Introduction, A Database Model, Relational Database Model, Integrity, RDBMS

→**Introduction**

A database is an organized collection of data that can be easily accessed, managed, and updated. Databases are crucial in managing information for applications, websites, businesses, and many other systems. They ensure efficient data storage and retrieval while supporting scalability, security, and consistency.

A Database Model

A database model is a conceptual framework used to define the structure of a database, including the relationships, constraints, and operations allowed on the data. It provides a blueprint for designing databases that meet specific requirements. The most common database models include:

- **Hierarchical Model**: Organizes data in a tree-like structure.
- **Network Model**: Uses a graph structure to represent relationships between entities.
- **Relational Model**: Represents data in tables (relations) with rows and columns.
- **Object-Oriented Model**: Combines database capabilities with object-oriented programming concepts.

Relational Database Model

The relational database model, introduced by E.F. Codd in 1970, organizes data into tables (relations), where each table consists of rows (tuples) and columns (attributes). Key features include:

1. **Tables (Relations)**: Data is stored in tabular format.
2. **Keys**:
    - **Primary Key**: Uniquely identifies a row in a table.
    - **Foreign Key**: Creates a link between tables, referencing the primary key of another table.
3. **Relationships**: Relationships are established through keys, enabling data normalization and reducing redundancy.
4. **SQL**: Structured Query Language is used for querying and managing data in relational databases.

Integrity

Integrity in databases refers to maintaining the accuracy, consistency, and reliability of the data throughout its lifecycle. There are two primary types of integrity in relational databases:

1. **Entity Integrity**: Ensures that each table has a primary key, and that key must contain unique, non-null values.
2. **Referential Integrity**: Ensures that foreign keys correctly reference primary keys in related tables, maintaining consistency across relationships.

RDBMS (Relational Database Management System)

An RDBMS is a software system that manages relational databases. It provides tools for creating, updating, and managing data while enforcing the principles of the relational model. Key features include:

- **Data Storage**: Efficiently stores data in tables.
- **Data Querying**: Allows retrieval and manipulation of data using SQL.
- **Concurrency**: Supports multiple users accessing the database simultaneously.
- **Security**: Implements user authentication and authorization to protect data.
- **Backup and Recovery**: Ensures data safety and restoration in case of failures.

Popular RDBMS software includes MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server, and SQLite.

SQL and Embedded SQL

→**SQL (Structured Query Language)**

SQL is a standard programming language specifically designed for managing and manipulating data in relational databases. It allows users to perform various operations, such as retrieving, inserting, updating, and deleting data, as well as defining and managing database structures.

Types of SQL Commands

1. **Data Definition Language (DDL)**:
    - Defines the structure of a database.
    - Examples:
        - CREATE TABLE: Creates a new table.

        - ALTER TABLE: Modifies an existing table.
        - DROP TABLE: Deletes a table.
2. **Data Manipulation Language (DML)**:
    - Manipulates data within database tables.
    - Examples:
        - INSERT: Adds new data to a table.
        - UPDATE: Modifies existing data.
        - DELETE: Removes data from a table.
3. **Data Query Language (DQL)**:
    - Retrieves data from one or more tables.
    - Example:
        - SELECT: Queries data from tables.
4. **Data Control Language (DCL)**:
    - Controls access to the database.
    - Examples:
        - GRANT: Gives permissions to users.
        - REVOKE: Removes user permissions.
5. **Transaction Control Language (TCL)**:
    - Manages database transactions.
    - Examples:
        - COMMIT: Saves changes permanently.
        - ROLLBACK: Undoes changes in a transaction.

Embedded SQL

Embedded SQL integrates SQL commands directly into a host programming language (e.g., C, Java, Python) to allow SQL operations within applications. This approach bridges the gap between the application and the database.

