHP, to allow users to establish a connection with a web application or third-party applications.
- To prevent data from being inaccessible when there is an overload, a recovery system with automated backups is provided by almost every DBMS software.
- With minimum data duplicity and redundancy, DBMS provides data protection and integrity.

Disadvantages of DBMS

- In some cases, DBMSs are highly complicated systems to set up and maintain.
- The cost of DBMS hardware and applications is comparatively high, exhausting an organization's budget.
- In certain organizations, all information is integrated into a common database, which may get destroyed due to electrical issues or it may get corrupted in storage media. Having backups is preferred in such situations but that increases the cost.
- Certain DBMS systems cannot run complex queries as they slow down the other processes that are running.
- To get an in-depth explanation of DBMS concepts, check out our three-hour video on Database Management and SQL:

SUMMARY

- A database is an organized collection of data and the records, which is stored in a system, usually a computer. The organization of data is achieved by structuring it according to the model of the database.
- A DBMS can take any one of the several approaches to manage data. Each approach constitutes a database model.Adata model is a collection of descriptions of data structures and their contained fields, together with the operations or functions that

Notes

manipulate them.Adata model is a comprehensive scheme for describng how data is to be represented for manipulation by humans or computer programs.

- DBA stands for database administrator, he is a person who is responsible for the features of a databse like databse recovery, integrity, availability, security and so on. DBAhas to check these features in order to maintain the database.
- A temporal database is formed by compiling, storing temporal data. The difference between temporal data and non-temporal data is that a time-period is appended to data expressing when it was valid or stored in the database. The data stored by conventional databases consider data valid at present time as in the time instance "now." When data in such a database is modified, removed, or inserted, the state of the database is overwritten to form a new state. The state prior to any changes to the database is no longer available. Thus, by associate time with data, it is possible to store the different database states. In essence, temporal data is formed by time-stamping ordinary data (type of data we associate and store in conventional databases). In a relational data model, tuples are time-stamped and in an object-oriented data model, objects/attributes are time stamped.
- A key is a field or combination of fields used to identify a record. When a key uniquely identifies a record it is referred to as the primary key.
- A database management system (DBMS) is a software system that integrates data in storage and provides easy access to them. The data themselves are placed on disk in a database, which can be thought of as an integrated collection of related files. Although not all databases are organized identically, many of them are composed of files, records and fields.
- 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.
- A schema is a group of related objects in a database. Within a schema, objects that are related have relationships to one another. It is overall design of the database. There is one owner of a schema, who has access to manipulate the structure of any object in the schema. A schema does not represent a person, although the schema is associated with a user account that resides in the database.
- The three models associated with a schema are as follows: The conceptual model, also called the logical model, is the basic database model, which deals with organizational structures that are used to define database structures such as tables and constraints.

Notes

- The internal model, also called the physical model, deals with the physical storage of the database, as well as access to the data, such as through data storage in tables and the use of indexes to expedite data access. The internal model separates the physical requirements of the hardware and the operating system from the data model.
- A table is the primary unit of physical storage for data in a database. The table is the most fundamental element found in a database schema. Columns and rows are associated with tables When a user accesses the database, a table is usually referenced for the desired data. Multiple tables might comprise a database, therefore a relationship might exist between tables. Because tables store data, a table requires physical storage on the host computer for the database.
- 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 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.

Notes

- 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.

KEY WORDS

- **A data model :** It is a collection of concepts for describing data, its relationships, and its constraints provides a clearer and more accurate description and representation of data. Data models can be broadly distinguished into 3 main categories- High-level or conceptual data models, Low levelor physical data models, Representational or implementation data models.
- **Hierarchical model:** Data is organized into a tree-like structure, implying a single upward link in each record to describe the nesting, and a sort field to keep the records in a particular order in each same-level list.
- **The network model :** It organizes datausing two fundamental constructs, called records and sets. Records contain fields (which may be organized hierarchically, as in the programming language COBOL). All relations (and, thus, tables) in a relational model have to adhere to some basic rules to qualify as relation
- **Attribute :** a property or description of an entity. Atoy department employee entity could have attributes describing the employee's name, salary, and years of service.
- **Domain : a set of possible values for an attribute.**
- **Entity:** an object in the real world that is distinguishable from other objects such as the green dragon toy.
- **Relationship:** an association among two or more entities.
- **Entityset:** a collection of similar entities such as all of the toys in the toy department.
- **Relationship set:** a collection of similar relationships
- **One-to-many relationship:** A key constraint that indicates that one entity can be
- **Participation constraint:** a participation constraint determines whether relationships must involve certain entities. Anexample is if every department entity has a manager entity. Participation constraints can either be total or partial. Atotalparticipation constraint says that everydepartment has a manager. Apartial participation constraint says that everyemployee does not have to be a manager.
- **Client/Server:** A server is a program that runs on a computer that directly manages the database. A client is a separate program (or process) that communicates with the database server through some kind of Remote Procedure Call (RPC) in order to **perform application-specific database operations.**
- **Cloud:** Cloud is a recently coined term used to describe an execution model for computing systems where functions and data are invoked by a name that refers to a

remote system whose location is irrelevant (hence the concept of it being "out there somewhere." like a cloud). Cloud-based systems allow thin-client interfaces to access this functionality through the internet, and frequently with wi-fi, reducing the power requirements of the client computers.

- **Column** - A single unit of named data that has a particular data type (e.g., number, text, or date). Columns only exist in tables.

REVIEW QUESTIONS

1. What do you understand by DBMS? Explain.
2. How file management system is different from DBMS? Explain.
3. List the advantages of DBMS.
4. What is Key? How primary key is different from foreign key?
5. Explain the concept of relationship with the helpof an example.
6. Explain the different function of DBMS.
7. Explain the role of DBA.
8. Explain the role of different users of DBMS.
9. What is Data Independence? Explain.
10. Explain the concept of DDL and DML commands in DBMS.
11. Explain whywe study the data models in databases.
12. What is data model and write down the basic characteristics of the databases?
13. What is the difference between E-R modeland Relational model?
14. What is the difference between network modeland hierarchical model?
15. How can you define the relational model and its characteristics?
16. Provide an example of multiple relationships between entities. Draw the E-R diagram.
17. Under what conditions is a relationship converted to an associative entity type? Give an example.
18. Explain why we study the E-R model of a database.
19. A department in a university stores the information about its students and courses in a database. The administrative assistant manages the database. At the end of the semester, he prepares a report about each course. Is the E-R diagram correct? If not, explain why and draw the correct diagram.

Notes

20. Explain the distinctions among the terms primary key, candidate key, and super key.
21. Define the concepts ofAggregation. Give two examples of where this concept is useful.
22. Explain the difference between a Strong and weak entity set.
23. Explain the distinction between total and partial design constraints. Explainyour answer.

FURTHER READINGS

1. "database, n". OED Online. Oxford University Press. June 2013. Retrieved July 12, 2013. (Subscription required.)
2. IBM Corporation (October 2013). "IBM Information Management System (IMS) 13 Transaction and Database Servers delivers high performance and low total cost of ownership". Retrieved Feb 20, 2014.
3. "Structured Query Language (SQL)". International Business Machines. October 27, 2006. Retrieved 2007-06-10. Wagner 2010.
4. Ramalho, J.C., Faria, L., Helder, S., & Coutada, M. (2013, December 31). Database Preservation Toolkit: A flexible tool to normalize and give access to databases. University of Minho. https://core.ac.uk/display/55635702?
5. Kroenke, David M. and David J. Auer. Database Concepts. 3rd ed. New York: Prentice, 2007.
6. Raghu Ramakrishnan and Johannes Gehrke, Database Management Systems
7. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, Database System Concepts

Unit 2

Relational Database Design

Notes

Structure

2.0 LEARNING OBJECTIVES

After reading this chapter students will be able to:

- provides some level of Underlying the structure of a database is the data model: a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints
- provides a general overview of the nature and purpose of database systems
- explain the how the concept of a database system has developed, what the common features of database systems are, what a database system does for the user, and how a database system interfaces with operating systems
- know the determining the user's information requirements explain the feasibility study and considerations
- Define a Relation; Know the purpose of Normalization; and understand the steps involved for normalization process.

Notes

2.1 INTRODUCTION

Most of the problems faced at the time of implementation of any system are outcome of a poor database design. In many cases it happens that system has to be continuously modified in multiple respects due to changing requirements of users. It is very important that a proper planning has to be done.

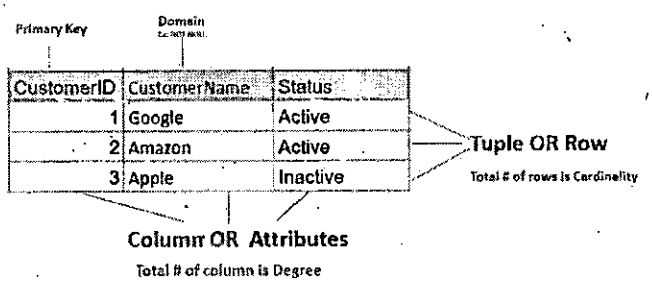
A relation in a relational database is based on a relational schema, which consists of number of attributes. A relational database is made up of a number of relations and corresponding relational database schema. The goal of a relational database design is to generate a set of relation schema that allows us to store information without unnecessary redundancy and also to retrieve information easily.

One approach to design schemas that are in an appropriate normal form. The normal forms are used to ensure that various types of anomalies and inconsistencies are not introduced into the database. The objective of this module to understand about Relational database management system. A type of database in which the data can be spread across several tables that are related together.

Data in related tables are associated by shared attributes. Any data element can be found in the database through the name of the table, the attribute (column) name, and the attribute values that uniquely identify each row. Afterlearning this module they differentiated between DBMSand RDBMS. They understand that a RDBMS requires few assumptions about how data is related or how it will be extracted from the database. As a result, the data can be arranged in different combinations.

A relational DBMS is special system software that is used to manage the organization, storage, access, security and integrity of data. This specialized software allows application systems to focus on the user interface, data validation and screen navigation. When there is a need to add, modify, delete or display data, the application system simply makes a "call" to the RDBMS. Although there are many different types of database management systems, relational databases are by far the most common. Other types include hierarchical databases and network databases.

The database management systems have been around since the 1960s; relational databases didn't become popular until the 1980s. A relational DBMS stores information in a set of "tables", each of which has a unique identifier or "primary key". The tables are then related to one another using "foreign keys". A foreign key is simply the primary key in a different table. Diagrammatically, a foreign key is depicted as a line with an arrow at one end.



In the Figure 2.1 , "Customer ID" is the primary key (PK) in one table and the foreign key (FK) in another. The arrow represents a one-to-many relationship between the twotables. The relationship indicates that one customer can have one or more orders. One and only one customer, however, can initiate a given order. By storing data in a RDBMS, undesirable data redundancy can be avoided. This not only makes data management easier, but it also makes for a flexible database that can respond to changing requirements.

2.2 WHAT IS RDBMS?

RDBMS stands for Relational Database Management System. RDBMS data is structured in database tables, fields and records. Each RDBMS table consists of database table rows. Each database table row consists of one or more database table fields.

RDBMS store the data into collection of tables, which might be related by common fields (database table columns). RDBMS also provide relational operators to manipulate the data stored into the database tables. Most RDBMS use SQL as database querylanguage.

The most popular RDBMS are MS SQL Server, DB2, Oracle and MySQL. The relational model is an example of record-based model. Record based models are so named because the database is structured in fixed format records of several types. Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes. The columns of the table correspond to the attributes of the record types.

The relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model. The relational model was designed by the IBM research scientist and mathematician, Dr. E.F.Codd. Many modern DBMS do not conform to the Codd's definition of a RDBMS, but nonetheless they are still considered to be RDBMS.

Two of Dr.Codd's main focal points when designing the relational model were to further reduce data redundancy and to improve data integrity within database systems. The relational model originated from a paper authored by Dr.codd entitled "A Relational Model of Data for Large Shared Data Banks", written in 1970. This unit included the following concepts that apply to database management systems for relational databases.

The relation is the only data structure used in the relational data model to represent both entities and relationships between them. Rows of the relation are referred to as tuples of the relation and columns are its attributes. Each attribute of the column are drawn from the set of values known as domain. The domain of an attribute contains the set of values that the attribute may assume.

From the historical perspective, the relational data model is relatively new. The first database systems were based on either network or hierarchical models. The relationaldata model has established itself as the primary data model for commercial data processing applications. Its success in this domain has led to its applications outside data processing in systems for computer aided design and other environments.

Notes

2.3 DIFFERENCE BETWEEN DBMS AND RDBMS

A DBMS has to be persistent, that is it should be accessible when the program created the data ceases to exist or even the application that created the data restarted. A DBMS also has to provide some uniform methods independent of a specific application for accessing the information that is stored.

RDBMS is a Relational Data Base Management System Relational DBMS. This adds the additional condition that the system supports a tabular structure for the data, with enforced relationships between the tables. This excludes the databases that don't support a tabular structure or don't enforce relationships between tables.

You can say DBMS does not impose any constraints or security with regard to data manipulation it is user or the programmer responsibility to ensure the ACID PROPERTY of the database whereas the RDBMS is more with this regard because RDBMS define the integrity constraint for the purpose of holding ACID PROPERTY

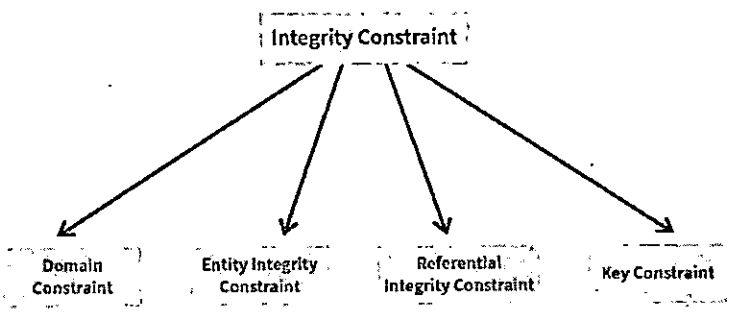
2.4 INTEGRITY CONSTRANTS

In database management systems (DBMS) there is a certain set of rules which are used to maintain the quality and consistency of data in the database. Every time there is an insertion, deletion, or updating of data in the database it is the responsibility of these integrity constraints to maintain the integrity of data and thus help to prevent accidental damage to the database.

In Database Management Systems, integrity constraints are pre-defined set of rules that are applied on the table fields(columns) or relations to ensure that the overall validity, integrity, and consistency of the data present in the database table is maintained.

Evaluation of all the conditions or rules mentioned in the integrity constraint is done every time a table insert, update, delete, or alter operation is performed. The data can be inserted, updated, deleted, or altered only if the result of the constraint comes out to be True. Thus, integrity constraints are useful in preventing any accidental damage to the database by an authorized user.

Types of Integrity Constraints



There are four types of integrity constraints in DBMS:

- Domain Constraint
- Entity Constraint
- Referential Integrity Constraint
- Key Constraint

Domain Constraint

Domain integrity constraint contains a certain set of rules or conditions to restrict the kind of attributes or values a column can hold in the database table. The data type of a domain can be string, integer, character, DateTime, currency, etc.

Example:

Consider a Student's table having Roll No, Name, Age, Class of students.

Roll No	Name	Age	Class
101	Adam	14	6
102	Steve	16	8
103	David	8	4
104	Bruce	18	12
105	Tim	6	A

In the above student's table, the value A in the last row last column violates the domain integrity constraint because the Class attribute contains only integer values while A is a character.

Entity Integrity Constraint

Entity Integrity Constraint is used to ensure that the primary key cannot be null. A primary key is used to identify individual records in a table and if the primary key has a null value, then we can't identify those records. There can be null values anywhere in the table except the primary key column.

Example:

Consider Employees table having Id, Name, and salary of employees

ID	Name	Salary
1101	Jackson	40000
1102	Harry	60000
1103	Steve	80000
1104	Ash	1800000
	James	36000

Notes

In the above employee's table, we can see that the ID column is the primary key and contains a null value in the last row which violates the entity integrity constraint.

Referential Integrity Constraint

Referential Integrity Constraint ensures that there must always exist a valid relationship between two relational database tables. This valid relationship between the two tables confirms that a foreign key exists in a table. It should always reference a corresponding value or attribute in the other table or be null.

Example:

Consider an Employee and a Department table where Dept_ID acts as a foreign key between the two tables

Employees Table

ID	Name	Salary	Dept_ID
1101	Jackson	40000	3
1102	Harry	60000	2
1103	Steve	80000	4
1104	Ash	1800000	3
1105	James	36000	1

Department Table

Dept_ID	Dept_Name
1	Sales
2	HR
3	Technical

In the above example, Dept_ID acts as a foreign key in the Employees table and a primary key in the Department table. Row having DeptID=4 violates the referential integrity constraint since DeptID 4 is not defined as a primary key column in the Departments table.

Key constraint

Keys are the set of entities that are used to identify an entity within its entity set uniquely. There could be multiple keys in a single entity set, but out of these multiple keys, only one key will be the primary key. A primary key can only contain unique and not null values in the relational database table.

Example:

Consider a student's table

Roll No	Name	Age	Class
101	Adam	14	6
102	Steve	16	8
103	David	8	4
104	Bruce	18	12
102	Tim	6	2

The last row of the student's table violates the key integrity constraint since Roll No 102 is repeated twice in the primary key column. A primary key must be unique and not null therefore duplicate values are not allowed in the Roll No column of the above student's table.

Conclusion

Integrity Constraints in Database Management Systems are the set of pre-defined rules responsible for maintaining the quality and consistency of data in the database. Evaluation against the rules mentioned in the integrity constraint is done every time an insert, update, delete, or alter operation is performed on the table.

- Integrity Constraints in DBMS are of 4 types:
- Domain Constraint
- Entity Constraint
- Referential Integrity Constraint
- Key Constraint

2.5 FUNCTIONAL DEPENDENCIES

A functional dependency (FD) is a relationship between two attributes, typically between the PK and other non-key attributes within a table. For any relation R, attribute Y is functionally dependent on attribute X (usually the PK), if for every valid instance of X, that value of X uniquely determines the value of Y. This relationship is indicated by the representation below :

X - - - -> Y

The left side of the above FD diagram is called the determinant, and the right side is the dependent. Here are a few examples.

In the first example, below, SIN determines Name, Address and Birthdate. Given SIN, we can determine any of the other attributes within the table.

SIN - - - -> Name, Address, Birthdate

For the second example, SIN and Course determine the date completed (DateCompleted). This must also work for a composite PK.

SIN, Course - - - -> DateCompleted

The third example indicates that ISBN determines Title.

ISBN - - - -> Title

Notes

Rules of Functional Dependencies

Consider the following table of data r(R) of the relation schema R(ABCDE) shown in Table 2.1.

Table 2.1. Functional dependency example, by A. Watt.

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_1	c_2	d_2	e_1
a_3	b_2	c_1	d_1	e_1
a_4	b_2	c_2	d_2	e_1
a_5	b_3	c_3	d_1	e_1

As you look at this table, ask yourself: What kind of dependencies can we observe among the attributes in Table R? Since the values of A are unique (a1, a2, a3, etc.), it follows from the FD definition that:

$A \rightarrow B, \quad A \rightarrow C, \quad A \rightarrow D, \quad A \rightarrow E$

It also follows that $A \rightarrow BC$ (or any other subset of ABCDE).

This can be summarized as $A \rightarrow BCDE$.

From our understanding of primary keys, A is a primary key.

Since the values of E are always the same (all e1), it follows that:

$A \rightarrow E, \quad B \rightarrow E, \quad C \rightarrow E, \quad D \rightarrow E$

However, we cannot generally summarize the above with $ABCD \rightarrow E$ because, in general, $A \rightarrow E, \quad B \rightarrow E, \quad AB \rightarrow E$.

Other observations:

Combinations of BC are unique, therefore $BC \rightarrow ADE$.

Combinations of BD are unique, therefore $BD \rightarrow ACE$.

If C values match, so do D values.

Therefore, $C \rightarrow D$

However, D values don't determine C values

So C does not determine D, and D does not determine C.

Looking at actual data can help clarify which attributes are dependent and which are determinants.

Inference Rules

Armstrong's axioms are a set of inference rules used to infer all the functional dependencies on a relational database. They were developed by William W. Armstrong. The following describes what will be used, in terms of notation, to explain these axioms.

Let R(U) be a relation scheme over the set of attributes U. We will use the letters X, Y, Z to represent any subset of and, for short, the union of two sets of attributes, instead of the usual X U Y.

Notes

2.6 NORMALIZATION

Normalization is a technique that is more applicable to record-based data models for e.g. a relational database model. Each of the processes can be carried out independently to arrive at normalized tables (depending on how detailed the decompositions is). If ER Modeling is done in detail, normalization may not be required at all. However, some people use the ER diagram as an input to normalization i.e. if the tables derived from ER diagrams are in the first normal form.

In this section, we will concentrate on normalization which is an important step in database design, particularly for relation DBMSs. The relational data model is based on a relation. When structuring data that is to be stored, the analyst must anticipate the need to access the data to meet unexpected requirements and to reduce redundancy. These can be achieved through the techniques of 'Normalization' that provides a systematic way of boiling data structures down to their simplest possible forms.

Database Design

Database design involves designing the conceptual model of the databases. This model isindependent of the physical representation of data. Before actually implementing the database, the conceptual model is designed using various techniques.

The requirements of all the users are taken into account to decide the actual data that needs to be stored in the system. Once the conceptual model is designed, it can then be mapped to the DBMS/ RDBMS that is actually being used. Two of the widely used approaches are Entity-Relationship (ER) Modeling and Normalization.

Meaning of 'Relation'

A 'Relation' is a two-dimensional table. It consists of 'rows' which represent records and 'columns'which show the attributes of the entity. Arelation is also called a file, it consists of a number of records which are also called as tuples. Record consists of a number of attributes which are also known as fields or domains. In order for a relational structure to be useful and manageable, the relation tables must first be 'normalized'.

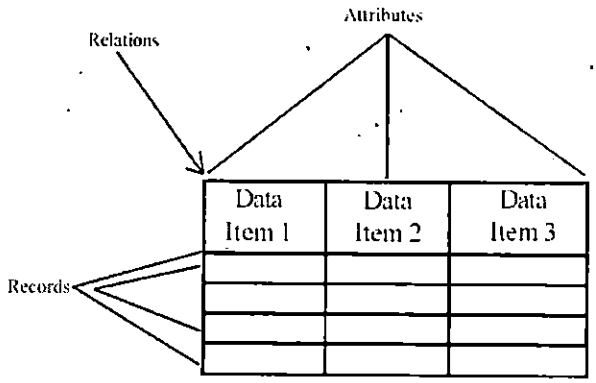


Figure 1: Components of a 'Relation'

Notes

Some of the properties of a relation are,

- No duplication: In the sense that no two records are identical.
- Unique key: Each relation has a unique key by which it can be accessed.
- Order: There is no significant order of data in the table

Un-normalized Set (yet relational!):

Employee

Man#	Name	Birthdate	JobHistory				Children	
123	MichaelSwart	Nov.22	JobDate	Title	SalaryHistory		Childname	Birthyear
			2000	Lackey	SalaryDate	Salary	Mini-me1	2000
					2000	$1,000,000	Mini-me2	2002
					2001	$2,000,000		
			2002	SeniorLackey	SalaryDate	Salary		
					2002	$3,000,000		
					2003	$4,000,000		

Normalized Set (1NF)

Employee'

Man#	Name	Birthdate
123	MichaelSwart	Nov.22

JobHistory'

Man#	JobDate	Title
123	2000	Lackey
123	2002	SeniorLackey

Children'

Man#	ChildName	BirthYear
123	Mini-me1	2000
123	Mini-me2	2002

SalaryHistory'

Man#	JobDate	SalaryDate	Salary
123	2000	2000	$1,000,000
123	2000	2001	$2,000,000
123	2002	2002	$3,000,000
123	2003	2003	$4,000,000

Figure 2 : Un-normalized Employee Record

Figure 2 shows a relation (un-normalized form) of the employee entity. As per the figure an unnormalized form of an employee record consists of the following,

Employee no., employee name, employee details (department code, grade, date of joining,

exit code and exit date), annual salary earned (MMYY, net paid), bank details (bank code, bank name, address, employees A/C no). In case we want the names of all the employees whose grade is 20, we can scan the employee relation, noting the grade. Here the unique key is the employee number.

Here it is clearly seen that the employee's annual salary earned details which are Month and Year paid, net paid, are being repeated. Therefore this relation is not following the normalization process. Now, the question arises 'what is normalization'?

Introduction to Normalization

Normalization is a process of simplifying the relationship between data elements in a record. It is the transformation of complex data stores to a set of smaller, stable data structures. Normalized data structures are simpler, more stable and are easier to maintain. Normalization can therefore be defined as a process of simplifying the relationship between data elements in a record.

Purpose of Normalization

Normalization is carried out for the following four reasons,

- To structure the data so that there is no repetition of data, this helps in saving space.
- To permit simple retrieval of data in response to query and report requests.
- To simplify the maintenance of the data through updates, insertions and deletions.
- To reduce the need to restructure or reorganize data when new application requirements arise.

Steps of Normalization

Normalization process starting with a data store developed for a data dictionary and analyst normalizes a data structure in three steps. Each step involves an important procedure to simplify the data structure.

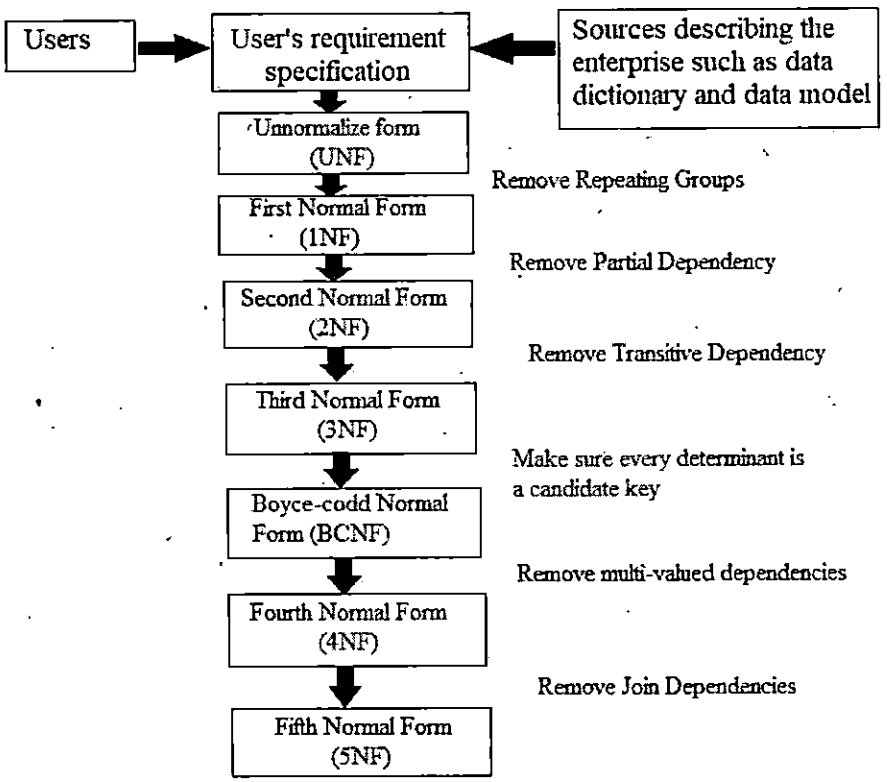


Figure 3. Steps of Normalization

It consists of basic three steps:

1. First Normal Form which decomposes all data groups into two-dimensional records.
2. Second Normal form which eliminates any relationship in which data elements do not fully depend on the primary key of the record.
3. Third Normal Form which eliminates any relationships that contain transitive dependencies

Notes

Functional Dependency

So, before we start to learn about 'Normalization and its steps', we have to understand the term 'Functional dependency'.

Let X and Y be two attributes of a relation R. Given the value of X, if there is only one value of Y corresponding to it, then y is said to be functionally dependent on X.

X Y

So, Y is functionally dependent on X.

Consider an example, given the value of item code, there is only one value of item name for it. Thus item name is functionally dependent on item code.

Item Code → Item Name

Functional dependency may also be based on a composite attributes. Suppose, if we write,

X, Z → Y

It means that there is only one value of Y corresponding to given values of X,Z. So, Y is functionally dependent on the composite X, Z. In figure 4, for example, Order no., and Item code together determine

Qty. and Price.

Order no., Item code → Qty. Price

Order no.	Order date	Item code	Quantity	Price/unit
1456	260289	3687	52	50.40
1456	260289	4627	38	60.20
1456	260289	3214	20	17.50
1886	040389	4629	45	20.25
1886	040389	4627	30	60.20
1788	040489	4627	40	60.20

Figure 4 : Normalized form of the Relation

There are mainly three types of functional dependencies,

(a) Full functional dependency

(b) Partial functional dependency

(c) Transitive dependency

(a) Full functional dependency :

Consider the following Relation

REPORT1 (S#, SName, C#, CTitle, Iname, Room#, Marks, Grade)

S# → Student Number

SName → Student Name

- C# → Course Number
- CTitle → Course Title
- Iname → Name of the Instructor who delivered the course
- Room# → Room number which is assigned to respective instructor
- Marks → Scored in Course COURSE# by the student STUDENT#
- Grade → Obtained by student STUDENT# in Course C#

Functional dependencies in the given example are,

- S# → SName
- C# → CTitle
- C# → Iname (Assuming one course is thought by only one instructor)
- IName → Room# (Assuming each instructor has his won non-shared room)
- S#.C# → Marks
- Marks → Grade
- S#.C# → Grade

Dependency Diagram :

REPORT1(S#, C#, CTitle, SName, Iname, Room#, Marks, Grade)

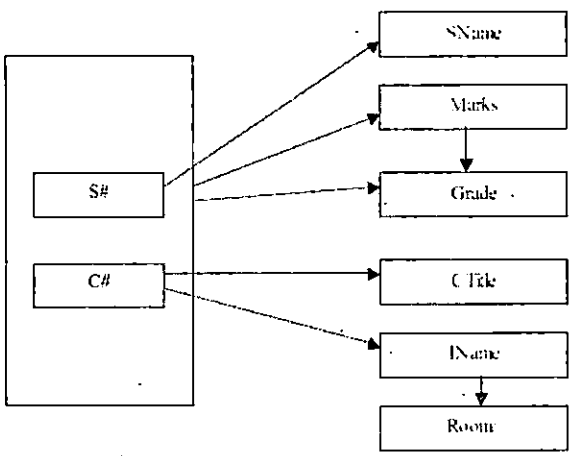


Figure 5 : Dependency Diagram

In above example, Marks is fully functionally dependent on S#, C# and not on subset of S#, C#.

This means, marks cannot be determined either by S# or C# alone. It can be determined only using S# and C# together. Hence Marks is fully functionally dependent on S# and C#.

Notes

(b) Partial functional dependency:

If in a relation R, X and Y are attributes. Attribute Y is partially dependent on the attribute X only if it is dependent on a sub-set of attribute X.\

In the above relationship (figure 7.4) CTitle, Iname are partially dependent on composite attributes S#, C# because C# alone defines the C Title, Iname.

(c) Transitive dependency:

In a give relation R, there are three attributes X, Y and Z. Y is functionally dependent on X and Z is functionally dependent on Y. Therefore there is an indirect dependency between X and Z. This is called 'Transitive (indirect) Dependency'.

$X \rightarrow Y$

$Y \rightarrow Z$

$X \rightarrow Z$

In the above example, Room# depends on Iname and in turn Iname depends on C#. Hence Room# transitively depends on C#.

Similarly, Grade depends on Marks, in turn Marks depends on S#, C# hence Grade depends transitively on S#, C#.

Normal Forms

Normalization reduces redundancy. Redundancy is the unnecessary repetition of data. It can cause problems with storage and retrieval of data. Redundancy can lead to

- Inconsistencies: Errors are more likely to occur when facts are repeated.
- Update Anomalies
  - o Inserting, modifying and deleting data may cause inconsistencies.
  - o There is a high likelihood of updating or deleting data in one relation, while omitting to make corresponding changes in other relations.

First Normal Form (1NF)

A relation is said to be in first normal form if all attributes defined on domains containing atomic values. Consider the following relation; this relation is un-normalized because each row contains multiple values.

The relational model does not permit tables that the un-normalized. In the relational mode, every relation is in first normal form. Hence, the tables arrived at form the entity-relationship diagram should be at least in first normal form.

ECODE	DEPT	PROJCODE	HOURS
EN101	SYSTEMS	P27	90
		P51	101
		920	60
EN305	SALES	P27	109
EN508	ADMIN	P51	Null
		P27	72

Figure 6 : Un-normalized Relation

Steps for above converting a database to 1NF:

- Put all double values in separate lines.

ECODE	DEPT	PROJCODE	HOURS
EN101	SYSTEMS	P27	90
EN101	SYSTEMS	P51	101
EN101	SYSTEMS	920	60
EN305	SALES	P27	109
EN508	ADMIN	P51	NULL
EN508	ADMIN	P27	72

Figure 7 : A relation in First Normal Form (1NF)

Second Normal Form (2NF)

Data in the tables in 1NF may be redundant. Consider the table in figure 7.6. This table stores the following attributes:

- ECODE : employee code.
- DEPT : department to which the employee belongs.
- PROJCODE : code of project on which employee is working.
- HOURS : number of hours worked on project.

The primary key here is composite (ECODE + PROJCODE). Few attributes in this table depends upon only part of the primary key as given below,

- PROJCODE + ECODE functionally determine HOURS.
- ECODE functionally determines DEPT. Attributes DEPT has no dependency on PROJCODE.

Notes

This situation could lead to the following problems,

(a) Insertion : The department of a particular employee cannot be recorded until the employee is assigned a project.

(b) Updation : For a given employee, the employee code and department is repeated several times.

Hence, if an employee is transferred to another department, this change will have to be recorded in every instance or record of the employee. Any omissions will lead to inconsistencies.

(c) Deletion : If an employee completes work on a project, his/her record will be deleted. The information regarding the department the employee belongs to will also be lost.

The table in figure 8 should, therefore, be decomposed without any loss of information as shown under,

ECODE
EN101
EN305
EN508

ECODE	PROJCODE	HOURS
EN101	P27	90
EN101	P51	101
EN101	920	60
EN305	P27	109
EN508	P51	NULL
EN508	P27	72

Figure 8 : No-loss decomposition in Second Normal Form (2NF)

Notice that the original table can be reconstructed with a join.

A table is said to be in Second Normal Form when it is in First Normal Form, and every attribute in the record is functionally dependent upon the whole key, and not just a part of the key.

The steps for converting a database to Second Normal Form (2NF):

- Identify the functional dependencies in the relation.
- Identify attributes that are dependent only to a part of key (partial dependency).
- Remove partial dependencies by placing them in new relations together with a copy of determinant.

Third Normal Form (3NF)

A table is said to be in 3NF when it is in 2NF and every non-key attribute is functionally dependent

on just the primary key. For example, consider the figure 7.8. The primary key is ECODE. The attribute DEPT is dependent on ECODE. The attribute DEPT-HEAD is dependent on DEPT-HEAD is the code of the department head. Notice that there is a transitive dependence between ECODE and DEPT-HEAD attributes.

ECODE	DEPT	DEPT-HEAD
E101	*SYSTEMS*	*E901*
E305	*SALES*	*E906*
E402	*SALES*	*E906*
E508	*ADMIN*	*E908*
E607	*FINANCE*	*E909*
E608	*FINANCE*	*E909*

Figure 9 : Table with Transitive Dependency

The problems with transitive dependency are:

- Insertion:

  The department head of a new department that does not have any employees as yet cannot be entered. This is because the primary key is unknown.

- Updation :

  For a given department, the department head's code is repeated several times. Hence, if a department head is moved to another department, the change will have to be made consistently across.

- Deletion :

  If a particular employee record is deleted, the information regarding the head of the department will also be deleted. Hence, there will be a loss of information.

The relation is therefore, reduced to the following two relations:

ECODE	DEPT
E101	*SYSTEMS*
E305	*SALES*
E402	*SALES*
E508	*ADMIN*
E607	*FINANCE*
E608	*FINANCE*

DEPT	DEPT-HEAD
SYSTEMS	*E901*
SALES	*E906*
ADMIN	*E908*
FINANCE	*E909*

Figure 10 : Removing Transitive Dependency

Notes

Each non-key attribute depends of the key, the whole key and nothing but the key.

Boyce-Code Normal Form (BCNF)

The original definition of 3NF was inadequate in some situations. It was not satisfactory for relation that:

- Had multiple candidate keys, where

  ⇒ Those candidate keys were composite.

  ⇒ The candidate keys overlapped (had at least one attribute in common).

Hence, a new normal form – the Boyce-Codd Normal Formal (BCNF) was introduced. We must understand that in relations where the above three conditions do not apply, we can stop at the third normal form.

In such cases, 3NF is the same as BCNF.

Let us examine BCNF in detail. Consider the table in figure 11. The relation PROJECT holds details on the number of hours spent by each employee working on each project.

ECODE	EMAILID	PROJCODE	HOURS
E1	bk@rediffmail.com	P2	48
E1	bk@rediffmail.com	P5	100
E1	bk@rediffmail.com	P6	15
E4	rahul@rediffmail.com	P5	250
E5	rahul@rediffmail.com	P5	75

Figure 11 : Project Table

Notice the following dependencies in this relation :

- The attributes ECODE and PROJCODE functionally determine HOURS.
- The attribute EMAILID and PROJCODE also functionally determine determines HOURS.
- ECODE functionally determine EMAILID.
- EMAILID functionally determine ECODE.

Notice that this relation has :

- Multiple candidate keys.
- The candidate keys are composite.
- The candidate keys overlap – PROJCODE is common.

This is a case for the Boyce-Codd Normal Form. This relation is in 3NF. The only non-key item is HOURS, and it is dependent on the whole key and only the key, i.e. PROJCODE+ECODE or PROJCODE+EMAILID. However, this relation has redundancies. If the emailed of and employee is changed, the change will have to be made in every tuple of

the relation, otherwise inconsistencies will creep in. This table needs to be further decomposed to eliminate dependence between the candidate key columns.

Figure 11 illustrates the non-loss decomposition of the table in figure 12.

ECODE	PROJCODE	HOURS
E1	P2	48
E1	P5	100
E1	P6	15
E4	P5	250
E4	P5	75

ECODE	EMAILID
E1	bk@rediffmail.com
E4	rahul@rediffmail.com

Figure 12 : Non-loss decomposition of Project table in BCNF

2.7 POPULAR RDBMS PACKAGES

A database is a collection of data that is related to a particular topic or purpose. As an example, employee records in a filing cabinet, a collection of sales leads in a notebook, are examples of collections of data or databases. A Database Management System (DBMS) is a system that stores and retrieves information in a database. It is used to help you organize your data according to a subject, so that it is easy to track and verify your data, and you can store information about how different subjects are related, so that it makes it easy to bring related data together.

A DBMS stores data in a table where the entries are filed under a specific category and are properly indexed. This allowed programmers to have a lot more structure when saving or retrieving data.

A relational database contains data in more than one table. Each table contains a database that is then linked to other tables with respect to their relationships.

This section provides a brief description about various popular RDBMS packages. Some of them are used for commercial purpose while others are available as an open source packages.

Data is the most important aspect in computing. Any program, whether big or small, needs data in order to process and produce its output; which often is some sort of data. Storing data has evolved a lot over the last few years. The first method of storing data before was in text files but this way very inefficient and difficult to deal with large amounts of data.

DBMS provides search functionalities in order to find a certain database entry. Once it is found, you can then pull out any other related information from that entry. DBMS is a very competent system for keeping track of data, but it doesn't scale very well. Dealing with huge databases, although possible, becomes a huge task in DBMS.

Relational database contains data in more than one table. Each table contains a database that is then linked to other tables with respect to their relationships. RDBMS is an improvement over the older DBMS. It provides the mechanism to overcome the restrictions that DBMS faces.

Notes

Various Types of RDBMS Packages

A DBMS that is based on relational model is called as RDBMS. Relation model is most successful mode of all three models. Designed by E.F.Codd, relational model is based on the theory of sets and relations of mathematics.

Relational model represents data in the form a table. A table is a two dimensional array containing rows and columns. Each row contains data related to an entity such as a student. Each column contains the data related to a single attribute of the entity such as student name. One of the reasons behind the success of relational model is its simplicity. It is easy to understand the data and easy to manipulate.

Another important advantage with relational model, compared with remaining two models is, it doesn't bind data with relationship between data item. Instead it allows you to have dynamic relationship between entities using the values of the columns.

Almost all Database systems that are sold in the market, now- a-days, have either complete or partial implementation of relational model.

Commercial RDBMS V/s. Open Source RDBMS

Basically Commercial (such as Oracle, MSSQL Server etc) will cost money, Non Commercial (MySQL, PostgreSQL) are 'free' or at least open source. They are developed in different ways and come with different levels of support.

SQL Server

Microsoft SQL Server is a relational database server which is developed by Microsoft. MS SQL Server is a software product whose primary function is to store and retrieve data as requested by other software applications from networks or other systems. There are at least a dozen different editions of Microsoft SQL Server aimed at different audiences and for different workloads, some of them are

Table 2.1: Various Versions of SQL Server

Various versions of SQL Server		
Version	*Year*	*Release Name*
1.0 (OS/2)	*1989*	*SQL Server 1.0 (16bit)*
1.1 (OS/2)	*1991*	*SQL Server 1.1 (16bit)*
4.21 (WinNT)	*1993*	*SQL Server 4.21*
6.0	*1995*	*SQL Server 6.0*
6.5	*1996*	*SQL Server 6.5*
7.0	*1998*	*SQL Server 7.0*
7.0	*1999*	*SQL Server 7.0 (with OLAP Tools)*
8.0	*2000*	*SQL Server 2000*
8.0	*2003*	*SQL Server 2000 (64-bit Edition)*
9.0	*2005*	*SQL Server 2005*
10.0	*2008*	*SQL Server 2008*
10.25	*2010*	*SQL Azure*
10.5	*2010*	*SQL Server 2008 R2*
11.0		*SQL Server 2012*

Characteristics

Microsoft SQL Server is an application used to create computer databases for the Microsoft Windows family of server operating systems. Microsoft SQL Server provides an environment used to generate databases that can be accessed from workstations, the Internet, or other media such as a personal digital assistant (PDA).

Microsoft SQL Server 2000 is a full-featured relational database management system (RDBMS) that offers a variety of administrative tools to ease the burdens of database development, maintenance and administration. Six of the more frequently used tools are: Enterprise Manager, Query Analyzer, SQL Profiler, Service Manager, Data Transformation Services and Books Online. Let's take a brief look at each:

- Enterprise Manager is the main administrative console for SQL Server installations. It provides you with a graphical 'birds-eye' view of all of the SQL Server installations on your network. You can perform high-level administrative functions that affect one or more servers, schedule common maintenance tasks or create and modify the structure of individual databases.
- Query Analyzer offers a quick method for performing queries against any of your SQL Server databases. It's a great way to quickly pull information out of a database in response to a user request, test queries before implementing them in other applications, create/ modify stored procedures and execute administrative tasks.
- SQL Profiler provides a window into the inner workings of your database. You can monitor many different event types and observe database performance in real time. SQL Profiler allows you to capture and replay system 'traces' that log various activities. It's a great tool for optimizing databases with performance issues or troubleshooting particular problems.
- Service Manager is used to control the MSSQLServer (the main SQL Server process), MSDTC (Microsoft Distributed Transaction Coordinator) and SQLServerAgent processes. An icon for this service normally resides in the system tray of machines running SQL Server. You can use Service Manager to start, stop or pause any one of these services.
- Data Transformation Services (DTS) provide an extremely flexible method for importing and exporting data between a Microsoft SQL Server installation and a large variety of other formats. The most commonly used DTS application is the 'Import and Export Data' wizard found in the SQL Server program group.
- Books Online is an often overlooked resource provided with SQL Server that contains answers to a variety of administrative, development and installation issues. It's a great resource to consult before turning to the Internet or technical support.

Strength

SQL Server includes an assortment of add-on services which are the strength of this application. While these are not essential for the operation of the database system, they provide value added services on top of the core database management system.

Notes

Some of them are given as under,

(i) **Service Broker:** Used inside an instance, it is used to provide an asynchronous programming environment. For cross instance applications, Service Broker communicates over TCP/IP and allows the different components to be synchronized together, via exchange of messages. The Service Broker, which runs as a part of the database engine, provides a reliable messaging and message queuing platform for SQL Server applications.

(ii) **Replication Services:** SQL Server Replication Services are used by SQL Server to replicate and synchronize database objects, either in entirety or a subset of the objects present, across replication agents, which might be other database servers across the network, or database caches on the client side. Replication follows a publisher/subscriber model i.e., the changes are sent out by one database server ('publisher') and are received by others ('subscribers').

SQL Server supports three different types of replications:

1. Transaction Replication : Each transaction made to the publisher database (master database) is synced out to subscribers, who update their databases with the transaction. Transactional replication synchronizes databases in near real time.

   - Merge replication: Changes made at both the publisher and subscriber databases are tracked, and periodically the changes are synchronized bi-directionally between the publisher and the subscribers. If the same data has been modified differently in both the publisher and the subscriber databases, synchronization will result in a conflict which has to be resolved - either manually or by using pre-defined policies.
   - Snapshot replication: Snapshot replication published a copy of the entire database (the then-snapshot of the data) and replicates out to the subscribers. Further changes to the snapshot are not tracked.

Analysis Services : SQL Server Analysis Services adds OLAP and data mining capabilities for SQL Server databases. The OLAP engine supports MOLAP, ROLAP and HOLAP storage modes for data. Analysis Services supports the XML for Analysis standard as the underlying communication protocol. The cube data can be accessed using MDX and LINQ queries.

Data mining specific functionality is exposed via the DMX query language. Analysis Services includes various algorithms - Decision trees, clustering algorithm, Naive Bayes algorithm, time series analysis, sequence clustering algorithm, linear and logistic regression analysis, and neural networks - for use in data mining.

(iii) **Reporting Services:** SQL Server Reporting Services is a report generation environment for data gathered from SQL Server databases. It is administered via a web interface. Reporting services features a web services interface to support the development of custom reporting applications. Reports are created as RDL files.

