CS352 – ADVANCED DATABASE SYSTEMS

UNIT 4 - SQL

Dr. John Conklin

Agenda

- SQL
  - Commands
  - Operators
  - Conditions
  - Syntax
- Joins
  - Inner
  - Outer
  - Left
  - Right
- Aliasing
- Grouping

SQL

What is SQL?

SQL (Structured Query Language): A standardized programming language used to manage and manipulate relational databases.

Key Components

1. **Data Definition Language (DDL)**:
    1. *CREATE*: Define new database objects (tables, indexes).
    2. *ALTER*: Modify existing database structures.
    3. *DROP*: Delete database objects.
2. **Data Manipulation Language (DML)**:
    1. *SELECT*: Retrieve data from the database.
    2. *INSERT*: Add new data into a table.
    3. *UPDATE*: Modify existing data.
    4. *DELETE*: Remove data from a table.

What is SQL?

3. **Data Control Language (DCL)**:
    1. *GRANT: Provide user access privileges.*
    2. *REVOKE: Remove user access privileges.*
4. **Transaction Control Language (TCL)**:
    1. *COMMIT: Save changes permanently.*
    2. *ROLLBACK: Undo changes.*

Example Query

```
SELECT * FROM Employees WHERE Department = 'Sales';
```

Why Learn SQL?

- **Universal Language**: Widely used across various database systems (MySQL, PostgreSQL, SQL Server, etc.).
- **Data Analysis**: Essential for querying and analyzing data.
- **Versatility**: Used in data science, web development, business intelligence, and more.

COMMANDS

Overview of SQL Commands

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language (DCL)
- Transaction Control Language (TCL)

Data Definition Language (DDL)

- CREATE: Create a new table or database

```
CREATE TABLE Employees (
    ID int,
    Name varchar(255),
    Position varchar(255)
);
```

- ALTER: Modify an existing database object

```
ALTER TABLE Employees
ADD COLUMN Salary int;
```

DROP: Delete an existing database object

```
DROP TABLE Employees;
```

Data Manipulation Language (DML)

- SELECT: Retrieve data from the database

```
SELECT * FROM Employees;
```

- INSERT: Add new data into a table

```
INSERT INTO Employees (ID, Name, Position)
VALUES (1, 'John Doe', 'Manager');
```

- UPDATE: Modify existing data

```
UPDATE Employees
SET Salary = 50000
WHERE ID = 1;
```

- DELETE: Remove data from a table

```
DELETE FROM Employees
WHERE ID = 1;
```

Data Control Language (DCL)

- GRANT: Provide user access privileges

```
GRANT SELECT ON Employees TO User1;
```

- REVOKE: Remove user access privileges

```
REVOKE SELECT ON Employees FROM User1;
```

Transaction Control Language (TCL)

- COMMIT: Save changes permanently

COMMIT;

- ROLLBACK: Undo changes

ROLLBACK;

- SAVEPOINT: Set a save point within a transaction

SAVEPOINT Savepoint1;

Example Scenario

- Combine multiple SQL commands in a real-world scenario

```
BEGIN TRANSACTION;

INSERT INTO Employees (ID, Name, Position) VALUES (2, 'Jane Smith', 'Developer');
UPDATE Employees SET Salary = 60000 WHERE ID = 2;

COMMIT;
```

OPERATORS

Overview of SQL Operators

- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Other Operators

Arithmetic Operators

- **Addition (+)**

```
SELECT Salary + 5000 FROM Employees;
```

- **Subtraction (-)**

```
SELECT Salary - 5000 FROM Employees;
```

- **Multiplication (*)**

```
SELECT Salary * 1.10 FROM Employees;
```

- **Division (/)**

```
SELECT Salary / 2 FROM Employees;
```

- **Modulus (%)**

```
SELECT Salary % 1000 FROM Employees;
```

Comparison Operators

- Equal to (=)

```
SELECT * FROM Employees WHERE Salary = 50000;
```

- Not equal to (<> or !=)

```
SELECT * FROM Employees WHERE Salary <> 50000;
```

- Greater than (>)

```
SELECT * FROM Employees WHERE Salary > 50000;
```

- Less than (<)

```
SELECT * FROM Employees WHERE Salary < 50000;
```

- Greater than or equal to (>=)

```
SELECT * FROM Employees WHERE Salary >= 50000;
```

- Less than or equal to (<=)

```
SELECT * FROM Employees WHERE Salary <= 50000;
```

Logical Operators

- AND

```
SELECT * FROM Employees WHERE Salary > 50000 AND Department = 'IT';
```

- OR

```
SELECT * FROM Employees WHERE Department = 'IT' OR Department = 'HR';
```