How Embedded SQL Works

1. **Integration**: SQL statements are embedded within the code of the host language using specific syntax or keywords.
2. **Pre-compilation**: The embedded SQL code is preprocessed to translate SQL statements into API calls or equivalent native functions.
3. **Execution**: The host program interacts with the database using the generated API or functions.

Advantages of Embedded SQL

- **Seamless Integration**: Allows applications to directly communicate with the database without separate interfaces.
- **Performance**: Reduces the overhead of context-switching between the application and database.
- **Flexibility**: Enables complex business logic to combine with database operations.

Example of Embedded SQL in C

c

Copy code

```
#include <stdio.h>

#include <sqlca.h>


int main() {

  EXEC SQL CONNECT TO my_database AS my_connection USER 'username' USING 'password';


  EXEC SQL DECLARE emp_cursor CURSOR FOR

    SELECT emp_name, emp_salary FROM employees;


  EXEC SQL OPEN emp_cursor;


  while (1) {

    EXEC SQL FETCH emp_cursor INTO :emp_name, :emp_salary;

    if (sqlca.sqlcode != 0) break;
```

```
        printf("Name: %s, Salary: %.2f\n", emp_name, emp_salary);
    }

    EXEC SQL CLOSE emp_cursor;
    EXEC SQL DISCONNECT my_connection;

    return 0;
}
```

Disadvantages of Embedded SQL

- **Portability**: Embedded SQL is tied to a specific programming language, limiting cross-platform use.
- **Complexity**: Embedding SQL can make code harder to read and maintain compared to using separate database APIs.

Writing Basic SQL SELECT Statements

→**Writing Basic SQL SELECT Statements**

The SELECT statement is one of the most fundamental SQL commands used to retrieve data from one or more tables in a relational database. Here's how you can write and understand basic SELECT statements:

General Syntax

sql

Copy code

```
SELECT column1, column2, ...
FROM table_name;
```

- **SELECT**: Specifies the columns to retrieve.
- **FROM**: Indicates the table(s) from which the data is fetched.

Examples

1. Selecting All Columns

To retrieve all columns from a table, use the * wildcard:

sql

Copy code

```
SELECT *
FROM employees;
```

This fetches all columns for every row in the employees table.

---

2. Selecting Specific Columns

To retrieve specific columns, list their names:

sql

Copy code

```
SELECT first_name, last_name
FROM employees;
```

This returns only the first_name and last_name columns for all rows.

---

3. Using Aliases

Aliases rename columns or tables temporarily for readability:

sql

Copy code

```
SELECT first_name AS "First Name", last_name AS "Last Name"
FROM employees;
```

This displays the columns with the aliases "First Name" and "Last Name."

---

4. Filtering Rows with WHERE Clause

The WHERE clause specifies conditions to filter the results:

sql

Copy code

```
SELECT first_name, last_name
FROM employees
WHERE department = 'Sales';
```

This retrieves employees whose department is "Sales."

---

5. Using Logical Operators

You can combine conditions with logical operators:

- **AND**: All conditions must be true.
- **OR**: At least one condition must be true.
- **NOT**: Negates a condition.

sql

Copy code

```
SELECT first_name, last_name
FROM employees
WHERE department = 'Sales' AND salary > 50000;
```

This retrieves employees in the "Sales" department with a salary greater than 50,000.

---

6. Sorting Results with ORDER BY

The ORDER BY clause sorts the results:

sql

Copy code

```
SELECT first_name, last_name, salary
FROM employees
ORDER BY salary DESC;
```

This sorts employees by their salary in descending order.

---

7. Limiting Rows with LIMIT or FETCH

Retrieve a specific number of rows:

sql

Copy code

```
SELECT first_name, last_name
FROM employees
LIMIT 5;
```

Fetches the first 5 rows from the table.

---

8. Using DISTINCT to Avoid Duplicates

The DISTINCT keyword removes duplicate values:

sql

Copy code

```
SELECT DISTINCT department
FROM employees;
```

This retrieves a list of unique departments.