Reports can be designed using recent versions of Microsoft Visual Studio (Visual Studio.NET 2003, 2005, and 2008) with Business Intelligence Development Studio, installed or with the included Report Builder.

Once created, RDL files can be rendered in a variety of formats including Excel, PDF, CSV, XML, TIFF (and other image formats), and HTML Web Archive.

(iv) Notification Services: Originally introduced as a post-release add-on for SQL Server 2000, Notification Services was bundled as part of the Microsoft SQL Server platform for the first and only time with SQL Server 2005. S

QL Server Notification Services is a mechanism for generating data-driven notifications, which are sent to Notification Services subscribers. A subscriber registers for a specific event or transaction (which is registered on the database server as a trigger); when the event occurs, Notification Services can use one of three methods to send a message to the subscriber informing about the occurrence of the event.

These methods include SMTP, SOAP, or by writing to a file in the filesystem. Notification Services was discontinued by Microsoft with the release of SQL Server 2008 in August 2008, and is no longer an officially supported component of the SQL Server database platform.

(v) Integration Services : SQL Server Integration Services is used to integrate data from different data sources. It is used for the ETL capabilities for SQL Server for data warehousing needs.

Integration Services includes GUI tools to build data extraction workflows integration various functionality such as extracting data from various sources, querying data, transforming data including aggregating, duplication and merging data, and then loading the transformed data onto other sources, or sending e-mails detailing the status of the operation as defined by the user.

Limitations

Although SQL Server has several strong features for Database creation, security, monitoring etc. but there are a few limitations with SQL Server 2005 as it introduced Database Mirroring, but it was not fully supported until the first Service Pack release (SP1).

In the initial release (RTM) of SQL Server 2005, database mirroring was available, but unsupported. In order to implement database mirroring in the RTM version, one had to apply trace flag 1400 at startup. Database mirroring is a high availability option that provides redundancy and failover capabilities at the database level.

Oracle

Oracle databases are used with many applications, for large and small organizations operating in a variety of sectors. Powerful, secure and reliable database applications can be built with Oracle, as the platform has a range of features for handling both the data itself and user access to it.

Oracle also provides tools for maintenance and developer functions. Oracle is a proprietary database system, and therefore has to be purchased with a commercial license before it can be used.

Notes

Characteristics

An Oracle database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A database server is the key to solving the problems of information management.

In general, a server reliably manages a large amount of data in a multiuser environment so that many users can concurrently access the same data. All this is accomplished while delivering high performance. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

Oracle Database is the first database designed for enterprise grid computing, the most flexible and cost effective way to manage information and applications. Enterprise grid computing creates large pools of industry-standard, modular storage and servers. With this architecture, each new system can be rapidly provisioned from the pool of components. There is no need for peak workloads, because capacity can be easily added or reallocated from the resource pools as needed. The database has logical structures and physical structures. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting the access to logical storage structures.

Oracle is an ORDBMS (Object Relational Database Management System) product. This means that Oracle databases model data in terms of objects and their relationships. Data modeled within an Oracle system is divided into conceptual entities, each of which has a set of attributes. ORDBMS modeling also has some of the characteristics of Object Oriented development, in which application responsibilities are divided between logical objects, each of which has a well-defined set of characteristics and behaviors.

Oracle database architecture models the data for a system as a single unit stored on a database server. This data is managed as a unit, but is designed to allow access by multiple users at any one time, while still maintaining data integrity. This means that even where multiple users have the ability to view the same data items, to update, insert and even remove data, an Oracle system is able to prevent corruption of the data. This is known as transaction management and is a key feature of Oracle systems.

Oracle databases are developed with various features to enhance the efficiency, reliability and security of systems in different organizational environments. Concurrency features in Oracle prevent the same item of data from being changed by more than one user at the same time. Consistency is also a key feature, meaning that when a user accesses a data item, they can be sure it is accurate at the time of access.

Management of user accounts within an Oracle database is also an important feature, as are the many administration tools, which facilitate any necessary maintenance work by database administrators (DBAs).

There are two aspects of Oracle database development: development of the database itself and development of the user interface application for accessing it. Development of a database is carried out through SQL programming, where databases can be created in the form of tables with columns. The columns are for specific data items and have set data types. Oracle databases can also be created through a user interface. The set of database queries required by an application will also be constructed in SQL code.

A database application consists of the database itself, and the application that provides an interface for user access. These interfaces can be developed using many different languages and technologies.

Oracle databases provide APIs (Application Programming Interfaces) to assist developers in some of the more common languages used. Oracle applications can be built in languages such as Java, PHP and PL/SQL among others.

Strength

Oracle includes several software mechanisms to fulfill the following important requirements of an information management system:

- Data concurrency of a multiuser system must be maximized. A primary concern of a multiuser database management system is how to control concurrency, which is the simultaneous access of the same data by many users. Without adequate concurrency controls, data could be updated or changed improperly, compromising data integrity. One way to manage data concurrency is to make each user wait for a turn.
- The goal of a database management system is to reduce that wait so it is either nonexistent or negligible to each user. All data manipulation language statements should proceed with as little interference as possible, and destructive interactions between concurrent transactions must be prevented.
- Destructive interaction is any interaction that incorrectly updates data or incorrectly alters underlying data structures. Neither performance nor data integrity can be sacrificed. Oracle resolves such issues by using various types of locks and a multiversion consistency model. These features are based on the concept of a transaction. It is the application designer's responsibility to ensure that transactions fully exploit these concurrency and consistency features.
- Data must be read and modified in a consistent fashion. The data a user is viewing or changing is not changed (by other users) until the user is finished with the data. Read consistency, as supported by Oracle, does the following:
    1. Guarantees that the set of data seen by a statement is consistent with respect to a single point in time and does not change during statement execution (statement-level read consistency).
    2. Ensures that readers of database data do not wait for writers or other readers of the same data.
    3. Ensures that writers of database data do not wait for readers of the same data.
    4. Ensures that writers only wait for other writers if they attempt to update identical rows in concurrent transactions.
- High performance is required for maximum productivity from the many users of the database system. To manage the multiversion consistency model, Oracle must create a read-consistent set of data when a table is queried (read) and simultaneously updated (written).
- When an update occurs, the original data values changed by the update are recorded

Notes

in the database undo records. As long as this update remains part of an uncommitted transaction, any user that later queries the modified data views the original data values. Oracle uses current information in the system global area and information in the undo records to construct a read-consistent view of a table's data for a query. Only when a transaction is committed are the changes of the transaction made permanent.

- Statements that start after the user's transaction is committed only see the changes made by the committed transaction. The transaction is key to Oracle's strategy for providing read consistency.

  This unit of committed (or uncommitted) SQL statements:

  1. Dictates the start point for read-consistent views generated on behalf of readers
  2. Controls when modified data can be seen by other transactions of the database for reading or updating.

     By default, Oracle guarantees statement-level read consistency. The set of data returned by a single query is consistent with respect to a single point in time. However, in some situations, you might also require transaction-level read consistency. This is the ability to run multiple queries within a single transaction, all of which are read-consistent with respect to the same point in time, so that queries in this transaction do not see the effects of intervening committed transactions.

     If you want to run a number of queries against multiple tables and if you are not doing any updating, you prefer a read-only transaction.

- Oracle also uses locks to control concurrent access to data. When updating information, the data server holds that information with a lock until the update is submitted or committed. Until that happens, no one else can make changes to the locked information. This ensures the data integrity of the system. Oracle provides unique non-escalating row-level locking.Oracle always locks only the row of information being updated. Because Oracle includes the locking information with the actual rows themselves, Oracle can lock an unlimited number of rows so users can work concurrently without unnecessary delays.
- Oracle locking is performed automatically and requires no user action. Implicit locking occurs for SQL statements as necessary, depending on the action requested. Oracle's lock manager automatically locks table data at the row level. By locking table data at the row level, contention for the same data is minimized. Oracle's lock manager maintains several different types of row locks, depending on what type of operation established the lock. The two general types of locks are exclusive locks and share locks. Only one exclusive lock can be placed on a resource (such as a row or a table); however, many share locks can be placed on a single resource. Both exclusive and share locks always allow queries on the locked resource but prohibit other activity on the resource (such as updates and deletes).
- Under some circumstances, a user might want to override default locking. Oracle allows manual override of automatic locking features at both the row level (by first querying for the rows that will be updated in a subsequent statement) and the table level.

Notes

- Oracle Database provides a high degree of self-management - automating routine DBA tasks and reducing complexity of space, memory, and resource administration. Oracle self-managing database features include the following: automatic undo management, dynamic memory management, Oracle-managed files, meantime to recover, free space management, multiple block sizes, and Recovery Manager (RMAN).
- Enterprise Manager is a system management tool that provides an integrated solution for centrally managing your heterogeneous environment. Combining a graphical console, Oracle Management Servers, Oracle Intelligent Agents, common services, and administrative tools, Enterprise Manager provides a comprehensive systems management platform for managing Oracle products.
- Automatic Storage Management automates and simplifies the layout of data files, control files, and log files. Database files are automatically distributed across all available disks, and database storage is rebalanced whenever the storage configuration changes. It provides redundancy through the mirroring of database files, and it improves performance by automatically distributing database files across all available disks. Rebalancing of the database's storage automatically occurs whenever the storage configuration changes.
- Traditionally, the operating systems regulated resource management among the various applications running on a system, including Oracle databases. The Database Resource Manager controls the distribution of resources among various sessions by controlling the execution schedule inside the database. By controlling which sessions run and for how long, the Database Resource Manager can ensure that resource distribution matches the plan directive and hence, the business objectives.
- Oracle provides for complete media recovery from all possible types of hardware failures, including disk failures. Options are provided so that a database can be completely recovered or partially recovered to a specific point in time. If some data files are damaged in a disk failure but most of the database is intact and operational, the database can remain open while the required table spaces are individually recovered. Therefore, undamaged portions of a database are available for normal use while damaged portions are being recovered.
- A data warehouse is a relational database designed for query and analysis rather than for transaction processing. It usually contains historical data derived from transaction data, but it can include data from other sources. It separates analysis workload from transaction workload and enables an organization to consolidate data from several sources. In addition to a relational database, a data warehouse environment includes an extraction, transportation, transformation, and loading (ETL) solution, an online analytical processing (OLAP) engine, client analysis tools, and other applications that manage the process of gathering data and delivering it to business users.
- Oracle has many SQL operations for performing analytic operations in the database. These include ranking, moving averages, cumulative sums, ratio-to-reports, and period-over-period comparisons. racle includes datatypes to handle all the types of rich Internet content such as relational data, object-relational data, XML, text, audio, video, image, and spatial. These datatypes appear as native types in the database. They can all be queried using SQL.

Notes

A single SQL statement can include data belonging to any or all of these datatypes.

- Oracle includes built-in spatial features that let you store, index, and manage location content (assets, buildings, roads, land parcels, sales regions, and so on.) and query location relationships using the power of the database. The Oracle Spatial Option adds advanced spatial features such as linear reference support and coordinate systems.

The Oracle database provides discretionary access control, which is a means of restricting access to information based on privileges. The appropriate privilege must be assigned to a user in order for that user to access a schema object. Appropriately privileged users can grant other users privileges at their discretion. Oracle manages database security using several different facilities:

1. Authentication to validate the identity of the entities using your networks, databases, and applications
2. Authorization processes to limit access and actions, limits that are linked to user's identities and roles.
3. Access restrictions on objects, like tables or rows.
4. Security policies
5. Database auditing

Other characteristics are given as under,

1. Oracle is very simple to install.
2. There is no need to install agents when you are going to install Oracle.

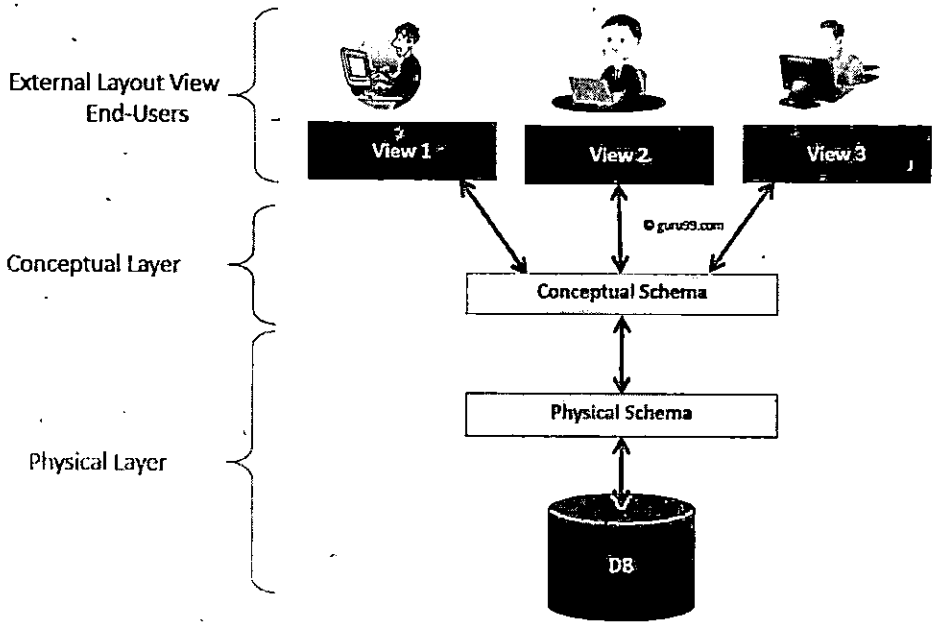
Limitations

- Oracle not provides any kind of GUI.
- There is no direct memory access for fine grained monitoring.
- There is no graphical tool like Excel.
- Users still need indepth knowledge of the Oracle database.
- Users still need a method for performance tuning & analyses.
- Analysis of performance puts extra load on the database.

2.8 PHYSICAL DATABASE DESIGN

The physical design of your database optimizes performance while ensuring data integrity by avoiding unnecessary data redundancies. The task of building the physical design is a job that truly never ends.

As a result, you need to continually monitor the performance and data integrity as time passes. Many factors necessitate periodic refinements to the physical design.



The physical design of your database optimizes performance while ensuring data integrity by avoiding unnecessary data redundancies. During physical design, you transform the entities into tables, the instances into rows, and the attributes into columns.

After completing the logical design of your database, you now move to the physical design. You and your colleagues need to make many decisions that affect the physical design, some of which are listed below.

- How to translate entities into physical tables
- What attributes to use for columns of the physical tables
- Which columns of the tables to define as keys
- What indexes to define on the tables
- What views to define on the tables
- How to denormalize the tables
- How to resolve many-to-many relationships
- What designs can take advantage of hash access

Physical design is the time when you abbreviate the names that you chose during logical design.

For example, you can abbreviate the column name that identifies employees,

EMPLOYEE_NUMBER, to EMPNO. In Db2 for z/OS®,

you need to abbreviate column names and table names to fit the physical constraint of a 30-byte maximum for column names and a 128-byte maximum for table names. For

Notes

more information about the conventions and rules for database object names, see Naming conventions and SQL identifiers.

The task of building the physical design is a job that truly never ends. You need to continually monitor the performance and data integrity characteristics of the database as time passes. Many factors necessitate periodic refinements to the physical design.

Db2 lets you change many of the key attributes of your design with ALTER SQL statements. For example, assume that you design a partitioned table so that it stores 36 months' worth of data. Later you discover that you need to extend that design to 84 months' worth of data. You can add or rotate partitions for the current 36 months to accommodate the new design.

The remainder of this information includes some valuable information that can help you as you build and refine the physical design of your database. However, this task generally requires more experience with Db2 than most readers of this introductory level information are likely to have.

Physical Database Design Process

Physical database design is the process of transforming a data model into the physical data structure of a particular database management system (DBMS). Normally, Physical Design is accomplished in multiple steps, which include expanding a business model into a fully attributed model (FAM) and then transforming the fully attributed model into a physical design model.

Conceptual, Logical, and Physical Data Models

You begin with a summary-level business data model that's most often used on strategic data projects. It typically describes an entire enterprise, which allows you to understand at a high level the different entities in your data and how they relate to one another. Due to its highly abstract nature, it may be referred to as a conceptual model.

Common characteristics of a conceptual data model:

A conceptual data model identifies important entities and the high-level relationships among them. This means no attribute or primary key is specified. Moreover, complexity increases as you expand from a conceptual data model.

A logical data model, otherwise known as a fully attributed data model, allows you to understand the details of your data without worrying about how the data will be implemented in the database. Additionally, a logical data model will normally be derived from and or linked back to objects in a conceptual data model. It is independent of DBMS, technology, data storage or organizational constraints.

Common characteristics of a logical data model:

- Unlike the conceptual model, a logical model includes all entities and relationships among them. Additionally, all attributes, the primary key, and foreign keys (keys identifying the relationship between different entities) are specified. As a result, normalization occurs at this level.

- The steps for designing the logical data model are as follows:
- First, specify primary keys for all entities.
- Then find the relationships between different entities.
- Find all attributes for each entity.
- Lastly, resolve many-to-many relationships.

Finally, the physical data model will show you exactly how to implement your data model in the database of choice. This model shows all table structures, including column name, column data type, column constraints, primary key, foreign key, and relationships between tables.

Correspondingly, the target implementation technology may be a relational DBMS, an XML document, a spreadsheet, or any other data implementation option.

Common characteristics of a physical data model:

- Describes data requirements for a single project or application.
- Specifies all tables and columns.
- Contains foreign keys used to identify relationships between tables.
- Physical considerations may cause the physical data model to be different from the logical data model.

The physical design is where you translate schemas into actual database structures. After that, you transform the entities into tables, instances into rows, and attributes into columns. At this time, you have to map:

- Entity to Table
- Attribute to Column
- Primary Key and Alternate Key to Unique Index
- Index to Non-unique Index
- Foreign Keys to Non-unique Index
- Transformation
- Choosing a physical data structure for the data constructs in the data model.
- Optionally choosing DBMS options for the existence constraints in the data model.
- Does not change the business meaning of the data.
- First transformation of a data model should be a "one to one" transformation.
- Should not denormalize the data unless required for performance.
- Based on shop design standards and DBA experience & biases.

Entity Subsetting – Choosing to transform only a subset of the attributes in an entity.

Dependent Encasement – Collapsing a dependent entity into its parent to form a repeating group of attributes or collapsing a dependent entity into its parent to form a new set of attributes.

Notes

- Category Encasement
- Category Discriminator Collapse
- Horizontal Split
- Vertical Split
- Data Migration

Synthetic Keys – The merging of similar entities using a made up key. Always loses business vocabulary and is never more than BCNF. Also used a lot in packages to make them easily extendable.

Adding Summaries – Adding new entities that are each a summary of data upon a single level of a dimension.

Adding Dimensions

Moreover, it is often necessary to apply multiple transforms to a single entity to get the desired physical performance characteristics. All physical design transformations are compromises.

Database Definition Language (DDL)

Except for data cubes, a one to one translation of the Physical Model into the specific DDL of the database:

- Entity to Table
- Attribute to Column
- Primary Key and Alternate Key to Unique Index
- Index to Non-unique Index
- Foreign Keys to Non-unique Index
- Data Cubes

Data cubes are the exception to the DDL one to one translation because:

- Proprietary physical databases
- Have their own structure definition language
- Have their own data loaders
- Physical Database Design: Watson-Watt's Law of Third Best
- Best never comes.
- Second Best takes too long.

Identify the Third Best – the design that can be validated in time to meet an identified.

In conclusion, physical database design is no easy feat. If you feel intimidated by it, you're not the only one. Learn more about Relational Junction for data warehousing and integration or contact a member of our team for more information!

2.9 DECOMPOSITION OF A RELATION SCHEMES

The process of breaking up or dividing a single relation into two or more sub relations is called as decomposition of a relation.

Properties of Decomposition

The following two properties must be followed when decomposing a given relation1. Lossless decompositionLossless decomposition ensures-

- No information is lost from the original relation during decomposition.
- When the sub relations are joined back, the same relation is obtained that was decomposed.

Every decomposition must always be lossless.

Types of Decomposition

Decomposition of a relation can be completed in the following two ways-

1. Lossless Join Decomposition-

- Consider there is a relation R which is decomposed into sub relations R1 , R2 , ... , Rn.
- This decomposition is called lossless join decomposition when the join of the sub relations results in the same relation R that was decomposed.
- For lossless join decomposition, we always have-

R1 ⋈ R2 ⋈ R3 ..... ⋈ Rn = R

where ⋈ is a natural join operator

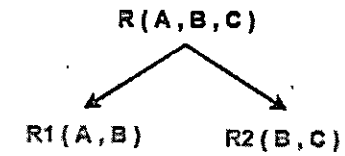
Example-

Consider the following relation R(A , B , C)-

A	B	C
1	2	1
2	5	3
3	3	3

Consider this relation is decomposed into two sub relations R1(A , B) and R2(B , C)-

Notes



The two sub relations are-

A	B
1	2
2	5
3	3

$R_1(A, B)$

B	C
2	1
5	3
3	3

$R_2(B, C)$

Now, let us check whether this decomposition is lossless or not. For lossless decomposition, we must have- R1 ⋈ R2 = R.

Now, if we perform the natural join (⋈) of the sub relations R1 and R2 , we get

A	B	C
1	2	1
2	5	3
3	3	3

This relation is same as the original relation R. Thus, we conclude that the above decomposition is lossless join decomposition.

NOTE:

Lossless join decomposition is also known as non-additive join decomposition.

- This is because the resultant relation after joining the sub relations is same as the decomposed relation.
- No extraneous tuples appear after joining of the sub-relations.

2. *Lossy Join Decomposition-*

- Consider there is a relation R which is decomposed into sub relations R1 , R2 , ... , Rn.
- This decomposition is called lossy join decomposition when the join of the sub relations does not result in the same relation R that was decomposed.
- The natural join of the sub relations is always found to have some extraneous tuples.
- For lossy join decomposition, we always have-

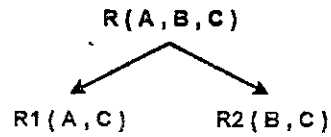
$$R1 \bowtie R2 \bowtie R3 \ldots\ldots \bowtie Rn \supset R$$

where ⋈ is a natural join operator

Example-

Consider the above relation R(A , B , C)-

Consider this relation is decomposed into two sub relations as R1(A , C) and R2(B , C)-



The two sub relations are-

A	C
1	1
2	3
3	3

R_1(A , B)

B	C
2	1
5	3
3	3

R_2(B , C)

Now, let us check whether this decomposition is lossy or not.

For lossy decomposition, we must have$R1 \bowtie R2 \bowtie R$

Now, if we perform the natural join ($\bowtie$) of the sub relations R1 and R2 we get

A	B	C
1	2	1
2	5	3
2	3	3
3	5	3
3	3	3

This relation is not same as the original relation R and contains some extraneous tuples.

Clearly, $R1 \bowtie R2 \bowtie R$. Thus, we conclude that the above decomposition is lossy join decomposition.

NOTE:

- Lossy join decomposition is also known as careless decomposition.
- This is because extraneous tuples get introduced in the natural join of the sub-relations.
- Extraneous tuples make the identification of the original tuples difficult.

Determining Whether Decomposition Is Lossless Or Lossy-

Consider a relation R is decomposed into two sub relations R1 and R2. Then,

- If all the following conditions satisfy, then the decomposition is lossless.
- If any of these conditions fail, then the decomposition is lossy.

Notes

Condition-01:

Union of both the sub relations must contain all the attributes that are present in the original relation R. Thus,

$$R_1 \cup R_2 = R$$

Condition-02:

Intersection of both the sub relations must not be null. In other words, there must be some common attribute which is present in both the sub relations. Thus,

$$R_1 \cap R_2 \neq \emptyset$$

Condition-03:

Intersection of both the sub relations must be a super key of either R1 or R2 or both. Thus,

$$R_1 \cap R_2 = \text{Super key of } R_1 \text{ or } R_2$$

2.10 INTRODUCTION TO DATA MINING AND DATA WAREHOUSING

Data mining is the process of determining data patterns. A data warehouse is a database system designed for analytics. Data mining is generally considered as the process of extracting useful data from a large set of data. Data warehousing is the process of combining all the relevant data.

Data Mining Vs Data Warehousing

Data warehouse refers to the process of compiling and organizing data into one common database, whereas data mining refers to the process of extracting useful data from the databases. The data mining process depends on the data compiled in the data warehousing phase to recognize meaningful patterns. A data warehousing is created to support management systems.

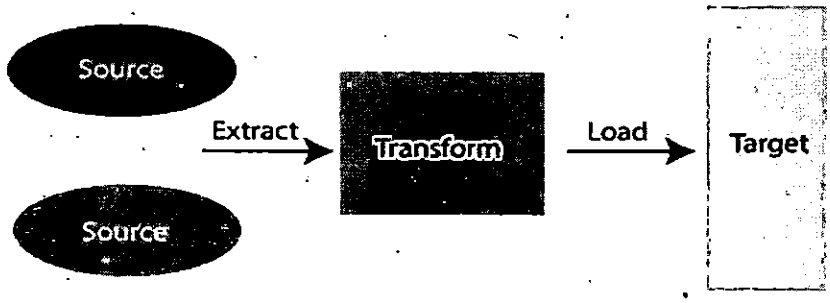
Data Warehouse

A Data Warehouse refers to a place where data can be stored for useful mining. It is like a quick computer system with exceptionally huge data storage capacity. Data from the various organization's systems are copied to the Warehouse, where it can be fetched and conformed to delete errors.

Here, advanced requests can be made against the warehouse storage of data.

Data warehouse combines data from numerous sources which ensure the data quality, accuracy, and consistency. Data warehouse boosts system execution by separating analytics processing from transnational databases.

Data flows into a data warehouse from different databases. A data warehouse works by sorting out data into a pattern that depicts the format and types of data. Query tools examine the data tables using patterns.



Data Warehousing Process

Data warehouses and databases both are relative data systems, but both are made to serve different purposes. A data warehouse is built to store a huge amount of historical data and empowers fast requests over all the data, typically using Online Analytical Processing (OLAP).

A database is made to store current transactions and allow quick access to specific transactions for ongoing business processes, commonly known as Online Transaction Processing (OLTP).

Important Features of Data Warehouse

The Important features of Data Warehouse are given below:

1. Subject Oriented

A data warehouse is subject-oriented. It provides useful data about a subject instead of the company's ongoing operations, and these subjects can be customers, suppliers, marketing, product, promotion, etc. A data warehouse usually focuses on modeling and analysis of data that helps the business organization to make data-driven decisions.

2. Time-Variant:

The different data present in the data warehouse provides information for a specific period.

3. Integrated

A data warehouse is built by joining data from heterogeneous sources, such as social databases, level documents, etc.

4. Non- Volatile

It means, once data entered into the warehouse cannot be change.

Notes

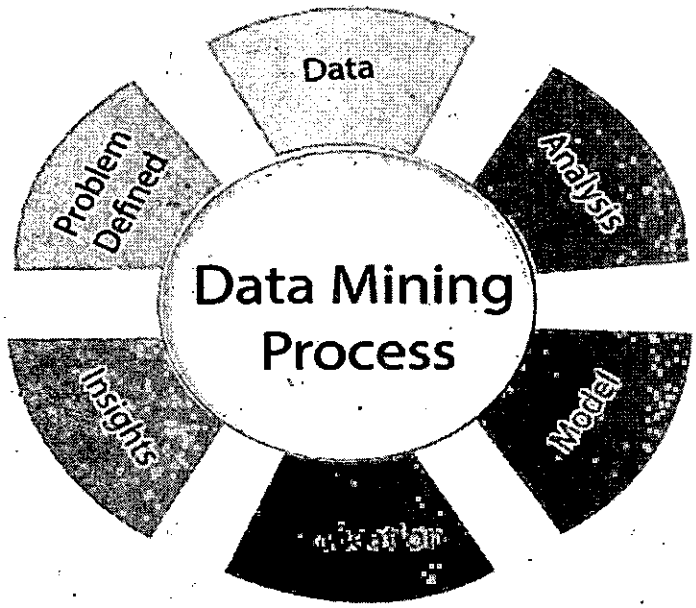
Advantages of Data Warehouse

- More accurate data access
- Improved productivity and performance
- Cost-efficient
- Consistent and quality data

Data Mining

Data mining refers to the analysis of data. It is the computer-supported process of analyzing huge sets of data that have either been compiled by computer systems or have been downloaded into the computer.

In the data mining process, the computer analyzes the data and extract useful information from it. It looks for hidden patterns within the data set and try to predict future behavior. Data mining is primarily used to discover and indicate relationships among the data sets.



Data mining aims to enable business organizations to view business behaviors, trends relationships that allow the business to make data-driven decisions. It is also known as knowledge Discover in Database (KDD). Data mining tools utilize AI, statistics, databases, and machine learning systems to discover the relationship between the data. Data mining tools can support business-related questions that traditionally time-consuming to resolve any issue.

Important features of Data Mining:

The important features of Data Mining are given below:

- It utilizes the Automated discovery of patterns.
- It predicts the expected results.

- It focuses on large data sets and databases
- It creates actionable information.

Advantages of Data Mining

i. Market Analysis:

Data Mining can predict the market that helps the business to make the decision. For example, it predicts who is keen to purchase what type of products.

ii. Fraud detection:

Data Mining methods can help to find which cellular phone calls, insurance claims, credit, or debit card purchases are going to be fraudulent.

iii. Financial Market Analysis:

Data Mining techniques are widely used to help Model Financial Market

iv. Trend Analysis:

Analyzing the current existing trend in the marketplace is a strategic benefit because it helps in cost reduction and manufacturing process as per market demand.

Differences between Data Mining and Data Warehousing

Data Mining	Data Warehousing
Data mining is the process of determining data patterns.	A data warehouse is a database system designed for analytics.
Data mining is generally considered as the process of extracting useful data from a large set of data.	Data warehousing is the process of combining all the relevant data.
Business entrepreneurs carry data mining with the help of engineers.	Data warehousing is entirely carried out by the engineers.
In data mining, data is analyzed repeatedly.	In data warehousing, data is stored periodically.
Data mining uses pattern recognition techniques to identify patterns.	Data warehousing is the process of extracting and storing data that allow easier reporting.
One of the most amazing data mining technique is the detection and identification of the unwanted errors that occur in the system.	One of the advantages of the data warehouse is its ability to update frequently. That is the reason why it is ideal for business entrepreneurs who want up to date with the latest stuff.
The data mining techniques are cost-efficient as compared to other statistical data applications.	The responsibility of the data warehouse is to simplify every type of business data.

Notes

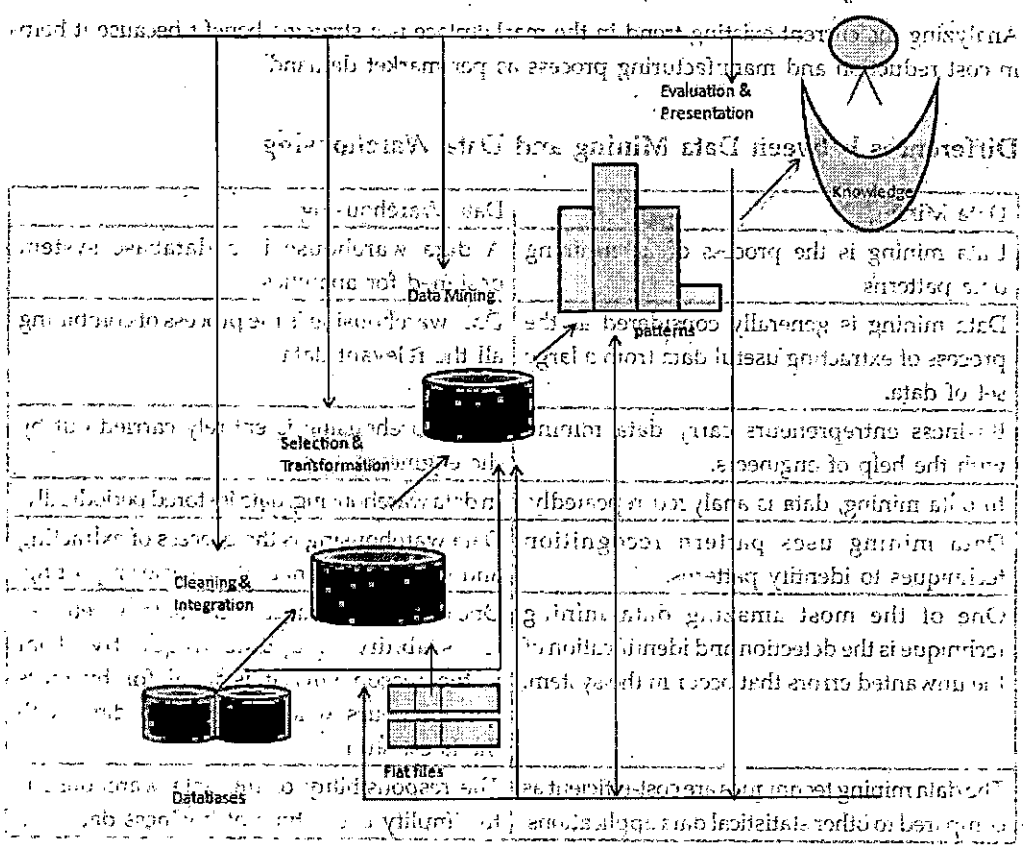
The data mining techniques are not 100 percent accurate. It may lead to serious consequences in a certain condition.	In the data warehouse, there is a high possibility that the data required for analysis by the company may not be integrated into the warehouse. It can simply lead to loss of data.
Companies can benefit from this analytical tool by equipping suitable and accessible knowledge-based data.	Data warehouse stores a huge amount of historical data that helps users to analyze different periods and trends to make future predictions.

2.11 KNOWLEDGE EXTRACTION THROUGH DATA MINING

Knowledge Extraction through Data Mining

Some people don't differentiate data mining from knowledge discovery while others view data mining as an essential step in the process of knowledge discovery. Here is the list of steps invoced in the knowledge discovery process:

The following diagram shows the process of knowledge discovery



Why we need Data Mining?

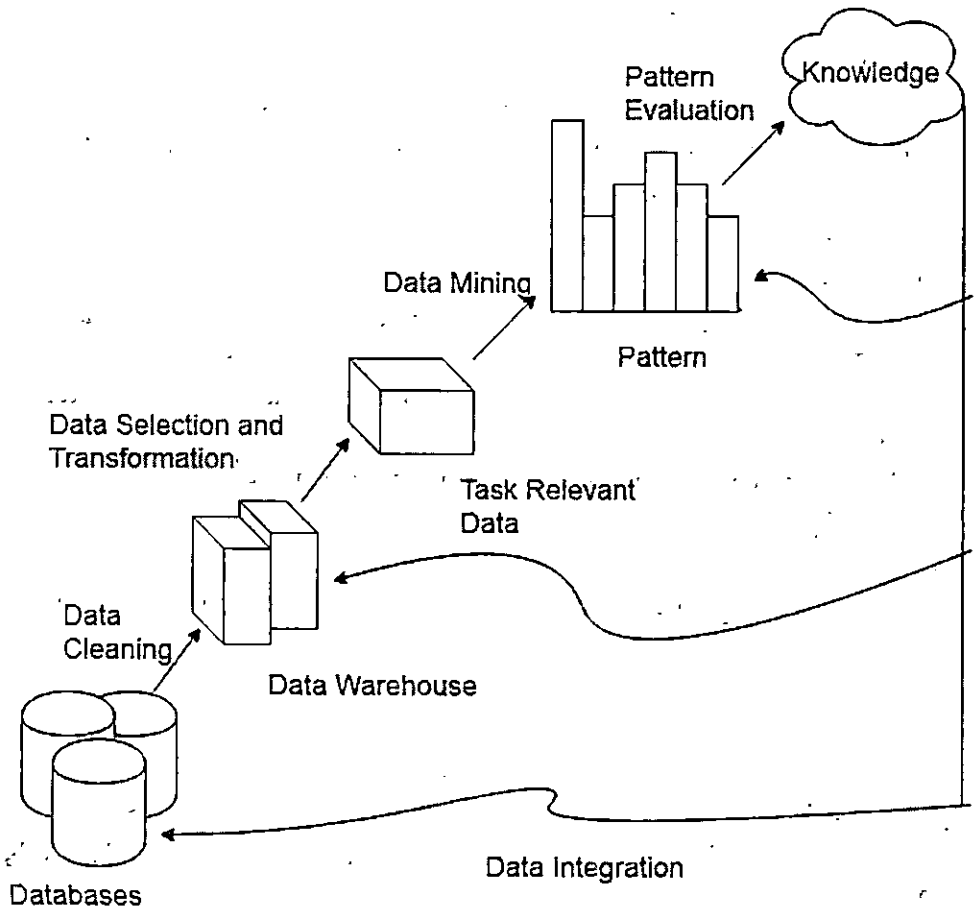
Volume of information is increasing everyday that we can handle from business transactions, scientific data, sensor data, Pictures, videos, etc. So, we need a system that will be capable of extracting essence of information available and that can automatically generate report, views or summary of data for better decision-making.

Why Data Mining is used in Business?

Data mining is used in business to make better managerial decisions by:

- Automatic summarization of data
- Extracting essence of information stored.
- Discovering patterns in raw data.
- Data Mining also known as Knowledge Discovery in Databases, refers to the nontrivial extraction of implicit, previously unknown and potentially useful information from data stored in databases.

Steps Involved in KDD Process:



Notes

Data Cleaning:

- Data cleaning is defined as removal of noisy and irrelevant data from collection.
- Cleaning in case of Missing values.
- Cleaning noisy data, where noise is a random or variance error.
- Cleaning with Data discrepancy detection and Data transformation tools.

Data Integration:

- Data integration is defined as heterogeneous data from multiple sources combined in a common source(DataWarehouse).
- Data integration using Data Migration tools.
- Data integration using Data Synchronization tools.
- Data integration using ETL(Extract-Load-Transformation) process.

Data Selection:

- Data selection is defined as the process where data relevant to the analysis is decided and retrieved from the data collection.
- Data selection using Neural network.
- Data selection using Decision Trees.
- Data selection using Naive bayes.
- Data selection using Clustering, Regression, etc.

Data Transformation:

Data Transformation is defined as the process of transforming data into appropriate form required by mining procedure. Data Transformation is a two step process:

- Data Mapping: Assigning elements from source base to destination to capture transformations.
- Code generation: Creation of the actual transformation program.

Data Mining

Data mining is defined as clever techniques that are applied to extract patterns potentially useful.

- Transforms task relevant data into patterns.
- Decides purpose of model using classification or characterization.

Pattern Evaluation

Pattern Evaluation is defined as identifying strictly increasing patterns representing knowledge based on given measures.

- Find interestingness score of each pattern.

- Uses summarization and Visualization to make data understandable by user.

Knowledge representation

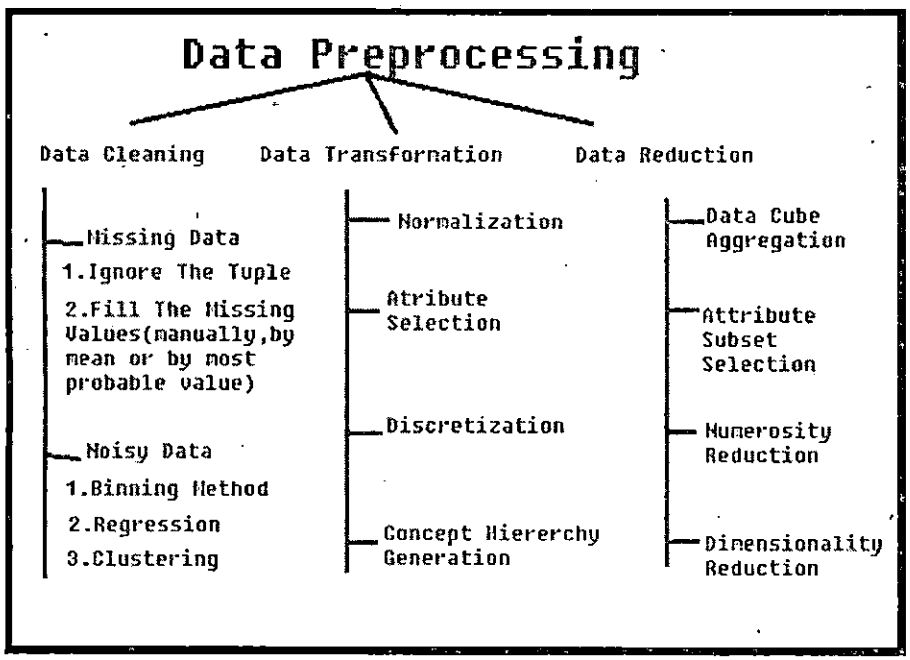
- Knowledge representation is defined as technique which utilizes visualization tools to represent data mining results.
- Generate reports.
- Generate tables.
- Generate discriminant rules, classification rules, characterization rules, etc.

Note:

KDD is an iterative process where evaluation measures can be enhanced, mining can be refined, new data can be integrated and transformed in order to get different and more appropriate results. Preprocessing of databases consists of Data cleaning and Data Integration.

Preprocessing in Data Mining:

Data preprocessing is a data mining technique which is used to transform the raw data in a useful and efficient format.



Steps Involved in Data Preprocessing:

1. Data Cleaning:

The data can have many irrelevant and missing parts. To handle this part, data cleaning is done. It involves handling of missing data, noisy data etc.

Notes

(a). Missing Data:

This situation arises when some data is missing in the data. It can be handled in various ways. Some of them are:

Ignore the tuples: This approach is suitable only when the dataset we have is quite large and multiple values are missing within a tuple.

Fill the Missing values: There are various ways to do this task. You can choose to fill the missing values manually, by attribute mean or the most probable value.

(b). Noisy Data:

Noisy data is a meaningless data that can't be interpreted by machines.It can be generated due to faulty data collection, data entry errors etc. It can be handled in following ways:

Binning Method: This method works on sorted data in order to smooth it. The whole data is divided into segments of equal size and then various methods are performed to complete the task. Each segmented is handled separately. One can replace all data in a segment by its mean or boundary values can be used to complete the task.

Regression: Here data can be made smooth by fitting it to a regression function.The regression used may be linear (having one independent variable) or multiple (having multiple independent variables).

Clustering: This approach groups the similar data in a cluster. The outliers may be undetected or it will fall outside the clusters.

2. Data Transformation:

This step is taken in order to transform the data in appropriate forms suitable for mining process. This involves following ways:

Normalization: It is done in order to scale the data values in a specified range (-1.0 to 1.0 or 0.0 to 1.0)

Attribute Selection: In this strategy, new attributes are constructed from the given set of attributes to help the mining process.

Discretization: This is done to replace the raw values of numeric attribute by interval levels or conceptual levels.

Concept Hierarchy Generation: Here attributes are converted from lower level to higher level in hierarchy. For Example-The attribute "city" can be converted to "country".

3. Data Reduction:

Since data mining is a technique that is used to handle huge amount of data. While working with huge volume of data, analysis became harder in such cases. In order to get rid of this, we uses data reduction technique. It aims to increase the storage efficiency and reduce data storage and analysis costs.

Notes

The various steps to data reduction are:

- Data Cube Aggregation: Aggregation operation is applied to data for the construction of the data cube.
- Attribute Subset Selection: The highly relevant attributes should be used, rest all can be discarded. For performing attribute selection, one can use level of significance and p-value of the attribute.the attribute having p-value greater than significance level can be discarded.
- Numerosity Reduction: This enable to store the model of data instead of whole data, for example: Regression Models.
- Dimensionality Reduction: This reduce the size of data by encoding mechanisms.It can be lossy or lossless. If after reconstruction from compressed data, original data can be retrieved, such reduction are called lossless reduction else it is called lossy reduction. The two effective methods of dimensionality reduction are:Wavelet transforms and PCA (Principal Component Analysis).

Data Integration in Data Mining

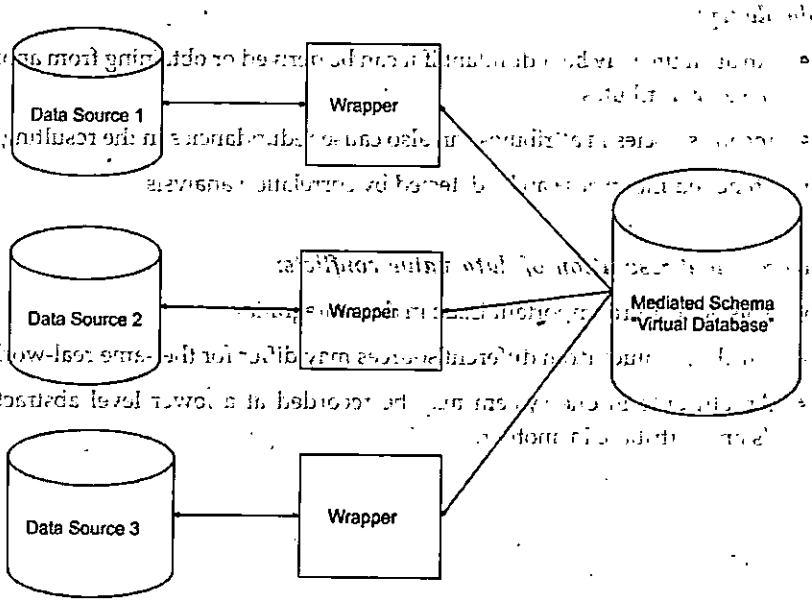
Data Integration is a data preprocessing technique that involves combining data from multiple heterogeneous data sources into a coherent data store and provide a unified view of the data. These sources may include multiple data cubes, databases, or flat files.

The data integration approaches are formally defined as triple <G, S, M> where,

G stand for the global schema,

S stands for the heterogeneous source of schema,

M stands for mapping between the queries of source and global schema.



Notes

There are mainly 2 major approaches for data integration - one is the "tight coupling approach" and another is the "loose coupling approach".

Tight Coupling:

Here, a data warehouse is treated as an information retrieval component.

In this coupling, data is combined from different sources into a single physical location through the process of ETL - Extraction, Transformation, and Loading.

Loose Coupling:

Here, an interface is provided that takes the query from the user, transforms it in a way the source database can understand, and then sends the query directly to the source databases to obtain the result.

And the data only remains in the actual source databases.

Issues in Data Integration:

There are no issues to consider during data integration: Schema Integration, Redundancy, Detection, and resolution of data value conflicts.

These are explained in brief below.

1. Schema Integration:

- Integrate metadata from different sources.
- The real-world entities from multiple sources are matched referred to as the entity identification problem.