- **NOT**

```
SELECT * FROM Employees WHERE NOT Department = 'HR';
```

Other Operators

- BETWEEN

```
SELECT * FROM Employees WHERE Salary BETWEEN 40000 AND 60000;
```

- IN

```
SELECT * FROM Employees WHERE Department IN ('IT', 'HR');
```

- LIKE

```
SELECT * FROM Employees WHERE Name LIKE 'J%';
```

- IS NULL

```
SELECT * FROM Employees WHERE Salary IS NULL;
```

Example Scenario

- **Using multiple operators in a query**

```
SELECT * FROM Employees
WHERE (Salary > 50000 AND Department = 'IT') OR Name LIKE 'J%';
```

CONDITIONS

Overview of SQL Conditions

- WHERE Clause
- AND, OR, NOT
- BETWEEN
- IN
- LIKE
- IS NULL

WHERE Clause

- The WHERE clause is used to filter records

```
SELECT * FROM Employees
WHERE Salary > 50000;
```

AND, OR, NOT Operators

- AND: All conditions must be true

```
SELECT * FROM Employees
WHERE Salary > 50000 AND Department = 'IT';
```

- OR: Any condition can be true

```
SELECT * FROM Employees
WHERE Department = 'IT' OR Department = 'HR';
```

- NOT: Negates a condition

```
SELECT * FROM Employees
WHERE NOT Department = 'HR';
```

BETWEEN Operator

- The BETWEEN operator selects values within a given range

```
SELECT * FROM Employees
WHERE Salary BETWEEN 40000 AND 60000;
```

IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause

```
SELECT * FROM Employees
WHERE Department IN ('IT', 'HR');
```

LIKE Operator

- The LIKE operator is used to search for a specified pattern in a column

```
SELECT * FROM Employees
WHERE Name LIKE 'J%';
```

IS NULL Operator

- The IS NULL operator is used to test for empty (NULL) values

```
SELECT * FROM Employees
WHERE Salary IS NULL;
```

Example Scenario

- Combining multiple conditions in a query

```
SELECT * FROM Employees
WHERE (Salary > 50000 AND Department = 'IT') OR Name LIKE 'J%';
```

PREDICATES

What are SQL Predicates?

- Predicates are conditions used in SQL statements
- Used to filter data and control the flow of queries
- Commonly used in WHERE, HAVING, and JOIN clauses

IS NULL Predicate

- **Tests for empty (NULL) values**

```
SELECT * FROM Employees WHERE Salary IS NULL;
```

EXISTS Predicate

- **Checks for the existence of rows in a subquery**

  Example 1: Checking for Existence To find customers who have placed at least one order, you can use the EXISTS keyword as follows:

  ```
  SELECT CustomerName FROM Customers c
  WHERE EXISTS (SELECT 1 FROM Orders o WHERE o.CustomerID = c.CustomerID);
  ```

  This query returns:

  ```
  CustomerName
  ----------------------
  Alice
  Bob
  ```

ANY Predicate

- **Compares a value to any value in a list or subquery**

```
SELECT * FROM Employees
WHERE Salary > ANY (SELECT Salary FROM Employees WHERE Department = 'IT');
```

ALL Predicate

- **Compares a value to all values in a list or subquery**

```
SELECT * FROM Employees
WHERE Salary > ALL (SELECT Salary FROM Employees WHERE Department = 'IT');
```

Example Scenario

- **Using multiple predicates in a query**

```
SELECT * FROM Employees
WHERE (Salary BETWEEN 50000 AND 80000)
AND Department IN ('IT', 'HR')
AND Name LIKE 'J%'
AND Salary IS NOT NULL;
```

SYNTAX

Basic SQL Commands

Categories of SQL Commands:

- - DDL (Data Definition Language)
- - DML (Data Manipulation Language)
- - DCL (Data Control Language)
- - TCL (Transaction Control Language)

Data Definition Language (DDL)

Key Commands:

- CREATE
- ALTER
- DROP

Examples:

```
CREATE TABLE Students (
  StudentID int,
  FirstName varchar(255),
  LastName varchar(255)
);
```

Data Manipulation Language (DML)

Key Commands:

- SELECT
- INSERT
- UPDATE
- DELETE

Examples:

```
SELECT * FROM Students;
INSERT INTO Students (StudentID, FirstName, LastName) VALUES (1, 'John', 'Doe');
```

Data Control Language (DCL)

Key Commands:

- GRANT
- REVOKE

Examples:

```
GRANT SELECT ON Students TO user_name;
```

Transaction Control Language (TCL)

Key Commands:

- COMMIT
- ROLLBACK
- SAVEPOINT

Examples:

```
COMMIT;
ROLLBACK;
```

The SELECT Statement

Basic Syntax:

SELECT column1, column2 FROM table_name;

Filtering Data:

Using WHERE clause

Example:

SELECT * FROM Students WHERE LastName = 'Doe';

Aggregation Functions

Common Functions:

- COUNT()
- SUM()
- AVG()
- MIN()
- MAX()

Examples:

```
SELECT COUNT(*) FROM Students;
SELECT AVG(Grade) FROM Enrollments;
```

Group By and Having Clauses

Group By: Used to group rows that have the same values

Example:

```
SELECT COUNT(StudentID), CourseID FROM Enrollments
GROUP BY CourseID;
```

Having: Used to filter groups

Example:

```
SELECT COUNT(StudentID), CourseID FROM Enrollments
GROUP BY CourseID
HAVING COUNT(StudentID) > 5;
```

Subqueries

What is a Subquery?

A query within another query

Examples:

```
SELECT * FROM Students
WHERE StudentID IN
    (SELECT StudentID FROM Enrollments WHERE CourseID = 101);
```

JOINS

Introduction to SQL Joins

What are SQL Joins?

- Used to combine rows from two or more tables
- Based on a related column between them

Types of Joins

Common Types of Joins:

- INNER JOIN
- LEFT JOIN (or LEFT OUTER JOIN)
- RIGHT JOIN (or RIGHT OUTER JOIN)
- FULL OUTER JOIN

INNER JOIN

Returns records that have matching values in both tables

Example:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

INNER JOIN Example

Example:

```
SELECT Students.FirstName, Courses.CourseName
FROM Students
INNER JOIN Enrollments ON Students.StudentID = Enrollments.StudentID
INNER JOIN Courses ON Enrollments.CourseID = Courses.CourseID;
```

LEFT JOIN

Returns all records from the left table, and the matched records from the right table

Example:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

LEFT JOIN Example

Example:

```
SELECT Students.FirstName, Courses.CourseName
FROM Students
LEFT JOIN Enrollments ON Students.StudentID = Enrollments.StudentID
LEFT JOIN Courses ON Enrollments.CourseID = Courses.CourseID;
```

RIGHT JOIN

Returns all records from the right table, and the matched records from the left table

Example:

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

RIGHT JOIN Example

Example:

```
SELECT Students.FirstName, Courses.CourseName
FROM Students
RIGHT JOIN Enrollments ON Students.StudentID = Enrollments.StudentID
RIGHT JOIN Courses ON Enrollments.CourseID = Courses.CourseID;
```

FULL OUTER JOIN

Returns all records when there is a match in either left or right table

Example:

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name;
```

FULL OUTER JOIN Example

Example:

```
SELECT Students.FirstName, Courses.CourseName
FROM Students
FULL OUTER JOIN Enrollments ON Students.StudentID = Enrollments.StudentID
FULL OUTER JOIN Courses ON Enrollments.CourseID = Courses.CourseID;
```

ALIASING

What is Aliasing?

- Aliasing refers to using temporary names to refer to database tables or columns in SQL queries.
- Aliases provide a way to simplify complex queries and improve readability.

Why Use Aliasing?

- **Simplifies Queries**: Makes complex SQL queries easier to read and understand.
- **Avoids Ambiguity**: Helps distinguish between columns with the same name in different tables.
- **Enhances Clarity**: Provides meaningful names that reflect the purpose of the data.

Table Aliases

Syntax: SELECT * FROM table_name AS alias_name;

Example:

```
SELECT e.name, d.department_name
FROM employees AS e
JOIN departments AS d
ON e.department_id = d.id;
```

Explanation: The alias **e** is used for the employees' table and **d** for the departments' table.

Column Aliases

Syntax: SELECT column_name AS alias_name FROM table_name;

Example:

SELECT first_name AS fname, last_name AS lname

FROM employees;

Explanation: The alias **fname** is used for first_name and **lname** for last_name.

Practical Example

Combining Table and Column Aliases:

```
SELECT e.first_name AS fname, e.last_name AS lname, d.department_name AS dept
FROM employees AS e
JOIN departments AS d
ON e.department_id = d.id;
```

Explanation: Using both table and column aliases for clarity and simplicity.

Benefits of Aliasing

- **Improves Readability**: Easier to follow and understand SQL queries.
- **Reduces Typing**: Shorter aliases save time and effort.
- **Prevents Errors**: Minimizes confusion with long table or column names.

Common Use Cases

- **Joining Multiple Tables**: Helps manage queries with multiple joins.
- **Subqueries**: Simplifies nested queries and improves readability.
- **Temporary Calculations**: Used for columns created on the fly (e.g., calculations).

Best Practices

- **Use Meaningful Aliases**: Choose aliases that convey the purpose of the table or column.
- **Consistency**: Stick to a consistent naming convention for aliases.
- **Document Queries**: Comment your code to explain the purpose of aliases.

Conclusion

- Aliasing is a powerful tool in SQL that enhances query readability and maintainability.
- Proper use of aliases can significantly simplify complex database queries.
- Implement best practices to make the most out of aliasing in your SQL code.

GROUPING

What is Grouping in SQL?

- Grouping in SQL is a method used to aggregate data across multiple records.
- It allows for performing aggregate functions like SUM, COUNT, AVG, MAX, and MIN on grouped data.

Why Use Grouping?

- **Simplifies Data Analysis**: Aggregates data to provide summarized insights.
- **Enhances Data Organization**: Groups related data for easier interpretation.
- **Facilitates Reporting**: Provides a basis for creating detailed reports.

Basic Syntax

The GROUP BY clause is used in a SELECT statement to group rows that have the same values in specified columns.

Syntax:

```
SELECT column1, aggregate_function(column2)
FROM table_name
GROUP BY column1;
```

Example Query

Example:

```
SELECT department, COUNT(employee_id) AS employee_count
FROM employees
GROUP BY department;
```

Explanation: This query counts the number of employees in each department.

Using HAVING Clause

The HAVING clause is used to filter groups based on conditions.

Syntax:

```
SELECT column1, aggregate_function(column2)
FROM table_name
GROUP BY column1
HAVING condition;
```

Example:

```
SELECT department, COUNT(employee_id) AS employee_count
FROM employees
GROUP BY department
HAVING COUNT(employee_id) > 5;
```

Explanation: This query filters to only include departments with more than 5 employees.

Aggregate Functions

Common aggregate functions used with GROUP BY:

- **SUM**: Calculates the total sum of a numeric column.
- **COUNT**: Counts the number of rows.
- **AVG**: Calculates the average value of a numeric column.
- **MAX**: Finds the maximum value in a column.
- **MIN**: Finds the minimum value in a column.

Practical Example

Example:

```
SELECT product_category, SUM(sales) AS total_sales
FROM sales_data
GROUP BY product_category;
```

Explanation: This query calculates the total sales for each product category.

Benefits of Grouping

- **Improves Data Insight**: Provides summarized views of large datasets.
- **Enhances Query Performance**: Reduces the amount of data processed by focusing on groups.
- **Simplifies Report Generation**: Aggregated data is easier to present and interpret.

Common Use Cases

- **Sales Reporting**: Summarizing sales by region, product, or time period.
- **Employee Analysis**: Counting employees in each department or location.
- **Financial Summaries**: Aggregating financial data like expenses and revenue.

Conclusion

- Grouping in SQL is a fundamental technique for data aggregation and analysis.
- It enhances the ability to extract meaningful insights from large datasets.
- Mastering the GROUP BY clause and aggregate functions is essential for effective SQL querying.

INDIVIDUAL PROJECT

Individual Project

- **Description:** Using the DBMS you chose in the previous Discussion Board assignment, download and install that software to prepare for the Database and Data Model to be created. Once the software is running and the database is available, complete the following:
- Create the physical data model for the logical data model you submitted in IP3. This should include all of the data definition language SQL.
- Your submission should include all DDL needed to:
  - Create the tables
  - Create the primary keys
  - Create the foreign keys
  - Add DML statements to:
    - Add data of 1 customer who buys from the company
    - Provide the DML to add 1 employee who interacts with customers
    - Give DML to change data of the employee, giving the commission a 25% increase
    - Give DML to delete the customer and employee data
- Write 3 SELECT statements:
  - To select the customer details
  - To select the employees' details
  - To show which employee services which customer
- Add the SQL for the DDL, DML, and SELECT statements to the " Advanced SQL " project template section."
- Name the document CS352_<First and Last Name>_IP4.doc.

Individual Project

- Submit your Word document and make sure that it contains the following:
- A screenshot of the ERD logical data model from previous assignments
- The DDL to create the tables, including the table definition and the primary and foreign key definitions
- The SQL to add data to the tables
  - Add data of 1 customer who buys from the company
  - Provide the DML to add 1 employee who interacts with customers
  - Give DML to change data of the employee, giving the commission a 25% increase
  - Give DML to delete the customer and employee data
- 3 SELECT statements, as follows:
  - To select the customer details
  - To select the employees' details
  - To show which employee services which customer
- **Please submit your assignment.**
- **For assistance with your assignment, please use your textbook and all course resources.**

Contact Information

Email:	Jconklin@coloradotech.edu
Phone:	602.796.5972
Website:	http://drjconklin.com
Office Hours:	Wednesdays: 6:00 PM – 7:00 PM (CST)
	Saturdays: 11:00 AM – 12:00 PM (CST)
Live Chats:	Thursday/Friday: 7:00 PM – 8:00 PM (CST)