---

9. Combining Multiple Columns

You can combine columns in the result:

sql

Copy code

```
SELECT first_name || ' ' || last_name AS "Full Name"
FROM employees;
```

This concatenates first_name and last_name into a single column called "Full Name."

Practice Exercise

1. Retrieve all columns from a table named students.
2. Retrieve the student_name and grade columns from the students table.
3. Fetch students who scored more than 85 in their exams.
4. Get the list of unique subjects from the subjects table.

Restricting and Sorting data

→**Restricting and Sorting Data in SQL**

When working with SQL queries, you often need to filter rows and sort the result set to meet specific requirements. The WHERE clause helps restrict data, while the ORDER BY clause allows sorting.

Restricting Data: The WHERE Clause

The WHERE clause filters rows based on specified conditions. Only rows that satisfy the condition(s) will be included in the result.

Syntax

sql

Copy code

SELECT column1, column2

FROM table_name

WHERE condition;

Operators Used in the WHERE Clause

1. **Comparison Operators**:
    - =: Equal to
    - != or <>: Not equal to
    - <: Less than
    - >: Greater than
    - <=: Less than or equal to
    - >=: Greater than or equal to
2. **Logical Operators**:

- AND: Combines multiple conditions; all must be true.
- OR: Combines conditions; at least one must be true.
- NOT: Negates a condition.

3. **Range Operators**:
   - BETWEEN: Checks if a value falls within a range.
   - IN: Matches values in a list.
   - LIKE: Matches patterns (with wildcards % and _).
4. **NULL Handling**:
   - IS NULL: Checks if a column contains NULL.
   - IS NOT NULL: Checks if a column does not contain NULL.

Examples of Restricting Data

1. **Filter by a Single Condition**:

sql

Copy code

```
SELECT first_name, last_name
FROM employees
WHERE department = 'HR';
```

2. **Filter by Multiple Conditions**:

sql

Copy code

```
SELECT first_name, salary
FROM employees
WHERE department = 'Sales' AND salary > 50000;
```

3. **Using BETWEEN**:

sql

Copy code

```
SELECT first_name, salary
FROM employees
WHERE salary BETWEEN 40000 AND 60000;
```

4. **Using IN**:

```
sql
Copy code
SELECT first_name, department
FROM employees
WHERE department IN ('HR', 'Finance', 'Sales');
```

5. **Using LIKE**:

```
sql
Copy code
SELECT first_name
FROM employees
WHERE first_name LIKE 'J%'; -- Names starting with 'J'
```

Sorting Data: The ORDER BY Clause

The ORDER BY clause sorts the result set based on one or more columns. By default, sorting is in ascending order (ASC), but you can explicitly specify descending order (DESC).

Syntax

```
sql
Copy code
SELECT column1, column2
FROM table_name
ORDER BY column1 [ASC | DESC], column2 [ASC | DESC];
```

Examples of Sorting Data

1. **Sort by a Single Column**:

sql

Copy code

```
SELECT first_name, salary
FROM employees
ORDER BY salary;
```

2. **Sort by Multiple Columns**:

sql

Copy code

```
SELECT first_name, department, salary
FROM employees
ORDER BY department ASC, salary DESC;
```

3. **Sort in Descending Order**:

sql

Copy code

```
SELECT first_name, hire_date
FROM employees
ORDER BY hire_date DESC;
```

Combining Restriction and Sorting

You can use WHERE and ORDER BY together in a single query:

sql

Copy code

```
SELECT first_name, department, salary
FROM employees
WHERE salary > 40000 AND department = 'IT'
ORDER BY salary DESC;
```

Practice Exercise

1. Retrieve the first_name and last_name of employees in the "Marketing" department, sorted by last_name in ascending order.
2. Fetch all rows from the products table where the price is between 100 and 500, and sort the results by price in descending order.
3. Find the list of customers whose names start with "A" or "B", sorted by their registration date.