2. Redundancy:

- An attribute may be redundant if it can be derived or obtaining from another attribute or set of attributes.
- Inconsistencies in attributes can also cause redundancies in the resulting data set.
- Some redundancies can be detected by correlation analysis.

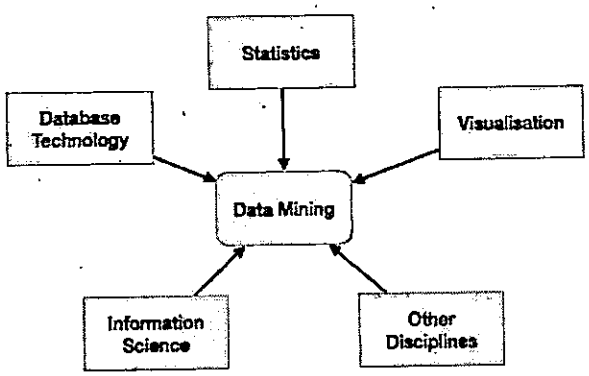
3. Detection and resolution of data value conflicts:

- This is the third important issue in data integration.
- Attribute values from different sources may differ for the same real-world entity.
- An attribute in one system may be recorded at a lower level abstraction than the "same" attribute in another.

Data Mining Classification

Data Mining System Classification

A data mining system can be classified according to the following criteria:



- Database Technology
- Statistics
- Machine Learning
- Information Science
- Visualization
- Other Disciplines

Apart from these, a data mining system can also be classified based on the kind of (a) databases mined, (b) knowledge mined, (c) techniques utilized, and (d) applications adapted.

Classification Based on the Databases Mined

We can classify a data mining system according to the kind of databases mined. Database system can be classified according to different criteria such as data models, types of data, etc. And the data mining system can be classified accordingly.

For example, if we classify a database according to the data model, then we may have a relational, transactional, object-relational, or data warehouse mining system.

Classification Based on the kind of Knowledge Mined

We can classify a data mining system according to the kind of knowledge mined. It means the data mining system is classified on the basis of functionalities such as:

- Characterization
- Discrimination
- Association and Correlation Analysis
- Classification
- Prediction

Notes

- Outlier Analysis
- Evolution Analysis

Classification Based on the Techniques Utilized

We can classify a data mining system according to the kind of techniques used. We can describe these techniques according to the degree of user interaction involved or the methods of analysis employed.

Classification Based on the Applications Adapted

We can classify a data mining system according to the applications adapted. These applications are as follows:

- Finance
- Telecommunications
- DNA
- Stock Markets
- E-mail

Integrating a Data Mining System with a DB/DW System

If a data mining system is not integrated with a database or a data warehouse system, then there will be no system to communicate with. This scheme is known as the non-coupling scheme.

In this scheme, the main focus is on data mining design and on developing efficient and effective algorithms for mining the available data sets.

The list of Integration Schemes is as follows:

- No Coupling – In this scheme, the data mining system does not utilize any of the database or data warehouse functions. It fetches the data from a particular source and processes that data using some data mining algorithms. The data mining result is stored in another file.
- Loose Coupling – In this scheme, the data mining system may use some of the functions of database and data warehouse system. It fetches the data from the data respiratory managed by these systems and performs data mining on that data. It then stores the mining result either in a file or in a designated place in a database or in a data warehouse.
- Semi–tight Coupling – In this scheme, the data mining system is linked with a database or a data warehouse system and in addition to that, efficient implementations of a few data mining primitives can be provided in the database.
- Tight coupling – In this coupling scheme, the data mining system is smoothly integrated into the database or data warehouse system. The data mining subsystem is treated as one functional component of an information system.

Notes

Why we need Data Mining?

Volume of information is increasing everyday that we can handle from business transactions, scientific data, sensor data, Pictures, videos, etc.

So, we need a system that will be capable of extracting essence of information available and that can automatically generate report, views or summary of data for better decision-making.

Why Data Mining is used in Business?

Data mining is used in business to make better managerial decisions by:

- Automatic summarization of data
- Extracting essence of information stored.
- Discovering patterns in raw data.

SUMMARY

- The goal of a relational database design is to generate a set of relation schema that allows us to store information without unnecessary redundancy and also to retrieve information easily.
- A database system is an integrated collection of related files, along with details of interpretation of the data contained therein. DBMS is a s/w system that allows access to data contained in a database. The objective of the DBMS is to provide a convenient and effective method of defining, storing and retrieving the information contained in the database.
- RDBMS stands for Relational Database Management System. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
- A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd. The data in RDBMS is stored in database objects called tables. The table is a collection of related data entries and it consists of columns and rows. Every table is broken up into smaller entities called fields. The fields in the EMPLOYEE table consist of ID, NAME, AGE, ADDRESS and SALARY.
- A field is a column in a table that is designed to maintain specific information about every record in the table. A record, also called a row of data, is each individual entry that exists in

Notes

a table.A record is a horizontal entity in a table.A column is a vertical entity in a table that contains all information associated with a specific field in a table.

- ANULL value in a table is a value in a field that appears to be blank which meansA field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.Afield with a NULL value is one that has been left blank during record creation.
- Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.
- A 'Relation' is a two-dimensional table. It consists of 'rows' which represent records and 'columns' which show the attributes of the entity. A relation is also called a file, it consists of a number of records which are also called as tuples. Record consists of a number of attributes which are also known as fields or domains. Normalization is a technique that is more applicable to record-based data models for e.g. a relational database model. Each of the processes can be carried out independently to arrive at normalized tables (depending on how detailed the decompositions is). Normalization is the bottom-up approach of logical database design. It is a step-by-step decomposition of complex records into simple records.
- Normalization reduces redundancy. Redundancy can lead to inconsistencies as well as Insertion, Updation and Deletion anomalies. The different stages of normalization are known as 'normal forms'. The most important and widely used of these are: 1NF, 2NF, 3NF, BCNF, 4NF etc. A relation is said to be in first normal form if all attributes defined on domains containing atomic values.
- A database is a collection of data that is related to a particular topic or purpose. A database management system (DBMS) is a system that stores and retrieves information in a database. A DBMS stores data in a table where the entries are filed under a specific category and are properly indexed. A DBMS that is based on relational model is called as RDBMS. Relational model is based on the theory of sets and relations of mathematics. Microsoft SQL Server is a relational database server which is developed by Microsoft. MS SQL Server is a software product whose primary function is to tore and retrieve data as requested by other software applications from networks or other systems.
- Microsoft SQL Server 2000 is a full-featured relational database management system (RDBMS) that offers a variety of administrative tools to ease the burdens of database development, maintenance and administration. An Oracle database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. Oracle Database is the first database designed for enterprise grid computing, the most flexible and cost effective way to manage information and applications.
- Oracle is an ORDBMS (Object Relational Database Management System) product. This means that Oracle databases model data in terms of objects and their relationships. Data modeled within an Oracle system is divided into conceptual entities, each of which has a set of attributes

KEY WORDS

- **Application Server** - A server that processes application-specific database operations made from application client programs. The DBMS is in-process with the application code for very fast internal access.
- **Aperiodic Server** - Software that is specific to a particular embedded system. Such application-specific code is generally built on a layered architecture of reusable components, such as a real-time operating system and network protocol stack or other middleware. If there is no such architecture, then this term may not be used. The application software is unlikely to be reusable across embedded platforms, simply because each embedded system has a different application.
- **Atomicity** - The property of a transaction that guarantees that either all or none of the changes made by the transaction are written to the database.
- **AVL-Tree** - An AVL tree is a self balancing binary search tree.
- **Big-Endian** - A data representation for a multibyte value that has the most significant byte stored at the lowest memory address. Note that only the bytes are reordered, never the nibbles or bits that comprise them. Every processor stores its data in either big-endian or little-endian format. Sun's SPARC, Motorola's 68k, and the PowerPC families are all big-endian. The Java virtual machine is big-endian as well. Similarly, every communications protocol must define the byte order of its multibyte values. TCP/IP uses big-endian representation.
- **BLOB** - An abbreviation for Binary Large OBject. In SQL, BLOB can be a general term for any data of type long varbinary, long varchar, or long wvarchar. It is also a specific term (and synonym) for data of type long varbinary.
- **Breakpoint** - A location in a program at which execution is to be stopped and control of the processor switched to the debugger. Mechanisms for creating and removing breakpoints are provided by most debugging tools.
- **B-tree** - An indexing method in which the values of the columns used in the index are efficiently maintained in sorted order that also provides fast access (three or four additional disk accesses) to an individual index entry.
- **Cache** - The computer memory that is set aside to contain a portion of the database data that has most recently been accessed by the database application program. A cache is used to minimize the amount of physical disk I/O performed by the DBMS.

REVIEW QUESTIONS

1. Explain different type of integrity constraints with example.
2. What do you meant by database integrity?
3. What are the advantages and limitation of triggers?

Notes

4. How are the nulls represented in database system?
5. What are the situations where you can use nulls?
6. What is Normalization? Discuss its need.
7. What are the design
8. What do you understand by 'Relational Database Management System'? How it is differ from DBMS?
9. Write a short note on Microsoft SQL Server with discussion of its strengths.
10. How Oracle is differing from other Relational Database Designs? Discuss some features of Oracle
11. What is decomposition of relation schemes?
12. What is the knowledge extraction through data mining? Expain.

FURTHER READINGS

1. "database, n". OED Online. Oxford University Press. June 2013. Retrieved July 12, 2013. (Subscription required.)
2. IBM Corporation (October 2013). "IBM Information Management System (IMS) 13 Transaction and Database Servers delivers high performance and low total cost of ownership". Retrieved Feb 20, 2014.
3. "Structured Query Language (SQL)". International Business Machines. October 27, 2006. Retrieved 2007-06-10. Wagner 2010.
4. Ramalho, J.C., Faria, L., Helder, S., & Coutada, M. (2013, December 31). Database Preservation Toolkit: A flexible tool to normalize and give access to databases. University of Minho. https://core.ac.uk/display/55635702?
5. Kroenke, David M. and Dayid J. Auer. Database Concepts. 3rd ed. New York: Prentice, 2007.
6. Raghu Ramakrishnan and Johannes Gehrke, Database Management Systems
7. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, Database System Concepts

Unit 3

Structured Query Language

Structure

Notes

Notes

3.0 LEARNING OBJECTIVES

After reading this chapter students will be able to:

- Describe the various types of SQL
- Describe the different data types and use of SQL
- Describe the DML, DML, DQL and DCL commands of SQL
- Describe the data administrations and TCL commands of SQL
- Describe the various queries of SQL to manipulate data

3.1 INTRODUCTION

Structured Query Language (SQL) is a programming language that is typically used in relational database or data stream management systems. It was developed by IBM in the early 1970s and is now an official standard recognized by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO).

SQL is not a complete programming language, but depends on the I/O and control facilities of a host language. It is both a dejure and a de facto standard. ANSI (American Nationa Standards Institute) has published three generations of SQL, as has ISO (International Organization for Standardization). X/Open, a consortium, has also published an SQL specification. Chamberlin and Boyce (1974) published the first paper on what became SQL, based on Codd's mathematical foundation for logical representation and manipulation of data (Codd 1974). In 1978, ANSI began to standardize a data definition language for the network database language then being designed by CODASYL; Technical Committee X3H2 was formed for this project, which soon evolved to encompass the entire network database language, published in 1986 as Databae Language NDL. X3H2 recognized the importance of the relational model and intitated a project based on Chamberlin's work. In cooperation with the corresponding ISO group, the SQL specification was developed and published in 1986. SQL-86 omitted support for referential integrity, but SQL-89 added basic referential integrity.

SQL-86 and SQL-89 were rightly criticiezed as inadequate for real applications. In 1922, a major new version, SQL-92, was published, containing features that allowed signifcant applications without vendor extensions (ANSI 1992 ; ISO 1992).

SQL has proved key in the success of relational database management systems and is central to many areas, ragning from traditional MIS applications to scientific reserach. The fourth generation of SQL is currently being prepared. SQL 3 adds significant new facilities, including support for object tehcnology, and is partioned into several parts that can progress independently. See Melton and Simon (1993) for a comprehensive introduction to the SQL language.

3.2 WHAT IS SQL?

Structured Query Language (SQL) is the standard language used to communicate with a relational database. The prototype was originally developed by IBM using Dr. E.F. Codd's paper ("A Relational Model of Data for Large Shared Data Banks") as a model. In 1979,

not long after IBM's prototype, the first SQL product, ORACLE, was released by Relational Software, Incorporated (which was later renamed Oracle Corporation). Today it is one of the distinguished leaders in relational database technologies.

The American National Standards Institute (ANSI) is an organization that approves certain standards in many different industries. SQL has been deemed the standard language in relational database communication, originally approved in 1986 based on IBM's implementation. In 1987, the ANSI SQL standard was accepted as the international standard by the International Standards Organization (ISO). The standard was revised again in 1992 (SQL-92) and once again in 1999 (SQL-99). The newest standard is now called SQL-2008, which was officially adopted in July of 2008.

Characteristics of SQL

- SQL databases tend to be mysterious when it comes to the information that is stored within them.
- The SQL Database Engine is the core service for storing, processing, and extracting data.
- The SQL Database Engine provides access and rapid transaction processing to meet the requirements of applications.
- The SQL Database Engine can be used to create relational databases for online transaction processing or online analytical processing data. It is likely that you have a number of applications that put information into your various SQL database engines. From this information set, you need to be able to retrieve meaningful information and that is where this course comes in.

3.3 TYPES OF SQL

Basically, there are two types of DBMSs: relational and non-relational, also referred to as SQL and NoSQL respectively.

The Relational Database

A relational database is a database divided into logical units called tables, where tables are related to one another within the database. A relational database allows data to be broken down into logical, smaller, manageable units, enabling easier maintenance and providing more optimal database performance according to the level of organization. In Figure 1, you can see that tables are related to one another through a common key (data value) in a relational database.

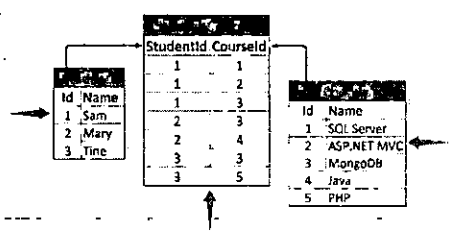


Figure 1. The relational database

Notes

Again, tables are related in a relational database, allowing adequate data to be retrieved in a single query (although the desired data may exist in more than one table). By having common keys, or fields, among relational database tables, data from multiple tables can be joined to form one large set of data. As you venture deeper into this book, you see more of a rela- tional database's advantages, including overall performance and easy data access.

Client/Server Technology

In the past, the computer industry was predominately ruled by mainframe computers—large, powerful systems capable of high storage capacity and high data processing capabilities. Users communicated with the mainframe through dumb terminals—terminals that did not think on their own but relied solely on the mainframe's CPU, storage, and memory. Each terminal had a data line attached to the mainframe. The mainframe environment definitely served its purpose and does today in many businesses, but a greater technology was soon to be introduced: the client/server model.

In the client/server system, the main computer, called the server, is accessible from a network—typically a local area network (LAN) or a wide area network (WAN). The server is normally accessed by personal computers (PCs) or by other servers, instead of dumb terminals. Each PC, called a client, is provided access to the network, allowing communication between the client and the server, thus explaining the name client/server. The main difference between client/server and mainframe environments is that the user's PC in a client/server environment is capable of thinking on its own, capable of running its own processes using its own CPU and memory, but readily accessible to a server computer through a network.

In most cases, a client/server system is much more flexible for today's overall business needs and is much preferred. Modern database systems reside on various types of computer systems with various operating systems. The most common types of operating systems are Windows-based systems, Linux, and command- line systems such as UNIX. Databases reside mainly in client/server and web environments. A lack of training and experience is the main reason for failed implementations of database systems.

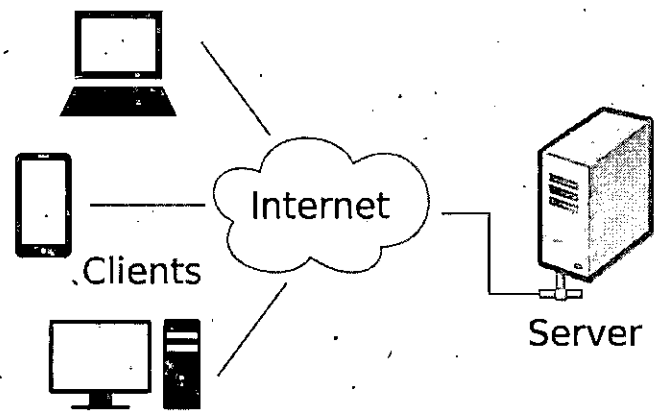


Figure 2. The client/server model.

Nevertheless, an understanding of the client/server model and web-based systems, which will be explained in the next section, is imperative with the rising (and sometimes unreasonable) demands placed on today's businesses as well as the development of Internet technologies and network computing. Figure 2 illustrates the concept of client/server technology.

Web-Based Database System

Business information systems are moving toward web integration. Databases are now accessible through the Internet, meaning that customers' access to an organization's information is enabled through an Internet browser such as Internet Explorer or Firefox. Customers (users of data) are able to order merchandise, check on inventories, check on the status of orders, make administrative changes to accounts, transfer money from one account to another, and so forth. A customer simply invokes an Internet browser, goes to the organization's website, logs in (if required by the organization), and uses an application built into the organization's web page to access data.

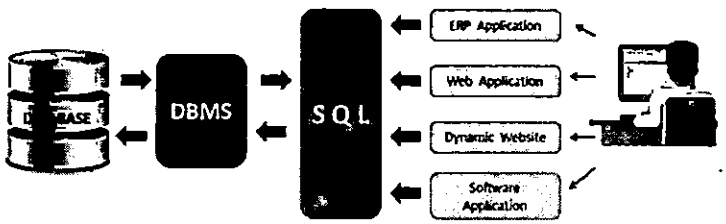


Figure 3

Most organizations require users to register with them and issue a login and password to the customer. Of course, many things occur behind the scenes when a database is being accessed via a web browser.

SQL, for instance, can be executed by the web application. This executed SQL is used to access the organization's database, return data to the web server, and then return that data to the customer's Internet browser. The basic structure of a web-based database system is similar to that of a client-server system from a user's standpoint (refer to Figure 1.3).

Each user has a client machine, which has a connection to the Internet and contains a web browser. The network in Figure 1.3 (in the case of a web-based database) just happens to be the Internet, as opposed to a local network. For the most part, a client is still accessing a server for information. It doesn't matter that the server might exist in another state or even another country.

The main point of web-based database systems is to expand the potential customer base of a database system that knows no physical location bounds, thus increasing data availability and an organization's customer base.

Notes

3.4 ADVANTAGES OF SQL

Programming using static SQL requires less effort than using embedded dynamic SQL. Static SQL statements are simply embedded into the host language source file, and the precompiler handles the necessary conversion to database manager run-time services API calls that the host language compiler can process.

Because the authorization of the person binding the application is used, the end user does not require direct privileges to execute the statements in the package.

For example, an application could allow a user to update parts of a table without granting an update privilege on the entire table. This can be achieved by restricting the static SQL statements to allow updates only to certain columns or to a range of values.

Static SQL statements are persistent, meaning that the statements last for as long as the package exists.

Dynamic SQL statem ents are cached until they are either invalidated, freed for space management reasons, or the database is shut down. If required, the dynamic SQL statements are recompiled implicitly by the DB2(R) SQL compiler whenever a cached statement becomes invalid.

The key advantage of static SQL, with respect to persistence, is that the static statements exist after a particular database is shut down, whereas dynamic SQL statements cease to exist when this occurs.

In addition, static SQL does not have to be compiled by the DB2 SQL compiler at run time, while dynamic SQL must be explicitly compiled at run time (for example, by using the PREPARE statement).

Because DB2 caches dynamic SQL statements, the statements do not need to be compiled often by DB2, but they must be compiled at least once when you execute the application.

There can be performance advantages to static SQL. For simple, short-running SQL programs, a static SQL statement executes faster than the same statement processed dynamically because the overhead of preparing an executable form of the statement is done at precompile time instead of at run time.

3.5 SQL DATA TYPES AND LITERALS

The following sections discuss the basic data types supported by ANSI SQL. Data types are characteristics of the data itself, whose attributes are placed on fields within a table. For example, you can specify that a field must contain numeric values, disallowing the entering of alphanumeric strings. After all, you would not want to enter alphabetic characters in a field for a dollar amount.

Defining each field in the database with a data type eliminates much of the incorrect data found in a database due to data entry errors. Field definition (data type definition) is a form of data validation that controls the type of data that may be entered into each given field.

Depending on your implementation of relational database management system (RDBMS), certain data types can be converted automatically to other data types depending upon their

format. This type of conversion in known as an implicit conversion, which means that the database handles the con- version for you. An example of this is taking a numeric value of 1000.92 from a numeric field and inputting it into a string field. Other data types cannot be converted implicitly by the host RDBMS and therefore must undergo an explicit conversion. This usually involves the use of an SQL function, such as CAST or CONVERT.

For example,

SELECT CAST('12/27/1974' AS DATETIME) AS MYDATE

The very basic data types, as with most other languages, are

- String types
- Numeric types
- Date and time types

Fixed-Length Strings :

Constant characters, those strings that always have the same length, are stored using a fixed-length data type. The following is the standard for an SQL fixed-length character:

Character (n) :

n represents a number identifying the allocated or maximum length of the particular field with this definition.

Some implementations of SQL use the CHAR data type to store fixed-length data. You can store alphanumeric data in this data type. An example of a constant length data type would be for a state abbreviation because all state abbreviations are two characters.

Spaces are normally used to fill extra spots when using a fixed-length data type; if a field's length was set to 10 and data entered filled only 5 places, the remaining 5 spaces would be recorded as spaces.

The padding of spaces ensures that each value in a field is a fixed length.

Varying-Length Strings :

SQL supports the use of varying-length strings, strings whose length is not constant for all data. The following is the standard for an SQL varying- length character:

Character Varying (n) :

n represents a number identifying the allocated or maximum length of the particular field with this definition.

Common data types for variable-length character values are the VARCHAR, VARBINARY, and VARCHAR2 data types. VARCHAR is the ANSI standard, which Microsoft SQL Server and MySQL use; Oracle uses both VARCHAR and VARCHAR2. The data stored in a character-defined column can be alphanumeric, which means that the data value may contain numeric

Notes

characters. VARBINARY is similar to VARCHAR and VARCHAR2 except that it contains a variable length of bytes. Normally, you would use a type such as this to store some kind of digital data such as possibly an image file.

Remember that fixed-length data types typically pad spaces to fill in allocated places not used by the field. The varying-length data type does not work this way. For instance, if the allocated length of a varying-length field is 10, and a string of 5 characters is entered, the total length of that particular value would be only 5. Spaces are not used to fill unused places in a column.

3.6 LARGE OBJECT TYPES

Some variable-length data types need to hold longer lengths of data than what is traditionally reserved for a VARCHAR field. The BLOB and TEXT data types are two examples of such data types in modern database implementations. These data types are specifically made to hold large sets of data. The BLOB is a binary large object, so its data is treated as a large binary string (a byte string). A BLOB is especially useful in an implementation that needs to store binary media files in the database, such as images or MP3s. The TEXT data type is a large character string data type that can be treated as a large VARCHAR field.

It is often used when an implementation needs to store large sets of character data in the database. An example of this would be storing HTML input from the entries of a blog site. Storing this type of data in the database enables the site to be dynamically updated.

Numeric Types

Numeric values are stored in fields that are defined as some type of number, typically referred to as NUMBER, INTEGER, REAL, DECIMAL, and so on. The following are the standards for SQL numeric values:

- BIT(n)
- BIT VARYING(n)
- DECIMAL(p,s)
- INTEGER
- SMALLINT
- BIGINT
- FLOAT(p,s)
- DOUBLE PRECISION(p,s)
- REAL(s)

p represents a number identifying the allocated or maximum length of the particular field for each appropriate definition.

s is a number to the right of the decimal point, such as 34.ss.

A common numeric data type in SQL implementations is NUMERIC, which accommodates the direction for numeric values provided by ANSI. Numeric values can be stored as zero,

positive, negative, fixed, and floating-point numbers. The following is an example using NUMERIC:

NUMERIC(5)

This example restricts the maximum value entered in a particular field to 99999. Note that all the database implementations that we use for the examples support the NUMERIC type but implement it as a DECIMAL.

Decimal Types

Decimal values are numeric values that include the use of a decimal point. The standard for a decimal in SQL follows, where p is the precision and s is the decimal's scale:

DECIMAL(p,s)

The precision is the total length of the numeric value. In a numeric defined DECIMAL(4,2), the precision is 4, which is the total length allocated for a numeric value. The scale is the number of digits to the right of the decimal point. The scale is 2 in the previous DECIMAL(4,2) example. If a value has more places to the right side of the decimal point than the scale allows, the value is rounded; for instance, 34.33 inserted into a DECIMAL(3,1) is typically rounded to 34.3. If a numeric value was defined as the following data type, the maximum value allowed would be 99.99:

DECIMAL(4,2)

The precision is 4, which represents the total length allocated for an associated value. The scale is 2, which represents the number of places, or bytes, reserved to the right side of the decimal point. The decimal point does not count as a character.

Allowed values for a column defined as DECIMAL(4,2) include the following:

- 12
- 12.4
- 2.44
- 12.449

The last numeric value, 12.449, is rounded off to 12.45 upon input into the column. In this case, any numbers between 12.445 and 12.449 would be rounded to 12.45.

Integers

An integer is a numeric value that does not contain a decimal, only whole numbers (both positive and negative).

Valid integers include the following

- 1
- 0
- -1
- 99
- -99
- 199

Notes

Floating-Point Decimals

Floating-point decimals are decimal values whose precision and scale are variable lengths and virtually without limit. Any precision and scale is acceptable. The REAL data type designates a column with single-precision, floating-point numbers. The DOUBLE PRECISION data type designates a column that contains double-precision, floating-point numbers. To be considered a single-precision floating point, the precision must be between 1 and 21 inclusive. To be considered a double-precision floating point, the precision must be between 22 and 53 inclusive. The following are examples of the FLOAT data type:

- FLOAT
- FLOAT(15)
- FLOAT(50)

Date and Time Types

Date and time data types are quite obviously used to keep track of information concerning dates and time. Standard SQL supports what are called DATETIME data types, which include the following specific data types:

- DATE
- TIME
- DATETIME
- TIMESTAMP

The elements of a DATETIME data type consist of the following:

- YEAR
- MONTH
- DAY
- HOUR
- MINUTE
- SECOND

NULL Data Types

As you should know from Hour 1, a NULL value is a missing value or a column in a row of data that has not been assigned a value. NULL values are used in nearly all parts of SQL, including the creation of tables, search conditions for queries, and even in literal strings.

The following are two methods for referencing a NULL value:

- NULL (the keyword NULL itself)

  The following does not represent a NULL value, but a literal string containing the characters N-U-L-L:

'NULL' When using the NULL data type, it is important to realize that data is not required in a particular field. If data is always required for a given field, always use NOT

NULL with a data type. If there is a chance that there might not always be data for a field, it is better to use NULL.

User-Defined Types

A user-defined type is a data type that the user defines. User-defined types allow users to customize their own data types to meet data storage needs and are based on existing data types.

User-defined data types can assist the developer by providing greater flexibility during database application development because they maximize the number of possibilities for data storage. The CREATE TYPE statement is used to create a user-defined type.

For example, you can create a type as follows in both MySQL and Oracle:

```
CREATE TYPE PERSON AS OBJECT
(NAME VARCHAR (30),
SSN VARCHAR (9));
```

You can reference your user-defined type as follows:

```
CREATE TABLE EMP_PAY
(EMPLOYEE PERSON,
SALARY DECIMAL(10,2),
HIRE_DATE DATE);
```

Notice that the data type referenced for the first column EMPLOYEE is PERSON. PERSON is the user-defined type you created in the first example.

3.7 TYPES OF SQL COMMANDS

The following sections discuss the basic categories of commands used in SQL to perform various functions. These functions include building database objects, manipulating objects, populating database tables with data, updating existing data in tables, deleting data, performing database queries, controlling database access, and overall database administration.

The main categories are

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Query Language (DQL)
- Data Control Language (DCL)
- Data administration commands
- Transactional control commands

Data Definition Language (DDL)

Data Definition Language (DDL) is the part of SQL that enables a database user to create and restructure database objects, such as the creation or the deletion of a table.

Notes

Some of the most fundamental DDL commands discussed during the following hours include

- CREATE TABLE
- ALTER TABLE
- DROP TABLE
- CREATE INDEX
- ALTER INDEX
- DROP INDEX
- CREATE VIEW
- DROP VIEW

1. Create Table: This command is used to create a new table in a database.

The SQL syntax for CREATE TABLE is

CREATE TABLE "table_name"

("column 1" "data_type_for_column_1",

"column 2" "data_type_for_column_2",

...)

So, if we are to create the customer table specified as above, we would type in

CREATE TABLE customer

(First_Name char(50),

Last_Name char(50),

Address char(50),

City char(50),

Country char(25),

Birth_Date date)

2. Alter table:

The ALTER TABLE statement is used to add or drop columns in an existing table.

ALTER TABLE table_name

ADD column_name datatype

ALTER TABLE table_name

DROP COLUMN column_name

Person:

Last Name	First Name	Address
Pettersen	Kari	Storgt 20

Example

To add a column named "City" in the "Person" table:

ALTER TABLE Person ADD City varchar(30)

Result:

LastName	FirstName	Address City
Pettersen	Kari	Storgt 20

3. Drop Table: This command is used to drop a table from a database.

The syntax for drop table is

DROP TABLE "table_name"

4. Create Index:

Indices are created in an existing table to locate rows more quickly and efficiently. It is possible to create an index on one or more columns of a table, and each index is given a name. The users cannot see the indexes, they are just used to speed up queries.

Note: Updating a table containing indexes takes more time than updating a table without, this is because the indexes also need an update. So, it is a good idea to create indexes only on columns that are often used for a search.

A Unique Index

Creates a unique index on a table. A unique index means that two rows cannot have the same index value.

CREATE UNIQUE INDEX index_name

ON table_name (column_name)

The "column_name" specifies the column you want indexed.

A Simple Index

Creates a simple index on a table. When the UNIQUE keyword is omitted, duplicate values are allowed.

CREATE INDEX index_name

ON table_name (column_name)

The "column_name" specifies the column you want indexed.

Example

This example creates a simple index, named "PersonIndex", on the LastName field of the Person table:

CREATE INDEX PersonIndex

ON Person (LastName)

If you want to index the values in a column in descending order, you can add the reserved word

DESC after the column name:

Notes

```
CREATE INDEX PersonIndex
ON Person (LastName DESC)
```

If you want to index more than one column you can list the column names within the parentheses, separated by commas:

```
CREATE INDEX PersonIndex
ON Person (LastName, FirstName)
```

5. **Drop Index**

You can delete an existing index in a table with the DROP INDEX statement.

Syntax for Microsoft SQLJet (and Microsoft Access):

```
DROP INDEX index_name ON table_name
```

Syntax for MS SQL Server:

```
DROP INDEX table_name.index_name
```

Syntax for IBM DB2 and Oracle:

```
DROP INDEX index_name
```

Syntax for MySQL:

```
ALTER TABLE table_name DROP INDEX index_name
```

6. **Create View**

In SQL, a VIEW is a virtual table based on the result-set of a SELECT statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from a single table.

Note: The database design and structure will NOT be affected by the functions, where, or join statements in a view.

Syntax

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

Note: The database does not store the view data. The database engine recreates the data, using the view's SELECT statement, every time a user queries a view.

Using Views

A view could be used from inside a query, a stored procedure, or from inside another view. By adding functions, joins, etc., to a view, it allows you to present exactly the data you want

to the user. The sample database Northwind has some views installed by default. The view "Current Product List" lists all active products (products that are not discontinued) from the Products table.

The view is created with the following SQL:

```
CREATE VIEW [Current Product List] AS
SELECT ProductID,ProductName
FROM Products
WHERE Discontinued=No
```

We can query the view above as follows:

```
SELECT * FROM [Current Product List]
```

Another view from the Northwind sample database selects every product in the Products table that has a unit price that is higher than the average unit price:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName,UnitPrice
FROM Products
WHERE UnitPrice>(SELECT AVG(UnitPrice) FROM Products)
```

We can query the view above as follows:

```
SELECT * FROM [Products Above Average Price]
```

Another example view from the Northwind database calculates the total sale for each category in 1997.

Note that this view select its data from another view called "Product Sales for 1997":

```
CREATE VIEW [Category Sales For 1997] AS
SELECT DISTINCT CategoryName,Sum(ProductSales) AS CategorySales
FROM [Product Sales for 1997]
GROUP BY CategoryName
```

We can query the view above as follows:

```
SELECT * FROM [Category Sales For 1997]
```

We can also add a condition to the query. Now we want to see the total sale only for the category "Beverages":

```
SELECT * FROM [Category Sales For 1997]
WHERE CategoryName='Beverages
```

Data Manipulation Language (DML)

Data Manipulation Language (DML) is the part of SQL used to manipulate data within objects of a relational database.

Notes

The three basic DML commands are

- INSERT
- UPDATE
- DELETE

1. INSERT Statement

The INSERT Statement adds one or more rows to a table. It has two formats:

INSERT INTO table-1 [(column-list)] VALUES (value-list)

and,

INSERT INTO table-1 [(column-list)] (query-specification)

The first form inserts a single row into table-1 and explicitly specifies the column values for the row. The second form uses the result of query-specification to insert one or more rows into table-1. The result rows from the query are the rows added to the insert table.

Both forms have an optional column-list specification. Only the columns listed will be assigned values. Unlisted columns are set to null, so unlisted columns must allow nulls. The values from the VALUES Clause (first form) or the columns from the query-specification rows (second form) are assigned to the corresponding column in column-list in order.

If the optional column-list is missing, the default column list is substituted. The default column list contains all columns in table-1 in the order they were declared in CREATE TABLE, or CREATE VIEW.

VALUES Clause

The VALUES Clause in the INSERT Statement provides a set of values to place in the columns of a new row. It has the following general format:

VALUES (value-1 [, value-2] ...)

value-1 and value-2 are Literal Values or Scalar Expressions involving literals. They can also specify NULL.

The values list in the VALUES clause must match the explicit or implicit column list for INSERT in degree (number of items). They must also match the data type of corresponding column or be convertible to that data type.

INSERT Examples

INSERT INTO p (pno, color) VALUES ('P4', 'Brown')

Before				**After**		
pno	**descr**	**color**		**pno**	**descr**	**color**
P1	Widget	Blue		P1	Widget	Blue
P2	Widget	Red	=>	P2	Widget	Red
P3	Dongle	Green		P3	Dongle	Green
				P4	NULL	Brown

INSERT INTO sp

SELECT s.sno, p.pno, 500

FROM s, p

WHERE p.color='Green' AND s.city='London'

Before				After		
sno	**pno**	**qty**		**sno**	**pno**	**qty**
S1	P1	NULL		S1	P1	NULL
S2	P1	200	=>	S2	P1	200
S3	P1	1000		S3	P1	1000
S3	P2	200		S3	P2	200
				S2	P3	500

2. UPDATE Statement

The UPDATE statement modifies columns in selected table rows. It has the following general format:

UPDATE table-1 SET set-list [WHERE predicate]

The optional WHERE Clause has the same format as in the SELECT Statement. See WHERE Clause. The WHERE clause chooses which table rows to update. If it is missing, all rows are in table-1 are updated.

The set-list contains assignments of new values for selected columns.

The SET Clause expressions and WHERE Clause predicate can contain subqueries, but the subqueries cannot reference table-1. This prevents situations where results are dependent on the order of processing.

SET Clause

The SET Clause in the UPDATE Statement updates (assigns new value to) columns in the selected table rows. It has the following general format:

SET column-1 = value-1 [, column-2 = value-2] ...

column-1 and column-2 are columns in the Update table. value-1 and value-2 are expressions that can reference columns from the update table. They also can be the keyword — NULL, to set the column to null.

Since the assignment expressions can reference columns from the current row, the expressions are evaluated first. After the values of all Set expressions have been computed, they are then assigned to the referenced columns. This avoids results dependent on the order of processing.

UPDATE Examples

UPDATE sp SET qty = qty + 20

Before After

Notes

sno.	pno	qty		sno.	pno	qty
S1	P1	NULL		S1	P1	NULL
S2	P1	200	=>	S2	P1	220
S3	P1	1000		S3	P1	1020
S3	P2	200		S3	P2	220

UPDATE s

SET name = 'Tony', city = 'Milan'

WHERE sno = 'S3'

Before				After		
sno	name	city		sno	name	city
S1	Pierre	Paris		S1	Pierre	Paris
S2	John	London	=>	S2	John	London
S3	Mario	Rome		S3	Tony	Milan

3. DELETE Statement

The DELETE Statement removes selected rows from a table. It has the following general format:

DELETE FROM table-1 [WHERE predicate]

The optional WHERE Clause has the same format as in the SELECT Statement. See WHERE Clause. The WHERE clause chooses which table rows to delete. If it is missing, all rows are in table-1 are removed.

The WHERE Clause predicate can contain subqueries, but the subqueries cannot reference table-

1. This prevents situations where results are dependent on the order of processing.

DELETE Examples

DELETE FROM sp WHERE pno = 'P1'

Before				After		
sno	pno	qty		sno	pno	qty
S1	P1	NULL		S3	P2	200
S2	P1	200	=>			
S3	P1	1000				
S3	P2	200				

DELETE FROM p WHERE pno NOT IN (SELECT pno FROM sp)

Before				After		
pno	descr	color		pno	descr	color
P1	Widget	Blue		P1	Widget	Blue
P2	Widget	Red	=>	P2	Widget	Red

P3	Dongle	Green

Data Query Language (DQL)

Though comprised of only one command, Data Query Language (DQL) is the most concentrated focus of SQL for modern relational database users. The base command is SELECT.

This command, accompanied by many options and clauses, is used to compose queries against a relational database. A query is an inquiry to the database for information. A query is usually issued to the database through an application interface or via a command-line prompt. You can easily create queries, from simple to complex, from vague to specific.

The SELECT statement is used to select data from a table. The tabular result is stored in a result table (called the result-set).

Syntax

```
SELECT column_name(s)
FROM table_name
```

Note: SQL statements are not case sensitive. SELECT is the same as select.

SQL SELECT Example

To select the content of columns named "LastName" and "FirstName", from the database table called "Persons", use a SELECT statement like this:

```
SELECT LastName,FirstName FROM Persons
```

The database table "Persons":

LastName FirstName Address City

HansenOla Timoteivn 10 Sandnes

Svendson Tove Borgvn 23 Sandnes

Pettersen Kari Storgt 20 Stavanger

The result

LastName FirstName

HansenOla

Svendson Tove

Pettersen Kari

Select All Columns

To select all columns from the "Persons" table, use a * symbol instead of column names, like this:

```
SELECT * FROM Persons
```

Result

LastName	FirstName	Address	City
HansenOla	Timoteivn	10 Sandnes	

Notes

Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

Data Control Commands

Data control commands in SQL enable you to control access to data within the database. These Data Control Language (DCL) commands are normally used to create objects related to user access and also control the distribution of privileges among users. Some data control commands are as follows:

- ALTER PASSWORD
- GRANT
- EVOKE
- CREATE SYNONYM

You will find that these commands are often grouped with other commands and might appear in a number of lessons throughout this book. Data Administration Commands Data administration commands enable the user to perform audits and perform analyses on operations within the database. They can also be used to help analyze system performance. Two general data administration commands are as follows:

- START AUDIT
- STOP AUDIT

Do not get data administration confused with database administration. Database administration is the overall administration of a database, which envelops the use of all levels of commands. Data administration is much more specific to each SQL implementation than are those core commands of the SQL language.

SQL-Transaction Statements

SQL-Transaction Statements control transactions in database access. This subset of SQL is also called the Data Control Language for SQL (SQL DCL).

There are 2 SQL-Transaction Statements:

- COMMIT Statement – commit (make persistent) all changes for the current transaction
- ROLLBACK Statement – roll back (rescind) all changes for the current transaction

Transactional Control Commands

Transactional control is the ability to manage various transactions that may occur within a relational database management system. When a transaction is executed and completes successfully, the target table is not immediately changed, although it may appear so according to the output. When a transaction successfully completes, there are transactional control commands that are used to finalize the transaction, either saving the changes made by the transaction to the database or reversing the changes made by the transaction.

There are three commands used to control transactions:

- COMMIT

- ROLLBACK
- SAVEPOINT
- The SET TRANSACTION Command
- Transactional control commands are only used with the DML commands INSERT, UPDATE, and DELETE. For example, you do not issue a COMMIT statement after creating a table. When the table is created, it is automatically committed to the database. Likewise, you cannot issue a ROLLBACK to replenish a table that was just dropped.
- When a transaction has completed, the transactional information is stored either in an allocated area or in a temporary rollback area in the database. All changes are held in this temporary rollback area until a transactional control command is issued. When a transactional control command is issued, changes are either made to the database or discarded; then, the temporary rollback area is emptied. Figure 3.1 illustrates how changes are applied to a relational database.

The COMMIT Command

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database. The COMMITcommand saves all transactions to the database since the last COMMIT or ROLLBACK command.

COMMIT [WORK];

The keyword COMMIT is the only mandatory part of the syntax, along with the character or command used to terminate a statement according to each implementation. WORK is a keyword that is completelyoptional; its only purpose is to make the command more user-friendly.

The ROLLBACK Command

The ROLLBACK command is the transactional control command used to undo transactions that have not already been saved to the database. The ROLLBACK command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

The syntax for the ROLLBACK command is as follows:

rollback [work];

Once again, the COMMIT statement, the WORK keyword is an optional part of the ROLLBACK syntax.

The SAVEPOINT Command

A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.

The syntax for the SAVEPOINT command is This command serves only in the creation of a SAVEPOINT among transactional statements. The ROLLBACK command is used to undo a group of transactions. The SAVEPOINT is a way of managing transactions by breaking large numbers of transactions into smaller, more manageable groups.

Notes

The SET TRANSACTION Command

The SET TRANSACTION Command Establishes the isolation level of the current transaction. If you use a SET TRANSACTION statement, it must be the first statement in your transaction. However, a transaction need not have a SET TRANSACTION statement.

3.8 AGGREGATE FUNCTIONS

SQL has a lot of built-in functions for counting and calculations.

Function Syntax

The syntaxfor built-in SQL functions is: SELECT function(column) FROM table

Aggregate functions operate against a collection of values, but return a single value.

Note : If used among many other expressions in the item list of a SELECT statement, the SELECT must have a GROUP BY clause!!

Five important aggregate functions: SUM, AVG, MAX, MIN, and COUNT.

They are calledaggregate functions because they summarize the results of a query, rather than listing all of the rows.

- SUM () gives the total of all the rows, satisfying anyconditions, of the givencolumn, where the givencolumn is numeric.
- AVG () gives the average of the given column.
- MAX () gives the largest figure in the given column.
- MIN () gives the smallest figure in the given column.
- COUNT(*) gives the number of rows satisfying the conditions.

Examples 1 :

SELECT SUM(SALARY), AVG(SALARY) FROM EMPLOYEESTABLE;

This query shows the total of all salaries in the table, and the average salary of all of the entries in the table.

SELECT MIN(BENEFITS) FROM EMPLOYEESTABLE WHERE POSITION = 'Manager';

This query gives the smallest figure of the Benefits column, of the employees who are Managers, which is 12500.

SELECT COUNT(*)

FROM EMPLOYEESTABLE WHERE POSITION = 'Staff';

This query tells you how many employees have Staff status.

Aggregate functions (like SUM) often need an added GROUP BYfunctionality

3.9 GROUP BY CLAUSE

GROUP BY Clauseis added to SQL because aggregate functions (like SUM) return the aggregate of all column values every time theyare called, and without the GROUP BY function it was impossible to find the sumfor each individual group of column values.

The syntaxfor the GROUP BY function is:

SELECT column,SUM(column) FROM table GROUP BY column

GROUP BY Example

This "Sales" Table :

Company	Amount
TVS	5500
IBM	4500
TVS	7100

And This SQL :

SELECT Company, SUM(Amount) FROM Sales

Returns this result :

Company	SUM (Amount)
TVS	17100
IBM	17100
TVS	17100

The above code is invalid because the column returned is not part of an aggregate. AGROUP BY clause willsolve this problem:

SELECT Company,SUM(Amount) FROM Sales GROUP BY Company

Returns this result :

Company	SUM (Amount)
TVS	12600
IBM	4500

3.10 HAVING CLAUSE

Having clause can also be added to SQL because the WHERE keyword could not be used against aggregate functions (like SUM), and without HAVING, it would be impossible to test for result conditions.

The syntax for the HAVING function is:

SELECT column, SUM(column) FROM table

GROUP BY column

Notes

HAVING SUM (column) condition value

This "Sales" Table :

Company	(Amount)
TVS	5500
IBM	4500
tvs	7100

This SQL :

SELECT Company,SUM(Amount) FROM Sales GROUP BY Company

HAVING SUM(Amount)>10000

Returns this result :

Company	(Amount)
TVS	12600

3.11 ORDER BY CLAUSE

The ORDER BY clause is optional. If used, it must be the last clause in the SELECT statement.

The ORDER BY clause requests sorting for the results of a query.

When the ORDER BY clause is missing, the result rows from a queryhave no defined order (they are unordered). The ORDER BYclause defines the ordering of rows based on columns fromthe SELECT clause. The ORDER BY clause has the following general format:

ORDER BY column-1 [ASC | DESC] [column-2 [ASC | DESC]]...

column-1, column-2, ... are column names specified (or implied) in the select list. If a select column is renamed (given a new name in the select entry), the new name is used in the ORDER BY list. ASC and DESC request ascending or descending sort for a column. ASC is the default.

ORDER BYsorts rows using the ordering columns in left-to-right, major-to-minor order. The rows are sorted first on the first column name in the list. If there are any duplicate values for the first column, the duplicates are sorted on the second column (within the first column sort) in the Order Bylist, and so on. There is no defined inner ordering for rows that have duplicate values for all Order Bycolumns.

Database nulls require special processing in ORDER BY. A null column sorts higher than all regular values; this is reversed for DESC.

In sorting, nulls are considered duplicates of each other for ORDER BY. Sorting on hidden information makes no sense in utilizing the results of a query. This is also why SQL only allows select list columns in ORDER BY.

For convenience when using expressions in the select list, select items can be specified bynumber (starting with 1). Names and numbers can be intermixed.

Example queries :

SELECT * FROM sp ORDER BY 3 DESC

sno	pno	qty
S1	P1	NULL
S3	P1	1000
S3	P2	200
S2	P1	200

SELECT name, city FROM s ORDER BY name

name	city
John	London
Mario	Rome
Pierre	Paris

SELECT * FROM sp ORDER BY qty DESC, sno

sno	pno	qty
S1	P1	NULL
S3	P1	1000
S2	P1	200
S3	P2	200

Orders table :

Company	OrderNumber
Sega	3412
ABC Shop	5678
TVS	2312
TVS	6798

Example :

To display the company names in alphabeticalorder:

SELECT Company, OrderNumber FROM Orders ORDER BYCompanyASC (asending)

Result:

Company	OrderNumber
ABC Shop	5678
Sega	3412
TVS	6798
TVS	2312

Notes

Example :

To displaythe company names in alphabetical order AND the OrderNumber in numerical order:

SELECT Company, OrderNumber FROM Orders

ORDER BYCompany, OrderNumber

Result:

Company	OrderNumber
ABC Shop	5678
Sega	3412
TVS	2312
TVS	6798

Example :

To display the company names in reverse alphabetical order:

SELECT Company, OrderNumber FROM Orders

ORDER BY Company DESC

Result:

Company	OrderNumber
TVS	6798
TVS	2312
Sega	3412
ABC Shop	5678

Example :

To display the company names in reverse alphabetical order AND the OrderNumber in numerical order:

SELECT Company, OrderNumber FROM Orders

ORDER BY Company DESC, OrderNumber ASC

Result:

Company	OrderNumber
TVS	2312
TVS	6798
Sega	3412
ABC Shop	5678

Notice that there are two equalcompanynames (TVS) in the result above. The only time you will see the second column inASC order would be whenthere are duplicated values in the first sort column, or a handful of nulls.The ORDER BY keyword is used to sort the result.

Sort the Rows :

The ORDER BY clause is used to sort the rows.

Orders :

Company	OrderNumber
Sega	3412
ABC Shop	5678
TVS	2312
TVS	6798

Example :

To display the company names in alphabeticalorder:

SELECT Company, OrderNumber FROM Orders

ORDER BY Company

Result :

Company	OrderNumber
ABC Shop	5678
Sega	3412
TVS	6798
TVS	2312

Example :

To display the company names in alphabetical order AND the OrderNumber in numerical order:

SELECT Company, OrderNumber FROM Orders

ORDER BYCompany, OrderNumber

Result :

Company	OrderNumber
TVS	2312

Notes

TVS	6798
Sega	3412
ABC Shop	5678

Notice that there are two equalcompanynames (TVS) in the result above. The only time you will see the second column inASC order would be whenthere are duplicated values in the first sort column, or a handfulof nulls.

3.12 JOIN

The FROM clause allows more than 1 table in its list, however simply listing more than one table will veryrarelyproduce the expected results. The rows from one table must be correlated with the rows of the others. This correlation is known as joining.

s Table

sno	name	city
S1	Pierre	Paris
S2	John	London
S3	Mario	Rome

sp Table

sno	pno	qty
S1	P1	NULL
S2	P1	200
S3	P1	1000
S3	P2	200

An example can best illustrate the rationale behind joins. The following query:

SELECT * FROM sp, p

Produces :

sno	pno	qty	pno	descr	color
S1	P1	NULL	P1	Widget	Blue
S1	P1	NULL	P2	Widget	Red
S1	P1	NULL	P3	Dongle	Green
S2	P1	200	P1	Widget	Blue
S2	P1	200	P2	Widget	Red
S2	P1	200	P3	Dongle	Green
S3	P1	1000	P1	Widget	Blue
S3	P1	1000	P2	Widget	Red
S3	P1	1000	P3	Dongle	Green
S3	P2	200	P1	Widget	Blue
S3	P2	200	P2	Widget	Red
S3	P2	200	P3	Dongle	Green

Each row in sp is arbitrarily combined with each row in p, giving 12 result rows (4

rows in sp X 3 rows in p.) This is known as a cartesian product.

A more usable query would correlate the rows from sp with rows from p, for instance matching on the common column — pno:

SELECT *

FROM sp, p

WHERE sp.pno = p.pno This Produces :

sno	pno	qty	pno	descr	color
S1	P1	NULL	P1	Widget	Blue
S2	P1	200	P1	Widget	Blue
S3	P1	1000	P1	Widget	Blue
S3	P2	200	P2	Widget	Red

Rows for each part in p are combined with rows in sp for the same part by matching on part number (pno). In this query, the WHERE Clause provides the join predicate, matching pno from p with pno from sp.

The join in this example is known as an innerequi-join. equimeaning that the join predicate uses = (equals) to match the join columns. Other types of joins use different comparison operators. For example, a querymight use a greater-than join.

The term inner means only rows that match are included. Rows in the first table that have no matching rows in the second table are excluded and vice versa (in the above join, the row in p with pno P3 is not included in the result.) An outer join includes unmatched rows in the result.

More than 2 tables can participate in a join. This is basically just an extension of a 2 table join. 3 tables — a, b, c, might be joined in various ways:

- a joins b which joins c
- a joins b and the join of a and b joins c
- a joins b and a joins c

Plus several other variations. With inner joins, this structure is not explicit. It is implicit in the nature of the join predicates. With outer joins, it is explicit;

This query performs a 3 table join: SELECT name, qty, descr, color FROM s, sp, p

WHERE s.sno = sp.sno AND sp.pno = p.pno

It joins *s* to *sp* and spto *p*, producing :

name	qty	descr	color
Pierre	NULL	Widget	Blue
John	200	Widget	Blue
Mario	1000	Widget	Blue
Mario	200	Widget	Red

Notes

Note that the order of tables listed in the FROM clause should have no significance, nor does the order of join predicates in the WHERE clause.

3.13 INNER JOIN

An inner join is a join in which the values in the columns being joined are compared using a comparison operator.

Inner joins can be specified in either the FROM or WHERE clause.. Inner joins specified in the WHERE clause are known as old-style inner joins..

SQL INNER JOIN Syntax

SELECT column_name(s) FROM table_name1 INNER JOIN table_name2

ON table_name1.column_name=table_name2.column_name

The INNER JOIN keyword return rows whenthere is at least one match in both tables Let's assume that we have the following two tables,

Table Store_Information

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

Table Geography

region_name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

We want to find out sales by store, and we only want to see stores with sales listed in the report.

To do this, we can use the following SQL statement using INNER JOIN: SELECTA1. store_name STORE, SUM(A2.Sales) SALES

FROM GeographyA1

INNER JOIN Store_Information A2 ONA1.store_name = A2.store_name.GROUP BYA1. store_name

Result:

STORE	SALES
Los Angeles	$1800
San Diego	$250
Boston	$700

Byusing INNER JOIN, the result shows 3 stores, eventhough we are selecting from theGeography table, which has 4 rows. The row "New York" is not selected because it is not present in the Store_Information table.

Outer Join

An inner join excludes rows from either table that don't have a matching row in the other table. An outer join provides the ability to include unmatched rows in the queryresults.

The outer join combines the unmatched row in one of the tables with an artificial row for the other table. This artificial row has all columns set to null.

The outer join is specified in the FROM clause and has the following generalformat:

table-1 { LEFT | RIGHT | FULL } OUTER JOIN table-2 ON predicate-1

Predicate -1 : is a join predicate for the outer join. It can only reference columns from the joined tables. The LEFT, RIGHT or FULL specifiers give the type of join:

- LEFT – only unmatched rows from the left side table (table-1) are retained
- RIGHT – only unmatched rows from the right side table (table-2) are retained
- FULL – unmatched rows from both tables (table-1 and table-2) are retained

Outer join example:

```
SELECT pno, descr, color, sno, qty
FROM p LEFT OUTER JOIN sp ON p.pno = sp.pno
```

pno	descr	color	sno	qty
P1	Widget	Blue	S1	NULL
P1	Widget	Blue	S2	200
P1	Widget	Blue	S3	1000
P2	Widget	Red	S3	200
P3	Dongle	Green	NULL	NULL

Self Join

Notes

A query can join a table to itself. Self joins have a number of real world uses. For example, a self join can determine which parts have more than one supplier :

```
SELECT DISTINCT a.pno
    FROM sp a, sp b WHERE a.pno = b.pno
    AND a.sno<>b.sno
```

As illustrated in the above example, self joins use correlation names to distinguish columns in the select list and where predicate. In this case, the references to the same table are renamed - a and b. Self joins are often used in sub queries.

3.14 SET OPERATIONS

Set Operators : Set operators combines results of two queries into a single result. Set operations are generally performed on two Lists obtained from distinct tables.

Union

The UNION command is used to select related information from two tables, much like the JOIN command. However, when using the UNION command all selected columns need to be of the same data type.

Note: With UNION, only distinct values are selected. SQL Statement 1

UNION

SQL Statement 2

Employees_Norway

E_ID	E_Name
01	Hansen, Ola
02	Svendson, Tove
03	Svendson, Stephen
04	Pettersen, Kari

Employees_USA :

E_ID	E_Name
01	Turner, Sally
02	Kent, Clark
03	Svendson, Stephen
04	Scott, Stephen

Using the UNION Command Example :

List all different employee names in Norway and USA: SELECT E_Name FROM Employees_ Norway UNION

SELECT E_Name FROM Employees_USA

Result :

E_Name
Hansen, Ola
Svendson, Tove
Svendson, Stephen
Pettersen, Kari
Turner, Sally
Kent, Clark
Scott, Stephen

Note: This command cannot be used to list allemployees in Norwayand USA. In the example above we have two employees with equalnames, and only one of them is listed. The UNION command only selects distinct values.

Union All

The UNION ALLcommand is equal to the UNION command, except that UNION ALL selects all values.

SQL Statement 1 UNION ALL

SQL Statement 2

Using the UNION ALL Command

Example :

List all employees in Norway and USA:

SELECT E_Name FROM Employees_Norway

UNION ALL

SELECT E_Name FROM Employees_USA

Result :

E_Name
Hansen, Ola
Svendson, Tove
Svendson, Stephen
Pettersen, Kari
Turner, Sally
Kent, Clark

Notes

Svendson, Stephen
. Scott, Stephen

Intersection

The Intersect operator is used to return the rows returned by both queries. The following command displays the rows that are common in the results of first and second queries.

```
SELECT          member_id
FROM            members
WHERE           category ='F'
INTERSECT
SELECT DISTINCT member_id
FROM bookissue
ORDER BY        member_id;
```

Minus

The Minus operator is used to return rows from theresult of the first querythat are not available in the result of the second query.

```
SELECT          member_id
FROM            members
WHERE           category ='F' MINUS
SELECT DISTINCT member_id FROM
bookissue
ORDER BY        memb
```

MEMBER I D
2
4
5
7

PRO P1	MEMBER I D

	1
	6

- Set Operator combine the result of two queries into one
- All set operators have equal precedence.
- When multiple set operators are present in the same query they are evaluated fromleft to right.
- The datatype of resulting columns should matchin both queries.
- The resultant column name would be the column name of first query.

3.16 ORACLE CREATING TABLES

Oracle CREATE TABLE statement with syntax, examples, and practice exercises.

Description

The Oracle CREATE TABLE statement allows you to create and define a table.

Syntax

The syntax for the CREATE TABLE statement in Oracle/PLSQL is:

```
CREATE TABLE table_name
(
  column1 datatype [ NULL | NOT NULL ],
  column2 datatype [ NULL | NOT NULL ],
  ...
  column_n datatype [ NULL | NOT NULL ]
);
```

Parameters or Arguments

table_name

The name of the table that you wish to create.

column1, column2, ... column_n

The columns that you wish to create in the table. Each column must have a datatype. The column should either be defined as "null" or "not null" and if this value is left blank, the database assumes "null" as the default.

Example

Let's look at an Oracle CREATE TABLE example.

```
CREATE TABLE customers
( customer_id number(10) NOT NULL,
  customer_name varchar2(50) NOT NULL,
  city varchar2(50)
);
```

This Oracle CREATE TABLE example creates a table called customers which has 3 columns.

The first column is called customer_id which is created as a number datatype (maximum 10 digits in length) and can not contain null values.

The second column is called customer_name which is a varchar2 datatype (50 maximum characters in length) and also can not contain null values.

The third column is called city which is a varchar2 datatype but can contain null values.

Now the only problem with this Oracle CREATE TABLE statement is that you have

Notes

not defined a primary key for the table. We could modify this CREATE TABLE statement and define the customer_id as the primary key as follows:

```
CREATE TABLE customers
( customer_id number(10) NOT NULL,
  customer_name varchar2(50) NOT NULL,
  city varchar2(50),
  CONSTRAINT customers_pk PRIMARY KEY (customer_id)
);
```

Learn about primary keys.

Learn about foreign keys.

Practice Exercise #1:

Create an Oracle table called suppliers that stores supplier ID, name, and address information.

Solution for Practice Exercise #1:

The Oracle CREATE TABLE statement for the suppliers table is:

```
CREATE TABLE suppliers
( supplier_id number(10) NOT NULL,
  supplier_name varchar2(50) NOT NULL,
  address varchar2(50),
  city varchar2(50),
  state varchar2(25),
  zip_code varchar2(10)
);
```

Practice Exercise #2:

Create an Oracle table called customers that stores customer ID, name, and address information.

But this time, the customer ID should be the primary key for the table.

Solution for Practice Exercise #2:

The Oracle CREATE TABLE statement for the customers table is:

```
CREATE TABLE customers
( customer_id number(10) NOT NULL,
  customer_name varchar2(50) NOT NULL,
```

```
    address varchar2(50),
    city varchar2(50),
    state varchar2(25),
    zip_code varchar2(10),
    CONSTRAINT customers_pk PRIMARY KEY (customer_id)
);
```

Applying Column Constraints

3.17 APPLYING COLUMN CONSTRAINTS IN SQL

SQL CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column. If you define a CHECK constraint on a column it will allow only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

SQL CHECK on CREATE TABLE

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

MySQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int CHECK (Age>=18)
);
```

To allow naming of a CHECK constraint, and for defining a CHECK constraint on

Notes

multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
);
```

SQL CHECK on ALTER TABLE

To create a CHECK constraint on the "Age" column when the table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CHECK (Age>=18);
```

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes');
```

DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL:

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT CHK_PersonAge;
```

MySQL:

```
ALTER TABLE Persons
DROP CHECK CHK_PersonAge;
```

SQL DEFAULT Constraint

The DEFAULT constraint is used to set a default value for a column.

The default value will be added to all new records, if no other value is specified.

SQL DEFAULT on CREATE TABLE

The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

My SQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
);
```

The DEFAULT constraint can also be used to insert system values, by using functions like GETDATE():

```
CREATE TABLE Orders (
    ID int NOT NULL,
    OrderNumber int NOT NULL,
    OrderDate date DEFAULT GETDATE()
);
```

SQL DEFAULT on ALTER TABLE

To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

MySQL:

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'Sandnes';
```

SQL Server:

```
ALTER TABLE Persons
ADD CONSTRAINT df_City
DEFAULT 'Sandnes' FOR City;
```

MS Access:

```
ALTER TABLE Persons
ALTER COLUMN City SET DEFAULT 'Sandnes';
```

Oracle:

```
ALTER TABLE Persons
```

Notes

MODIFY City DEFAULT 'Sandnes';

DROP a DEFAULT Constraint

To drop a DEFAULT constraint, use the following SQL:

MySQL:

ALTER TABLE Persons

ALTER City DROP DEFAULT;.

SQL Server / Oracle / MS Access:

ALTER TABLE Persons

ALTER COLUMN City DROP DEFAULT;

SQL Server:

ALTER TABLE Persons

ALTER COLUMN City DROP DEFAULT;

SQL CREATE INDEX Statement

The CREATE INDEX statement is used to create indexes in tables.

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

CREATE INDEX index_name

ON table_name (column1, column2, ...);

CREATE UNIQUE INDEX Syntax

Creates a unique index on a table. Duplicate values are not allowed:

CREATE UNIQUE INDEX index_name

ON table_name (column1, column2, ...);

Note: The syntax for creating indexes varies among different databases. Therefore: Check the syntax for creating indexes in your database.

CREATE INDEX Example

The SQL statement below creates an index named "idx_lastname" on the "LastName" column in the "Persons" table:

```
CREATE INDEX idx_lastname
ON Persons (LastName);
```

If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

```
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
```

3.18 QUERIES AND SUBQUERIES

Tables used in queries :

EMP: Contains information about the employees of the sample company

DEPT: Contains information about the departments in the company

Structure of EMP Table:

EMPNO	NUMBER(4)	Employee number
ENAME	VARCHAR(20)	Employee name
JOB	CHAR (10)	Designation
MGR	NUMBER (4)	Respective manager's EMPNO
HIREDATE	DATE	Date of joining
SAL	NUMBER (9,2)	Basic salary
COMM	NUMBER (7,2)	Commission
DEPTNO	NUMBER (2)	Department number

Structure of DEPT Table :

Column names	Types	Description
DEPTNO	NUMBER(2)	Department number
DNAME	VARCHAR2 (20)	Name of the department
LOC	VARCHAR2 (10)	Location of the department

Queries with join

- Joins are used to combine columns from different tables
- The connection between tables is established through the WHERE clause
- Types of joins: Equi Joins, Cartesian Joins, Outer Joins, Self Joins

One of the most important features of SQL is the ability to define relationships between multiple tables and draw information from them in terms of these relationship, all within a single command. With joins, the information from any number of tables can be related.

To join two tables, the retrieval criteria will typically specify the condition that a column in the first table (which is defined as foreign key) is equal to a column in the second table

Notes

(which is the primary key referenced by the foreign key) Ajoin's where clause may contain additional conditions. In a join, the table names are listed in the FROM clause, separated by commas.

SELECT < select - list >

FROM < table1 > < table 2 >,.....< tableN >

WHERE < table1. column1> = < table2. column 2 and

< table2 column 3 = < tableN. columnN >

.....

additional - conditions

The variables are defined as follows:-

<select-list> is the set of columns and expressions from <table1> through <tableN>

<table1> through <tableN> are the tables from which column values are retrieved.

<column1> through <columnN> are the columns in <table> through <tableN> Additional conditions are optional query criteria.:-

We introduce another table, INCR, which holds the information about the salary increments of the employee. The structure of the INCR table is:

EMPNO	NUMBER (4)
AMTNUMBER (7,2)	
DATEINCR	DATE

Equi Join

When two tables are joined together using equality of values in one or more columns, they make an Equi Join.

Table prefixes are utilized to prevent ambiguity and the WHERE clause specifies the col- umns being joined.

Examples:

List the employee numbers, names, department numbers and the department name:

SELECT empno, ename, emp. deptno, dname FROM emp, dept

WHERE emp. deptno = dept. deptno;

Here the deptno column exists in both the tables. To avoid ambiguity, the column name should be qualified with the table name (or with an alias).

Both the table names need to be specifed (emp and dept.) The WHERE clause defines the joining condition i.e., joining the deptno of emp table to the deptno of dept table. Here it checks for the equality values in these columns.

EMPNO.	ENAME	EMP. DEPTNO	DNAME
7369	SMITH	20	RESEARCH
7499	ALLEN	30	SALES
7521	WARD	30	SALES
7566	JONES	20	RESEARCH
7654	MARTIN	30	SALES
7698	BLAKE	30	SALES
7782	CLARK	10	ACCOUNTING
7788	SCOTT	20	RESEARCH
7839	KING	10	ACCOUNTING
7844	TURNER	30	SALES
7876	ADAMS	20	RESEARCH
7900	JAMES	30	SALES
7902	FORD	20	RESEARCH
7934	MILLER	10	ACCOUNTING
7945	ALLEN	20	RESEARCH
7526	MARTIN	20	RESEARCH
7985	SCOTT	30	SALES

14 rows selected. Using Table Aliases:

It can be very tedious to type table mames repeatedly. Temporary labels (or aliases) can be used in the FROM clause. These temporary names are valid only for the current select statement.Table aliases should also specified in the select clause. Table aliases can be up to 30 character in length but the shorter they are the better.

Note: The advantages of using table aliases in that it effectively speeds up the query.

SELECT e. empno, e.ename, e.deptno, d.dname FROM emp e, dept d WHERE e. deptno = d. deptno;

Cartesian Join

When no WHERE clause is specified, each row of one table matches every row of the other table. This results in a Cartesian product.

SELECT empno, ename, dname, loc FROM emp, dept;

If the umber of rows are 14 and 4 in emp and dept tables respectively, then the total number of rows produced is 56.

Cartesian product is finding out all the possible combination of columns from different tables.

Notes

Example:

Consider the following tables and the data present:

Tab1 : Holds the principal amount.

Tab2 : Holds year and rate of interest.

Tab1 Tab2

PRINCIPAL	YEAR	RATE
.....	.....	.....
1000	1	10
2000	2	11
3000	3	11.5
	4	12

Finding the possible combinations of calculation of amount, a Cartesian join of theTab1 and Tab2 is required. The formula for calulation ofAmount is Princopal* (1+(rate/100)year

SELECT PRINCIPAL, YAER, RATE, PRINCIPAL* POWER (1+RATE/100), YEAR) FROM TAB1, TAB2;

Will produce the following output

PRINCIPAL	YEAR	RATE	PRINCIPLE*POWER (1+ RATE/100), YEAR)
.....	.....	.....	.....
1100	1	10	1100
2000	1	10	2200
3000	1	10	3300
1000	2	11	1232.1
2000	2	11	2464.2
3000	2	11	3696.3
1000	3	11.5	2772.1959
2000	3	11.5	2772.3918
3000	3	11.5	4158.5876
1000	4	12	1573.5194
2000	4	12	3147.0387
3000	4	12	4720.5581

Outer Join

If there are any values in one table that do not have corresponding values (s) in the other, in an equi join that row will not be selected. Such rows can be forcefully selected by using the outer join symbol (+) The corresponding columns for that row will have NULLs.

Example:

In the emp table, no record of the employees belonging to the department 40 is present. There- fore, in the example above for equi join, the row of department 40 from the dept table will not be dis- played.

Dispaly the list of the employees working in each department Display the department information even if no empployee belongs to that department:

SELECT empno, ename, emp. deptno, dnamo, loc FROM emp, dept

WHERE emp. deptno (+) = demp. deptno;

EMPNO	ENAME	EMP.	DEPTNDNAME
7369	SMITH	20	RESEARCH
7499	ALLEN	30	SALES
7521	WARD	30	SALES
7566	JONES	20	RESEARCH
7654	MARTIN	30	SALES
7698	BLAKE	30	SALES
7782	CLARE	10	ACCOUNTING
7788	SCOTT	20	RESEARCH
7839	KING	10	ACCOUNTING
7844	TURNER	30	SALES
7876	ADAMS	20	RESEARCH
7900	JAMES	30	SALES
7902	FORD	20	RESEARCH
7934	MILLER	10	ACCOUNTING
7945	ALLEN	20	RESEARCH
7526	MARTIN	20	RESEARCH
7985	SCOTT	30	SALES
		40	OPERATIONS

14 rows selected.

Notes

If the symbol (+) is placed on the other side of the equation then all the employee details with no corresponding department name and location, will be displayed with NULL values in DNAME and LOC column.

Rules to place (+) operator:-

- The outer join symbol (+) can not be on both the side.
- We can not "outer join" the some table to more than one other table in a single SELECT statement.
- A condition involving an outer join may not use the IN operator or be linked to another condition by the OR operator.

Self Join

To join a table to itself means taht each row of the table is combined with itself and with every other row of the table. The self join can be viewed as a join of two copies of the same table. The table is not actually copied, but SQL performs the command as though it were.

The syntax of the command for joining a table to itself is almost the same as that for joinning two different table. To distinguish the column names from one another, aliases for the actual the table name are used, since both the table have the same name. Table name aliases are defined in the FROM

clause of the query.

To define the alias, one space is left after the table name and the alias.

Example:

* EMPTABLE

EMPNO	ENAME	MGR
7839	KING	
7566	JONES	7839
7876	ADAMS	7788
7934	MILLER	7782
.....	.....	.....

Consider the emp table shown above. Primary key of the emp table is empno. Details of each employee's manager is just another row in the EMPtable whose EMPNO is stored in MGR column of some other row.

Notes

So every employee except manager has a Manager. Therefore MGR is a foreign key taht reference empno. To list out the names of the manager with the employee record one will have to join EMPwithItself.

SELECT WORKER. ename, MANAGER. ename 'Manager'

FROM emp WORKER, emp MANAGER

WHERE WORKER. mgr = MANAGER. empno;

Where WORKER and MANAGER are two aliases for the EMP table and as a virtual

EMP TABLE

EMPNO	ENAME	MGR
7839	KING	
7566	JONES	7839
7876	ADAMS	7788
7934	MILLER	7782
.....	.....	.....

WORKER

EMPNO	ENAME	MGR
7839	KING	
7566	JONES	7839
7876	ADAMS	7788
7934	MILLER	7782
.....	.....	.....

MANAGER

EMPNO	ENAME	MGR
7839	KING	
7788	SCOTT	7566
7782	CLARK	7839
7934	MILLER	7782
.....	.....	.....

The output will be:

ENAME	MANAGER
.....	.....
SCOTT	JONES
FORD	JONES
ALLEN	BLAKE
WARD	BLAKE
JAMES	BLAKE

Notes

TURNER	BLAKE
MARTIN	BLAKE
MILLER	CLARK
ADAMS	SCOTT
JONES	KING
CLARK	KING
BLAKE	KING
SMITH	FORD

13 rows selected.

Self Learning Exercise:-

2. In the previous query, only 13 rows (Not 14) have been retrieved. Why ?
3. List all employees who joined the company before their manager.

SELECT e. ename, e.hiredate, m.ename manager, m.hiredate FROM emp e, emp m WHERE e.mgr = m.empno and e. hiredate < m.hiredate;

ename hiredate manager hiredate

.....	.....	.....	.....
ALLEN	15-AUG-83	BLAKE	11-JUN-84
WARD	26-MAR-84	BLAKE	11-JUN-84
MARTIN	05-DEC-83	BLAKE	11-JUN-84
TURNER	04-JUN-84	BLAKE	11-JUN-84
MILLER	21-NOV-83	CLARK	14-MAY-84
JONES	31-OCT-83	KING	09-JUN-84
BLAKE	11-JUN-84	KING	09-JUL-84
CLARK	14-MAY-84	KING	09-JUL-84
SMITH	13-JUN-83	FORD	05-DEC-83

3.19 SET OPERATORS

SET Operator are used to combine information of similar type from one or more than one table.

Datatype of correponding columns must be the same The types of SET operator in ORACLE are :

- UNION: Rows of first query plus rows of second query, less duplicate rows

- INTERSECT : Common rows from all the queries
- MINUS : Rows unique to the first query

SET operator combine two or more queries into one result. Suppose we want following three details from dept table

- List of all the different designations in department 20 and 30
- List the jobs common to department 20 and 30
- List the jobs unique to department 20 :

To get these combination of information the SET operators UNION, INTERSECTand MINUS are used.

Union

The UNION clause merges the outputs of two of more queries into a single set of rows and columns. The syntax of UNION operator is

select <stmt1> union

select <stmt2>

{order-by-clause}

The variables are defined as follows:

select stmt1 and select stmt 2 are valid SELECT statement.

order-by-clause is optional and it references the columns by number rather than by name.

The queries are all executed independently, but their output is merged. Only the final query ends with a semicolon.

Examples:-

- Dispaly the different designations in department 20 and 30:

```
SELECT job FROM emp
WHERE deptno = 20
UNION
SELECT job FROM emp
WHERE deptno = 30;
```

The output will be:

```
JOB
.....
CLERK
SALESMAN
MANAGER
```

Notes

ANALYST

Points to be kept in mind while using UNION operator.

- The two select statement may not contain an ORDER BYclause; hawever, the final result of the entire UNION operation can be ordered.
- The number of columns retrieved by first select must be equal to number of columns retrieved by second select.
- The date types of columns retrieved by the select statements should be same.
- The optional order by clause differs from the usual ORDER BY clause an a SELECT statement because the columns used for ordering must be reference by a number rather than name. The reason that the columns must be referenced by number is the SQL does not require that the column name retrieved by first select be identical to theolumns names retrieved by second select

Example:-

```
select empno, ename from emp
where deptno = 10
UNION
select empno, ename from emp
where deptno = 30
order by 1;
```

EMPNO	ENAME
.....	.....
7499	ALLEN
7521	WARD
7654	MARTIN
7698	BLAKE
7839	KING
7844	TURNER
7900	JAMES
7934	MILLER

Intersect

The intersect operator returns the rows that are common between two sets of rows. The syntax of INTERSECT operator is same as UNION operator. Only UNION key word is replaced by INTERSECT.

```
select stmt1
INTERSECT
```

select stmt2

{order-by clause}

Example:-

List the jobs common to department 20 and 30:

SELECT job FROM emp WHERE deptno = 20

INTERSECT

SELECT job FROM emp WHERE deptno = 30,

Job

.....

CLERK

MANAGER

Minus operator returns the rows unique to first query. The syntax using the MINUS operator resembles the syntax for the union operator:

select stmt1

MINUS

select stmt2

{order-by clause}

The requirements and considerations for using the MINUS operator are essentially the same as those for the INTERSECT and UNION operator. To illustrate the use of the MINUS operator, consider the following example.

List the jobs unique to department 20:

SELECT job FROM emp WHERE deptno = 20

MINUS

SELECT job FROM emp

WHERE deptno = 10

MINUS

SELECT job FROM emp

WHERE deptno = 30;

Job

.....

ANALYST

Classroom Exercise:-

Can we rewrite the query to find jobs that are unique to department 20 as:

SELECT job FROM emp WHERE deptno = 20

Notes

```
MINUS
SELECT job FROM emp WHERE deptno IN (10, 30);
```

Sub Queries

- The result of inner query is dynamically substituted in the condition of outer query
- There is no practical limitation to the level of nesting of queries in Oracle 9
- When using relational operators, ensure that the sub query returns a single column output

In some cases, the DISTINCT clause cab be used to ensure single valued output

SQL has an ability to nest queries within one another. Asubquery is a SELECT statement that is nested within another SELECT statement and which returns intermediate results. SQL frist evaluates the inner query (or sub query) within the WHERE clause.

The inner query generates values that are tested in the predict of the outer query, determining when it will be true. The return value of inner query is then substituted in the condition of the outer query.

Advantages of Nested queries

- Subqueries allows a developer to build powerful commands out of simple ones.
- The nested subquery is very useful when you need to select rows from a table with a condition that depends on the data in the table itself.

Example:

List the employees belonging to the department of MILLER:

Here we do not know the department to which MILLER belongs. So, we have to determine the epartment of MILLER and use that department number to find out the other employees of that department.

```
SELECT deptno FROM emp
WHERE ename = 'MILLER';
DEPTNO
.....
10
SELECT ename FROM emp
WHERE deptno = 10;
   ENAME
   .....
   KING
   CLARK
```

MILLER

Combining the above two queries:

SELECT ename FROM emp

WHERE deptno = (SELECT deptno FROM emp WHERE ename = 'MILLER');

* list the names of the employee drawing the highest salary:

SELECT ename FROM emp

WHERE sal = (SELECT MAX (sal) FROM emp),;

Using Aggregate Functions In Subqueries

Aggregate function produces single value for any number of rows. We want to see all employee details whose salary is greater than avarege salary of employees whose hiredata is before'01-04-81' For this we need to use aggregate function in inner query.

SELECT * from emp where sal >

(select avg (sal) from emp where hiredata < '01-APR- 81');

Subqueries in Having

We can also use subqueries within the Having clause. These subqueries can use their own aggregate functions as long as do not produce multiple values or use GROUPBY or HAVING themselves.

List the employee number, name, total number of increments and total increments amount for the employee who has got maximum number of increments:

SELECT incr. empno, ename, COUNT (*), SUM (amt) FROM emp, incr WHERE incr. empno = emp. empno

GROUPBY incr. empno, ename

HAVING COUNT (*) = (SELECT MAX (COUNT (*) from incr

GROUP BY empno);

The output will be:

EMPNO	ENAME	COUNT(*)	SUM (AMT)
.....	.....	.....	.....
7369	SMITH	3	500

List the job with highest average salary.

SELECT job, AVG (sal)

FROM emp

GROUP BY job

HAVINGAVG (sal) = (SELECT MAX (AVG (sal)

FROM emp

GROUP BY job);

Notes

The output will be:

JOB	AVG (SAL)
.....	.....
PRESIDENT	5000

The inner query first finds the average salary for each different job group, and the MAX function picks the highest average salary. That value (5000) is used in the HAVING clause. The GROUP BYclause in the main query is needed because the main query's SELECT list contains both an aggregate and non-aggregate column.

Distinct Clause with Subqueries

Distinct clause is used in some cases to force a subquery to generate a single value. Suppose we want to find the details of the department whose manager's empcode '7698'.The query for this is shown below.

```
select * from dept
where deptno = (select distinct deptno from emp where mgr = '7698');
```

The inner query will give deptno whose manager's empcode is '7698'Without distinct clause the inner query would have returned more than one row as there are more than one employee whose manager's empcode is '7698'.

Subqueries that return more than one row

When a query returns more than one row we need to use multirow comparision operator.

Example:-

List the names of the employees, who have got an increment:

```
SELECT ename FROM emp
WHERE empno IN (SELECT empno FROM incr);
```

Here, the inner query returns multiple values, hence the IN operator is used instead of a relational operator.

List the names of the employees, who earn lowest salary in each department:

```
SELECT ename, sal, deptno FROM emp
WHERE sal IN (SELECT MIN (sal) FROM emp GROUP BY deptno);
```

Here the inner query has a GROUP BY clause. This means it may return more than one value. In this case, the IN operator must be used because it expects a list of values.

The following points should be kept in mind while writting subqueries:

1. The inner query must be enclosed in parentheses.
2. The inner query must on the right hand side of the condition.
3. The subquery may not have an order by clause.
4. The ORDER BY clause appears at the end of the main select statement.

5. Subqueries are always executed from the most deeply nested to the least deeply nested, unless they are correlated subqueries.

Correlated Subquery

A correlated subquery is a nested subquery which is executed once for each 'candidate row' considered by the main query and which executed uses a value from a column in the outer query.

In a correlated subquery, the column value used in inner sub query refers to the column value present in the outher query forming a subquery. The subquery is executed repeatedly once for each row of the main (outer) query table.

List the employee numbers and names, who have got more than 1 increments:

```
SELECT empno ename FROM emp
WHERE 1 <
        (SELECT COUNT (*) FROM incr
        WHERE empno = emp. empno);
```

EMPNO	ENAME
.....	.....
7369	SMITH
7788	SCOTT
7900	JAMES
7934	MILLER

List employee details who earn salary greater than the average salary for their department.

```
SELECT empno, ename, sal, deptno
FROM emp e
WHERE sal > (select AVG (sal) FROM emp WHERE deptno = e. deptno);
```

EMPNO	ENAME	SAL	DEPTNO
.....	.....	.....	.....
7839	KING	5000	10
7566	JONES	2975	20
7788	SCOTT	3000	20
7902	FORD	1600	30
7698	BLAKE	2850	30

5 rows selected.

Remember, a correlated subquery is signalled by a column name, a table name or table alias in the WHERE clause that refers to the value of a column in each candidate row ofthe outer select.

Also the correlated subquery executes repeatedly for each candidate row in the main

Notes

query. Correlated subquery is used to answer multipart questions whose answer depends on the value of each row of the parent query. The inner select is normaly executed once for each candidate row.

Self Learning Exercise:-

4. **How are nested queries different from joined queries ?**

Using Special Operators in Subqureies:-

Some Special operators used in subqueries are:

EXISTS

ANY SOME

ALL Operators

EXISTS

This operator is used to check the existence of values

This operator produces a Boolean result

It takes a subquery as an argument and evaluates it to True, if it produces any output or False, if it does not

ANY, SOME and ALL

Used along with the relational operators

Similar to IN operator, but only used in subqueries

The SOME and ANY operator can be used interchangeably

EXISTS operator

The EXISTS operator is frequenly used with correlated subqueries. It tests whether a value is there (NOT EXISTS ensure for nonexistence of values.) If the value exists it returns TRUE, if it does not exists it returns FALSE.

NOT EXISTS operator is more relible if the subquery returns any NULL values.

Examples:

List all employee who have atleast one person reporting to them.

```
SELECT empno, ename, job, deptno
FROM emp e
WHERE EXISTS        (SELECT empno from emp
                     WHERE emp. mgr = e. empno)
        ORDER BY empno;
```

empno	ename	job	deptno

Notes

.....	.....	.....	.....
7566	JONES	MANAGER	20
7698	BLAKE	MANAGER	30
7782	CLARK	MANAGER	10
7788	SCOTT	ANALYST	20
7839	KING	PRESISENT	10
7902	FORD	ANALYST	20

list the employee details if and only if more than 10 employees are present in department number 10:

```
SELECT * FROM emp
WHERE DEPTNO = 10 AND EXISTS (SELECT COUNT (*) FROM emp
                    WHERE deptno = 10
                    GROUP BY deptno
                    HAVING COUNT (*) > 10);
```

list the name of employee from the employee table where the increment amount is greater than 1000 and the number of employees receiving the same increment is greater than 5:

```
SELECT ename FROM emo
WHERE empno IN(SELECT empno FROM incr
                    WHERE amt > 1000
                    AND EXISTS (SELECT COUNT (*)
                              FROM incr GROUP BY amt
                              HAVING count (*) > 5));
```

List all the employees datails who do not manage any one.

```
SELECT ename, job from emp e
where not exists (select mgr frm emp where mgr = e. empno);
```

The output is

ename	job
.....	.....
SMITH	CLERK
ADAMS	CLERK
ALLEN	SALESMAN
WARD	SALESMAN
MARTIN	SELESMAN
TURNER	SELESMAN

Notes

JAMES	CLERK
MILLER	CLERK

Self Learning Exercise

5. What will be the output if we use NOT IN operator instead of NOT EXISTS in the above query?

ANY operator

The ANY operator compares the lowest value from the set.

List the employee names whose salary is greater than the lowest of an employeebelonging to de- partment number 20:

SELECT ename FROM emp

WHERE sal > ANY (SELECT sal FROM emp WHERE deptno = 21);

List the employee details of those employees whose salary is greater than any of the managers:

SELECT EMPNO, ENAME SAL FROM EMP WHERE SAL > ANY (SELECT SAL FROM EMP WHERE JOB = 'MANAGER');

ALL Operator

In case of All operator the predicate is true if every value selected by the subquery satisfies the condition in the predicate of the outer query.

Example:-

List the employee names whose salary is greater than the highest salary of all employee belonging to department number 20:

SELECT ename FROM emp

WHERE sal > ALL(SELECT sal FROM emp WHERE deptno = 20);

The inner query return salary of all employees who belong to department number 20. The outer query selects employee name of that employee whose salary is greater than all the employees' salary who belong to department number 20.

List the details of the employee earning more than the highest paid MANAGER:

SELECT empno, ename, sal FROM emp WHERE sal > ALL (SELECT sal FROM emp WHERE job = 'MANAGER');

3.20 INSERTING ROWS; VIEWS; SNAPSHOTS

In this part, we'll continue to work on views using the sample database and data that we created so far.

Notes

To briefly summarize the series, in the first part about Creating views in SQL Server, the idea was to get familiar with the CREATE VIEW SQL syntax, all the different things we can do with views, and creating a really basic view.

In the second part about Modifying views in SQL Server, we upped the difficulty a little bit and created a more complex view with aggregates in it. Furthermore, we got familiar with the ALTER VIEW statement used to change the output by changing the definition and structure of a query.

Although this section can be read independently from the first two, it's highly advisable to head over and read the previous two parts to get the full picture and because it will be easier to follow along.

Now it's time to start using Data Manipulation Language (DML) that is used to manipulate data itself and see how we can insert data into a table through a view.

However, before we actually insert data through a view, let's see how we can rename a view. I also want to show you one neat thing that we can do WITH CHECK OPTION which is a part of the CREATE VIEW SQL syntax. This option can be useful when inserting data through a view which you'll see later in this article.

Renaming views

Views are renamed using the sp_rename system stored procedure. By definition, this SP is used for changing the name of a user-created object in the current database. Having said that, changing any part of an object name, including views, can break scripts and dependencies, which is why it's not recommended to use this statement to rename views or any other user-created object. The smart thing to do would be to drop the object and re-create it with the new name using the CREATE VIEW SQL statement.

Renaming views using a third-party software solution

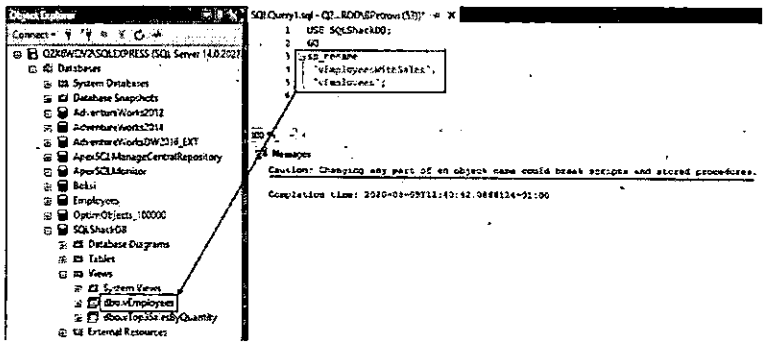
If you want to play it safe, I'd suggest a third-party software solution for safely renaming SQL objects. I, personally, use SQL Server Management Studio and Visual Studio add-ins for SQL Server called ApexSQL Search and ApexSQL Refactor. Both add-ins have a feature called Safe rename that renames tables, procedures, views, functions, and columns without breaking dependencies.

This is a series about using T-SQL, so we're going to do it using the stored procedure mentioned above. Make sure that you're are connected to the SQLShackDB database or specify the database name in the USE statement to avoid getting errors from SQL Server. The example below changes the database context to the SQLShackDB database and changes the view's name:

```
USE SQLShackDB;
GO
sp_rename
'vEmployeesWithSales',
'vEmployees';
```

Notes

If we head over to Object Explorer and refresh the Views folder, we should see that the name changed to vEmployees. There's also a warning from SQL Server about changing any part of an object name that I've mentioned earlier:

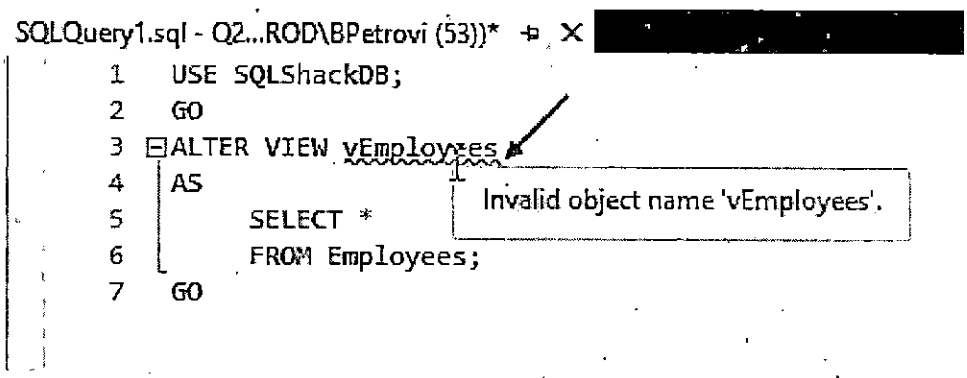


Now, let's completely change the definition of the view by executing the code from below:

```
USE SQLShackDB;
GO
ALTER VIEW vEmployees
AS
    SELECT *
    FROM Employees;
GO
```

We could have just dropped the view and use the CREATE VIEW SQL statement to re-create it with the new definition, but we can't break script and dependencies with this simple view, so it's safe to just rename it using the stored procedure.

We can't help but notice the name of the view underlined in red. We've all seen this at some point. This indicates an error in SSMS, in this particular case about an invalid object name, but it's not:

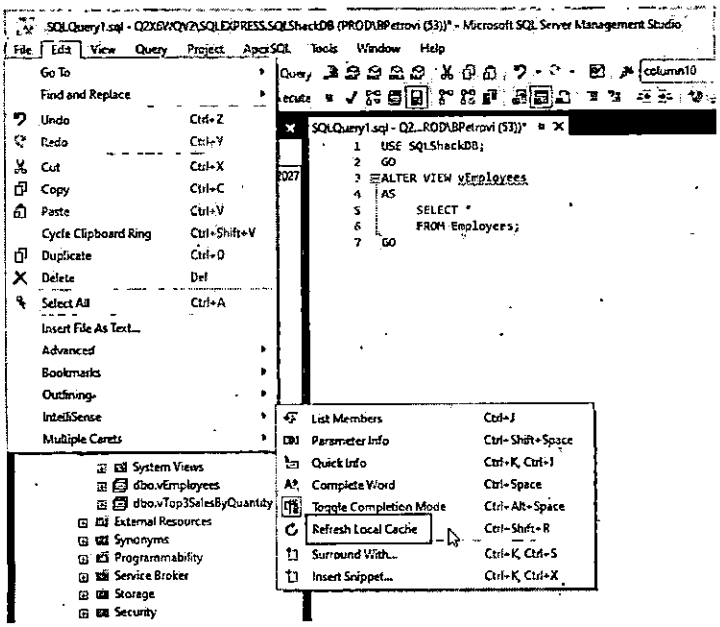


This is coming from SSMS's IntelliSense. We just changed the name of the view and the metadata has not been updated yet. The same would have happened to the CREATE VIEW SQL statement.

Right after the creation of an object, when queried, it's very likely for the view as the source to be underlined in red until the cache is refreshed.

An easy way to fix this, although it's not an actual error (the above script can be executed without throwing an error), is to refresh the IntelliSense's cache manually.

In SSMS, go to Edit on the main menu. Expand the Intellisense options and select Refresh Local Cache as shown below:



As soon as we refresh the cache, metadata is up-to-date (sometimes it takes a couple of seconds), and Intellisense removes the red squiggles line.