Single Row Functions

→**Single Row Functions in SQL**

Single-row functions are functions that operate on individual rows and return one result per row. They are commonly used to manipulate and format data during queries. These functions are applied in the SELECT or WHERE clauses.

Types of Single-Row Functions

1. String Functions

These functions operate on text strings.

Function	**Description**	**Example**
UPPER(string)	Converts a string to uppercase.	SELECT UPPER('hello') AS result; → HELLO
LOWER(string)	Converts a string to lowercase.	SELECT LOWER('HELLO') AS result; → hello
INITCAP(string)	Capitalizes the first letter of each word.	SELECT INITCAP('hello world') AS result; → Hello World
LENGTH(string)	Returns the length of a string.	SELECT LENGTH('hello') AS result; → 5
SUBSTR(string, start, length)	Extracts a substring.	SELECT SUBSTR('database', 1, 4) AS result; → data
INSTR(string, substring)	Returns the position of a substring.	SELECT INSTR('database', 'base') AS result; → 5

Function	Description	Example
TRIM(string)	Removes leading and trailing spaces.	SELECT TRIM(' hello ') AS result; → hello
CONCAT(string1, string2)	Concatenates two strings.	SELECT CONCAT('Hello', ' World') AS result; → Hello World

---

2. Numeric Functions

These functions operate on numeric data.

Function	Description	Example
ROUND(number, n)	Rounds a number to n decimal places.	SELECT ROUND(123.456, 2) AS result; → 123.46
TRUNC(number, n)	Truncates a number to n decimal places.	SELECT TRUNC(123.456, 2) AS result; → 123.45
MOD(number, divisor)	Returns the remainder of a division.	SELECT MOD(10, 3) AS result; → 1
ABS(number)	Returns the absolute value of a number.	SELECT ABS(-5) AS result; → 5
POWER(base, exponent)	Returns the result of raising a base to a power.	SELECT POWER(2, 3) AS result; → 8

---

3. Date Functions

These functions operate on date and time data.

Function	Description	Example
SYSDATE	Returns the current system date and time.	SELECT SYSDATE AS result; → Current Date/Time
CURRENT_DATE	Returns the current date.	SELECT CURRENT_DATE AS result; → Current Date

Function	Description	Example
ADD_MONTHS(date, n)	Adds n months to a date.	SELECT ADD_MONTHS(SYSDATE, 2) AS result; → 2 months later
MONTHS_BETWEEN(date1, date2)	Finds the number of months between two dates.	SELECT MONTHS_BETWEEN('2025-01-01', '2024-01-01') AS result; → 12
NEXT_DAY(date, 'day')	Finds the next occurrence of a day of the week.	SELECT NEXT_DAY(SYSDATE, 'Monday') AS result;
TRUNC(date, 'unit')	Truncates a date to a specific unit.	SELECT TRUNC(SYSDATE, 'MONTH') AS result; → First day of the month

---

4. Conversion Functions

These functions convert data from one type to another.

Function	Description	Example
TO_CHAR(date, format)	Converts a date to a string in a specific format.	SELECT TO_CHAR(SYSDATE, 'YYYY-MM-DD') AS result;
TO_DATE(string, format)	Converts a string to a date using a format mask.	SELECT TO_DATE('2025-01-01', 'YYYY-MM-DD') AS result;
TO_NUMBER(string)	Converts a string to a numeric value.	SELECT TO_NUMBER('123') AS result; → 123

Example Queries Using Single-Row Functions

1. **Formatting Names**:

sql

Copy code

SELECT UPPER(first_name), LOWER(last_name)

FROM employees;

2. **Calculating Age**:

sql

Copy code

```
SELECT first_name, TRUNC(MONTHS_BETWEEN(SYSDATE, birth_date) / 12) AS age
FROM employees;
```

3. **Formatting Dates**:

sql

Copy code

```
SELECT TO_CHAR(hire_date, 'DD-MON-YYYY') AS formatted_date
FROM employees;
```

4. **Extracting Substrings**:

sql

Copy code

```
SELECT SUBSTR(first_name, 1, 3) AS short_name
FROM employees;
```

Practice Exercises

1. Write a query to display the first name in uppercase and the last name in lowercase for all employees.
2. Display the current date and time in the format YYYY-MM-DD HH24:MI:SS.
3. Find the absolute difference between two numbers in a table.
4. Extract the first 5 characters from a product description.