So, let's move on and execute the ALTER VIEW statement to change also the definition of the view to a simple SELECT statement that retrieves all the columns from the Employees table:

```
SQLQuery1.sql - Q2...ROD\BPetrovi (53))*
      1   USE SQLShackDB;
      2   GO
      3  ALTER VIEW vEmployees
      4   AS
      5        SELECT *
      6        FROM Employees;
      7   GO

100 %
Messages
Commands completed successfully.

Completion time: 2020-03-09T15:02:27.7880825+01:00
```

Notes

So, if we query the vEmployees view now, it should return all columns from the Employees table as shown below:

```
USE SQLShackDB;
GO
 SELECT * FROM vEmployees;
```

Data modifications through views

Finally, let's see how we can do data modifications through the vEmployees view. This is a very neat thing that we can do. Just think about it. We started with the CREATE VIEW SQL statement, we then created a very simple view, and now we're going to use that view to insert a record into our Employees table.

Let's say that we need to insert an employee through our view. The below code is just an example of inserting data through a view using the SELECT statement:

```
USE SQLShackDB;
GO
 INSERT INTO vEmployees
      SELECT 3,
            'Bojan',
            NULL,
            'Petrovic',
            'Author',
            '1/1/2017',
            2080,
            '100000.00';
```

To explain the INSERT INTO statement, I'm simply using SELECT after the name of our view, which is a very simple way to insert data into tables as we're inserting new data based on the result of the SELECT statement.

The vEmployees view has 8 columns. I like to expand the Columns folder of the view in Object Explorer just to see which columns cannot be a nullable value and also because it's easier to write the SELECT statement and match all values one-by-one. After executing the above script, we should have a "1 row affected" message returned meaning that the new record went into the table successfully:

SQL Snapshots

Snapshot is a recent copy of the table from the database or a subset of rows/columns of a table. The SQL statement that creates and subsequently maintains a snapshot normally reads data from the database residing server. A snapshot is created on the destination system with the create snapshot SQL command. The remote table is immediately defined and populated from the master table.

Notes

These are used to dynamically replicate data between distributed databases. Two types of snapshots are available.

Simple snapshots

Complex snapshots

Simple snapshot :

In simple snapshot, each row is based on a single row in a single remote table. This consists of either a single table or a simple SELECT of rows from a single table.

Example:

```
CREATE SNAPSHOT emp_snap
as select * from emp;
```

Complex snapshot :

In complex snapshot, a row may be based on more than one row in a remote table via GROUP BY operation or result of Multi-Table Join. This consists of joined tables, views, or grouped and complex SELECT statement queries.

Example:

```
CREATE SNAPSHOT sampleSnps1
AS SELECT student.rollno, student.name
FROM student
UNION ALL
SELECT new_student.rollno, new_student.name
FROM new_student;
```

Advantages :

Response time is improved when local read-only copy of table exists.

Once snapshot is built on remote database, if node containing data from which the snapshot is built is not available. Snapshot can be used without need to access the unavailable database.

Ease network loads.

Data subsetting.

Disconnected computing.

Mass deployment.

Disadvantages :

Snapshots are not reachable when primary database goes offline.

It does not support full text indexing.

Notes

Snapshot runs out of disk if data changes frequently faster.

As no.of snapshots increases, disk space becomes problematic.

Applications :

- Protects data.
- Maintains history of data.
- Used in testing application software.
- Used in data mining.
- Recovers data when information is lost because of human error or corruption of data.

3.21 INDEXING AND SEQUENCING

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.

For example, if you want to reference all pages in a book that discusses a certain topic, you first refer to the index, which lists all the topics alphabetically and are then referred to one or more specific page numbers.

An index helps to speed up SELECT queries and WHERE clauses, but it slows down data input, with the UPDATE and the INSERT statements. Indexes can be created or dropped with no effect on the data.

Creating an index involves the CREATE INDEX statement, which allows you to name the index, to specify the table and which column or columns to index, and to indicate whether the index is in an ascending or descending order.

Indexes can also be unique, like the UNIQUE constraint, in that the index prevents duplicate entries in the column or combination of columns on which there is an index.

The CREATE INDEX Command

The basic syntax of a CREATE INDEX is as follows.

```
CREATE INDEX index_name ON table_name;
```

Single-Column Indexes

A single-column index is created based on only one table column. The basic syntax is as follows.

```
CREATE INDEX index_name
ON table_name (column_name);
```

Unique Indexes

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table. The basic syntax is as follows.

CREATE UNIQUE INDEX index_name

on table_name (column_name);

Composite Indexes

A composite index is an index on two or more columns of a table. Its basic syntax is as follows.

CREATE INDEX index_name

on table_name (column1, column2);

Whether to create a single-column index or a composite index, take into consideration the column(s) that you may use very frequently in a query's WHERE clause as filter conditions.

Should there be only one column used, a single-column index should be the choice. Should there be two or more columns that are frequently used in the WHERE clause as filters, the composite index would be the best choice.

Implicit Indexes

Implicit indexes are indexes that are automatically created by the database server when an object is created. Indexes are automatically created for primary key constraints and unique constraints.

The DROP INDEX Command

An index can be dropped using SQL DROP command. Care should be taken when dropping an index because the performance may either slow down or improve.

The basic syntax is as follows –

DROP INDEX index_name;

You can check the INDEX Constraint chapter to see some actual examples on Indexes.

When should indexes be avoided?

Although indexes are intended to enhance a database's performance, there are times when they should be avoided.

The following guidelines indicate when the use of an index should be reconsidered.

- Indexes should not be used on small tables.
- Tables that have frequent, large batch updates or insert operations.
- Indexes should not be used on columns that contain a high number of NULL values.
- Columns that are frequently manipulated should not be indexed.

Sequences

Sequences are a feature that some DBMS products implement to provide users with a mechanism to generate unique values - the Sequence ensures that each call to it returns a unique value. This is particularly important when the Sequence's result is used as a Primary Key. These can be generated with a schema for loading onto the DBMS server.

Notes

Sequences are provided so that database users are not forced to implement their own unique value generator. Not all DBMS products support Sequences; those that do not instead provide functionality for columns to be initialized with an incrementing value.

In Enterprise Architect, Sequences can be modeled in one of two ways:

- As individual objects (the default method) or
- As Operations in a container

Functionally the two methods result in the same DDL being produced. The main difference is visual - by having several Operations in one container, you have fewer elements and less clutter on the diagram.

Individual objects

Sequences modeled as individual objects are UML Classes with the stereotype «dbsequence»; you create these either:

Within the Database Builder or

By dragging the 'Sequence' icon from the 'Data Modeling' Toolbox pages onto a diagram

Add a Database Sequence using the Database Builder

Steps:

- Open the Database Builder.
- .Load or create a Data model.
- Right-click on the Sequences Package and select 'Add New Sequence'.
- Overtype the default name with the appropriate name for the Sequence, and press the Enter key.
- Double-click on the new Sequence, or right-click on it and select 'SQL Object Properties'.

The 'SQL Object Editor' dialog displays.

Add a Database Sequence to a diagram

Steps:

- Open your Data Modeling diagram and, if necessary, display the 'Data Modeling' page of the Diagram Toolbox (click on Search to display the 'Find Toolbox Item' dialog and specify 'Data Modeling').
- Drag the 'Sequence' icon onto the diagram.
- Database Sequence

This generates the Sequence element:

- Database sequence element in Sparx Systems Enterprise Architect.
- Right-click on the new Sequence element and select 'SQL Object Properties'.

- The 'SQL Object Editor' dialog displays.
- SQL Object Editor

The 'SQL Object Editor' dialog is shared by a number of SQL-based database objects (Views, Procedures and Functions); it helps you to manage the various properties of the SQL-based object.

Option

- If it has already been set, the default database type displays.
- If the default has not been set, or you want to change the database type for this Sequence, click on the drop-down arrow and select the target DBMS to model.
- Set the Database Type

Notes: If necessary, type in a comment on the current Sequence.

Definition

Type the full SQL Sequence definition including the CREATE SEQUENCE syntax.

The code editor provides Intelli-sense for basic SQL keywords, functions and names of all objects in the current data model.

Operations in a Container

Database Sequences modeled as operations have a container object, this being a UML Class with the stereotype «sequences» (with an 's' on the end). Each Sequence is an operation with the stereotype «sequ». The system provides a dedicated Maintenance window through which the modeler can easily manage the Sequences defined as operations.

3.22 TRANSACTION PROCESSING IN DBMS

A transaction is a program including a collection of database operations, executed as a logical unit of data processing. The operations performed in a transaction include one or more of database operations like insert, delete, update or retrieve data. It is an atomic process that is either performed into completion entirely or is not performed at all. A transaction involving only data retrieval without any data update is called read-only transaction.

In a system with a number of simultaneous transactions, a schedule is the total order of execution of operations. Given a schedule S comprising of n transactions, say T1, T2, T3..... Tn; for any transaction Ti, the operations in Ti must execute as laid down in the schedule S.

A single DBMS operation as viewed by an user, for example, to update the grade of a student in the relation ENROL (Student_Name, Course, Grade), involves more than one task. Since the data resides on a secondary nonvolatile storage medium, it will have to be brought into the Volatile primary memory for manipulation.

This requires that the data be transferred between secondary storage and primary storage. The transfer is usually performed in blocks of the implementation-specified size. The transfer task consists of locating the blocks of in the secondary storage device containing the required

Notes

tuple (which may be preceded by searching an index), obtaining the necessary locks on the block or the tuple involved in the update, and reading in this block.

This task is followed by making the update to the tuple in memory, which in turn is followed by another transfer task, written the tuple back to secondary device, and releasing the locks.

In order to reduce the number of accesses to disk, the blocks are read into blocks of main memory called buffers. We can thus assume that a program performs input/ output using, for example, the get and put operations, and the system transfers the required block from secondary memory to main memory using the read and write operations. The block read (Write) tasks need not be performed in case the system uses buffered input (output) and the required data (space) is already in the primary memory buffer. In such a case the get (put) operation of the program can input (output) the required data from (to) the appropriate buffer. If the required data is not in the buffer, the buffer manager does a read operation and obtains the required data, after which the data is input from the buffer to thte program executing the get statement. If there is no more space left in the buffer, the put operation causes the buffer to be writtern to the secondary storage (with a write) and then the put operation transfers the data from main memory to the space made available in the buffer.

The above DBMS operation of changing the grade of a student in a given course initiated by a user and appearing to her or him as a single operation actually requires a number of distinct tasks or steps to be performed by the DBMS. This is illustrated by the skeleton program given on the next page.

In this program the comment indicates the definition of the action update ENROL of the record for a given student in a given course; this action is being referenced later with the keywords commit and rollback. The statements defined for the update operation are assumed to modify a temporaty copy of the selected portion of the database (the main memory copy of the block of nonvolatile storage containing the tuple for the relation ENROL). Here we are using error to indicate whether there are any errors during the execution of the statements defined for the action update ENROL. If there were any errors, we want to undo any changes made to the database by the statements defined for the update action. This involves simply discarding the temporary copy of the affected portion of the database. The database itself is not changed if a temporary copy of the database is being used. If there were no errors, we want the changes made by the update operations to become permanent by being reflected in the actual database.

Each high level operation can be divided into a number of low level tasks or operations. For example, a data update operation can be divided into three tasks –

- read_item() – reads data item from storage to main memory.
- modify_item() – change value of item in the main memory.
- write_item() – write the modified value from main memory to storage.

Database access is restricted to read_item() and write_item() operations. Likewise, for all transactions, read and write forms the basic database operations.

Transaction Operations

The low level operations performed in a transaction are –

- begin_transaction – A marker that specifies start of transaction execution.
- read_item or write_item – Database operations that may be interleaved with main memory operations as a part of transaction.
- end_transaction – A marker that specifies end of transaction.
- commit – A signal to specify that the transaction has been successfully completed in its entirety and will not be undone.
- rollback – A signal to specify that the transaction has been unsuccessful and so all temporary changes in the database are undone. A committed transaction cannot be rolled back.

Transaction States

A transaction may go through a subset of five states, active, partially committed, committed, failed and aborted.

- Active – The initial state where the transaction enters is the active state. The transaction remains in this state while it is executing read, write or other operations.
- Partially Committed – The transaction enters this state after the last statement of the transaction has been executed.
- Committed – The transaction enters this state after successful completion of the transaction and system checks have issued commit signal.
- Failed – The transaction goes from partially committed state or active state to failed state when it is discovered that normal execution can no longer proceed or system checks fail.
- Aborted – This is the state after the transaction has been rolled back after failure and the database has been restored to its state that was before the transaction began.

The following state transition diagram depicts the states in the transaction and the low level transaction operations that causes change in states.

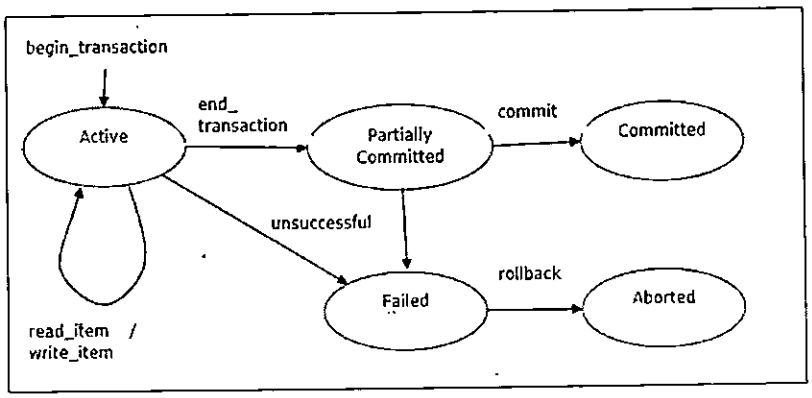


Figure 4. Desirable Properties of Transactions

Any transaction must maintain the ACID properties, viz. Atomicity, Consistency, Isolation, and Durability.

Notes

- Atomicity – This property states that a transaction is an atomic unit of processing, that is, either it is performed in its entirety or not performed at all. No partial update should exist.
- Consistency – A transaction should take the database from one consistent state to another consistent state. It should not adversely affect any data item in the database.
- Isolation – A transaction should be executed as if it is the only one in the system. There should not be any interference from the other concurrent transactions that are simultaneously running..
- Durability – If a committed transaction brings about a change, that change should be durable in the database and not lost in case of any failure.

3.23 CONCURRENT EXECUTION

Larger computer systems are typically used by many users in a multiprogramming mode; programs are executed concurrently.(multiple programs execute simultaneously); One reason for the use of multiprogramming is to exploit the different characteristics of the various program to maximize the utilization of the equipment; thus, while one program awaits the completion of an input/output operation, the processor can be used to do the computation of another program.

Another reason for choosing multiprogramming is the need to share a resource by these different programms : a database is such a shared resource. The primary objective of the database system (at least on a large mainframe) is to allow many users and application programs to access data from the database in a concurrent manner.

The sharing of the database for read-only access does not cause any problem, but if one of the transactions runnning concurrently tries to modify some data-item, it could lead to inconsistencies.

Furthermore, if more than one transaction is allowed to simultaneously modify a data-item in the database, it could lead to incorrect values for the data-item and an inconsistent database.

Such would be the result even if each of the transactions were correct and a consistent database would remain so if each of agents access the airline reservations system simultaneously to see if a seat is available on a given flight; if both agents make a reservation against the last available seat on that flight, overbooking of the flight would result. This potential problem of leaving the database in an inconsistent state with concurrently running transactions would be able to access only disjoint data for modificatons.

One method of enforcing mutual exclusion is by some type of locking mechanism that locks a shared resource (for example a data-item) used by a transaction for the duration of its usage by the transaction. The locked data-item can only be used by the transaction that locked it.

The other concurrent transactions are locked out and have to wait their turn at using the data-item. However, a locking scheme must be fair. This requires that the lock manager, which is the DBMS subsystem managing the locks, must not cause some concurrent transaction to be permanently blocked from using the shared restore. This is reformed to is avoiding the

starvation or livelock situation.

The other danger to be avoided is that of deadlock, wherein a number of transactions are waiting in a circular chain, each waiting for the release of resources held by the next transaction in the chain.

In other methods of concurrency control, some form of a priori ordering with a single or many versions of data is used. These methods are called timestamp ordering and multiversion schemes. The optimistic approach, on the other hand, assumes that the data-items used by concurrent transactions are most likely be disjoint.

Concurrency and Possible Problems:

In the last chapter we stressed that a correct transaction, when completed, leaves the database in a consistent state provided that the database was in a consistent state at the start of the transaction. Nevertheless, during the life of a transaction, the database could be inconsistent, although if the inconsistencies are not accessible to other transactions, they would not cause a problem.

In the case of concurrent operations, where a number of transactions are running and using the database, we cannot make any assumptions about the order in which the statements belonging to different transactions will be executed. The order in which these statements are executed is called a schedule.

Consider the two transactions in Figure 5 Each transaction reads some data-item, performs some operation on the data-item that could change its value, and then writes out the modified data-item.

In figure 5 and n subsequent example in this chapter, we assume that the read operation reads in the database value of the named variable to a local variable with an identical name. Any modifications by a transaction are made on this local copy.

The modifications made by the transactions are indicated by the operators f1 and f2 in Figure 5 These modification are not reflected in the database until the write operation is executed, at which point the modifications in the value of the named variable are said to be commited. ineffect the write operation is asignal for committing the modifications and reflecting the changes to the physical database.

Transaction T_1	Transaction T_2
Read (*Avg_faculty_Salary*)	Read (*Avg_Staff_Salary*)
Avg_Faculty_Salary : =	*Avg_Staff_Salary* :
f_1(*Avg_Faculty_Salary*)	f_2(*Avg_Staff_Salary*)
Write(*Avg_Faculty_Salary*)	Write(*Avg_Staff_Salary*)

Figure 5: Two concurrent transactions

Notes

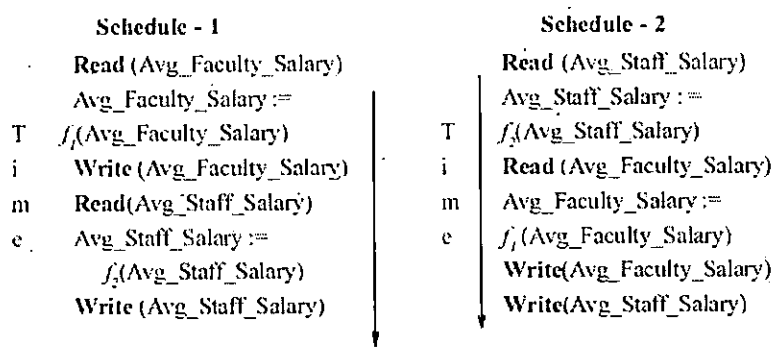


Figure 6: Possible interleaving of concurrent transactions of Figure 8

Figure 6 gives two give two possible schedules for executing the transactions of Figure 3.4 in an interleaved manner. Since the transactions of Figure 5 are accessing and modifying distinct data-items, (Avg_Faculty_salary, Avg_staff_Salary), there is no problem in executing these transactions concurrently. In other words, regardless of the order of interleaving of the statements of these transactions, we will get a consistent database on the termination of these transactions.

Lost Update Problem

Consider the transactions of Figure 7 These transactions are accessing the same data-item A. Each of the transactions modifies the data-item and writes it back. Again let us consider a number of possible interleavings of the execution of the statements of these transactions. These schedules are given in Figure 8.

Starting with 200 as the initial value of A, let us see what the value of A would be if the transactions are run without any interleaving. In other words, the transactions are run to completion, without any interruptions, one at a time in a seial manner.

If transaction T3 is run first, then at the end of the transaction the value of A will have changed from 200 to 210. Running transaction T4 after the completion of T3 will change the value of A from 210 to 231 Running the transactions in the order T4 followed by T3 result in a final value for A of 230.

The result obtained with neither of the two interleaved execution schedules of figure 8 agrees with either of the results of executing these same transactions serially. Obviously something is wrong !

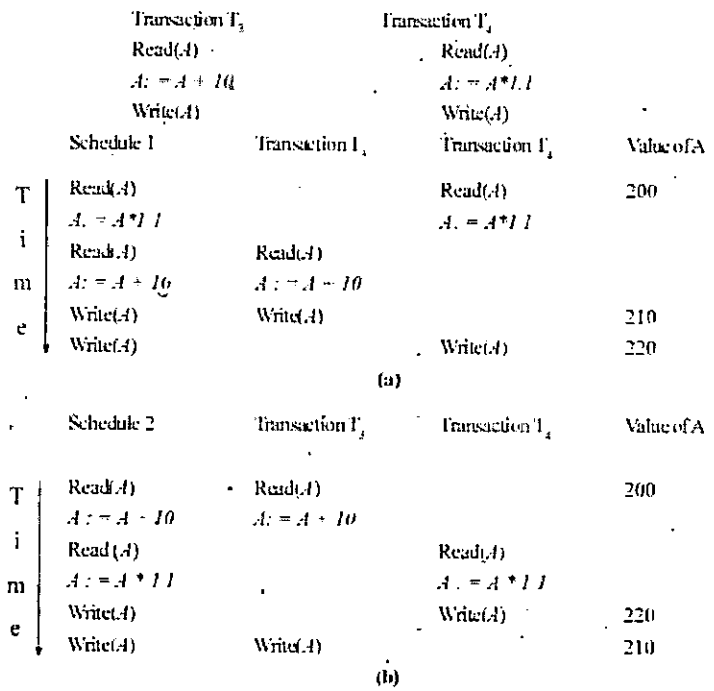


Figure 7 : Two Transactions Modifying the Same Data-Item

In each of the schedules given in Figure 7 we have lost the update made by one of the transactions. In schedule 1, the update made by transaction T3 is lost; in schedule 2, the update made by transaction T4 is lost. Each schedule exhibits an example of the so-called lost update problem of the concurrent execution of a number of transactions.

It is obvious that the reason for the lost update problem is that even though we have been able to enforce that the changes made by one concurrent transaction are not accessible by the other transactions until it commits, we have not enforced the atocity requirement. This demands that only one transaction can modify a given data-item at a given time and other transactions should be locked out from even viewing the unmodified value (in the database) until the modifications (being made to a local copy of the data) are committed to the database.

Inconsistent Read Problem

The lost update problem was caused by concurrent modifications of the same data-item. However, concurrency can also cause problems when only one transaction modifies a given set of data while that set of data is being used by other transaction.

Transaction T5	**Transaction T6**
Read(A)	Sum : = 0
A : = A-100	Read(A)
Write(A)	Sum : = Sum + A
Read(B)	Read(B)

Notes

B : = B + 100 Sum : = Sum + B

Write(B) Write(Sum)

Figure 8 : Two transactions; one modified while the other reads

Consider the transactions of Figure 8 Suppose A and B represent some data-items containing intege valued data, for example, two accounts in a bank (or a quantity of some part X in two different locations, etc.).

Let us assume that transaction T5 transfers 100 units from A to B. Transaction T6 is concurrently running and it wants to dind the total of the current values of data items A and B (the sum of the balance in case A and B represent two accounts, or the total quantity of part X in the two different locations, etc.)

Figure 9 gives a possible schedule for the concurrent execution of the transactions of Figure 9 with the intial value of A and B being 500 and 1000, respectively. We notice from the schedule that transaction T6 uses the value of A before the transfer was made, but it uses the modified value of B after the transfer. T

he result is that transaction T6 erroneously determines the total of A and B as being 1600 instead of 1500. We can also come up with another schedule of the concurrent execution of these transactions that will give the total of A and B as 1400, and of course other schedules that will give the correct answer.

	Schedule	Transaction T5	Transaction T6	Value of Database items		
				A	B	Sum
	Read (A)	Read (A)		500	1000	
	Sum : - 0		Sum : = 0			0
T	Read (A)		Read (A)			
i	A : - A - 100	A : - A - 100				
m	Writer (A)	Writer (A)		400		
e	Sum :- Sum+A		Sum : = Sum+A			500
	Read(B)	Read (B)				
	B : + B + 100	B : + B + 100				
	Write (B)	Write (B)				
	Read (B)		Read (B)		1100	
	Sum : - Sum + B		Sum : = Sum + B			
	Write (Sum)		Write (Sum)			1600

Figure 9 : Example of Inconsistent Reads

The reason we got an incorrect answer in the schedule of Figur 12.9 was because that transaction T6 was using values of data-items A and B while they were being modified by transaction T5. Locking out transaction T6 from these data-items indicidually would not have solved the problem of the inconsistent read.

The problem would have been resolved in this example only if transaction T5 had not released the exclusive usage of the data item A after locking data-item B. We discuss this scheme, called two phase locking.

Semantics of Concurrent Transactions

In concurrent operations, where a number of transactions are running and modifying parts

of the database, we not only have to hide the changes made by a transaction from other transactions, but we also have to make sure that only one transaction has exclusive access to these data-items for at least the duration of the original transaction's usage of the data-items. This requires that an appropriate locking mechanism be used to allow exclusive access of these data-items to the transaction requiring them. In the case of the transactions of Figure 12.6 no such locking was used with the consequence that the result is not the same as the result we would have obtained had these transactions run consequently.

Now let us see why the results obtained when we run two transactions, one after the other, need not be the same for different orderings. The modification operations performed by two transactions are not necessarily commutative.

The operations A : = (A + 10) + 20 give the same result as A : = (A + 20) + 10 for the same initial value for A (which is assumed to be an integar valued data-item); this is so because the addition operation is commutative. Similarly, (A * 10) * 20 = (A * 20) * 10.

However, commuting the order of operations, as illustrated by the following expressions, does not always give the same result :

Salary : = (Salary + 1000) * 1.1

Salary : = (Salary * 1.1) + 1000

In the above example we have two transformations. In the first the salary is initially modified by adding 1000 to it and then the result is augmented by 10% to give the revised Salary. In the second the Salary is first augmented by 10% and then 1000 is added to the result, which becomes the revised Salary.

The reasonable approach, to make sure that the intended result is obtained in all cases (i.e. to make sure that transaction Ti is completed before transaction Tj is run), would be to code the operations in a single transaction and not to divide the operations into two or more transactions.

Thus, if the above set of operations on Salary were written as two transactions as given below, we cannot be sure which of the above two results would be obtained with their concurrent execution.

Transaction Ti	Transaction Tj
Read Salary	Read Salary
Salary : = Salary * 1.1	Salary : = Salary + 1000
Write Salary	Write Salary

In efffect, the division of a transaction into interdependent transactions run serially in the wrong order would give erroneous results. Furthermore, these interdependent transactions must not be run concurrently, otherwise the concurrent execution will lead to results that could be incorrect again and not agree with the result obtained by any serial execution of the same transactions. It is logical error to divide a single set of operations into two or more transactions. We assume hereafter that transactions aresemantically correct.

From the definition of a transaction, we see the status of a transaction and the observation of its actions is not visible from outside until the transaction terminates. Once a transaction ends, the user may be notified of its success or failure and the changes made by the transaction are accessible. In order for a transaction to achieve these characteristics, it should have the

Notes

properties of atomicity, consistency, isolation and durability (ACID).

The atomicity property of a transaction implies that it will run to completion as an indivisible unit, at the end of which either no changes have occurred to the database or the database has been changed in a consistent manner. At the end of a transaction the updates made by the transaction will be accessible to other transactions and processes outside the transaction.

The consistency property of a transaction implies that if the database was in a consistent state before the start of a transaction, then on termination of a transaction the database will also be in a consistent state.

The isolation property of a transaciton indicates that actions performed by a transaction will be isolated or hidden from outside the transaction untill the transaction terminates. This property gives the transaction a measure of relative independence.

The durability property of a transaction ensures that the commit action of a transaction, on its termination, will be reflected in the database. The permanence of the commit action of a transaction requires that any failures after the commit operation will not cause loss of the updates made by the transaction.

Serializability

A serializable schedule of 'n' transactions is a parallel schedule which is equivalent to a serial schedule comprising of the same 'n' transactions. A serializable schedule contains the correctness of serial schedule while ascertaining better CPU utilization of parallel schedule.

Equivalence of Schedules

Equivalence of two schedules can be of the following types:

- Result equivalence – Two schedules producing identical results are said to be result equivalent.
- View equivalence – Two schedules that perform similar action in a similar manner are said to be view equivalent.
- Conflict equivalence – Two schedules are said to be conflict equivalent if both contain the same set of transactions and has the same order of conflicting pairs of operations.

We assume that the database is in a consistent state before a transaction starts. Atransaction starts when the first statement of the transaction is executted; it becomes active and we assume that it is in the modify state, when it modifies the database.

At the end of the modify state, there is a transaction into one of the following states start to commit, abort, or error. If the transaction completes the modification state satisfactorily, it enters the start-to-commit state where it instructs the DBMS to reflect the changes made by it into the database, the transaction is said to be in the commit state and from there the transaction is terminated, the database once again being in a consistent state. In the interval of time between the start-to-commit state and the commit state, some of the data changed by the transaction in the buffers may or may not have been propagated to the database on the nonvolatile storage.

There is a possibility that all the modifications made by the transaction cannot be

propagated to the database due to conflicts or hardware failures. In this case the system forces the transaction to the abort state.

The abort state could also be entered from the modify state if there are system errors for example, division by zero or an unrecoverable parity error. In case the transaction detects an error while in the modify state, it decides to terminate itself (suicide) and enters the error state and then, the rollback state.

If the system aborts a transaction, it may have to initiate a rollback to undo partial changes made by the transaction. An aborted transaction that made no changes to the database is terminated without the need for a rollback, hence there are two paths from the abort state to the end of the transaction.

A transaction that, on the execution of its last statement, enters the start to commit state and from there the commit state is guaranteed that the modifications made by it are propagated to the database.

The transaction outcome can be either successful (if the transaction goes through the commit state), suicidal (if the transaction goes through the rollback state), or murdered (if the transaction goes through the abort state). In the last two cases, there is no trace of the transaction left in the database, and only the log indicates that the transaction was ever run.

Any messages given to the user by the transaction must be delayed till the end of the transaction, at which point the user can be notified as to the success or failure of the transaction and in the latter case, the reasons for the failure.

In the above examples consider the transactions are independent. An execution schedule of these transactions as shown in figure 10 is called a serial execution. In a serial execution, each transaction runs to completion before any statements from any other transaction are executed.

In Schedule A given in Figure 10 a, transaction T3 is run to completion before transaction T4 is executed. In schedule B, transaction T4 is run to completion before transaction T3 is started. If the initial value of A in the database were 200, Schedule A would result in the value of A being changed to 231. Similarly, Schedule B with the same mitial value of A would give a result of 230.

This may seem odd, but in a shared environment, the result obtained by independent transactions that modify the same data-item always depends on the order in which these transactions are run; and any of these results is considered to be correct.

If there are two transactions and if they refer to and use distinct data-items, the result obtained by the interleaved execution of the statements of these transactions would be the same regardless of the order in which these statements are executed (Provided there are no other concurrent transactions that refer to any of these data-items).

In this unit, we assume that the concurrent transactions share some data-items, hence we are interested in a correct ordering of execution of the statements of these transactions.

Notes

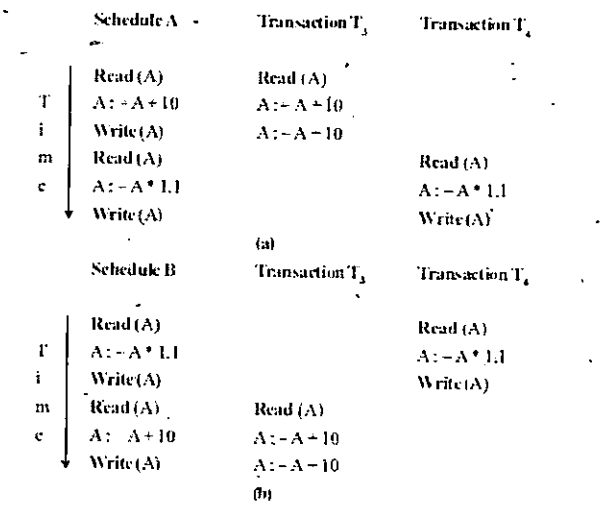


Figure 10 : Two serial Schedules

A nonserial schedule wherein the operations from a set of concurrent transactions are interleaved is considered to be serializable if the execution of the operations in the schedule leaves the database in the same state as some serial execution of these transactions. With two transactions, we can have at most two district serial schedules, and starting with the same state of the database, each of these serial schedules could give a different final state of the database. Starting with an initial value of 200 for A, the serial schedule illustrated in Figure 10 a would give the final value of A as 231, and for the serial schedule illustrated in part b the final value of A would be 230. If we have n concurrent transactions, it is possible to have n!, where n! = n * (n-1) * (n - 2) * ..... * 3 * 2 * 1 distint serial schedules, and possibly that many district resulting modifications to the database. For a serializable schedule, all we require is that the schedule gives a result that is the same as any one of these possibly distinct results.

When n transactions are run concurrently and in an interleaved manner, the number of possible schedules is much larger than n!. We would like to find out if a given interleaved schedule produces the same results as one of the serial schedules. If the answer is positive, then the given interleaves schedule is said to be serializable.

Definition - Serializable Schedule :

Given an interleaved execution of a set of n transactions; the following conditions hold for each transaction in the set :

All transactions are correct in the sense that if any one of the transactions is executed by itselt on a consistent database, the resulting database will be consistent.

Any serial execution of the transactions is also correct and preserves the consistency of the database; the results obtained are correct. (This implies that the transactions are logically correct and that no two transactions are interdependent.) The given interleaved execution of these transactions is said to be serializable if it produces the same result as some serial execution of the transactions. Since a serializable schedule gives the same result as some

serial schedule and since that serial schedule is correct, then the serializable schedule is also correct. Thus, given any schedule, we can say it is correct if we can show that it is serializable.

Algorithm given stablishes the serializability of an arbitrarily interleaved execution of a set of transactions on a database. The algorithm does not consider the nature of the computations performed by a transaction nor the exact effect of each such computational operation on the database. In effect, the algorithm ignores the semantics of the operations performed by the transactons including the commuting property of algebraic of logical computations of the transactions.

We may conclude from the algorightm that a given schedule is not serializable, when in effect it is, if some of the semantics and the algebraic commutability were not ignored.

However, the algorithm will never lead us to conclude that a schedule is serializable, when it does not produce the same result as some serial schedule. The computation involved in analyzing each transaction and seeing if its operations could be safely interleaved with those of other concurrent transactions is not justified by the greater degree of concurrency of the resulting "better" serializable schedule.

In Algorithm, we make the following assumptions :

- Each transaction is a modifying transaction, i.e., it would change the value of at least one database item.
- For each such item A that a transaction modifies, it would first read the value a of the item from the database (this is the read-before-write protocol)
- Having read the value it would transform a to f(a). where f is some transaction-dependent computation of transformation.
- It would then write this new value to the database.

Before presenting the algorithm we present the notion of precedence graph

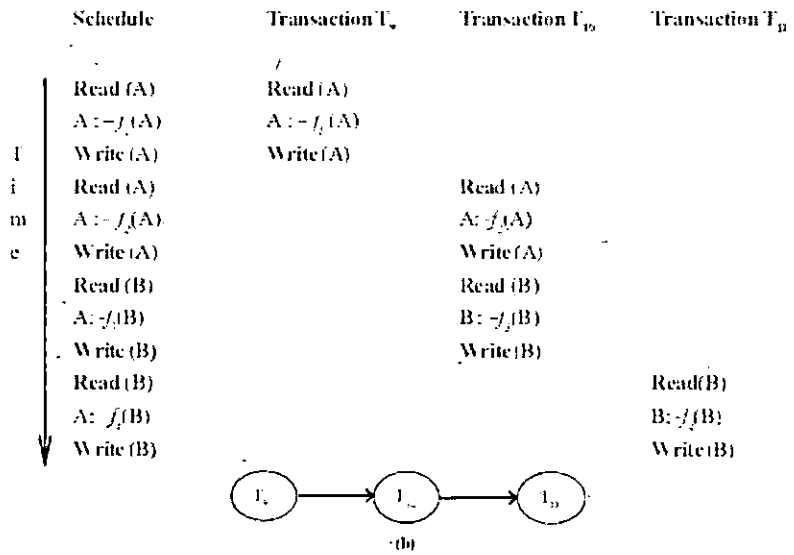


Figure 11 : (a) A schedulue and (b) an acyclic precedence graph.

Notes

Precedence Graph

Precedence graph G(V, E) consists of a set of nodes or vertices V and a set of directed arcs or edges E. Figure 6 gives an example of a schedule and the corresponding precedence graph. The schedule is for three transactions T9 ,T10 and T11 and the corresponding precedence graph has the vertices T9, T10, and T11 there is an edge from T9 to T10 and another edge from T10 to T11 if T9 , T10 and T11 represent three transactions, the precedence graph represents the serial execution of these transactions.

In a precedence graph, a directed edge from a node Ti to a node Tj. i # j, indicates one of the following conditions regarding the read and write operations in transactions Ti and Tj with respect to some database item A:

- Tj performs the operation Read(A) to read the value written by Ti performing the operation Write(A)
- Tj performs the operation Write(A) after Ti performs the operation Read(A).

If we limit ourselves to the read-before-write protocol only, we have to look for an edge corresponding to these conditions only.

In figure 12 all the statements in transaction T9 are executed before transaction T10 is started. Similarly, all the operations of T10 are completed before starting T11. The precedence graph corresponding to the schedule of part a is given in part b.

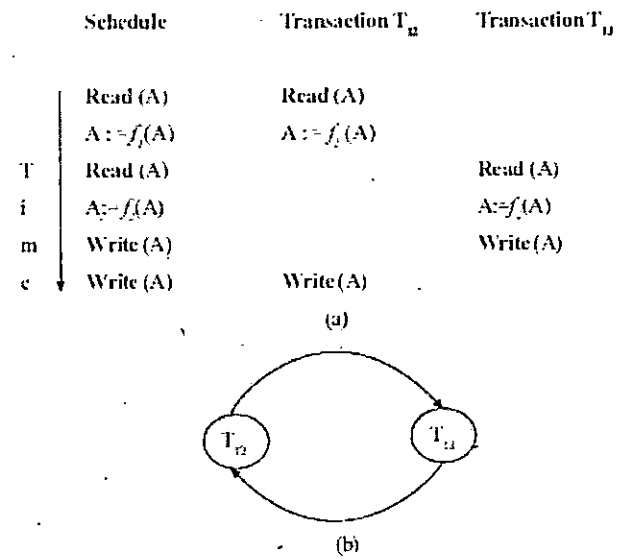


Figure 12 : (a) Schedule and (b) a cyclic precedence graph.

Figure 12 a gives a schedule and Figure 12.12 b gives precedence graph for transactions T12 and T13 in the precedence graph there is an edge from T12 to T13 as well as edge from T13 to T 12 The edge T13 to T12 is included because T12 executes a write operation after T13 executes a write operations for the same database item A. The edge T12to T13 is included because T13 executes a write optration after T12 executes a read operation for the same database item A. We see that the precendence graph has a cycle, since we can start from

one of the nodes of the graph and, following the directed edges, return to the starting node.

A precedence graph is said to be acyclic if there are no cycles in the graph. The graph of Figure 12 b has no cycles. The graph of Figure 12b is cyclic, since it has cycle.

The precedence graph for serializable schedule S must be acyclic, hence it can be converted to a serial schedule.

To test for the serializability of the arbitraty schedule S for transactions T1,..... Tk we convert the schedule into a precedence graph and then test the precedence graph for cycles. If no cycles are detected the schedule is serializable; otherwise it is not. If there are n nodes in the graph for schedule S, the number of operations required to check if there is a cycle in the graph is proportional to n_2.

Serializability Algorithm : Read-before-write Protocol :

In the read-before-write protocol we assume that a transaction will read the data-item before it modifies it and after modifications, the modified value is written back the the database. In the Algorithm, we give the method of testing whether a schedule is serializable. We create a precedence graph and test for a cycle in the graph. If we find a cycle. The schedule is nonserializable; otherwise we find a linear ordering of the transactions.

In Examples 3.1 and 3.2 we illustrate the application of this algorithm.

Example 3.1 : Consider the schedule of Figure A. The precendence graph for this schedule is given in Figure B . The graph has three nodes corresponding to the three transactions T14, T15 and T16.

There is an arc from T14 to T 15 because T14 writes data-item A before T15 reads it. Similarly, there is an arc from T15 to T16 because T15 writes data-item B before T16 reads it.

Finally, there is an arc from T16 to T14 because T16 writes data-item C before T14 reads it. The precedence graph of Figure B has a cycle formed by the directed edges from T14 to T15, from T15 to T16 and from T16 back to T14 Hence, the schedule of Figure A is not serializable. We cannot execute the three transactions serially to get the same result as the given schedule.

Schedule	Transaction T_{14}	Transaction T_{15}	Transaction T_{16}
Read(A)	Read(A)		
Read(B)		Read(B)	
$A := f_{14}(A)$	$A := f_{14}(A)$		
Read(C)			Read(C)
$B := f_{15}(B)$		$B := f_{15}(B)$	
Write(B)		Write(B)	
$C := f_{16}(C)$			$C := f_{16}(C)$
Write(C)			Write(C)
Write(A)	Write(A)		
Read(B)			Read(B)
Read(A)		Read(A)	
$A := f_{15}(A)$		$A := f_{15}(A)$	
Read(C)	Read(C)		
Write(A)		Write(A)	
$C := f_{14}(C)$	$C := f_{14}(C)$		
Write(C)	Write(C)		
$B := f_{16}(B)$			$B := f_{16}(B)$
Write(B)			Write(B)

Figure : (A) An Execution Schedule Involving Three Transactions

Notes

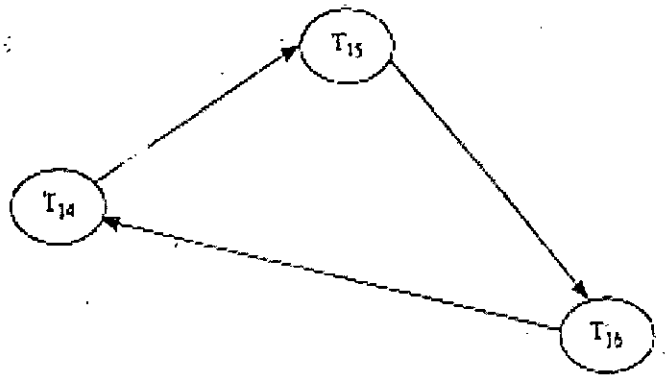


Figure 13: (B) A precedence Graph with a Cycle.

Serializability Algorithm : Read-Only and Write-Only Protocols :

The Algorithm is for a set of transactions that follow the read before write protocol. Some transactions, in addition to having a set of data-items that are read before rewritten have another set of data-items that are only read and a further set of data-items that are only written. In such a case some additional edges must be added to the graph.

Recovery

In designing a reliable system we try to anticipate different types of failures and provide for the means to recover without loss of information. Some very rare failures may not be catered to for economic reasons. Recovery from failures that are not thought of, overlooked, or ignored may not be possible. In common practice, the recovery system of a DBMS is designed to anticipate and recover from the following types of failure :

- Failures without loss of data : This type of failure is due to errors that the transaction discovers before it reaches the start to commit state. It can also be due to the action of the system, which resets its state to that which existed before the start of the transaction. No loss of data is involved in this type of failure, especially in the case where the transactions are run in a batch mode; these transactions can be rerun later in the same sequence.
- Failure with loss of volatile storage : Such a failure can occur as a result of software or hardware errors. The processing of an active transaction is terminated in an unpredictable manner before it reaches its commit or rollback state and the contents of the volatile memory are lost.
- Failure with loss of nonvolatile storage : This is the sort of failure that can occur after the failure of a nonvolatile storage system; for example, a head crash on a disk drive or errors in writing to a nonvolatile device.
- Failure with a loss of stable storage : This type involves loss of data stored on stable storage. The cause of the loss could be due to natural or man-made disasters. Recovery from this type of failure requires manual regeneration of the database. The probability

of sush a failure is reduced to a vary small value by having multiple copies of data in stable storage, stored in physically secure environments in geographically dispersed locations. The basic technique to implement the database transaction paradigm in the presence of failures of various kinds is by using date redundancy in the from of logs, check-points and archival copies of the database.

Logs

The log, which is usually written to stable storage, contains the redundant date required to recover from volatile storage failures and also from errors discovered by the transaction or the database system.

For each transaction the each transaction the following data is recorded on the log:

- A start of transaction marker.
- The transaction identifier.
- The record identifiers, which include the identifiers for the record occurrences.
- The operation(s) performed on the records (insert, delete, modify).
- The previous value(s) of the modified data. This information is required for undoing the changes made by a partially completed transaction; it is called the undo log. Where the modification made by the transaction is the insertion of a new record, the previous values can be assumed to be null.
- The updated value(s) of the modified record(s). This information is required for making sure that the changes made by a committed transaction are in fact reflected in the database and can be used to redo these modification. This information is called the redo part of the log. In case the modification made by the transaction is the deletion of a record, the updated values can be assumed to be null.
- A commit transaction marker if the transaction is committed; otherwise an abort or rollback-transaction marker.

The log is written before any updates are made to the database. This is called the write-ahead log strategy. In this strategy a transaction is not allowed to modify the physical database until the undo portion of log (i.e. the portion of the log that contains the value (s) of the modified data) is written to stable storage.

Furthermore, the log write-ahead strategy requires that a transaction is allowed to commit only after the redo portion of the log and the commit transaction marker are written to the log.

In effect, both the undo and redo portin of the log will be written to stable storage before a transaction commit. Using this strategy, the partial updates made by an uncommitted transaction can be undone using the undo portion of the log, and a failure occurring between the writing of the log and the completor of updating the database corresponding to the action implied by the log can be redone.

Let us see how the log information can be used in the case of a system crash with the loss of volatile information. Consider a number of transaction, as shown in figure 13. The figure shows the system start-up at the time t_0 and a number of concurrent transaction T0, T1, ... , Ti+6 are made on the database. Suppose a system crash occures at time tx.

Notes

We have stored the log information for transaction T0 through Ti+2 on stable storage, and we assume that this will be available when the system comes up after.

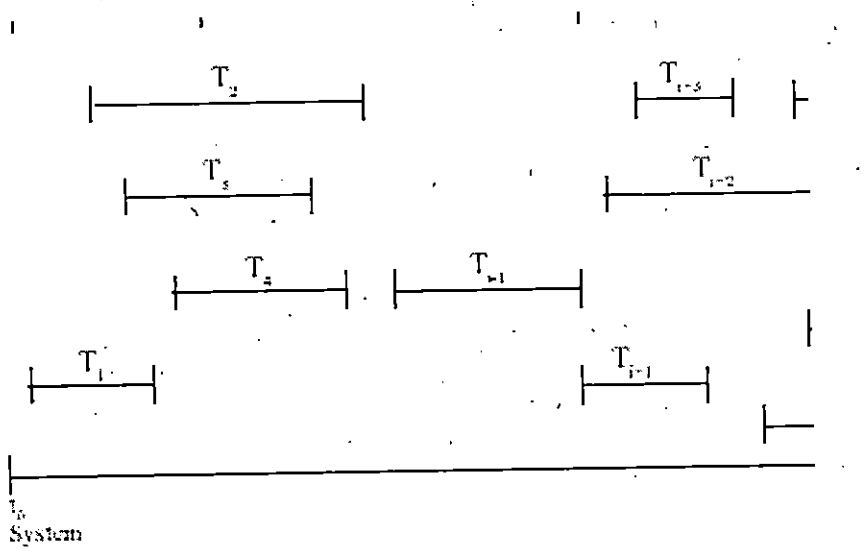


Figure 14 : DBMS operation to a system crash

The crash, Furthermore, we assume that the database existing on the nonvolatile storage will alsobe availble. It is clear that the transaction that were not committed at the time of the system crash will have to be undone. The changes made by these uncommitted transactions will have to be rolled back.

The transactions that have not been committed can be found by examining the log and those transactions that have a start of transactions marker but no commit or abort transactions marker are considered to have been active at the time of the crash. These transactions have to be rolled back to restore the database to a consistent state. In figure 14 the transactions Ti and Ti+6 started before the crash, but they had not been committed and, hence, are undone.

However, it is not clear from the log to what extent the changes made by committed transactions have actually been propagated to the database on the nonvolatile storage. The reason for this uncertainly is the fact that buffers (implemented in volatile storage) are used by the system to hold the modified data.

Some of the changed data in these buffers may or may not have been propagated to the database on the nonvolatile storage. In the absence of any method of finding out the extent of the loss, we will be forced to redo the effects of all committed transactions. For figures 14, this involves redoing the changes made by all transactions from t0 Under such a scenario, the longer the system operates without a crash, the longer it will take to recover from the crash.

In the above, we have assumed that the log informatin is available up to the time of the system crash in nonvolatile storage.

However, the log information is also collected in buffers. In case of a system crash with loss of volatile information, the log information collected in buffers will also be lost and transactions that had been completed for some period prior to the system crash may be missing their respective end-of-transactions markers in the log. Such transactions, if

rolled back, will likely be partially undone. The write-ahead log strategy avoids this type of recovery problem, since the log information is forced to be copied to stable storage before the transactions commits.

These problems point to the conclusion that some means must be devised to propogate to stable storage at regular intervals all the log informaton, as well as modificatins to the database existing at a given time. Then the recovery operation after a system crash will not have to reprocess all transactions from the time of start-up of the system.

The crash, Furthermore, we assume that the database existing on the nonvolatile storage will also be availble. It is clear that the transaction that were not committed at the time of the system crash will have to be undone. The changes made by these uncommitted transactions will have to be rolled back. The transactions that have not been committed can be found by examining the log and those transactions that have a start of transactions marker but no commit or abort transactions marker are considered to have been active at the time of the crash. These transactions have to be rolled back to restore the database to a consistent state. In figure 12.13 the transactions Ti and Ti+6 started before the crash, but they had not been committed and, hence, are undone.

However, it is not clear from the log to what extent the changes made by committed transactions have actually been propagated to the database on the nonvolatile storage. The reason for this uncertainly is the fact that buffers (implemented in volatile storage) are used by the system to hold the modified data.

Some of the changed data in these buffers may or may not have been propagated to the database on the nonvolatile storage. In the absence of any method of finding out the extent of the loss, we will be forced to redo the effects of all committed transactions.

For figures 12.13, this involves redoing the changes made by all transactions from t0 Under such a scenario, the longer the system operates without a crash, the longer it will take to recover from the crash.

In the above, we have assumed that the log informatin is available up to the time of the system crash in nonvolatile storage. However, the log information is also collected in buffers. In case of a system crash with loss of volatile information, the log information collected in buffers will also be lost and transactions that had been completed for some period prior to the system crash may be missing their respective end-of-transactions markers in the log. Such transactions, if rolled back, will likely be partially undone.

The write-ahead log strategy avoids this type of recovery problem, since the log information is forced to be copied to stable storage before the transactions commits.

These problems point to the conclusion that some means must be devised to propogate to stable storage at regular intervals all the log informaton, as well as modificatins to the database existing at a given time. Then the recovery operation after a system crash will not have to reprocess all transactions from the time of start-up of the system.

The frequency of checkpointing is a design consideration of the recovery system. A checkpoint can be taken at fixed intervals of time (say, every 15 minutes). If this approach is used, a choice has to be made regarding what to do with the transactions that are active when the checkpoint signal is generated by a system timer.

In one alternative, called transaction-consistent checkpoint, the transactions that are active when the system timer signals a checkpoint are allowed to be started until the checkpoint

Notes

are allowed to complete, but no new transactions (requiring modifications to bhe database) are allowed to be started until the checkpoint is completed. This scheme, though attractive, makes the database unavailable at regular intervals and may not be acceptable for certain online applications.

In addition, this approach is not appropriate for long transactions. In the second variation, called action consistent checkpoint, active transactions are allowed to complete the current step before the checkpoint and no new actions can be started on the database until the checkpoint is completed; during the chekpoint no actions are permitted on the database. Another alternative, called transaction-oriented checkpoint, is to take a checkpoint at the end of each transaction by forcing the log of the transaction onto stable storage. In effect, each commit transaction is a chekpoint.

How does the checkpoint information help in recovery ? To answer this question, reconsider the set of transactions of Figure 14, shown in Figure 15 with the addition of a checkpoint being taken at time t. Suppose, as before, the crash occurs at time, tx. Now the fact that a checkpoint was taken at time tc indicates that at that time all log and data uffers were propagated to storage. Transactions T0, ..., Ti-1 as well as transactions Ti+1 and Ti+3 were committed, and their modifications are reflected in the database. With the checkpoint scheme these transactions are not required to be redone during the recovery operation following a system crash occurring after time tc. A transaction such as Ti (which started before checkpoint time tc), as well as transaction Ti+6 (which started after checkpoint time tc,) were not committed at the time of the crash and have to be rolled back. Transactions such as Ti+4 and Ti+5 which started after checkpoint time tc and were committed before the system crash, have to be redone.

Similarly, transactions such as Ti+2, which started before the checkpoint time and were committed before the system crash, will have to be redone. However, if the commit transaction information is missing for any of the transactions Ti+2, Ti+4, or Ti+5, then they have to be undone.

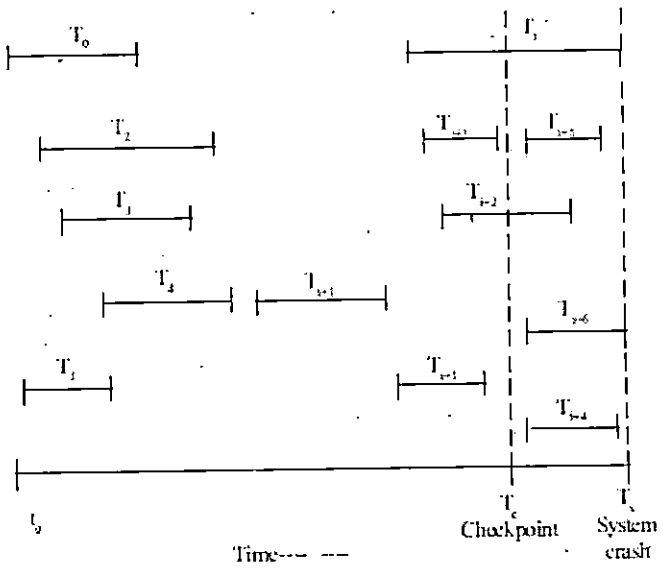


Figure.15 : Checkpointing

Notes

Let us now see how the system can perform a recovery at time tx. Suppose all transactions that started before the checkpoint time but were not committed at that time, as well as the transactions started after the checkpoint time, are placed in an undo list, which is a list of transactions to be undone.

The undo list for the transactions of Figure 15 is given below:

UNDO List (Ti, Ti+2, Ti+4, Ti+5, Ti+6)

Now the recovery system scans the log in a backward direction from the time tx of system crash. If it finds that a transaction in the undo list has committed, that transaction is removed from the undo list and placed in the redo list. The redo list contains all the transactions that have to be redone. The reduced undo list and the redo list for the transactions of Figure 15 are given below :

REDO List; (Ti+4, Ti+5, Ti+2)

UNDO List; (Ti, Ti+6)

Obviously, all transactions that were committed before the checkpoint time need not be considered for the recovery operation. In this way the amount of work required to be done for recovery from a system crash is reduced. Without the checkpoint scheme, the redo list will contain all transactions except Ti and Ti+6. A system crash occurring during the checkpoint operation, requires recovery to be done using the most recent previous checkpoint.

The recovery scheme described above takes a pessimistic view about what has been propagated to the database at the time of a system crash with loss of volatile information. Such pessimism is adopted both for transactions committed after a checkpoint and transactions not committed since a checkpoint. It assumes that the transactions committed since the checkpoint have not been able to propagate their modifications to the database and the transactions still in progress have done so.

Note that in some systems the term checkpoint is used to denote the correct state of system files recorded explicitly in a backup file and the term checkpointing is used to denote a mechanism used to restore the system files to a previous consistent state. However, in a system that uses the transaction paradigm, checkpoint is a startegy to minimize the search of the log and the amount of undo and redo required to recover from a system failure with loss of volatile storage.

Archival Database and Implementation of the Storage Hierarchy of a Database System Figure 15 gives the different categories of data used in a database system. These storage types are sometime called the storage hierarchy. It consists of the archival database, physical database, archival log, and current log.

Physical database : This is online copy of the database that is stored in nonvolatile storage and used by all active transactions.

Current database : The current version of the database is made up of the physical database plus modifications implied by buffers in the volatile storage.

Archival Database in Stable Storage :

This is the copy of the database at a given time, stored on stable storage. It contains the entire database in a quiescent mode (i, e, no transactions were active when the database was

Notes

copied to the stable storage) and could have been made by simple dump routines to dump the physical database (which in quiescent state would be the same as the current or online database) onto stable storage. The purpose of the archival database is to recover from failures that involve loss of nonvolatile storage. The archiving process is a relatively time-consuming operation and during this period the database is not accessible.

Consequently, archiving is done at infrequent intervals. The frequency of archiving is a loss of nonvolatile data being the arbitrator. All transactions that have been executed on the database from the time of archiving have to be redone in a global recovery operation. No undoing is required in the global recovery operation since the archival database is a copy of the database in a quiescent state, and only the committed transactions since the time of archiving are applied to this database.

Current log : This contains the log information (including the checkpoint) required for recovery from system failures involving loss of volatile information

Archival log : This log is used for failure involving loss of nonvolatile information. The log contains information on all transactions made on the database from the time of the archival copy. This log is written in chronological order. The recovery from loss of nonvolatile storage uses the archival copy of the database and the archival log to reconstruct the physical database to the time of the nonvolatile storage failure.

With the above storage hierarchy of a database, we can use the following terms to denote different combinations of this hierarchy.

The on line or current database is made up of all the records (and the auxiliary structures such as indexes) that are accessible to the DBMS during its operation. The currrent database consists of the data stored in nonvolatile storage (Physical database) as well as the data stored in buffers (in the volatile storage) and not yet propagated to the nonvolatile storage

The materialized database is that portion of the database that is still intact after a failure. All the data stored in the buffers would have been lost and some portion of the database would be in an inconsistent state. The log information is to be applied to the materialized database by the recovery system to restore the database to as close a state as possible to the online database prior to the crash.

Obviously, it will not be possible in all cases to return to exactly the same state as the precrash online database. The intent is to limit the amount of lost data and the loss of completed transactions.

Do, Undo, and Redo

A transaction on the current database transforms it from the current state to a new state. This is the so-called do operation. The undo and redo operations are functions of the recovery subsystem of the database system used in the recovery process. The undo operation undoes or reverses the actions (Possibly partially executed) of a transaction and restores the database to the state that existed before the start of the transaction. The redo operation redoes the action of a transaction and restores the database to the state it would be in at the end of the transaction. The undo operation is also called into play when a transaction decides to terminate itself (Suicidal termination).

Figure 16(b) shows the transformation of the database as a result of a transaction do,

redo, and undo.

The undo and redo operations for a given transaction are required to be idempotent; that is, for any transaction, performing one of these operations once is equivalent to performing it any unmber of times.

Thus :

undo (any action) = undo (undo(..undo(any action)....))

Redo(any action) = redo (redo(..redo(any action)..))

The reason for the requirement that undo and redo be idempotent is that the recovery process, while in the process of undoing or redoing the actions of a transaction, may fail without a trace, and this type of failure can occur any number of times before the recovery is completed successfully.

Transaction Undo :

A transaction that discovers an error while it is in progress and consequently needs to abort itself and roll back any changes made by it uses the transaction undo feature. A transaction also has to be undone when the DBMS forces the transaction to abort. A transaction undo removes all database changes, partial of otherwise, made by the transaction.

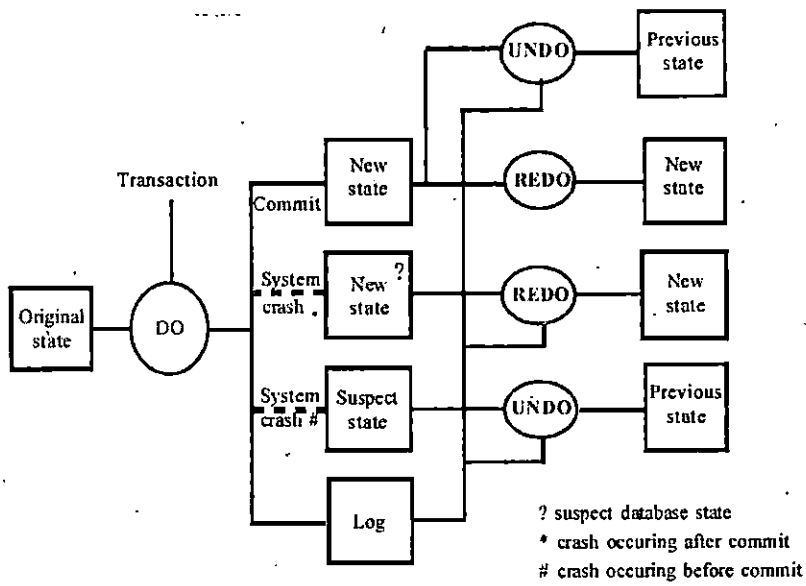


Figure 16(b) Do, Undo, and redo operations

Transaction Redo:

Transaction redo involves performing the changes made by a transaction that commited before a system crash. With the write-ahead log strategy, a commited transaction implies that the log for the transaction would have been written to nonvolatile storage, but the physical database may or may not have been modified before the system failure. A transaction redo

Notes

modifies the physical database to the new values for a committed transaction. Since the redo operation is idempotent, redoing the partial or complete modifications made by a transaction to the physical database will not pose a problem for recovery.

Global Undo :

Transactions that are partially complete at the time of a system crash with loss of volatile storage need to be undone by undoing any changes made by the transaction. The global undo operation, intiated by the recovery system, involves undoing the partial or otherwise updates made by all uncommitted transactions at the time of a system failure.

Global Redo :

The global redo operation is required for recovery from failures involving nonvolatile storage loss. The archival copy of the database is used and all transactions committed since the time of the archival copy are redone to obtain a database updated to a point as close as possible to the time of the nonvolatile storage loss. The effects of the transaction in progress at the time of the nonvolatile loss will not be reflected in the recovered database. The archival copy of the database could be anywhere from months to days old and the number of transactions that have to be redone could be large. The log for the committed transaction needed for performing a global redo operation has to be stored on stable storage so that they are not lost with the loss of nonvolatile storage containing the physical database.

Reflecting updates to the Database and Recovery :

Let us assume that the physical database at the start of a transaction is equivalent to the current database, i.e., all modifications have been reflected in the database on the nonvolatile storage. Under this assumption. Whenever a transaction is run against a database, we have a number of options as to the strategy that will be followed in reflecting the modifications made by the transaction as it is executed.

The strategies we will explore are the following :

Update in place : In this approach the modifications appear in the database in the original locations and in the case of a simple update, the new values will replace the old values. Concurrent execution of a number of transactions implies that the operations from these transactions may be interleaved. This is not the same as serial execution of the transactions where each transaction is run to completion before the next transaction is started. Concurrent access to a database by a number of transactions requires some type of concurrency control to preserve the consistency of the database, to ensure that the modifications made by the transactions are not lost, and to guard against transations reading data that is inconsistent. The serializability criterion is used to test whether or not an interleaved execution of the operations from a number of concurrent transactions is correct.

The Serializability test consists of generating a precedence graph from a interleaved execution schedule. If the precedence graph is acyclic, the schedule is serializable, which means that the database will have the same state at the end of the schedule as some serial execution of the transactions. In this unit, we introduce a number of concurrency control schemes.

3.24 CONCURRENCY CONTROL

If all schedules in a concurrent environment are restricted to serializable schedules, the result obtained will be consistent with some serial execution of the transactions and will be considered correct. However, using only serial schedules unnecessarily limits the degree of concurrency. Furthermore, testing for serializability of a schedule is not only computationally expensive but it is an after-the-fact technique and im practical. Thus, one of the following concurrency control schemes is applied in a concurrent database environment to ensure that the schedules produced by concurrent transactions are serializable. The schemes we discuss are locking, timestamp-based order.

The intent of locking is to ensure serializability by ensuring mutual exclusion in accessing data-items. In the timestamp-based ordering scheme, the order of execution of the transactions is selected a priori by assigning each transaction an unique value. This value, usually based on the system clock, is called a timestamp. The values of the timestamp of the transactions determine the sequence in which transactions contesting for a given data-item will be executed. Conflicts in the timestamp scheme are resolved by abort and rollback.

Locking and types of Locking

From the point of view of locking, a database can be considered as being made up of a set of data items. Alock is a variable associated with each such data-item. Manipulating the value of a lock is called locking. The value of a lock variable is used in the locking scheme to control the concurrent access and manipulation of the associated data-item. Locking the items being used by a transaction can prevent other concurrently running transactions from using these locked items. The locking is done by a subsystem of the database management system usually called the lock manager.

So that concurrency is not restricted unnecessarily, at least two types of locks are defined: exclusive lock and shared lock.

Exclusive lock : The exclusive lock is also called an update or a write lock. The intention of this mode of locking is to provide exclusive use of the data item to one transaction. If a transaction T locks a data item Q in an exclusive mode, no other transaction can access Q, not even to read Q, until the lock is released by transaction T.

Shared lock : The shared lock is also called a read lock. The intention of this mode of locking is to ensure that the data item does not undergo any modifications while it is locked in this mode.

Any number of thansactions can concurrently lock and access a data item in the shared mode, but none of these transactions can modify the data item. A data item locked in a shared mode cannot be locked in the exclusive mode until the shared lock is released by all transactions holding the lock. A data item locked in the exclusive mode cannot be locked in the shared mode until the exclusive lock on the data item is released.

The protocol of sharing is as follows. Each transaction, before accessing a data item, requests that the data item be locked in the appropriate mode. If the data item is not locked, the lock request is honored by the lock manager. If the data item is already locked, the request may or may not be granted, depending on the mode of locking requested and the current mode in which the data item is locked. If the mode of locking requested is shared

Notes

and if the data item is already locked in the shared mode, the lock request can be granted. If the data item is locked in an exclusive mode, then the lock request cannot be granted, regardless of the mode of the request. In this case the requesting transaction has to wait till the lock is released.

The compatibility of a lock request for a data item with respect to its current state of locking is given in Figure 17. Here we are assuming that the request for locking is made by a transaction not already holding a lock on the data item.

Lock mode of request	Current state of locking of data-item		
	Unlocked	Shared	Exclusive
Unlock		yes	yes
Shared	yes	yes	no
Exclusive	yes	no	no

Figure 17 : Compatibility of locking

If transaction Tx makes a request to lock data item Ain the shared mode and if Ais not locked or if it is already locked in the shared mode, the lock request is granted. This means that a subsequent request from another transaction, Ty, to lock data item A in the exclusive mode would not be granted and transaction Ty will have to wait until A is unlocked. While Ais locked in the shared mode, if transaction TZ makes a request to lock it in the shared mode. this request can be granted. Both TX and TZ can concurrently use data item A.

If transaction Tx makes a request to lock data-item Ain the shared mode and if Ais locked in the exclusive mode, the request made by transaction Tx cannot be granted.

Similarly, a request by transaction Tz to lock A in the exclusive mode while it is already locked in the exclusive mode would also result in the request not being granted, ant Tz would have to wait until the lock on Ais released. From the above we see that any lock request for a data-item can only be granted if it is compatible with the current mode of locking of the data-item. If the request is not compatible, the requesting transaction has to wait until the mode becomes compatible.

The releasing of a lock on a data-item changes its lock status. If the data-item was locked in an exclusive mode, the release of lock request by the transaction holding the exclusive lock on the data-item would result in the data-item being unlocked.Any transaction waiting for a release of the exclusive lock would have a chance of being granted its request for locking the data-item. If more than one transaction is waiting, it is assumed that the lock manager would use some fair scheduling teachnique to choose one of these waiting transactions.

If the data-item was locked in a shared mode, the release of lock request by the transaction holding the shared lock on the data-item may not result in the data-item being unlocked. This is because more than one transaction may be holding a shared lock on the data-item. Only when the transaction releasing the lock is the only transaction having the shared lock does the data-item become unlocked.

The lock manager may keep a count of the number of transactions holding a shared lock on a data-item. It would increase this value by one when an additional transaction is granted a shared lock and decrease the value by one when a transaction holding a shared lock releases the lock.

The data-item would then become unlocked when the number of transactions holding a shared lock on it becomes zero. This count could be stored in an appropriate data structure along with the data-item but it would be accessible only to the lock manager.

The lock manager must have a priority scheme whereby it decides whether to allow additional transactions to lock a data item in the share-mode in the following situation:

- The data-item is already locked in the shared mode.
- There is at least one transaction waiting to lock the data-item in the exclusive mode.

Allowing a higher priority to share lock requests could result in possible starvation of the transactions waiting for an exclusive lock. Similarly, the lock manager has to deal with a situation where a data-item is locked in an exclusive mode and there are transactions waiting to lock the data-item in the shared mode and the exclusive mode.

3.25 GRANTING OF LOCKS

In the following discussions we assume that a transaction makes a request to lock data-itemA by executing the statement Locks(A) or Lockx(A). The former is for requesting a shared lock; the latter, an exclusive lock. A lock is released by simply executing an Unlock(A) statement. We assume that the transactions are correct. In other words, a transaction would not request a lock on a data-item for which it already holds a eock, nor would a transaction unlock a data-item if it does not hold a lock for it.

A transaction may have to hold onto the lock on a data-item beyond the point when it last needs it to preserve consistency and avoid the inconsistent read problems . We illustrate this point by reworking the example of Figure 18 here each transaction request locks for the data-itemsA and B : transaction T5 in exclusive mode and transaction T6 in shared mode. The transactions with the lock requests are given in Figure 18 As shown there, the transactions attempt to release the locks on the data-items as soon as possible.

Now consider Figure 19 which given a possible schedule of execution of the transactions of Figure 18 The locking scheme did not resolve the inconsistent read problem; the reason is that transactions T5 and T6 are performing an operation made up of many steps and all these have to be executed in an atomic manner. The database is in an inconsistent state after transaction T5 has taken 100 units from A but not added it to B. Allowing transaction T6 to read the vlues ofA and B before transaction

T5 is complete leads to the inconsistent read problem.

A possible solution to the inconsistent read problem is shown in Figure 19 Here transactions T5 and T6 are rewritten as transactions T20 and T21. The possible schedules of concurrent executions of these transactions are shown in Figure 20 and 21 Both of these solutions extend the period of time for which they keep some data-items locked even though the transactions no longer need these items. This extended locking forces a serialization of the two transactions and given correct results.

Transaction T5	Transaction T6
Lockx (A)	Lockx (Sum)
Read (A)	Sum : = 0

Notes

A : = A - 100	Locks (A)
Write (A),	Read (A)
Unlock (B)	Sum : = Sum + A
Lockx (B)	Unlock (A)
Read (B)	Locks (B)
B : - B + 100	Read (B)
Write (B)	Sum : = Sum + B
Unlock(B)	Write (Sum)
	Unlock (B)
	Unlock (Sum

Figure 18: Two Transactions with Lock Requests

	Shedule	Transaction T_1	Transaction T_2
	Lockx (Sum)		Lockx (Sum)
	Sum : = 0		Sum : = 0
	Locks(A)		Locks (A)
	Read(A)		Read (A)
	Sum : = Sum + A		Sum : = Sum + A
T	Unlock (A)		Unlock (A)
i	Lockx(A)	Lockx (A)	
m	Read(A)	Read (A)	
e	A : = A - 100	A : = A – 100	
	Write (A)	Write (A)	
	Unlock (A)	Unlock(A)	
	Lockx (B)	Lockx (B)	
	Read (B)	Read (B)	
	B : = B + 100	B : = B + 100	
	Write(B)	Write (B)	
	Unlock (B)	Unlock (B)	
	Locks (B)		Locks(B)
	Read (B)		Read (B)
	Sum : = Sum + B		Sum : = Sum + B
	Write(Sum)		Write (Sum)
	Unlock (B)		Unlock (B)
	Unlock (Sum)		Unlock (Sum)

Figure 19 : A possible Schedule Causing an Inconsistent Read

Some data-items locked even though the transactions no longer need these items. This extended locking forces a serialization of the two transactions and gives correct results.

3.26 TWO-PHASE LOCKING

The correctness of the schedules of Figure 20 of the transactions lead us to the observation that both these solutions involve transactions whose locking and unlocking operations are monotonic, in the sense that all locks are first acquired before any of the locks are released. Once a lock is released; no additional locks are requested.

In other words, the release of the locks is delayed until all locks on all data-items required by the transaction have been acquired.

This method of locking is called two-phalse locking. It has two phases, a growing phase where in the number of locks increase from zero to the maximum for the transaction, and contracting phase where in the number of locks held decreases from the maximum to zero. Both of these phases are monotonic; the number of locks are only increasing in the first phase and decreasing in the second phase. Once a transaction starts releasing locks, it is not allowed to request any further locks. In this way a transaction is obliged to request all locks it may need during its life before it releases any. This leads to a possible lower degree of concurrency.

	Shedule	Transaction T_{20}	Transaction T_{21}
	Lock x (A)	Lock x (A)	
	Read (A)	Read (A)	
	A : - A – 100	A : – A – 100	
	Write (A)	Write (A)	
	Lock x (B)	Lock x (B)	
T	Unlock (A)	Unlock (A)	
i	Read (B)	Read (B)	
m	B : – B + 100	B : – B + 100	
e	Write (B)	Write (B)	
	Unlock (B)	Unlock (B)	
	Lock x (Sum)		Lock x (Sum)
	Sum : – 0		Sum : – 0
	Lock s (A)		Lock s (A)
	Read (A)		Read (A)
	Sum : – Sum + A		Sum : – Sum + A
	Lock s (B)		Lock s (B)
	Read (B)		Read (B)
	Sum : Sum + B		Sum : Sum + B
	Write (Sum)		Write (Sum)
	Unlock (B)		Unlock (B)
	Unlock (A)		Unlock (A)
	Unlock (Sum)		Unlock (Sum)

Figure 20 : Another Solution to The Inconcistent Read Problem.

The two-phase locking protocol ensures that the schedules involving transactions using this protocol will always be serializable. For instance, if S is a schedule containing the interleaved operations from a number of transactions, T1, T2,..... Tk and all the transactions are using the two-phase locking protocol, schedule S is serializable. This is because if the schedule is

Notes

not serializable, the precdence graph for S will have a cycle made up of a subset of {T1, T2, ...,Tk} Assume the cycle consists of Ta → Tb → Tc .....Tx → Ta. This means that a lock operation by T6 is followed by an unlock operation by Ta; a lock operation by Tc is followed by an unlock operation by Tb....., and finally a lock operation by Ta is followed by an unlock operation by Tx. However this is a contradiction of the assertion that Ta is using the two phase protocol. Thus the assumption that there was a cycle in the precedence graph is incorrect and hence S is serializable.

The transactions of Figure 20 use the two-phase locking protocol, and the schedules derived from the concurrent execution of these transactions given n Figures 13.5 and 13.6 are serializable. However, the transactions do not follow the two-phase locking protocol and the schedule is not serializable.

3.27 TIME STAMP-BASED ORDER

In the timestamp-based method, a serial order is created among the concurrent transaction by assigning to each transaction a unique nondecreasing number. The usual value assigned to each transaction is the system clock value at the start of the transaction, hence the name timestamp ordering.

A variation of this scheme that is used in a distributed environment includes the site of a transaction appended to the system wide clock value. This value can then be used in deciding the order in which the conflict between two transactions is resolved. Atransaction with a smaller timestamp value is considered to be an "older" transaction thann another transaction with a larger timestamp value.

The serializability that the system enforces is the chronological order of the timestamps of the concurrent transactions. If two transaction Ti and Tj with the time stamp values ti and tj respectively, such that ti < tj, are to run concurrently, then the schedule produced by the system is equivalent to running the older transaction Ti first, followed by the younger one, Tj.

The contention problem between two transactions in the timestamp ordering system is resolved by rolling back one of the conflicting transactions.Aconflict is said to occur when an older transaction tries to read a value that is written by a younger transaction or when an older transaction tries to modify a value already read or written by a younger transaction. Both of these attempts signify that the older transaction was "too late" in performing the required read/write operations and it could be using values from different "generations" for different data-items.

In order for the system to determine if an older transaction is processing a value already read by or written by a younger transaction, each data-item has, in addition to the value of the item, two timestamps: a write timestamp and a read timestamp. Data-item X is thus represented by a triple X: {x, Wx, Rx} where each component of the triple is interpreted as given below :

x, the value of the data-item X

Wx, the write timestamp value, the largest timestamp value of any transaction that was allowed to write a vlue of X.

Rx, the read timestamp value, the largest timestamp value of any transaction that was allowed to read the current value X.

Now let us see how these timestamp values find their way into the data structure of a data-item and how all these values are modified. A transaction Ta with the timestamp value of ta issues a read operation for the data-item X with the values {x, Wx, Rx}.

- This request will succeed if $t_a > Wx$ since transaction T_a is younger than the transaction that last wrote (or modified) the value of X. Transaction T_a is allowed to read the value x of X and if the value ta is larger than Rx, then ta becomes the new value of Rx.
- This request will fail if $t_a \leq W_x$, i.e. transaction T_a is an older transaction than the last transaction that wrote the value of X.

The failure of the read request is due to the fact that the older transaction was trying to read a value that had been overwritten by a younger transaction. Transaction Ta is too late to read the previous outdated value and any other values it has acquired are likely to be inconsistent with the updated value of X. It is thus safe to abort and roll back T_a, T_a, is assigned a new timestamp and restarted.

Atransaction Ta with the timestamp value of ta issues a write operation for the data-item X with the values {x, W_x, R_x}

- If $t \geq Wx$ and $t_a \geq Rx$, i.e. both the last transaction that updated the value of X and the last transaction that read the value of X are older than transaction Ta, then Ta is allowed to write the value of X and ta becomes the current value of Wx, the write timestamp.
- If $t < R$, it means that a younger transaction is already using the corrent value of X and it would be an error to update the value of X. Transaction T_a is not allowed to modify the value of X. T_a is rolled back and its timestamp is reset to the current system-generated timestamp value and restarted.
- If $Rx < t_a < Wx$. this means that a younger transaction has already updated the value of X, and the value that Ta is writing must be based on an obsolete value of X and is obsolete. Transaction Ta is not allowed to modify the value of X; its write operation is ignored.

The reason for ignoring the write operation in the last alternative is as follows. In the serial order of transaction processing, transaction Ta with the timestamp of ta wrote the value for the data-item X. This was followed by another write operation to the same data-item by a younger transaction with a timestamp of Wx. No transaction read the data-item between the writing by Ta and the time Wx. Hence, ignoring the writing by Ta indicates that the value written by Ta was immediately overwritten by a younger transaction at time Wx.

Let us illustrate the timestamp ordering by considering transactions T_{22} and T_{23} given. Each of these transactions has a local variable Sum and the intent is to show a user the sum of two data-items Aand B.

However, transaction T_{23} not only reads these values, it also transfers 100 units from Ato B and writes the modified values to the database. Now let us suppose that $t_{23} > t_{22}$ This means that transaction T23 is younger than transaction T22. Also, let the data-items Aand B be stored as follows (here the Wi's and Ri's have some values assumed to be less then t22 and t23) :

A : 400, Wa, Ra B: 500, Wb, Rb

Notes

3.28 TRIGGERS, PROCEDURES FUNCTIONS AND PACKAGE

Informix Dynamic Server stores triggers and stored procedures with the server. Oracle stores triggers and stored subprograms with the server. Oracle has three different kinds of stored subprograms: functions, stored procedures, and packages. For detailed discussion on all these objects, see the PL/SQL User's Guide and Reference, Release 1(9.0.1).

Triggers

Triggers provide a way of executing PL/SQL code on the occurrence of specific database events. For example, you can maintain an audit log by setting triggers to fire when insert or update operations are carried out on a table. The insert and update triggers add an entry to an audit table whenever the table is altered.

The actions that Informix Dynamic Server triggers perform are constrained to multiple insert, update, delete, and execute procedure clauses; whereas, Oracle allows triggers to execute arbitrary PL/SQL code. Oracle triggers are similar to stored procedures in that they can contain declarative, execution, and exception handling code blocks.

Additionally, Oracle enables triggers to be invoked by many events other than table insert, update and delete operations. However, there are restrictions.

A trigger is also a set of SQL statements in the database which automatically execute whenever any special event occurs in the database, like insert, delete, update, etc.

Syntax

```
create trigger trigger_name
before | after
{insert | update | delete}
on table_name
for each row
---Query here
```

Stored Procedure

A stored procedure is a set of pre-compiled Structured Query Languages (SQL), so it can be reused and shared by multiple programs. It can access or modify data in a database.

Syntax

```
Create proc Proc_name
@permater
as begin
—Query here
end
OR
```

```
Create proc Proc_name
@permeter datatype
@rslt output datatype
as begin
—Query here
select @rslt
end
```

SQL Funcion

A function is a database object in SQL Server. Basically, it is also a set of SQL statements that accept only input parameters and produce output in a single value form or tabular form.

Syntax

```
create function funname(@parmeter datatype)
returns Returntype
as
begin Returntype
end
```

Difference between Stored Procedure, SQL Function, and Trigger

Executable

- Store procedure: We can execute the stored procedures when required.
- Function: We can call a function whenever required. Function can't be executed because a function is not in pre-compiled form.
- Trigger: Trigger can be executed automatically on specified action on a table like, update, delete, or update.

Calling

- Stored procedure: Stored Procedures can't be called from a function because functions can be called from a select statement and Stored Procedures can't be called from. But you can call Store Procedure from Trigger.
- Function: Function can be called from Store Procedure or Trigger.
- Trigger: Trigger can't be called from Store Procedure or Function.

Parameter

- Store procedure: Stored Procedures can accept any type of parameter. Stored Procedures also accept out parameter.

Notes

- Function: Function can accept any type of parameter. But function can't accept out parameter.
- Trigger: We can't pass a parameter to trigger.

Return

- Store procedure: Stored Procedures may or may not return any values (Single or table) on execution.
- Function: Function must return any value.
- Trigger: Trigger never return value on execution.

3.29 EMERGING TRENDS IN DATABASE

Database management systems are standard tools that enable the storage and retrieval of data within modern information systems. Applications in domains such as Multimedia, Geographical Information Systems, and digital libraries demand a completely different set of requirements in terms of the underlying database models. The conventional relational database model is no longer appropriate for these types of data.

Furthermore the volume of data is typically significantly larger than in classical database systems. Finally, indexing, retrieving and analyzing these data types require specialized functionality.

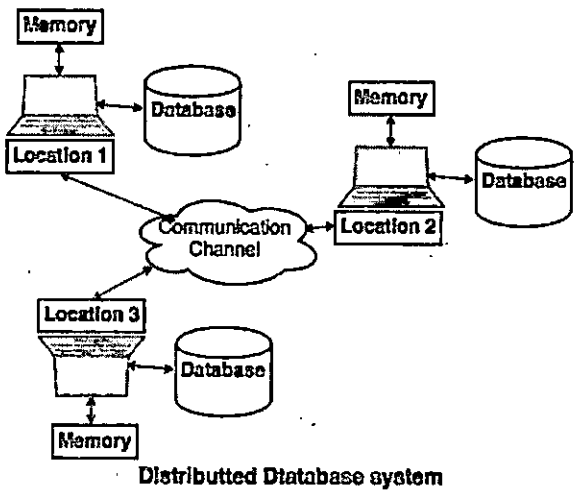


Figure 21 Introduction to Distributed Databases

A Distributed database system is a database in which the data is stored at several computers that are located at geographically distributed locations connected through a network. Each site or node has its own database management system, transaction management software

which also includes local logging, logging and recovery. The characteristics of distributed databases are:

- The computers are connected through some communication media.
- Such systems do not share the disks or memory.
- The applications can access data stored at local as well as at remote locations.
- Distributed databases show data independence that is the user can specify the query without specifying the location at which data is stored.Distributed transaction atomicity is also a characteristic of distributed databases as all the changes to the database are made permanent only if the transaction commits and no changes are made if the transaction aborts.

The computers at one site are connected to the computers located at various other sites as shown in Figure 22.

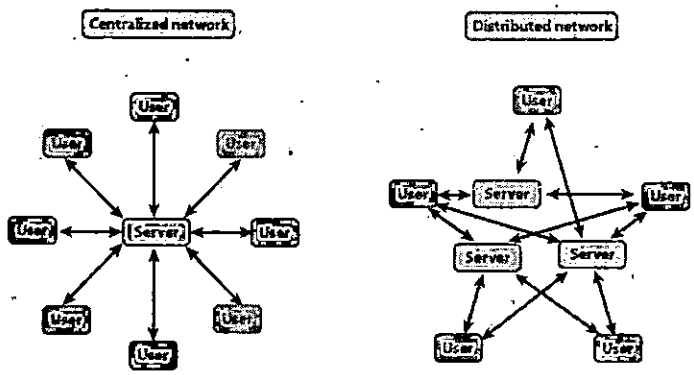


Figure 22: ADistributed Network

The distributed databases are of following types :

- Homogeneous Distributed Databases : Such kind of databases serve the purpose of location transparency i.e. all the sites have identical DBMS software and all the clients are aware of one another. Also the clients cooperate with each other
- Heterogeneous Distributed Databases : In a heterogeneous distributed database, different sites have different DBMS and may use different schemas and software. Such sites may not be aware of one another and have cooperation to only some extent. Such a system is also known as multi-database system or federated database system.

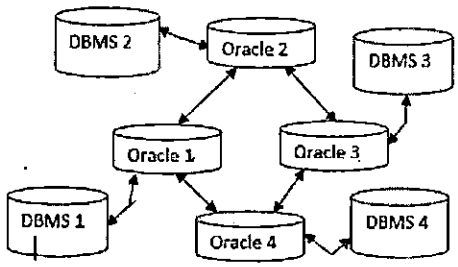


Figure 23: Heterogeneous v/s Homogeneous Distributed Databases

Notes

A distributed database must have some additional capabilities as compared to a centralized system. Some of these capabilities are as follows:

- Network Transparency : The user must not be concerned about the details of the locations where data is stored. This is further categorized into Location Transparency in which data is retrieved independent of the location and Naming transparency which indicates that once a name has been specified, the data can be accessed from anywhere.
- Replication Transparency : Refers to the fact that multiple copies of the data may be stored at several locations and the user is unaware of the details.
- Fragmentation Transparency : This can be further classified into :
  1. Horizontal fragmentation: where the database tuples or rows are distributed.
  2. Vertical fragmentation: where the database columns are distributed.
  3. Hybrid fragmentation includes an intermix of the above two types of fragmentation.
- Local Autonomy and Independence : The data stored at one location must be independent of the data stored at other locations.
- Distributed Query Processing and Transaction Management : The data stored at various locations is synchronized so as to maintain the integrity of the database. A distributed database must be independent of the hardware, operating system, network and DBMS that is used.
- Distributed Catalog Management : The directory or catalog contains metadata about the entire database.
- Security : Since a distributed database contains several computers connected at several locations, it should be secure enough. Also, access and authorization to the database must be checked.

Because of its distributed nature and its capability to store and access data at several locations, a distributed database has several advantages and disadvantages.

- The advantages include, sharing of data resulting into increased availability, efficiency and performance. Due to its extensible nature, a distributed database is also scalable.
- On the contrary it suffers from the disadvantage that recovery from failure becomes more complex due to its distributed nature. Again there can be breach of security due to increased transparency.
- Other disadvantages include increased cost, increased overhead of creation and maintenance of the database and lack of proper standards for synchronizing various sites.
- A distributed database may have a varieed architecture. Some of the architectures that are available in distributed databases are client/server architecture, collaborating server and middleware systems.
- The client server architecture has a collection of clients connected to a server that fulfils the requests of the clients.
- The clients provide the user interface and server is responsible for managing data and

executing transactions as shown in Figure 24.

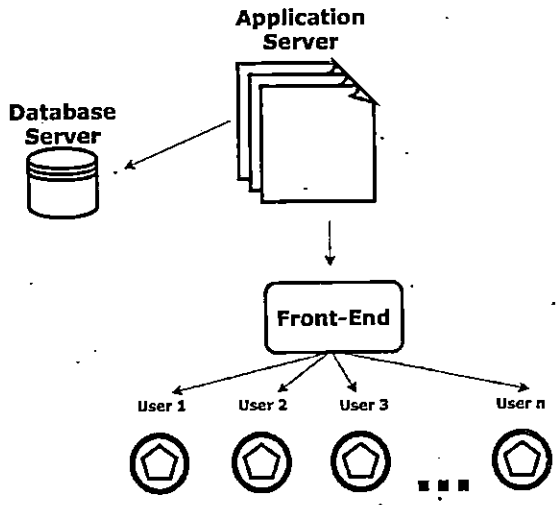


Figure 24 Client Server Architecture

The client server architecture is relatively simple in implementation and is comparatively less expensive but an increase in the number of client computers may increase the cost and complexity of the system. This architecture also suffers from the drawback that if the server breaks down then the entire system becomes difficult to manage.

To overcome this problem, the architecture of Collaborating server systems may be used. In this architecture several servers may cooperate with each other to solve the client requests.

Another architecture that may be used for distributed databases is middleware systems. In this architecture, a layer of software called the middleware is used that coordinates the execution of queries and transactions across multiple servers as illustrated in Figure 25.

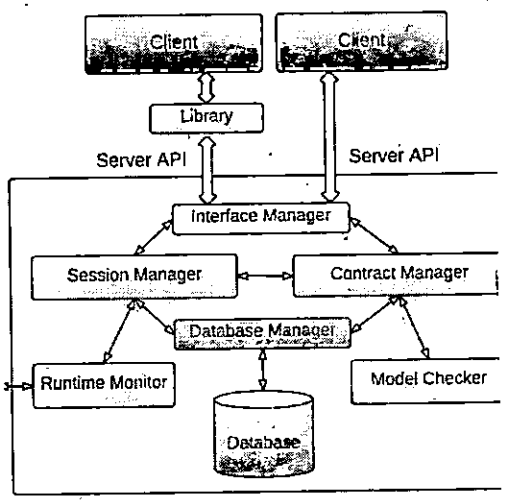


Figure 25: Middleware Architecture

Notes

Due to the distributed nature of the database several concurrency related problems arise. For example if one site fails, its database must be recovered to the state it was earlier in.

Similarly, if a communication link fails then two or more sites may be disconnected from each other. In an extreme case the network may be partitioned into two.

Another problem that arises in distributed databases is distributed commit i.e. if a site wants to commit a transaction that is stored at multiple locations and some of the participating sites fail. Two-phase commit protocol or three-phase commit protocol may be used to solve this problem. Two-phase commit (2PC) protocol has two phases: voting phase and decision phase. In the voting phase, the coordinator asks all the participating sites whether they wish to commit and in the decision phase, a decision whether to commit or abort all the sites is made. This protocol suffers from the drawback that if the coordinator fails then all the sites are blocked.

Three phase commit protocol (3PC) is an extension of 2PC.This protocol postpones the decision to commit until a specified number of sites commit the transaction. 3PC has an additional phase of pre-commit in which even if the coordinator fails, the new coordinator checks if one of the site has the decision of the old coordinator. A distributed database may access data from remote or local databases. A Homogeneous distributed database system may have the same database whereas a heterogeneous distributed database may have different databases on different systems. A distributed homogeneous database has the same database installed on different computers that are available on one or more machines. A single application may access different databases simultaneously in the distributed environment.

The location and other details are transparent to the clients connected to the distributed databases. A single copy of all the database objects may be stored at distributed locations or in some cases replication of data may be done.

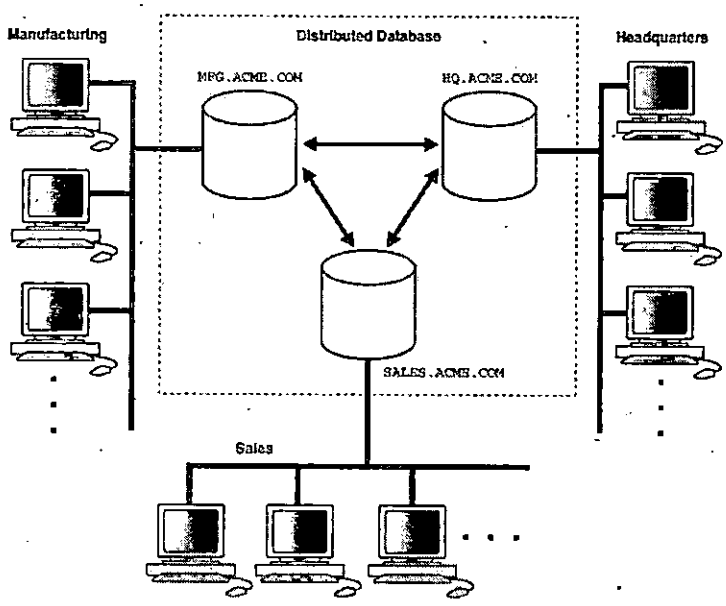


Figure 26. Distributed Database Architecture

The term replication refers to the fact that multiple copies of the data are maintained and stored at several locations. Replication may be done to ensure availability of data even if one site fails and thus improves the performance of the database. In such a case the replicated data remains available in case a site fails.