Displaying Data from Multiple Tables

→**Displaying Data from Multiple Tables**

In SQL, data from multiple tables can be combined and displayed using **joins**, **set operators**, and **subqueries**. These techniques help in retrieving related data stored across different tables in a relational database.

---

1. Using Joins

A **join** is used to combine rows from two or more tables based on a related column between them.

Types of Joins

1. **Inner Join**:
    - Combines rows from both tables where the condition is satisfied.
    - Non-matching rows are excluded.

sql

Copy code

```
SELECT employees.first_name, departments.department_name
FROM employees
INNER JOIN departments
ON employees.department_id = departments.department_id;
```

2. **Left Join (Left Outer Join)**:
    - Returns all rows from the left table and the matching rows from the right table.
    - Non-matching rows in the right table are filled with NULL.

sql

Copy code

```
SELECT employees.first_name, departments.department_name
FROM employees
LEFT JOIN departments
ON employees.department_id = departments.department_id;
```

3. **Right Join (Right Outer Join)**:
    - Returns all rows from the right table and the matching rows from the left table.
    - Non-matching rows in the left table are filled with NULL.

sql

Copy code

```
SELECT employees.first_name, departments.department_name
```

```
FROM employees
RIGHT JOIN departments
ON employees.department_id = departments.department_id;
```

4. **Full Join (Full Outer Join)**:
    - Returns all rows from both tables. Non-matching rows are filled with NULL.

```
sql
Copy code
SELECT employees.first_name, departments.department_name
FROM employees
FULL JOIN departments
ON employees.department_id = departments.department_id;
```

5. **Cross Join**:
    - Produces the Cartesian product of two tables.
    - Each row from the first table is paired with every row from the second table.

```
sql
Copy code
SELECT employees.first_name, departments.department_name
FROM employees
CROSS JOIN departments;
```

6. **Self Join**:
    - A table is joined with itself to compare rows within the same table.

```
sql
Copy code
SELECT e1.first_name AS Employee, e2.first_name AS Manager
FROM employees e1
INNER JOIN employees e2
ON e1.manager_id = e2.employee_id;
```

---

2. Using Set Operators

Set operators combine the results of two or more SELECT statements. Tables must have the same number of columns with compatible data types.

- **UNION**: Combines the results of two queries and removes duplicates.

sql

Copy code

```
SELECT first_name FROM employees
UNION
SELECT first_name FROM customers;
```

- **UNION ALL**: Combines the results of two queries without removing duplicates.

sql

Copy code

```
SELECT first_name FROM employees
UNION ALL
SELECT first_name FROM customers;
```

- **INTERSECT**: Returns rows common to both queries.

sql

Copy code

```
SELECT first_name FROM employees
INTERSECT
SELECT first_name FROM customers;
```

- **EXCEPT** (or **MINUS** in some databases): Returns rows from the first query not in the second.

sql

Copy code

```
SELECT first_name FROM employees
EXCEPT
```

```
SELECT first_name FROM customers;
```

---

3. Using Subqueries

A **subquery** is a query within another query to retrieve data.

- **In the SELECT clause**:

sql

Copy code

```
SELECT first_name,
    (SELECT department_name
     FROM departments
     WHERE departments.department_id = employees.department_id) AS department
FROM employees;
```

- **In the WHERE clause**:

sql

Copy code

```
SELECT first_name, last_name
FROM employees
WHERE department_id = (SELECT department_id
                       FROM departments
                       WHERE department_name = 'HR');
```

- **In the FROM clause**:

sql

Copy code

```
SELECT dept_name, avg_salary
FROM (SELECT department_name AS dept_name, AVG(salary) AS avg_salary
    FROM employees
    INNER JOIN departments
```

ON employees.department_id = departments.department_id

GROUP BY department_name);

---

4. Example Scenario

Tables:

1. **employees**:

employee_id	first_name	department_id	manager_id
1	John	10	2
2	Sarah	20	NULL

2. **departments**:

department_id	department_name
10	HR
20	IT

Query: Retrieve employee names and their department names.

sql

Copy code

```
SELECT e.first_name, d.department_name
FROM employees e
INNER JOIN departments d
ON e.department_id = d.department_id;
```

Result:

first_name	department_name
John	HR
Sarah	IT

Practice Exercises

1. Write a query to retrieve all employees and their department names, including those without departments.
2. Display the names of employees who share the same department as "John."
3. Combine the lists of employee and customer first names using a set operator.

Aggregation Data Using Group Functions

→**Aggregation Data Using Group Functions**

Group functions in SQL are used to perform calculations on a set of rows, returning a single summarized result for each group of data. They are often used with the GROUP BY clause to group rows based on one or more columns.

Types of Group Functions

1. **COUNT**: Counts the number of rows.
2. **SUM**: Calculates the total sum of a numeric column.
3. **AVG**: Computes the average value of a numeric column.
4. **MIN**: Finds the smallest value in a column.
5. **MAX**: Finds the largest value in a column.
6. **STDDEV**: Computes the standard deviation.
7. **VARIANCE**: Computes the variance.

Syntax

sql

Copy code

```
SELECT group_function(column_name)
FROM table_name
[WHERE condition]
[GROUP BY column_name(s)]
[HAVING condition];
```

---

1. Common Group Functions

1.1 Using COUNT

sql

Copy code

SELECT COUNT(employee_id) AS total_employees

FROM employees;

Returns the total number of employees.

1.2 Using SUM

sql

Copy code

SELECT SUM(salary) AS total_salary

FROM employees;

Calculates the total salary of all employees.

1.3 Using AVG

sql

Copy code

SELECT AVG(salary) AS average_salary

FROM employees;

Calculates the average salary.

1.4 Using MIN and MAX

sql

Copy code

SELECT MIN(salary) AS lowest_salary, MAX(salary) AS highest_salary

FROM employees;

Finds the lowest and highest salary.

---

2. Grouping Data with GROUP BY

The GROUP BY clause groups rows that have the same values in specified columns and applies the group function to each group.

Example

sql

Copy code

SELECT department_id, AVG(salary) AS average_salary

FROM employees

GROUP BY department_id;

This calculates the average salary for each department.

---

3. Filtering Groups with HAVING

The HAVING clause filters groups created by GROUP BY based on a condition. It is similar to the WHERE clause but works on aggregated data.

Example

sql

Copy code

SELECT department_id, AVG(salary) AS average_salary

FROM employees

GROUP BY department_id

HAVING AVG(salary) > 50000;

This retrieves departments with an average salary greater than 50,000.

---

4. Using Multiple Group Functions

You can use multiple group functions in the same query.

sql

Copy code

SELECT department_id, COUNT(employee_id) AS total_employees, MAX(salary) AS highest_salary

FROM employees

GROUP BY department_id;

This shows the total employees and the highest salary in each department.

---

5. Combining Group Functions with Joins

You can combine data from multiple tables and use group functions.

sql

Copy code

SELECT d.department_name, COUNT(e.employee_id) AS total_employees

FROM employees e

INNER JOIN departments d

ON e.department_id = d.department_id

GROUP BY d.department_name;

This calculates the number of employees in each department.

Practice Exercises

1. Calculate the total number of employees, grouped by job title.
2. Find the departments where the total salary exceeds 1,000,000.
3. List the highest and lowest salaries for each department.
4. Calculate the average salary for each manager and display only those with an average salary above 60,000.