OBJECT - ORIENTED DATABASE

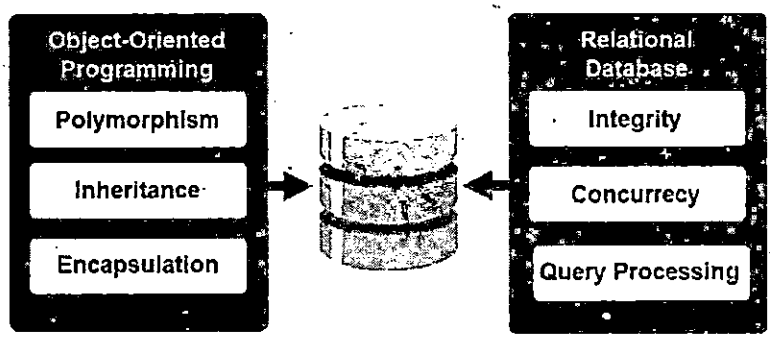


Figure 27. Object Oriented Database Management System

In a heterogeneous database, at least one of the databases is of different type as opposed to other sites in the distributed environment. In such cases, a gateway may be required to access databases of different types.

Some more issues that need to be addressed in a distributed environment include network authentication so that only valid users can access the network. There may be a global user that can access all the databases in the distributed environment, or there may be local users accessing the data at local sites.

In case of global users, authentication needs to be done so that it can be checked which databases the user can access, what are the roles corresponding to the databases and what schema the user can connect to.

Several database models like hierarchical, network and relational models were suggested. However due to the increased complexity and the need of flexibility led to the development of Object oriented data model (OODM) and Object-relational data model (ORDM) (or Extended relational data model). These models support the concept of object oriented programming. Apart from providing the capabilities of traditional databases like persistence, concurrency control, recovery, querying, atomicity, durability etc., Object oriented databases support the properties of Object oriented programming like encapsulation, inheritance, polymorphism etc.

As opposed to relational databases that support simple data types, Object based models support complex data types. Object Oriented Database Management System (OODBMS) is designed to manage the Object oriented database (OODB). Some of the popular OODBMS include Orion, IRIS, Versant and Vbase.

An OODBMS must have several features to support Object oriented design. These features include, support for encapsulation, classes and object. Such classes must also provide inheritance. The Object identity indicates the existence of the object. An object may show its identity as:

Notes

- Intra-procedure. The object exists in a particular procedure only
- Intra-program. Such an object exits throughout the program
- Inter-program. Such an object is accessible across several programs
- Persistent. These object persist even after the completion of the program

Object Data Management Group (ODMG) was a consortium developed for object oriented DBMSs. This standard provides the object model, Object Definition Language (ODL) and Object Query Language (OQL).

This standard was designed to provide portable applications that could store objects in any database management system. The Object Definition Language designed in a manner similar to the Data definition Language of RDBMs is independent of any kind of programming language. Object Query Language is the query language defined for ODMG binding with programming languages like SmallTalk, Java and C++.

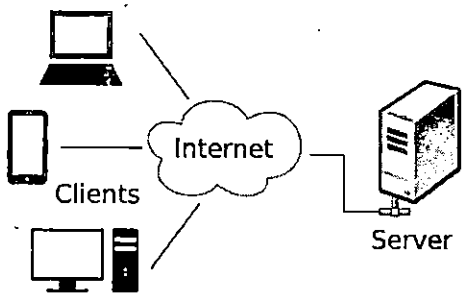


Figure 28. Client Server Systems

Apart from the general requirements of a computer system like the network software and the operating system, a client server system also has following components:

- Client, the resource user and
- Server, the resource and service provider

The applications are deployed on the client whereas the database management system resides on the server. The applications residing on the client make requests for services while the database on the server processes those requests and sends the results back to the client. The client server system can be scaled horizontally by adding or removing clients or vertically by migrating to a larger and faster server machines. Sometimes more than one server may be added to the architecture.

A Client server system consists of clients that are intelligent workstations through which the user can interact with the system, the server processing the requests, the communication networks connecting the clients and the server and several applications interacting with the clients and the server. The general architecture of the client server. Often, but not always several clients may be connected to a single server.

The clients have their own operating system and run one or more applications using the client machine's CPU, memory etc. The application communicates with the database system residing on the server using the database driver which acts as the middleware to establish a connection with the DBMS over the network.

The server machines are characterized by the database management system. These systems also have a listening daemon that accepts connections from the clients.Apart from the client and the server, the middleware plays an important role. The middleware is a small portion of software that sits between the client and the server, establishes the connection and sends requests and responses between the two. Such kind of architecture is described as two-tier architecture.

Sometimes, an application server is added to the client-server architecture to make it three-tier architecture. Some of the advantages of such architecture include scalability, flexibility, reduced cost, improved services to the clients and competitive advantage.

The middleware may interact through Remote Procedure Calls (RPC) or Message oriented middleware (MOM). Several commonly used client-server middleware include application program interface (API), Open database connectivity (ODBC), java database connectivity (JDBC), common object request broker architecture (CORBA) and Distributed Component Object Model (DCOM). Remote procedure calls provide a transparent mechanism such that a client communicates as if it is directly communicating with the server. Whereas message oriented middleware allows many to many communication via message queues.

Different types of software tools that are included in the development of client-server database systems include client builders used to develop the clients, database programming and administration tools that are used to develop and maintain the database and CASE/data modeling tools that help in the design and implementation of the database.

Like any other system, client server system also has its own advantages and drawbacks. Some of the advantages of the client server system include:

- Increased modularity
- Simple to implement
- Economy of communication
- In most of the cases it is hardware and software independent
- Improved functionality and efficiency
- Improved performance
- Due to the maintenance of the database on the server, it is more robust The disadvantages of such a system are as follows:
- Difficult failure detection due to increased modularity
- Security issues may arise with the increase in the number of clients
- Network traffic may cause problems when the number of clients is large
- Some of the commonly used database management systems like Oracle Server, Oracle Developer, Sybase Adaptive Server RDBMS utilize the functionality of client server systems.

Failure and Recovery

The database may fail due to several reasons.And the failure may affect part of the database or the entire database. This results into a short-term or long-term loss to the valuable information. Some of the common types of failure that may affect the database system include:

Notes

- Hardware failures including memory errors, disk crashes, bad disk sectors, system crash etc.
- Software failures that include failure of the software that may the failure of system software or application software
- System crashes are due to hardware or software problems and the entire system may stop functioning altogether. In client server architecture, failure of a single client may affect only the system concerned whereas failure of the server may affect the processing of all the systems connected to the server.
- Network failures may affect the network resulting into loss of communication between the database and the systems connected to the network.
- Application software errors cause logical errors in the application software
- Natural disasters such as flood, fire, earthquake may affect the functioning of the entire system or even the entire network
- Deliberate attacks include failures due to intentional corruption of the data, software or hardware of the system. Such attacks may be made to intentionally cause damage to the system.
- Media failure such as damage to the primary or secondary storage may cause loss of data.

Some types of failures are easy to fix and take less time to recover. One of the important aspects of failure and recovery is maintenance of backup. Depending upon the type of failure, the type and frequency of backups may vary.

Frequent backups may be required when database changes are frequent such as addition and deletion of tables, insertions and deletions of rows or columns in existing tables and frequent updates are done to the database.

Recovery algorithms are techniques to ensure transaction atomicity and durability despite failures. A recovery algorithm is used, that ensures atomicity by undoing the actions of transactions that do not commit and durability by making sure that all actions of committed transactions survive even if failures occur. Such an algorithm takes the database back to the most recent consistent state that was available before the failure.

In order to restore the database to the consistent state, the current state may be reconstructed by applying the changes again or any changes made to the database may be reversed so that the state of the database is maintined.

The system log or trail or journal is a useful tool that maintains the information about the changes that were made to the database. The basic unit of recovery of the database is a transaction. It should be ensured that either all the changes are made to the database or none are made. Two types of transaction recovery are as follows :

- Forward Recovery or REDO
- Backward Recovery or UNDO

Forward recovery or roll-forward is a technique in which the intermediate results of the transactions are stored in the buffers in the main memory. From the main memory, this data may be transferred to the secondary storage whenever required.

Two of the commonly used techniques for recovery techniques include log-based recovery and shadow paging.

The recovery techniques are as discussed below :

1. Deferred Update Techniques :

This technique defers the update to the database until the transaction commits. Till the commit, all the updates are stored in buffers or the local transaction workspace. These updates are recorded in the log and then written to the database.

In case the transaction fails before commit, no UNDO is required as the changes have not been saved to the database. But in case of a crash, it may be essential to REDO the committed transactions from the log in order to record their effects on the database. This technique is also known as NO-UNDO/REDO algorithm.

2. Immediate Update Techniques :

Before committing, the database may be updated by some operations. In order to perform recovery, the changes must be written to the log. If a transaction fails before commit, it must be rolled back to the consistent state by performing UNDO on the database. Also it may be required to REDO some of the transactions. This technique is also known as UNDO/REDO algorithm. A variation of the algorithm requires only REDO and is known as UNDO/ NO-REDO algorithm.

Database recovery restores the database to the most consistent state that existed before the failure. The consistent state is achieved by either completing all the transactions or aborting the transactions that cannot be completed. Maintaining a database backup is essential for recovery of the database. This backup can be achieved on secondary storage devices such as CDs, tapes etc.

Also, backup may be local or situated at a remote place. The level of backup required may also vary from one scenario to another. Full system backup may be required to maintain a copy of the entire system database. But due to the cost involved in the entire system backup, differential backup, differential backup may be done that maintains a copy of only some part of the database.

Forward Recovery (REDO):

The database recovery is achieved by flushing the data to the database buffers. This data is further transferred from the buffers to the secondary storage. The update operation becomes permanent only when this data is transferred to the secondary storage.

The failure may occur before writing the updates to the database or after the COMMIT operation. If the transaction has already committed, the redo operation writes the transaction updates to the database. This operation is called forward recovery or REDO or roll forward.

Backward Recovery (UNDO)

This type of recovery occurs when an error occurs in between the normal operation of the

Notes

database. Such recovery is called Backward Recovery, REDO or roll-backward. For instance backward recovery may be required on abnormal termination of the program or due to some wrongly entered value.

Such a recovery is essential to maintain the atomicity of the transactions. The transaction is started from the current state and is traversed in the backward direction. During the traversal the changes made to the database are undone.

Shadow Paging:

An alternative to log-based recoveries is shadow paging. It was introduced in 1977 by Lorie. In this technique, the database is considered to be made up of logical units of storage of fixed size blocks or pages. The virtual or logical blocks are mapped onto the physical blocks of same size. This mapping is done with the help of a page table. To page tables are maintained, the current page table and shadow page table. During the initial stages both the page tables are the same and point to the same blocks of physical storage.

3.30 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 eûcient queryprocessing that diûer 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 ûelds 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 spacerepresenting 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 needsis the capabilityto dealwith large collections of relatively simple geometric objects, for example, a set of 100,000 polygons.

This is somewhat diûerent from areas like CAD databases (solid modeling, etc.) where geometric entities are composed hierarchicallyinto complex structures, although the issues are certainlyrelated.

Aspatial database system is a database system which oûers SDTs in its data model and query language. It supports SDTs in its implementation, providing at least spatialindexing and eûcient algorithms for spatial join.

Temporal Database

Temporal database stores data relating to timeinstances. It offers temporal data types and stores information relating to past, present, and future time, for example, the history of the stock market or the movement of employees within an organization. Thus, a temporal database stores acollection of time related data.

A temporal database is formed bycompiling, storing temporal data. The difference between temporal data and non-temporal data is that a time-period is appended to data expressing when it was valid or stored in the database. The data stored by conventional databases consider data valid at present time as in the time instance "now." When data in such a database is modified, removed, or inserted, the state of the database is overwritten to form a new state.

The state prior to any changes to the database is no longer available. Thus, byassociate timewith data, it is possible tostore the different database states. In essence, temporal data is formed by time-stamping ordinary data (type of data we associate and store in conventional databases). In a relational data model, tuples are time-stamped andin an object-oriented data model, objects/attributes are timestamped.

Each ordinary data has two time values attached to it, a start time, and an end time to establish the time interval of the data. In a relational data model, relations are extended to have two additional attributes, one for start time and another for end time.

Different Forms of Temporal Databases:

Time can be interpreted as valid time (when data occurred or is true in reality) or transaction time (when data was entered into the database). Ahistorical database stores data with respect to valid time. Arollback database stores data with respect to transaction time. Abitemporal database stores data with respect to both valid and transaction time – they store the history of data with respect to valid time and transaction time.

Knowledge Database

The concepts of Knowledge Base Management System (KBMS) and the Knowledge Warehouse (KW) are analogues of Database Management System (DBMS) and Data Warehouse.

To arrive at a standard practice on the KBMS, and a standard definition of the Knowledge Warehouse, it's reasonable to begin with "straw man" definitions of both these concepts, next develop a general concept of this paper is a working paper, or "straw man," circulated

Notes

for purposes of collaboration within the Knowledge Management ConsortiumInternational's (KMCI) Artificial Knowledge Management Systems Committee (AKMSC). It is intended that this paper be used by the Committee, along with contributions of other committee members to arrive at a collaborative.

Standard Recommended Practice onArtificial Knowledge Base Management Systems, a product of the Committee and the KMCI.2 what a standard practice might encompass, and then subject these products to vigorous criticism and analysis bythe AKMSC.

To produce this straw man is the purpose of this paper, we will proceed byconsidering some basic distinctions among data, information, and knowledge, then discuss DBMSs, the DW, DW evolution, and Data Warehousing as a process, and then move from there to develop the analogous concepts in the knowledge and knowledge management sphere.

Data Warehouse

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.

Aparadox 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 diûerent 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.

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 materialfor 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 form traditional structural data to unstructured data like text ûles 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 fundamentalrecognition that information systems must be distinguished into operationaland informational systems. Operational systems support the day-to-day conduct of the business, and are optimized for fast response time of predeûned 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.

Characteristics of Data in Data Warehouse

Data in the Data Warehouse is integrated from various, heterogeneous operational systems like database systems, ûat ûles, etc. Before the integration, structural and semantic diûerences 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 highpayoû. Some examples where data cleaning becomes necessaryare: inconsistent ûeld lengths, inconsistent descriptions, inconsistent valueassignments, missingentries, and violation ofintegrityconstraints.

The need to access historical dataare one of theprimaryincentives 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 periodicalsnapshots 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 beneûts.

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.

Data Warehouse Architectures:

Data warehouses and their architectures vary depending upon the speciûcs of an organization's situation. Three common data warehouse architectures which are discussed in this section are:

Basic Data Warehouse Architecture

(a) Data Warehouse Architecture with a Staging Area

(b) Data Warehouse Architecture with a Staging Area and Data Marts

(c) Basic Data Warehouse Architecture

Data warehouses contain consolidated data from many sources, augmented with summary information and covering a long time period. Warehouses are muchlarger than other kinds of databases; sizes ranging from several gigabytes to terabytes are common.

Notes

Typical workloads involve ad hoc, fairly complex queries and fast response times are important. These characteristics differentiate warehouse applications from OLTP applications, and different DBMS design and implementation techniques must be used to achieve satisfactory results.

A distributed DBMS with good scalabilityand high availability (achieved by storing tables redundantly at more than one site) is required for verylarge warehouses.

An organization's daily operations access and modify operational databases. Data from these operational databases and other externalsources (e.g., customer profiles supplied by external consultants) are extracted by using gateways, or standard external interfaces supported by the underlying DBMSs.

A gateway is an application program interface that allows client programs to generate SQL statements to be executed at a server. Standards such as Open Database Connectivity (ODBC) and Open Linking and Embedding for Databases (OLE-DB) from Microsoft and Java Database Connectivity (JDBC) are emerging for gateways.

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. Adata 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.

Meta Data

In general meta data are deûned 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 ofthe data warehouse data, the tasks performed during the construction, maintenance and access of a data ware house, etc. Adata warehouse without adequate metadata are like "a ûling cabinet stuûed 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 ware house, 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 meta data describes the data elements as they exist in the ware house. 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 conûguration phase, a conceptual view of the ware house is ûrst speciûed according to user requirements which are often termed as data warehouse design. Then the involved data sources and the waydata 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.

Data Warehouse Design

Data warehouse design methods consider the read-oriented character of warehouse data and enables theeûcient queryprocessing 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 canbe 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 diûerent 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 traûc to the DW.

Data Tends to Exist at Multiple Levels of Dimensions :

Data can be deûned 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 eûcient and ûexible to achieve good and eûcient decision. Data Warehouse Should have a Capability for Rewriting History, that is, Allowing for "what-if" Analysis:

Notes

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 effcient 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.

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 conceptualand 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 beenassociated with highlynormalized 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 betweenthem, the focus is on identifying which information belongs to a centralfact table and which information belongs to its associated dimension tables.

In a nutshell, the logical designshould 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 datawarehouse schema.

Some of the logical warehouse design tools from Oracle are Oracle Warehouse Builder, Oracle Designer, which is a generalpurpose 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 designed for data ware house 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.

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 usefulis the snowûake 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 inwarehouse schema that store business measurements. Fact tables typically contain facts and foreign keys to the dimension tables. Fact tables represent data, usuallynumeric and additive that can be analyzed and examined. Dimension tables, also known as lookup or reference tables; containthe relativelystatic data in thewarehouse. Dimensiontablesstores 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 canbe aggregated bysimplearithmeticaladdition. 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 ofdetail 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.

Notes

Hierarchies:

Hierarchies are logical structures that use ordered levels as ameans of organizing data. Ahierarchy can be used to deûne data aggregation. For example, in a time dimension, a hierarchymight aggregate data from the month level to the quarter level to the year level.

A hierarchy can also be used to deûne a navigational drill path and to establisha familystructure. Withina hierarchy, eachlevelis logically connected to the levels above and below it. Data values at lower levels aggregate into the data values at higher levels. Adimension can be composed of more than one hierarchy.

Hierarchies impose a familystructure 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 designphase is converted into a description of the physical database structure. Physical design decisions are mainly driven by query performance and database maintenance aspects.

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 ûles, which are physical structures within the operating system. Adata ûle 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 bydiûerences. 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.

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 canbe saved bycompressing heap-organized tables. Atypicaltype 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 manyforeign keys.

Views:

A view is a tailored presentation of the data contained in one or more tables or other views. Aview 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 diûer 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 speciûc 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 deûnes hierarchical relationships between columns or column sets. Ahierarchical 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. Atypical dimension is city, state (or province), region, and country.

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 classiûed as follows:

a) Traditional queryand 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.

Notes

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.

Multi-dimensional 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, artiûcial 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. Beneûts of data visualization include the ability to better observe trends and patterns, and to identify correlations and clusters.

Difference Between Database and Data Warehouse

A Computer Database is a structured collection of records or data that is stored in a computer system. The structure is achieved by organizing the data according to a database model. The model in most common use today is the relational model. Other models such as the hierarchical model and the network model use a more explicit representation of relationships (see below for explanation of the various database models).

A computer data base relies upon software to organize the storage of data. This software is known as a database management system(DBMS). best examples are Mysql, Oracle etc.

Adata warehouse is a repository of an organization's electronically stored data. Data warehouses are designed to facilitate reporting and analysis.

However, the means to retrieve and analyze data, to extract, transformand load data, andto manage the datadictionary are also considered essentialcomponents of a data warehousing system. Manyreferences to data warehousing use this broader context.

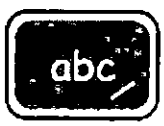
Thus, an expanded definition for data warehousing includes business intelligence tools, tools to extract, transform, and load data into the repository, and tools to manage and retrieve meta data.

SUMMARY

- SQL (Structured Query Language) is a database sublanguage for querying and modifying
- relational databases. It was developed by IBM Research in the mid 70's and standardized by ANSI in
- 1986. SQL is a version of Relational Calculus. The basic structure in SQL in the statement. Semicolons separate multiple SQL statements.
- Using different aggregate functions, joins and different set operations we can perform more complex queries on DBMS and get desired outputs in different formats.
- Joins are used to combine columns from different tables..
- The different types of joins are: equi joins, cartesian joins, outer joins, self joins and nonequi joins.
- Joing condition is specified in the WHERE clause of the SELECT statement.
- When two tables are joined together using equality of values in one or more columns, they make an Equi Join.
- Without any joining condition the join becomes a cartesian join.
- Join a table with itself is know is self join.
- Self Join is possible by providing table name aliases for the table.
- With joins, the names of all the table to be joined, need to be specified.
- To select a row forcfully which cannot be selected using equi join outer join symbol (+). is used.
- Set operator are used to combine result form different queries. the operator used are UNION, INTERSECT and MINUS.
- The UNION clause merges the outputs of two or more queries into a single set of rows and columms.
- The intersect operater returens tha rows thet are commen between two sets of rows.
- Minus operator returns the rows unique to first query
- Nested queries are used in a situation where the condition of the query is dependent on the outcome of an inner query. Subqueries can also used within tha HAVING clause.
- A co-related subquery is a nested subquery which is executed once for each 'candidate row considered by themain query and which on execution uses a value from a column in the outer query
- The special operators used with subqueries are :EXIST, SOMEANY and ALL.
- The EXISTS operator is used to be test for the presence of the value. If the value exists it returns TRUE; if it does not exist returns FALSE.

Notes

- The ANY operator compares tha lowest value from the set.
- ALL operator returns TRUE if every value selected by the subquery satisfise the condition in the predicate of the outer query
- In this unit we discussed the recovery of the data contained in a database system after failures of various types. The aim of the recovery scheme is to allow database operations to be resumed after a failure with minimum loss of information and at an economically justifiable cost. The checkpoint information is used to limit the amount of recovery operations to be done following a system crash resulting in the loss of volatile storage. The archival database is the copy of the database at a given time stored to stable storage.
- Whenever a transaction is run against a database , a number of options can be used in reflecting the modifications made by the transactions. the options we have examined are update in place and indirect update with careful replacement: the shadow page scheme and the update vai log scheme are two versions of the later. In the update in place scheme, the transaction updates the physical database and the modified record replaces the old record in the database. The write ahead log strategy is used.
- Database Administration is the function of manging and maintaining database management system (DBMS) software.
- Client/Server Computing is the logical extension of modular programming. Modular programming has as its fundamental assumption that sepration of a large piece of software into its constituent parts creats the possibility for easier development and better maintainbility.
- Distributed computing system consists of a number of processing elements, not necessarily homogeneous that are interconnected by a computer network, and that cooprate in performaing certain assigned task



KEY WORDS

- **Multi-version Concurrency Control (MVCC)** - MVCC is a concurrency control method which allows for multiple types of database access to occur simultaneously. RDM implements this through the use of database snapshots.
- **Natural Join** - A join formed between two tables where the values of identically named and defined columns are equal.
- **Network Model** - A database in which inter-record type relationships are organized using one-to-many sets. This differs from a Hierarchical Model in that it allows a record type to be a member of more than one set. Individual rows can be retrieved using API functions that allow an application to navigate through individual set instances.
- **Network** - An inter-connection of computers and computing devices, all of which can send and receive messages from one another. The world's largest network is the internet, in which billions of computers are connected.
- **NoSQL** - A classification of data storage systems that are not primarily designed to be

relationally accessed through the common SQL language. NoSQL systems are characterized by dynamic creation and deletion of key/value pairs and are structured to be highly scalable to multiple computers.

- **Object-oriented** - A computing programming paradigm that defines the computing problem to be solved as a set of objects that are members of various object classes each with its own set of data manipulation methods. Individual objects which have been instantiated (created) can be manipulated only using those prescribed methods.
- **Open Source Software (OSS)** - Software that is released under a Software License that (1) permits each recipient of the software to copy and modify the software; (2) permits each recipient to distribute the software in modified or unmodified form; and (3) does not require recipients to pay a fee or royalty for the permission to copy, modify, or distribute the software.

REVIEW QUESTIONS

1. How DDL commands are different from DML? Explain with the help of suitable example.
2. Explain the different data types of SQL with the help of example.
3. What do you understand by sub queries? Give example and explain.
4. What will be the output if we use NOT IN operator instead of NOT EXISTS in the above query?
5. Explain the use of join in RDBMS with the help of example.
6. Discuss the advantages of SQL over old database software packages.
7. What do you understand by aggregate functions in SQL? Explain.
8. Explain the basic difference between where and having clause of select statement.
9. Explain the different set operations of SQL with the help of suitable example.
10. Explain the use of order by clause of select statement.
11. List all employees name along with their manager's name.Also list the name of that who has no manager. (Employee KING has no Manager)
12. Dispaly the department that has no employee.
13. List the employee details who earn minimum salary for their job.
14. List the ename, salary, deptno for those employees who earn salary greater than average salary for their department. Sort the output in department number order.
15. List the employee details who earn highest salary for their job.
16. List the details of those employees who are among the five highest earners of the company.

Notes

17. How is the checkpoint information used in the recovery operation following a system crash?
18. What is concurrency in dbms? Explain.
19. What is lost update problem? Discuss.
20. Describe the inconsistent read problem in dbms
21. What is serializability and how it works in concurrency?
22. Write a note on precedence graph?
23. What is locking scheme and how it is utilized in database?
24. Write a note on granularity of locking
25. Describe the hierarchy of locks and intention mode locking.
26. How client/server databases are different from distributed databases?
27. Explain the working of CASE tools with the help of example.
28. What do you understand by repositories explain?
29. Explain the different types of failures of databases with the help of example.
30. Discuss the concept of databases security by giving example of any practical database.

FURTHER READINGS

1. "database, n". OED Online. Oxford University Press. June 2013. Retrieved July 12, 2013. (Subscription required.)
2. IBM Corporation (October 2013). "IBM Information Management System (IMS) 13 Transaction and Database Servers delivers high performance and low total cost of ownership". Retrieved Feb 20, 2014.
3. "Structured Query Language (SQL)". International Business Machines. October 27, 2006. Retrieved 2007-06-10. Wagner 2010.
4. Ramalho, J.C., Faria, L., Helder, S., & Coutada, M. (2013, December 31). Database Preservation Toolkit: A flexible tool to normalize and give access to databases. University of Minho. https://core.ac.uk/display/55635702?
5. Kroenke, David M. and David J. Auer. Database Concepts. 3rd ed. New York: Prentice, 2007.
6. Raghu Ramakrishnan and Johannes Gehrke, Database Management Systems
7. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, Database System Concepts

Unit 4

Database Utilities

Notes

Structure

4.0 LEARNING OBJECTIVES

After reading this chapter students will be able to:

- Know about DBMS
- Discuss the database utilities
- Know about the security and security services
- Describe the object/basic database administration
- Understand the RDA
- Discuss the use of Remote Data Access

Notes

4.1 INTRODUCTION

A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data. This is a collection of related data with an implicit meaning and hence is a database. The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient. By data, we mean known facts that can be recorded and that have implicit meaning.

The management system is important because without the existence of some kind of rules and regulations it is not possible to maintain the database. We have to select the particular attributes which should be included in a particular table; the common attributes to create relationship between two tables; if a new record has to be inserted or deleted then which tables should have to be handled etc. These issues must be resolved by having some kind of rules to follow in order to maintain the integrity of the database.

Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.

Because information is so important in most organizations, computer scientists have developed a large body of concepts and techniques for managing data.

A Database management system is a computerized record-keeping system. It is a repository or a container for collection of computerized data files. The overall purpose of DBMS is to allow he users to define, store, retrieve and update the information contained in the database on demand. Information can be anything that is of significance to an individual or organization.

Databases touch all aspects of our lives. Some of the major areas of application are as follows:

- Banking
- Airlines
- Universities
- Manufacturing and selling
- Human resources

4.2 DATABASE UTILITIES

The database utility is the interface between the ABAP Dictionary and the relational database underlying the R/3 System. You can use the database utility to edit all the database objects that are generated from objects of the ABAP Dictionary. These are database tables that are generated from transparent tables or phys ical table pools or table clusters, indexes, database views.

It is mainly used when a table is changed in the ABAP Dictionary. At that time, we

must ensure that the database structure of the table is adjusted to the change in the ABAP Dictionary duringactivation.

You can call the database utility from the initial screen of the ABAP/4 Dictionary (SE11) with Utilities -> Database utility or usingT-code SE14.

The database utility is the interface between the ABAP Dictionary and the relational database underlying the SAP system. The database utility allows you to edit (create, delete and adjust to changes to their definition in the ABAP Dictionary) database objects derived from objects of the ABAP Dictionary. In the following figure, you can see the place of the database utility.

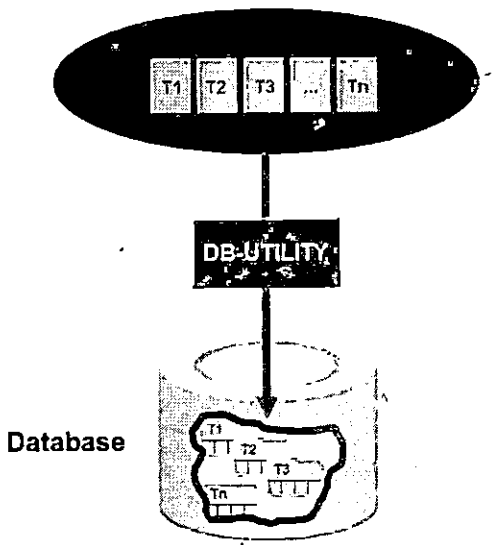


Figure 1.

The database structure of a table can be adjusted to its changed ABAP Dictionary definition inthree ways:

- By deleting and recreating the database table The table is deleted in the data base. The revised version of the table is then activated in the ABAP Dictionary and created again in the database. Data in the ta ble is lost during this process.
- By changing the data base ca talog(ALTER TABLE) on ly the defi nition of the table is cha nged in the database. Data in the table is retai ned. The indexes on the ta ble might have to be rebu ilt.
- By convert ing th e table.The database table is renamed and serves as a temporary buffer for the data. The revised version of the ta ble is activated in the ABAP Dictionary a nd created in the database. The data is reloaded from the temporary buffer to the new database table (with MOVE CORRESPO N DI NG) and t he indexes on the table are bu ilt.

The database utility provides a number of options for administering and monitoring requests for database modifications. You can perform these functions directly in the initial

Notes

screen of the database utility modifications. You can perform these functions directly in the initial screen of the database utility. You can:

- Schedule jobs for mass processing
- Display requests for mass processing
- Display logs for mass processing
- Display temporary tables without restart logs

4.3 SECURITY IN DBMS

Database security refers to the range of tools, controls, and measures designed to establish and preserve database confidentiality, integrity, and availability. This article will focus primarily on confidentiality since it's the element that's compromised in most data breaches.

Database security must address and protect the following:

- The data in the database
- The database management system (DBMS)
- Any associated applications
- The physical database server and/or the virtual database server and the underlying hardware
- The computing and/or network infrastructure used to access the database

Database security is a complex and challenging endeavor that involves all aspects of information security technologies and practices. It's also naturally at odds with database usability. The more accessible and usable the database, the more vulnerable it is to security threats; the more invulnerable the database is to threats, the more difficult it is to access and use. (This paradox is sometimes referred to as Anderson's Rule. (link resides outside IBM).

Database Security and Threats

Data security is an imperative aspect of any database system. It is of particular importance in distributed systems because of arge number of users, fragmented a nd replicated data, multiple sites and distributed control.

Threats in a Database

- **Availability loss**- Availability loss refers to non-availability of database objects by legitimate users.
- **Integrity loss**-Integrity loss occurs when unacceptable operations are performed upon the database either accidentally or maliciously. This may happen while creating, inserting, updating or deleting data. It results incorrupted data leading to incorrect decisions.
- **Confidentiality loss**- Confidentiality loss occurs due to unauthorized or unintentional disclosure of confidential information. It may result inillega lactions, security threats and loss in public confidence.

Measures of Control

The measures of control can be broadly divided into the following categories:

- **Access Control** - Access control includes security mechanisms in a database management system to protecc against unauthorized access. A user can gain access ro the database after clearing the login process through only valid user accounts. Each user account is password protected.
- **Flow Control** - Distributed systems encompass a lot of data flow from one site to another and also within a site. Flow control preven ts da ta from being transferred in such a way that it can be accessed by unauthorized agents. A flow policy lists out the channels through which information can flow. It also defines security classes for data as wellas transactions .
- Data Encryption - Data encryption refers to coding data when sensitive data is to be communicated over public channels. Even if an unauthorized agent gains access of the data, he cannot understand it since it is in an incomprehensible format.

Cryptography is the science of encoding information before sending via unreliable communication paths so that only in an authorised receiver can decode and use it.

The coded message is called cipher text and the original message is called plain text. The process of converting plain text to cipher text by the sender is called encoding or encryption. The process of converting cipher text to plain text by the receiver is called decoding or decryption.

The entire procedure of communicating using cryptography can be illustrated through the following diagram :

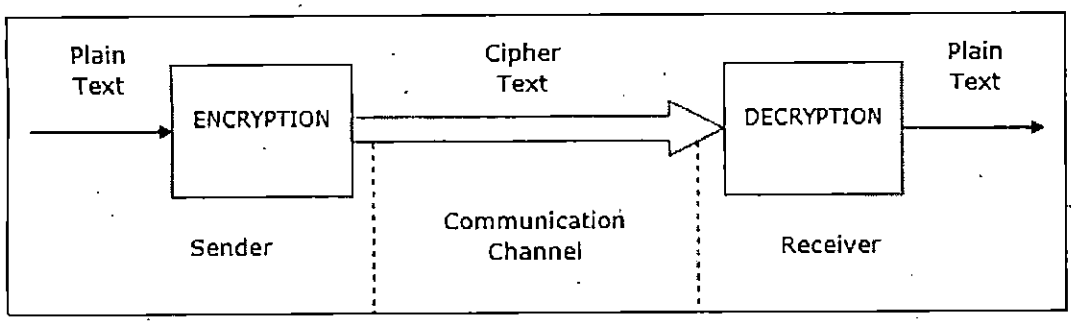


Figure 2.

Why is it important

By definition, a data breach is a failure to maintain the confidentiality of data in a database. How much harm a data breach inflicts on your enterprise depends on a number of consequences or factors:

1. **Compromised intellectual property:** Your intellectual property—trade secrets, inventions, proprietary practices—may be critical to your ability to maintain a competitive advantage in your market. If that intellectual property is stolen or exposed, your competitive advantage

Notes

may be difficult or impossible to maintain or recover.

2. **Damage to brand reputation:** Customers or partners may be unwilling to buy your products or services (or do business with your company) if they don't feel they can trust you to protect your data or theirs.
3. **Business continuity (or lack thereof):** Some business cannot continue to operate until a breach is resolved.
4. **Fines or penalties for non-compliance:** The financial impact for failing to comply with global regulations such as the Sarbannes-Oxley Act (SAO) or Payment Card Industry Data Security Standard (PCI DSS), industry-specific data privacy regulations such as HIPAA, or regional data privacy regulations, such as Europe's General Data Protection Regulation (GDPR) can be devastating, with fines in the worst cases exceeding several million dollars per violation.
5. **Costs of repairing breaches and notifying customers:** In addition to the cost of communicating a breach to customer, a breached organization must pay for forensic and investigative activities, crisis management, triage, repair of the affected systems, and more.

4.4 SECURITY SERVICES

Secured communication,requires the following four basic services:

- **Privacy:** A person (say Sita) should be able to send a message to another person (say Ram) p.rivately. It implies that to all others the message should be unintelligible.
- **Authentication:** After the message is received by Ram, he should be sure that the message has been sent by nobody else but by Sita.
- **Integrity:** Ram should be sure that the message has not been tampered on transit.
- **Nonrepudiation:** Ram should be able to prove at a later stage that the message was indeed received from Sita.
- **Integrity:** Ram should be sure that the message has not been tampered on transit.
- Nonrepudiation: Ram should be able to prove at a later stage that the message was indeed received from Sita.

4.5 PRIVACY

Privacy can be achieved using symmetric key cryptography. In this case, the key is shared between the sender (Sita) and the receiver (Ram) as shown in Fig. 3. Privacy can also be achieved by using public-key cryptography as shown in Fig. 4. However, in this case the owner should be verified.

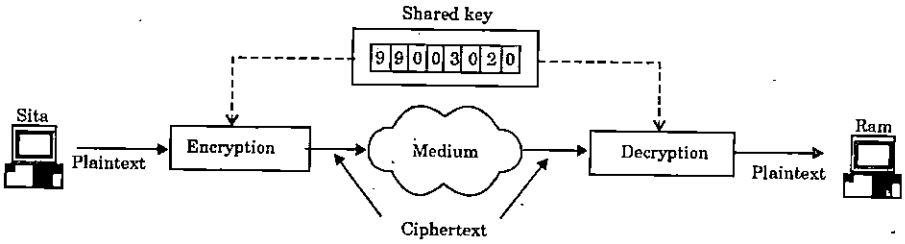


Figure 3. Privacy using private-key cryptography.

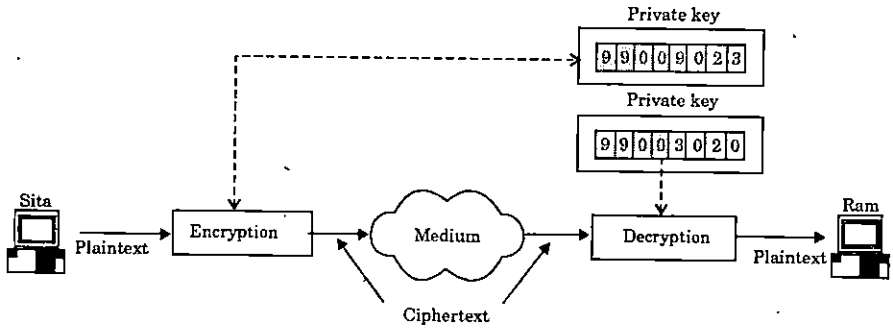


Figure 4. Privacy using public-key cryptography.

4.6 AUTHENTICATION, INTEGRITY AND NONREPUDIATION USING DIGITAL SIGNATURE

By message authentication we mean that the receiver should be sure about sender's identity. One approach to provide authentication is with the help of digital signature. The idea is similar to signing a document. Digital Signature provides the remaining three security services; Authentication, Integrity and Nonrepudiation.

Digital Signature

There are two alternatives for Digital Signature:

- Signing the entire document
- Signing the digest

In the first case the entire document is encrypted using private key of the sender and at the receiving end it is decrypted using the public key of the sender as shown in Fig. 5. For a large message this approach is very inefficient.

In the second case a miniature version of the message, known as digest, is encrypted using the private key of the sender and then the signed digest along with the message is sent to the receiver as shown in Fig. 6. The receiver decrypts the signed digest using the public key of the sender and the digest created using the received message is compared with the

Notes

decrypted digest as shown in Fig. 7. If the two are identical, it is assumed that the sender is authenticated. This is somewhat similar to error detection using parity bit.

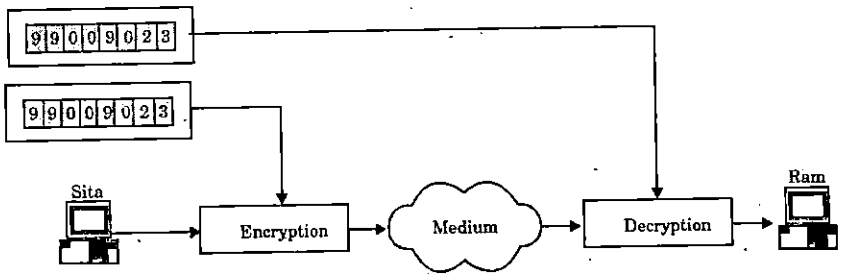


Figure 5. Authentication by signing the whole document.

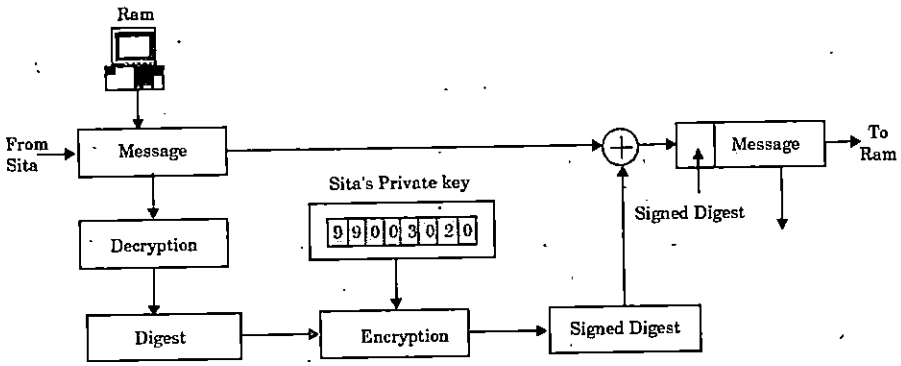


Figure 6. Sender site for authentication by signed digest.

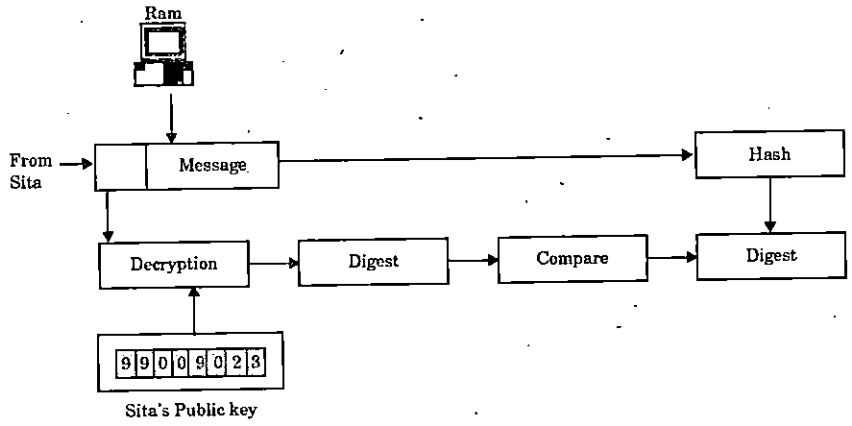


Figure 7. Receiver site for authentication by signed digest.

Some key features of this approach are mentioned below:

- Digital signature does not provide privacy
- Hash function is used to create a message digest
- It creates a fixed-length digest from a variable-length message

- Most common Hash functions:
- MD5 (Message Digest 5): 120-bit
- SHA-1 (Secure Hash algorithm 1): 160-bit
- Important properties:
- One-to-One
- One-way

4.7 OBJECT/BASIC DATABASE

A database object is any defined object in a database that is used to store or reference data. Anything which we make from create command is known as Database Object. It can be used to hold and manipulate the data.Some of the examples of database objects are : view, sequence, indexes, etc.

Table – Basic unit of storage; composed rows and columns

View – Logically represents subsets of data from one or more tables

Sequence – Generates primary key values

Index – Improves the performance of some queries

Synonym – Alternative name for an object

Different database Objects :

Table – This database object is used to create a table in database.

Syntax :

```
CREATE TABLE [schema.]table
        (column datatype [DEFAULT expr][, ...]);
```

Example :

```
CREATE TABLE dept
      (deptno NUMBER(2),
       dname VARCHAR2(14),
       loc VARCHAR2(13));
```

Output :

```
DESCRIBE dept;
```

View – This database object is used to create a view in database.A view is a logical table based on a table or another view. A view contains no data of its own but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called base tables. The view is stored as a SELECT statement in the data dictionary.