Sub Queries, Manipulating Data and Creating &Managing Tables

→**Subqueries, Manipulating Data, and Creating & Managing Tables in SQL**

---

1. Subqueries

A **subquery** is a query nested within another query. Subqueries can be used to retrieve data that will be used in the main query.

Types of Subqueries

1. **Single-row subquery**: Returns one row.
2. **Multiple-row subquery**: Returns multiple rows.
3. **Multiple-column subquery**: Returns multiple columns.

Syntax

sql

Copy code

```
SELECT column_name
FROM table_name
WHERE column_name = (SELECT column_name FROM another_table WHERE condition);
```

Examples

1. **Single-row Subquery**

sql

Copy code

```
SELECT first_name, salary
FROM employees
WHERE salary = (SELECT MAX(salary) FROM employees);
```

Retrieves the employee with the highest salary.

2. **Multiple-row Subquery**

sql

Copy code

```
SELECT first_name, department_id
FROM employees
WHERE department_id IN (SELECT department_id FROM departments WHERE location_id = 100);
```

Retrieves employees in departments located at location 100.

3. **Correlated Subquery** A subquery that references columns from the outer query.

sql

Copy code

```
SELECT first_name, salary
FROM employees e1
WHERE salary > (SELECT AVG(salary) FROM employees e2 WHERE e1.department_id = e2.department_id);
```

Retrieves employees earning above the average salary in their department.

---

2. Manipulating Data

Manipulating data involves inserting, updating, and deleting data in tables.

1. Inserting Data

sql

Copy code

```
INSERT INTO table_name (column1, column2)
VALUES (value1, value2);
```

- Example:

sql

Copy code

```
INSERT INTO employees (first_name, last_name, salary)
VALUES ('John', 'Doe', 50000);
```

- Insert data from another table:

sql

Copy code

```
INSERT INTO employees_backup (employee_id, first_name, salary)
SELECT employee_id, first_name, salary
FROM employees;
```

2. Updating Data

sql

Copy code

```
UPDATE table_name
SET column1 = value1, column2 = value2
WHERE condition;
```

- Example:

sql

Copy code

```
UPDATE employees
SET salary = salary * 1.1
WHERE department_id = 10;
```

3. Deleting Data

sql

Copy code

```
DELETE FROM table_name
WHERE condition;
```

- Example:

sql

Copy code

```
DELETE FROM employees
WHERE department_id = 20;
```

---

3. Creating and Managing Tables

1. Creating Tables

sql

Copy code

```
CREATE TABLE table_name (
   column1 data_type [constraint],
```

```
    column2 data_type [constraint],
    ...
);
```

- Example:

```
sql
Copy code
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    hire_date DATE,
    salary DECIMAL(10, 2)
);
```

2. Adding Constraints

Constraints enforce rules on the data in a table:

- **PRIMARY KEY**: Ensures unique and non-null values.
- **FOREIGN KEY**: Maintains referential integrity.
- **NOT NULL**: Ensures a column cannot have NULL.
- **UNIQUE**: Ensures unique values in a column.
- **CHECK**: Ensures values satisfy a condition.
- **DEFAULT**: Assigns a default value to a column.
    - Example:

```
sql
Copy code
CREATE TABLE departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(100) NOT NULL,
```

```
    location_id INT,
    CONSTRAINT fk_location FOREIGN KEY (location_id) REFERENCES locations(location_id)
);
```

3. Modifying Tables

- **Adding a Column**

sql

Copy code

```
ALTER TABLE table_name
ADD column_name data_type;
```

Example:

sql

Copy code

```
ALTER TABLE employees
ADD job_title VARCHAR(50);
```

- **Modifying a Column**

sql

Copy code

```
ALTER TABLE table_name
MODIFY column_name new_data_type;
```

Example:

sql

Copy code

```
ALTER TABLE employees
MODIFY salary DECIMAL(12, 2);
```

- **Dropping a Column**

sql

Copy code

ALTER TABLE table_name

DROP COLUMN column_name;