Syntax :

Notes

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
                [(alias[, alias]...)]
                AS subquery
                [WITH CHECK OPTION [CONSTRAINT constraint]]
                [WITH READ ONLY [CONSTRAINT constraint]];
```

Example :

```
CREATE VIEW salvu50
        AS SELECT employee_id ID_NUMBER, last_name NAME,
        salary*12 ANN_SALARY
        FROM employees
        WHERE department_id = 50;
```

Output :

```
SELECT *
FROM salvu50;
```

Sequence – This database object is used to create a sequence in database.A sequence is a user created database object that can be shared by multiple users to generate unique integers. A typical usage for sequences is to create a primary key value, which must be unique for each row.The sequence is generated and incremented (or decremented) by an internal Oracle routine.

Syntax :

```
CREATE SEQUENCE sequence
            [INCREMENT BY n]
            [START WITH n]
            [{MAXVALUE n | NOMAXVALUE}]
            [{MINVALUE n | NOMINVALUE}]
            [{CYCLE | NOCYCLE}]
            [{CACHE n | NOCACHE}];
```

Example :

```
CREATE SEQUENCE dept_deptid_seq
                INCREMENT BY 10
                START WITH 120
                MAXVALUE 9999
                NOCACHE
                NOCYCLE;
```

Check if sequence is created by :

```
SELECT sequence_name, min_value, max_value,
```

increment_by, last_number
FROM user_sequences;

Index: This database object is used to create a indexes in database.An Oracle server index is a schema object that can speed up the retrieval of rows by using a pointer.Indexes can be created explicitly or automatically. If you do not have an index on the column, then a full table scan occurs.

An index provides direct and fast access to rows in a table. Its purpose is to reduce the necessity of disk I/O by using an indexed path to locate data quickly. The index is used and maintained automatically by the Oracle server. Once an index is created, no direct activity is required by the user.Indexes are logically and physically independent of the table they index. This means that they can be created or dropped at any time and have no effect on the base tables or other indexes.

Syntax :

CREATE INDEX index
ON table (column[, 'column]...);

Example :

CREATE INDEX emp_last_name_idx
ON employees(last_name);

Synonym – This database object is used to create a indexes in database.It simplify access to objects by creating a synonym(another name for an object). With synonyms, you can Ease referring to a table owned by another user and shorten lengthy object names:To refer to a table owned by another user, you need to prefix the table name with the name of the user who created it followed by a period. Creating a synonym eliminates the need to qualify the object name with the schema and provides you with an alternative name for a table, view, sequence,procedure, or other objects. This method can be especially useful with lengthy object names, such as views.

In the syntax:

PUBLIC : creates a synonym accessible to all users

synonym : is the name of the synonym to be created

object : identifies the object for which the synonym is created

Syntax :

CREATE [PUBLIC] SYNONYM synonym FOR object;

Example :

CREATE SYNONYM d_sum FOR dept_sum_vu;

4.8 EDITING DATABASE OBJECTS

You can use the database utility to edit all the database objects that are generated from objects of the ABAP Dictionary. These are database tables that are generated from transparent tables or physical table pools or table clusters, indexes, and database views.

Notes

To call the database utility for a object, proceed as follows:

- Open the Repository Browser (Transaction Code SE80) and navigate to the required object.
- Open the object in display mode by double-clicking on it.
- Choose Utilities → Database Object → Database Utility.

Since different functions are required for different object types, a separate maintenance screen is displayed for each object type.

Analyzing and Continuing Terminated Conversions

If a conversion terminates, you can find the reason using the database utility. After correcting the error you can continue the conversion with the database utility. For more information, see:

- Conversion Problems
- Continuing Terminated Conversions
- Finding Terminated Conversions

Administration of Requests for Database Modifications

The database utility provides a number of options for administering and monitoring requests for database modifications. You can perform these functions directly on the initial screen of the database utility (transaction SE14).

Executing and Monitoring Incremental Conversions

You can reorganize a table and change its structure with an incremental table conversion. In contrast to a standard conversion, when an incremental conversion is performed the system can continue using the table during the data transfer. This means that there is much more time available for the data transfer and larger sets of data can be converted than with the standard conversion method. There is a short reduction in productivity only when initializing the incremental conversion and when switching to the new table.

The monitor for the incremental conversion can be displayed from the initial screen of the database utility with DB requests → Incremental.

To display more information about the flow of an incremental conversion in the monitor of the incremental conversion, choose with the quick info text Information. An assistant helps you to perform an incremental conversion.

4.9 CONVENTIONAL ENCRYPTION METHODS

In conventional cryptography, the encryption and decryption is done using the same secret key. Here, the sender encrypts the message with an encryption algorithm using a copy of the secret key. The encrypted message is then send over public communication channels. On receiving the encrypted message, the receiver decrypts it with a corriesponding decryption

algorithm using the same secret key.

Security in conventional cryptography depends on two factors:

- Asound algorithm which is known to all.
- A randomly generated, preferably long secret key known only by the sender and the receiver.

The most famous conventional cryptography algorithm is **Data Encrypted Standard** or **DES.**

The advantage of this method is its easy applicability. However, the greatest problem of conventional cryptography is sharing the secret key between the communicating parties. The ways to send the key are cumbersome and highly susceptible to evasdropping.

4.10 USER AUTHENTICATION USING SYMMETRIC KEY CRYPTOGRAPHY

User authentication is different from message authentication. In case of message authentication, the identity of the sender is verified for each and every message. On the other hand, in user authentication, the user authentication is performed once for the duration of system access.

In the first approach, the sender (Sita) sends her identity and password in an encrypted message using the symmetric-key ksr and then sends the message as shown in Fig. 8. However, an intruder (say Ravana) can cause damage without accessing it. He can also intercept both the authentication message and the data message, store them and then resends them, which is known as replay attack

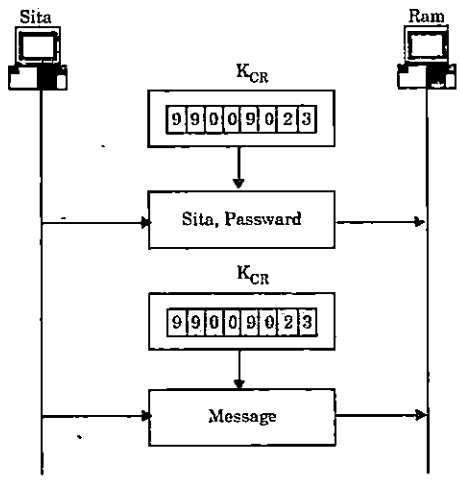


Figure 8. User authentication using symmetric key cryptography.

Notes

Public Key Cryptography

In contrast to conventional cryptography, public key cryptography uses two different keys, referred to as the public key and the private key. Public-key cryptography, or asymmetric cryptography, is a cryptographic system that uses pairs of keys. Each pair consists of a public key (which may be known to others) and a private key (which may not be known by anyone except the owner).

The generation of such key pairs depends on cryptographic algorithms which are based on mathematical problems termed one-way functions. Effective security requires keeping the private key private; the public key can be openly distributed without compromising security.

In such a system, any person can encrypt a message using the intended receiver's public key, but that encrypted message can only be decrypted with the sender's private key. This allows, for instance, a server program to generate a cryptographic key intended for a suitable symmetric-key cryptography, then to use a client's openly-shared public key to encrypt that newly generated symmetric key.

The server can then send this encrypted symmetric key over an insecure channel to the client; only the client can decrypt it using the client's private key (which pairs with the public key used by the server to encrypt the message).

With the client and server both having the same symmetric key, they can safely use symmetric key encryption (likely much faster) to communicate over otherwise-insecure channels. This scheme has the advantage of not having to manually pre-share symmetric keys (a fundamentally difficult problem) while gaining the higher data throughput advantage of symmetric-key cryptography.

With public-key cryptography, robust authentication is also possible. A sender can combine a message with a private key to create a short digital signature on the message. Anyone with the sender's corresponding public key can combine that message with a claimed digital signature; if the signature matches the message, the origin of the message is verified (i.e., it must have been made by the owner of the corresponding private key).

Public key algorithms are fundamental security primitives in modern cryptosystems, including applications and protocols which offer assurance of the confidentiality, authenticity and non-repudiability of electronic communications and data storage. They underpin numerous Internet standards, such as Transport Layer Security (TLS), S/MIME, PGP, and GPG.

Some public key algorithms provide key distribution and secrecy (e.g., Diffie–Hellman key exchange), some provide digital signatures (e.g., Digital Signature Algorithm), and some provide both (e.g., RSA). Compared to symmetric encryption, asymmetric encryption is rather slower than good symmetric encryption, too slow for many purposes. Today's cryptosystems (such as TLS, Secure Shell) use both symmetric encryption and asymmetric encryption, often by using asymmetric encryption to securely exchange a secret key which is then used for symmetric encryption.

Two of the best-known uses of public key cryptography are:

- Public key encryption, in which a message is encrypted with the intended recipient's public key. For properly chosen and used algorithms, messages cannot in practice be decrypted by anyone who does not possess the matching private key, who is thus presumed to be the owner of that key and so the person associated with the public key.

This can be used to ensure confidentiality of a message.

- Digital signatures, in which a message is signed with the sender's private key and can be verified by anyone who has access to the sender's public key. This verification proves that the sender had access to the private key, and therefore is very likely to be the person associated with the public key. It also proves that the signature was prepared for that exact message, since verification will fail for any other message one could devise without using the private key.

The solution is to use a combination of conventional and public key cryptography. The secret key is encrypted using public key cryptography before sharing between the communicat ing parties. Then, the message is send using conventionalcryptography with the aid of the shared secret key.

Using nonce, a large random number used only once.

To prevent the replay attack, the receiver (Ram) sends nonce, a large random number that is used only once to the sender (Sita) to challenge Sita. In response Sita sends an encrypted version of the random number using the symmetric key. The procedure is shown in Fig. 9.

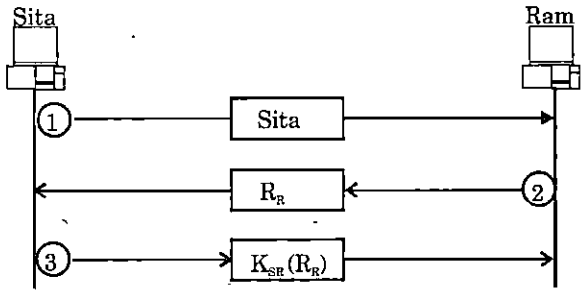


Figure 9 User authentication using a nonce.

Bidirectional Authentication

In the bidirectional authentication approach, Ram sends nonce to challenge Sita and Sita in turn sends nonce to challenge Ram as shown in Fig. 10. This protocol uses extra messages for user authentication. Protocol with lesser number of messages is possible as shown in Fig. 10.

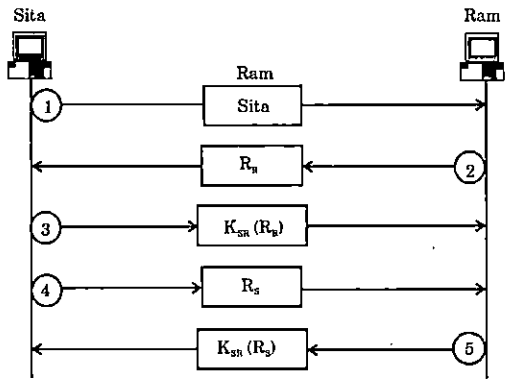


Figure 10. Bidirectional authentication using a nonce.

Notes

4. 11 USER AUTHENTICATION USING PUBLIC KEY CRYPTOGRAPHY

Public key cryptography can also be used to authenticate a user. The procedure is shown in Fig. 10.

ER = Public key of Ram,

Es = Public key of Sita

rs = nonce by Sita,

RR = nonce by Ram

Ks = Session key sent by Ram

4.12 KEY MANAGEMENT

Although symmetric-key and public-key cryptography can be used for privacy and user authentication, question arises about the techniques used for the distribution of keys. Particularly, symmetric-key distribution involves the following three problems:

- For *n* people to communicate with each other requires n(n-l)/2 keys. The problem is aggravated as n becomes very large.
- Each person needs to remember (n-1) keys to communicate with the the remaining (n-1) persons.

How the two parties will acquire the shared key in a secured manner?

- In view of the above problems, the concept of session key has emerged. A session key is created for each session and destroyed when the session is over. The Diffie-Hellman protocol is one of the most popular approach for providing one-time session key for both the parties.

Diffie-Hellman Protocol

Key features of the Diffie-Hellman protocol are mentioned below and the procedure is given in Fig. 11.

- Used to establish a shared secret key
- Prerequisite: N is a large prime number such that (N–1)/2 is also a prime number.
- G is also a prime number. Both N and G are known to Ram and Sita..
- Sita chooses a large random number x and calculates R_1 = Gx mod N and sends it to Ram
- Ram chooses another large random number y and calculates R_2 = Gymod N and sends it to Sita

- Ram calculates K = $(R_1)y$ mod N
- Sita calculates K = $(R_2)x$ mod N

Key Management using KDC

It may be noted that both R_1 and R_2 are sent as plaintext, which may be intercepted by an intruder. This is a serious flaw of the Diffie-Hellman Protocol. Another approach is to use a trusted third party to assign a symmetric key to both the parties. This is the basic idea behind the use of key distribution center (KDC).

Key Management using Kerberos

Another popular authentication protocol known as Kerberos. It uses an authentication server (AS), which performs the role of KDC and a ticket-granting server (TGS), which provides the session key (KAB) between the sender and receiver parties. Apart from these servers, there is the real data server say Ram that provides services to the user Sita.

The operation of Kerberos is depicted with the help of Fig. 16.12. The client process (Sita) can get a service from a process running in the real server Ram after six steps as shown in the Fig..

The steps are as follows:

- Step 1: Sita uses her registered identity to send her message in plaintext.
- Step 2: The AS server sends a message encrypted with Sita's symmetric key Ks. The message contains a session key Kse, which is used by Sita to contact the TGS and a ticket for TGS that is encrypted with the TGS symmetric key KTG.
- Step 3: Sita sends three items to the TGS; the ticket received from the AS, the name of the real server, and a timestamp encrypted by Kse. The timestamp prevents replay by Ram.
- Step 4: The TGS sends two tickets to Sita. The ticket for Sita encrypted with Kse and the ticket for Ram encrypted with Ram's key. Each of the tickets contains the session key ksr between Sita and Ram.
- Step 5: Sita sends Ram's ticket encrypted by ksr.
- Step 6: Ram sends a message to Sita by adding 1 to the timestamp confirming the receipt of the message using ksr as the key for encryption.

Following this Sita can request and get services from Ram using ksr as the shared key.

4.13 APPLICATION LAYER SECURITY

Based on the encryption techniques we have discussed so far, security measures can be applied to different layers such as network, transport or application layers. However, implementation of security features in the application layer is far simpler and feasible compared to implementing at the other two lower layers. In this subsection, a protocol known as Pretty Good Privacy (POP), invented by Phil Zinmermann, that is used in the application layer to provide all the

Notes

four aspects of security for sending an email is briefly discussed. POP uses a combination of private-key and public key for privacy. For integrity, authentication and nonrepudiation, it uses a combin

4.14 VIRTUAL PRIVATE NETWORK (VPN)

With the availability of huge infrastructure of public networks, the Virtual Private Network (VPN) technology is gaining popularity among enterprises having offices distributed throughout the country. Before we discuss about the VPN technology, let us first discuss about two related terms: intranet and extranet.

Intranet is a private network (typically a LAN) that uses the internet model for exchange of information. A private network has the following features:

- It has limited applicability because access is limited to the users inside the network
- Isolated network ensures privacy
- Can use private IP addresses within the private network

Extranet is same as the intranet with the exception that some resources can be allowed to access by some specific groups under the control of network administrator.

Privacy can be achieved by using one of the three models: Private networks, Hybrid Networks and Virtual Private Networks.

Private networks: A small organization with a single site can have a single LAN whereas an organization with several sites geographically distributed can have several LANs connected by leased lines and routers as shown in Fig. 11.

In this scenario, people inside the organization can communicate with each other securely through a private internet, which is totally isolated from the global internet.

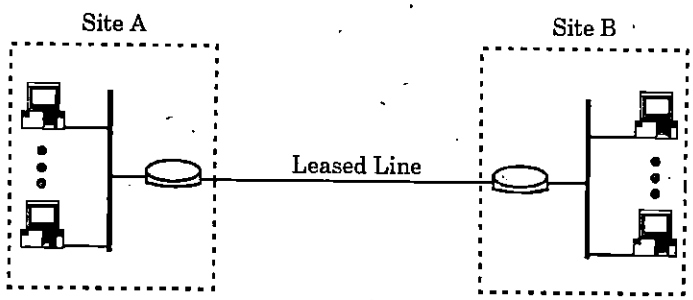


Figure 11 Private network with two LAN sites.

Hybrid Networks: Many organizations want privacy for inter-organization level data exchange, at same time they want to communicate with others through the global internet.

One solution to achieve this is to implement a hybrid network as shown in Fig. 12. In this case, both private and hybrid networks have high cost of implementation, particularly private WANs are expensive to implement.

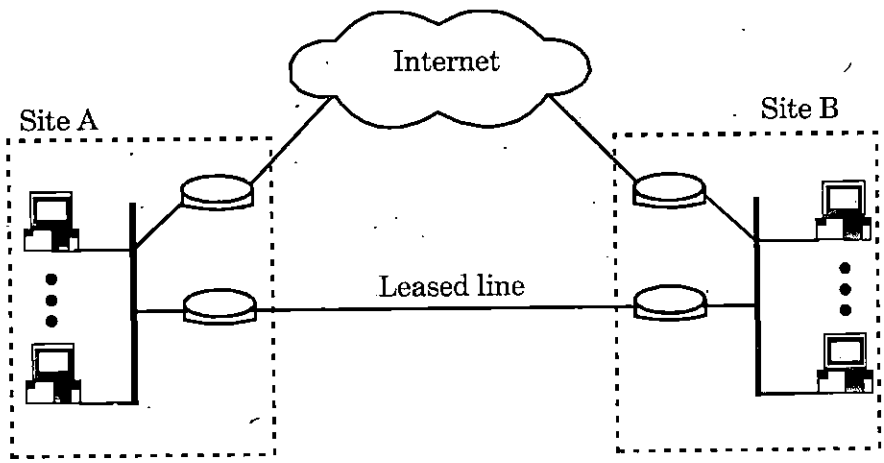


Figure 12 Hybrid network with two LAN sites.

Virtual Private Networks (VPN): VPN technology allows both private communication and public communications through the global internet as shown in Fig.13. VPN uses IPSec in the tunnel mode to provide authentication, integrity and privacy.

In the IPSec tunnel mode the datagram to be sent is encapsulated in another datagram as payload. It requires two sets of addressing as shown in Fig. 14.

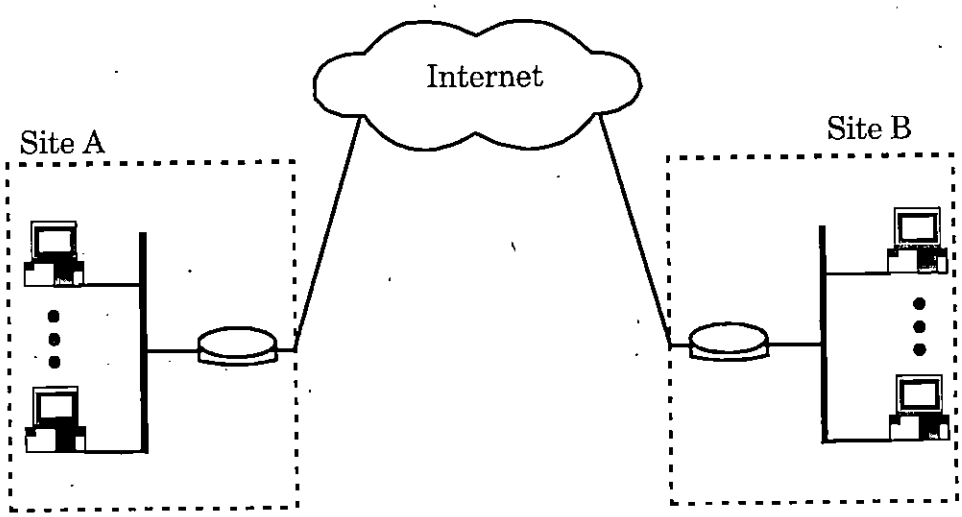


Figure 13 VPN linking two LANs.

Notes

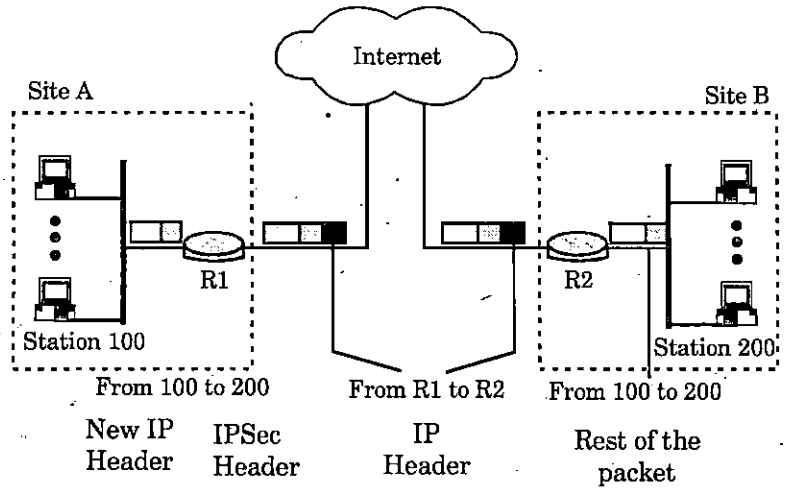


Figure 14 VPN linking two LANs.

4.15 DIGITAL SIGNATURES

A Digital Signature (OS) is an authentication technique based on public key cryptography used in e-commerce applications. It associates a unique mark to an individual within the body of his message. This helps others to authenticate valid senders of messages.

Typically, a user's digital signature varies from message to message in order to provide security against counterfeit ing.The method is as follows:

- The sender takes a message, calcu lates the message digest of the message and signs it digest with a private key.
- The sender then appends the signed digest along with the plaintext message.
- The message is sent over communica tion channel.
- The receiver removes the appended signed digest and verifies the digest using the corresponding public key.
- The receiver then ta kes the pla intext message and ru ns it through the same message digest algorithm.
- If the results of step 4 and step 5 match, then the receiver knows that the message has integrity and authentic.

4.16 ADJUSTMENT OF DATABASE STRUCTURES

Use

To enable ABAP programs to access database tables correctly, the runtime object of a table

must correspond to the structure of the table in the database. If the table is changed in the ABAP Dictionary, you must ensure that the database structure of the table is adjusted to the change in the ABAP Dictionary during activation (when the runtime object is rewritten).

This process is clearly explained in the figure below.

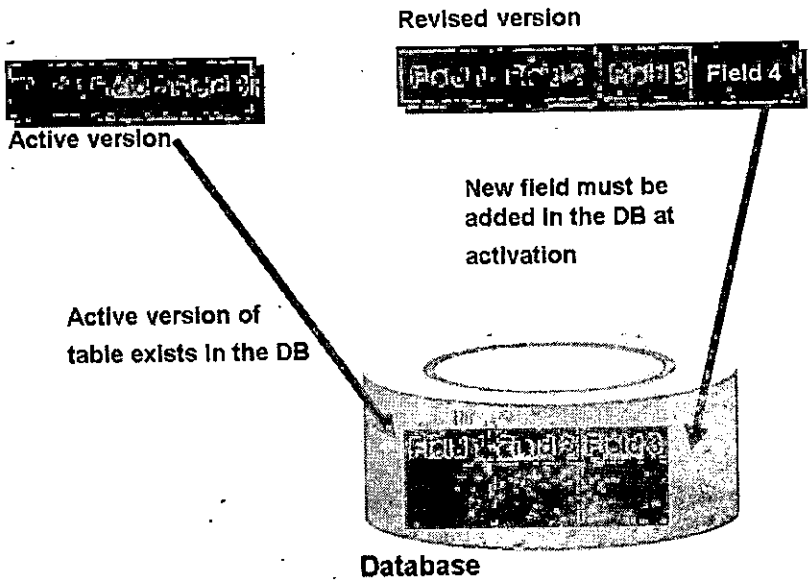


Figure 15

Features

The database structure does not need to be changed for certain changes in the ABAP Dictionary. For example, the database table does not need to be adjusted to a change in the field sequence (except for key fields) in the ABAP Dictionary.

The field sequence in the ABAP Dictionary does not have to correspond to the field sequence in the database. In this case, the changed structure is simply activated in the ABAP Dictionary and the database structure remains unchanged, but the runtime object must be rewritten.

The database structure of a table can be adjusted to its changed ABAP Dictionary definition in three ways:

- By deleting and re-creating the database table
- The table is deleted in the database. The revised version of the table is then activated in the ABAP Dictionary and created again in the database.
- You lose the data in the table during this process.

By changing the database catalog (ALTER TABLE)

Only the definition of the table is changed in the database. Sometimes when you use ALTER TABLE, there are additional actions possible. For example, if you change a column from null to not null, you get time-intensive actions like:

Notes

```
UPDATE""
SET "F01"='...'
WHERE 'F01' IS NULL
```

When you use ALTER TABLE, the data in the table is retained (for some databases, when you add a not null column, the system writes the default value into each line).

Caution

The indexes on the table have to be rebuilt.

By converting the table (for more information, see Conversion Process)

The system renames the database table and it serves as a temporary buffer for the data. The revised version of the table is activated in the ABAP Dictionary and created in the database. The data is reloaded from the temporary buffer to the new database table (with MOVE-CORRESPONDING) and the indexes on the table are built.

The procedure actually used by the system in a particular case depends on:

- The type of structural change
- The database system used
- Whether data already exists in the table

If the table does not contain any data, the existing table is deleted in the database and recreated. If there is data in the table, the system tries to change the structure using ALTER TABLE. If the database system used cannot execute the structural change with ALTER TABLE, the table is converted.

Conversion is the most resource-intensive method of adjusting structures. However, structural changes involving changes to the database catalog can also result in costly internal data reorganizations in some database systems. For information about the processes occurring in the database for structural changes with ALTER TABLE, see your database system documentation.

Caution

You must not adjust the database structure during production. At least all the applications that access the table must be deactivated during the structural adjustment. Since the table data is not consistent during the structural adjustment (in particular during conversion), programs may behave incorrectly when they access this data.

Common threats and challenges

Many software misconfigurations, vulnerabilities, or patterns of carelessness or misuse can result in breaches. The following are among the most common types or causes of database security attacks and their causes.

Insider threats

An insider threat is a security threat from any one of three sources with privileged access to the database:

- A malicious insider who intends to do harm
- A negligent insider who makes errors that make the database vulnerable to attack
- An infiltrator—an outsider who somehow obtains credentials via a scheme such as phishing or by gaining access to the credential database itself
- Insider threats are among the most common causes of database security breaches and are often the result of allowing too many employees to hold privileged user access credentials.

Human error

Accidents, weak passwords, password sharing, and other unwise or uninformed user behaviors continue to be the cause of nearly half (49%) of all reported data breaches.(Link resides outside IBM)

Exploitation of database software vulnerabilities

Hackers make their living by finding and targeting vulnerabilities in all kinds of software, including database management software. All major commercial database software vendors and open source database management platforms issue regular security patches to address these vulnerabilities, but failure to apply these patches in a timely fashion can increase your exposure.

SQL/NoSQL injection attacks

A database-specific threat, these involve the insertion of arbitrary SQL or non-SQL attack strings into database queries served by web applications or HTTP headers. Organizations that don't follow secure web application coding practices and perform regular vulnerability testing are open to these attacks.

Buffer overflow exploitations

Buffer overflow occurs when a process attempts to write more data to a fixed-length block of memory than it is allowed to hold. Attackers may use the excess data, stored in adjacent memory addresses, as a foundation from which to launch attacks.

Denial of service (DoS/DDoS) attacks

In a denial of service (DoS) attack, the attacker deluges the target server—in this case the database server—with so many requests that the server can no longer fulfill legitimate requests from actual users, and, in many cases, the server becomes unstable or crashes.

Notes

In a distributed denial of service attack (DDoS), the deluge comes from multiple servers, making it more difficult to stop the attack. See our video "What is a DDoS Attack"(3:51) for more information:

Malware

Malware is software written specifically to exploit vulnerabilities or otherwise cause damage to the database. Malware may arrive via any endpoint device connecting to the database's network.

Attacks on backups

Organizations that fail to protect backup data with the same stringent controls used to protect the database itself can be vulnerable to attacks on backups.

These threats are exacerbated by the following:

- Growing data volumes: Data capture, storage, and processing continues to grow exponentially across nearly all organizations. Any data security tools or practices need to be highly scalable to meet near and distant future needs.
- Infrastructure sprawl: Network environments are becoming increasingly complex, particularly as businesses move workloads to multicloud or hybrid cloud architectures, making the choice, deployment, and management of security solutions ever more challenging.
- Increasingly stringent regulatory requirements: The worldwide regulatory compliance landscape continues to grow in complexity, making adhering to all mandates more difficult.
- Cybersecurity skills shortage: Experts predict there may be as many as 8 million unfilled cybersecurity positions by 2022.

Best practices

Because databases are nearly always network-accessible, any security threat to any component within or portion of the network infrastructure is also a threat to the database, and any attack impacting a user's device or workstation can threaten the database. Thus, database security must extend far beyond the confines of the database alone.

When evaluating database security in your environment to decide on your team's top priorities, consider each of the following areas:

- *Physical security:* Whether your database server is on-premise or in a cloud data center, it must be located within a secure, climate-controlled environment. (If your database server is in a cloud data center, your cloud provider will take care of this for you.)
- *Administrative and network access controls:* The practical minimum number of users should have access to the database, and their permissions should be restricted to the minimum levels necessary for them to do their jobs. Likewise, network access should be limited to the minimum level of permissions necessary.
- *End user account/device security:* Always be aware of who is accessing the database

and when and how the data is being used. Data monitoring solutions can alert you if data activities are unusual or appear risky. All user devices connecting to the network housing the database should be physically secure (in the hands of the right user only) and subject to security controls at all times.

- *Encryption:* ALL data—including data in the database, and credential data—should be protected with best-in-class encryption while at rest and in transit. All encryption keys should be handled in accordance with best-practice guidelines.
- *Database software security:* Always use the latest version of your database management software, and apply all patches as soon as they are issued.
- *Application/web server security:* Any application or web server that interacts with the database can be a channel for attack and should be subject to ongoing security testing and best practice management.
- *Backup security:* All backups, copies, or images of the database must be subject to the same (or equally stringent) security controls as the database itself.
- *Auditing:* Record all logins to the database server and operating system, and log all operations performed on sensitive data as well. Database security standard audits should be performed regularly.

Controls and Policies

In addition to implementing layered security controls across your entire network environment, database security requires you to establish the correct controls and policies for access to the database itself, These include:

- Administrative controls to govern installation, change, and configuration management for the database.
- Preventative controls to govern access, encryption, tokenization, and masking.
- Detective controls to monitor database activity monitoring and data loss prevention tools. These solutions make it possible to identify and alert on anomalous or suspicious activities.
- Database security policies should be integrated with and support your overall business goals, such as protection of critical intellectual property and your cybersecurity policies and cloud security policies. Ensure you have designated responsibility for maintaining and auditing security controls within your organization and that your policies complement those of your cloud provider in shared responsibility agreements. Security controls, security awareness training and education programs, and penetration testing and vulnerability assessment strategies should all be established in support of your formal security policies.

Data Protection Tools and Platforms

Today, a wide array of vendors offer data protection tools and platforms. A full-scale solution should include all of the following capabilities:

- **Discovery:** Look for a tool that can scan for and classify vulnerabilities across all your databases—whether they're hosted in the cloud or on-premise—and offer

Notes

recommendations for remediating any vulnerabilities identified. Discovery capabilities are often required to conform to regulatory compliance mandates.

- **Data activity monitoring:** The solution should be able to monitor and audit all data activities across all databases, regardless of whether your deployment is on-premise, in the cloud, or in a container. It should alert you to suspicious activities in real-time so that you can respond to threats more quickly. You'll also want a solution that can enforce rules, policies, and separation of duties and that offers visibility into the status of your data through a comprehensive and unified user interface. Make sure that any solution you choose can generate the reports you'll need to meet compliance requirements.
- **Encryption and tokenization capabilities:** In case of a breach, encryption offers a final line of defense against compromise. Any tool you choose should include flexible encryption capabilities that can safeguard data in on-premise, cloud, hybrid, or multicloud environments. Look for a tool with file, volume, and application encryption capabilities that conform to your industry's compliance requirements, which may demand tokenization (data masking) or advanced security key management capabilities.
- **Data security optimization and risk analysis:** A tool that can generate contextual insights by combining data security information with advanced analytics will enable you to accomplish optimization, risk analysis, and reporting with ease. Choose a solution that can retain and synthesize large quantities of historical and recent data about the status and security of your databases, and look for one that offers data exploration, auditing, and reporting capabilities through a comprehensive but user-friendly self-service dashboard.

Database security and IBM Cloud

IBM-managed cloud databases feature native security capabilities powered by IBM Cloud Security, including built-in identity and access management, visibility, intelligence, and data protection capabilities. With an IBM-managed cloud database, you can rest easy knowing that your database is hosted in an inherently secure environment, and your administrative burden will be much smaller.

IBM also offers the IBM Security Guardium smarter data protection platform, which incorporates data discovery, monitoring, encryption and tokenization, and security optimization and risk analysis capabilities for all your databases, data warehouses, file shares, and big data platforms, whether they're hosted on-premise, in the cloud, or in hybrid environments.

In addition, IBM offers managed Data Security Services for Cloud, which includes data discovery and classification, data activity monitoring, and encryption and key management capabilities to protect your data against internal and external threats through a streamlined risk mitigation approach.

4.17 REMOTE DATA ACCESS

Remote database access (RDA) is a protocol standard for database access. Despite early efforts to develop proof of concept implementations of RDA for major commercial RDBMSs

(including Oracle, Rdb, NonStop SQL and Teradata), this standard has been largely ignored by commercial database vendors.

RDA is a communications protocol for remote database access that has been adopted as an International Standard by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC). It has also been adopted as an American National Standard by ANSI and as a Federal Information Processing Standard (FIPS) for the U.S. federal government.

RDA was published as a combined ANSI/ISO/IEC standard in 1993. It consists of two parts:

Part 1 – Generic RDA ANSI/ISO/IEC 9579-1:1993

Part 2 – SQL Specialization ANSI/ISO/IEC 9579-2:1993

Remote Database Access provides standard protocols for establishing a remote connection between a database client and a database server. The client is acting on behalf of an application program while the server is interfacing to a process that controls data transfers to and from a database. The goal is to promote the interconnection of database applications among heterogeneous environments.

Description of RDA

RDA describes the connection of a database client to a database server. It includes features for:

- Communicating database operations and parameters from the client to the server,
- In return, transporting result data from the server to the client,
- Database transaction management
- Exchange of information

The RDA standard provides an RDA Service Interface to an RDA Communication Element that exists both at the client site and at the server site. The RDA Communication Element converts RDA service requests into underlying ACSE and TP service requests as part of an open systems interconnection.

The RDA Service Interface consists of service elements for association control, for transfer of database operations and parameters from client to server, for transfer of resulting data from server to client, and for transaction management.

Association control includes establishing an association between the client and server remote sites and managing connections to specific databases at the server site. Database operations are sent as character strings conforming to the SQL language. Resulting data and/or errors and exceptions are described and represented using the ISO ASN.1 standard. Transaction management includes capabilities for both one-phase and two-phase commit protocols.

The RDA standard describes RDA services in terms of an RDA Client, and RDA Server, and an RDA Service as follows:

> An RDA client is an application-process, within an open system, that requests database services from another application-process called a database server. A

Notes

database server is an application-process, within the same or another open system, that supplies database storage facilities and provides, through OSI communication, database services to RDA clients. An RDA client and a database server communicate by means of the RDA Service, supported by an RDA service- provider. The part of the database server that uses the RDA service-provider to communicate with an RDA client is called an RDA server.

The RDA client has the ability to initiate RDA service requests, while the RDA server can only issue RDA service responses to reply to such requests.

A data resource is a named collection of data and/or capabilities on the database server known to both the RDA client and the RDA server. The meaning of the data content and capabilities of a data resource depend upon the application of RDA, which is determined by each RDA specialization standard (e.g. the SQL specialization). The RDA client opens a data resource in order to gain access to the data content or capabilities of that resource through Database Language services (e.g. SQL).

Data resources may be nested, with subordinate data resources grouped within their parent data resource. The RDA client is required to open a parent data resource before it can open subordinate data resources.

An RDA transaction is a logically complete unit of processing as determined by the RDA client. Execution during an RDA transaction of a sequence of database access services that change data resources enables the set of changes to be handled as an atomic unit. When the RDA transaction is terminated, either the whole set of changes is applied to the data resources or no changes are applied. The RDA client requests termination of an RDA transaction by requesting the RDA server either to commit or to roll back the complete set of changes of that transaction. Changes made to the data content of data resources during an RDA transaction are not made available to other RDA clients until that RDA transaction is terminated at the RDA server. RDA provides a choice of two application- contexts for managing RDA transactions:

1) a basic application-context for one-phase commitment, and
2) a TP application-context for two-phase commitment.

The RDA protocol for the basic application-context is completely specified in the RDA standard, whereas the protocol for the TP application context is dependent upon the ISO/IEC Distributed Transaction Processing standard (ISO/IEC 10026).

An RDA operation models a request by an RDA client that is transferred to an RDA server for processing. RDA operations enable an RDA client to request any of five types of RDA services:

a) RDA Dialogue Management services, to start and end RDA dialogues;
b) RDA Transaction Management services, to start and end RDA transactions;

c) RDA Control services, to report the status or cancel existing operations;

d) Resource Handing services, to enable or disable access by RDA clients to data resources;

e) Database Language services, to access and modify data resources.

An RDA client may request RDA operations without waiting for the results of previously requested RDA operations. Thus an RDA server may have several RDA operations outstanding for a particular RDA dialogue.

An RDA dialogue is a cooperative relationship between and RDA client and an RDA server. The RDA client initializes the RDA dialogue and requests RDA operations that are to be performed by the RDA server. An RDA dialogue is uniquely identified within the scope of the OSI environment, and all RDA operations occur within the bounds of an RDA dialogue. An RDA dialogue can exist only in the context of an established application-association, and ceases to exist if the association is released.

A failed RDA dialogue cannot be recovered; the process of recovery after a failure is a local matter beyond the scope of the RDA 1993 standard, and recovery actions outside the RDA service-provider may be necessary.

In the event of dialogue failure, it is a requirement that all changes made to data resources by any RDA transaction that is not already terminating when RDA dialogue failure occurs be rolled back by the database server during its recovery process. If an RDA dialogue is terminating when RDA dialogue failure occurs, then it may either be committed or rolled back.

The OSE Implementor's Workshop (OIW) has specified implementation agreements for the Basic Application Context of the RDA standard, with profiles for: Immediate Execution, Stored Execution, Status, and Cancel. Future work is in progress by the OIW to specify corresponding profiles for the Transaction Processing (TP) Application Context. Only the Immediate Execution profile is required for all RDA conforming implementations; the other profiles of the Basic Application Context and the TP Application Context are optional enhancements to this basic requirement as follows:

RDA Stored Execution. Support for the Immediate Execution profile and, in addition, support for the RDA Stored Execution Functional Unit as specified in the RDA'93 standard.

- RDA Status. Support for the Immediate Execution profile and, in addition, support for the RDA Status Functional Unit as specified in the RDA'93 standard.
- RDA Cancel. Support for the Immediate Execution profile and, in addition, support for the RDA Cancel Functional Unit as specified in the RDA'93 standard.
- RDA TP Application Context. Support for the Immediate Execution profile and, in addition, support for the RDA SQL TP Application Context as specified in the RDA'93 standard, and dependent upon ISO/IEC 10026 (Distributed Transaction Processing).

Notes

General Use of RDA

RDA is appropriate for remote access to a database in any context where lower layer transport protocols have already been established. RDA protocols have been shown to work properly in both OSI and Internet communications environments. The Internet RFC 1006 is the guide used for executing RDA over a TCP/IP connection.

It is expected that RDA will be used for interconnection among SQL database management products from different vendors. Interconnection among database products from the same vendor will likely continue to use vendor specific communication and interchange forms.

RDA Enhancements

The existing RDA/SQL Specialization standard does not yet support all of the facilities in the SQL-92 (ISO 9075:1992) language standard. Instead, it more closely approximates support for just the Entry SQL level of SQL-92. Work is in progress on an Amendment 1 to the RDA/SQL Specialization to support all SQL-92 features. Expected progression dates for Amendment 1 are CD registration in 1995 and formal adoption as an International Standard by 1997.

SQL-92 data types not yet fully supported by RDA are:

DATE, TIME, TIMESTAMP, INTERVAL, CHARACTER VARYING, NATIONAL CHARACTER, BIT, and BIT VARYING.

Also, very long character and bit strings, e.g. blobs, are not yet fully supported. SQL-92 facilities for special Descriptor and Diagnostic areas are not supported implicitly in RDA.

Amendment 1 to the RDA/SQL Specialization will provide data type encodings for the above types and may also include facilities for Descriptor and Diagnostics information to flow on each SQL statement. In most other cases, RDA simply carries an SQL statement from Client to Server, so additional RDA facilities are not needed.

SUMMARY

- Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.
- Because information is so important in most organizations, computer scientists have developed a large body of concepts and techniques for managing data.
- A Database management system is a computerized record-keeping system. It is a repository or a container for collection of computerized data files. The overall purpose of DBMS is to allow he users to define, store, retrieve and update the information contained in the database on demand. Information can be anything that is of

significance to an individual or organization.

- The database utility is the interface between the ABAP Dictionary and the relational database underlying the R/3 System. You can use the database utility to edit all the database objects that are generated from objects of the ABAP Dictionary. These are database tables that are generated from transparent tables or phys ical table pools or table clusters, indexes, database views.
- It is mainly used when a table is changed in the ABAP Dictionary. At that time, we must ensure that the database structure of the table is adjusted to the change in the ABAP Dictionary duringactivation.
- Database security refers to the range of tools, controls, and measures designed to establish and preserve database confidentiality, integrity, and availability. This article will focus primarily on confidentiality since it's the element that's compromised in most data breaches.
- A database object is any defined object in a database that is used to store or reference data.Anything which we make from create command is known as Database Object. It can be used to hold and manipulate the data.Some of the examples of database objects are : view, sequence, indexes, etc.
- In conventional cryptography, the encryption and decryption is done using the same secret key. Here, the sender encrypts the message with an encryption algorithm using a copy of the secret key. The encrypted message is then send over public communication channels. On receiving the encrypted message, the receiver decrypts it with a corriesponding decryption algorithm using the same secret key.
- A Digital Signature (OS) is an authentication technique based on public key cryptography used in e-commerce applications. It associates a unique mark to an individual within the body of his message. This helps others to authenticate valid senders of messages.
- Remote database access (RDA) is a protocol standard for database access. Despite early efforts to develop proof of concept implementations of RDA for major commercial RDBMSs (including Oracle, Rdb, NonStop SQL and Teradata), this standard has been largely ignored by commercial database vendors.
- The RDA Service Interface consists of service elements for association control, for transfer of database operations and parameters from client to server, for transfer of resulting data from server to client, and for transaction management.
- Association control includes establishing an association between the client and server remote sites and managing connections to specific databases at the server site. Database operations are sent as character strings conforming to the SQL language. Resulting data and/or errors and exceptions are described and represented using the ISO ASN.1 standard. Transaction management includes capabilities for both one-phase and two-phase commit protocols.

Notes

KEY WORDS

- **Relational Database:** A relational database is one which employs the relational model, in which the raw data is organized into sets of tuples, and the tuples organized into relations. This relational model imposes structure on its contents, in contrast to unstructured or semi-structured data of the various NoSQL architectures.
- **Database Management System (DBMS):** A database management system is a software system which facilitates the organization of housed data into a particular database architecture, be it relational (Relational Database Management System, or RDBMS), document store, key-value store, column-oriented, graph, or other. Popular DBMSs include MongoDB, Cassandra, Redis, MySQL, Microsoft SQL Server, SQLite, and Oracle, among many, many, many others.
- **Primary Key:** In the relational model, a primary key is a single attribute, or combination of attributes, which can be used to uniquely identify a row of data in a givn table. Common primary keys include vendor ID, user ID, email address, or combination of attributes considered together such as first name, last name, and city of residence, all considered together as a single entity. It should be noted that what is an acceptable primary key in one situation may not uniquely identify data instances in another.
- **Foreign Key:** Again in the relational model, a foreign key is an attribute or collection of attributes from one relational table whose values must match another relational table's primary key. A common use for such an organizational scheme would be to link a street address in one table to a city in another, and perhaps to a country in a third. This eliminates repetitive data input, and reduces the possibility of error, increasing data accuracy.
- **Scalability:** A software system is scalable when its performance and overall system throughput continues to improve as more computing resources are made available for its use. This usually comes in the form of the number of CPUs and cores available in the computer on which the software system is run.
- **Scalar Function:** Either a built-in SQL function or a user-defined function that returns a single value computed only from the values of any required arguments at the time the function is called.
- **Schema:** A representation of the structure of a database. It may be graphical or textual. Graphical representations typically involve the use of boxes that represent database tables and arrows that represent inter-table relationships. Textual schema representations utilize Database Definition Language (DDL) statements to describe a database design.
- **Searched Update/Delete :** An SQL update or delete statement in which the rows to be updated/deleted are those for which the conditional expression specified in the WHERE clause is true.

Notes

- **SQL:** The standardized and commonly accepted language used for defining, querying and manipulating a relational database. The etymology of "SQL" is unclear, possibly a progression from "QueL" (Query Language) to "SeQueL" to "SQL." However, some experts don't like the expansion "Structured Query Language" because its structure is inconsistent and a historical patchwork. See Wikipedia
- **SQL PL:** A SQL based programming language. This allows for a SQL programmer to use programming constructs like variables, conditionals and loops purely through the use of SQL statements.
- **Transaction Log:** A sequential record of all of the database changes made by each transaction in the order they were issued. The transaction log is used to ensure that a database conforms to the ACID properties. Transaction logs are also used to mirror or replicate data to other databases.
- **Transaction:** A set of logically related database modifications that is written to the database as a unit. The database changes associated with a given transaction are guaranteed by the DBMS to be written completely to the database; in the event of a system failure, none are written. The state of the database both before and after a transaction will be consistent with its design.
- Hierarchical Model: A special case of a network model database in which each record type can participate only as the member of one set.
- **Hot Spot:** In a database, a hot spot is a single shared row of a table that is used and updated so often that it creates a performance bottleneck on the system.
- **I/O:** Input/output. For a DBMS, this is normally a disk drive, used to create database durability.
- IEC: International Electrotechnical Commission. Along with the ISO, the IEC controls the SQL standard (ISO/IEC 9075) and many others as well.
- **IIOT:** Abbreviation of Industrial Internet of Things.
- Implicit Locking: Done by SQL to automatically apply the locks needed to safely execute an SQL statement in a multiuser (i.e., shared database) operational environment.
- **IMDB:** Abbreviation of In-Memory Database

REVIEW QUESTIONS

1. Define database utility.
2. What is do you mean by security? Explain security services.
3. How will you edit database objects? Explain.
4. Describe the conventional encryption method
5. What is RDA? Explain the general uses of RDA.
6. Briefly explain the description of RDA.

Notes

FURTHER READINGS

1. "database, n". OED Online. Oxford University Press. June 2013. Retrieved July 12, 2013. (Subscription required.)
2. IBM Corporation (October 2013). "IBM Information Management System (IMS) 13 Transaction and Database Servers delivers high performance and low total cost of ownership". Retrieved Feb 20, 2014.
3. "Structured Query Language (SQL)". International Business Machines. October 27, 2006. Retrieved 2007-06-10. Wagner 2010.
4. Ramalho, J.C., Faria, L., Helder, S., & Coutada, M. (2013, December 31). Database Preservation Toolkit: A flexible tool to normalize and give access to databases. University of Minho. https://core.ac.uk/display/55635702?
5. Kroenke, David M. and David J. Auer. Database Concepts. 3rd ed. New York: Prentice, 2007.
6. Raghu Ramakrishnan and Johannes Gehrke, Database Management Systems
7. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, Database System Concepts

SWAMI VIVEKANAND
SUBHARTI UNIVERSITY
UGC Approved Meerut

Where Education is a Passion ...

ACCREDITED WITH GRADE A NAAC