Example:

sql

Copy code

ALTER TABLE employees

DROP COLUMN job_title;

4. Dropping Tables

sql

Copy code

DROP TABLE table_name;

Example:

sql

Copy code

DROP TABLE employees;

Practice Exercises

1. Create a table named projects with the following columns:
    - project_id: Integer (Primary Key)
    - project_name: String (Not Null)
    - start_date: Date
    - budget: Decimal
2. Insert three rows of data into the projects table.
3. Write a query to increase the budget of all projects by 10%.
4. Create a subquery to display employees who work in a department where the department name is "Sales."
5. Delete all rows from the projects table where the budget is less than 5000.

Creating Views and Controlling User Access

→**Creating Views and Controlling User Access in SQL**

---

1. Creating Views

A **view** is a virtual table based on the result of a SQL query. Views simplify complex queries, enhance security by limiting data access, and provide a layer of abstraction.

Syntax

sql

Copy code

CREATE VIEW view_name AS

SELECT column1, column2, ...

FROM table_name

WHERE condition;

Examples

1. **Simple View**

sql

Copy code

CREATE VIEW employee_summary AS

SELECT first_name, last_name, department_id, salary

FROM employees

WHERE salary > 50000;

This view shows details of employees earning more than 50,000.

2. **Complex View**

sql

Copy code

CREATE VIEW department_salary AS

SELECT d.department_name, AVG(e.salary) AS average_salary

FROM employees e

INNER JOIN departments d

ON e.department_id = d.department_id

GROUP BY d.department_name;

This view displays the average salary for each department.

3. **Querying a View**

sql

Copy code

SELECT * FROM employee_summary;

Updating a View

You can modify an existing view using CREATE OR REPLACE.

sql

Copy code

CREATE OR REPLACE VIEW employee_summary AS

SELECT first_name, last_name, salary

FROM employees

WHERE department_id = 10;

Dropping a View

To remove a view:

sql

Copy code

DROP VIEW view_name;

Example:

sql

Copy code

DROP VIEW employee_summary;

---

2. Controlling User Access

Database systems allow administrators to control user access using privileges. This ensures security and limits user actions.

Granting Privileges

Use the GRANT statement to give permissions to users.

sql

Copy code

GRANT privilege_type ON object TO user;

Examples

1. **Granting Table Privileges**

sql

Copy code

GRANT SELECT, INSERT, UPDATE ON employees TO user1;

Allows user1 to view, insert, and update data in the employees table.

2. **Granting Privileges on a View**

sql

Copy code

GRANT SELECT ON employee_summary TO user2;

Allows user2 to view data from the employee_summary view.

3. **Granting All Privileges**

sql

Copy code

GRANT ALL PRIVILEGES ON employees TO user3;

4. **Granting Privileges with Grant Option** This allows the user to grant the same privileges to others.

sql

Copy code

GRANT SELECT ON employees TO user4 WITH GRANT OPTION;

Revoking Privileges

Use the REVOKE statement to remove user permissions.

sql

Copy code

REVOKE privilege_type ON object FROM user;

Examples

1. **Revoke Specific Privileges**

sql

Copy code

REVOKE INSERT, UPDATE ON employees FROM user1;

2. **Revoke All Privileges**

sql

Copy code

REVOKE ALL PRIVILEGES ON employees FROM user3;

---

3. User Roles

Database roles are used to manage privileges more efficiently. Instead of assigning privileges to individual users, they can be assigned to a role, which is then granted to users.

Creating a Role

sql

Copy code

CREATE ROLE manager_role;

Granting Privileges to a Role

sql

Copy code

GRANT SELECT, INSERT ON employees TO manager_role;

Assigning a Role to a User

sql

Copy code

```
GRANT manager_role TO user1;
```

Revoking a Role

sql

Copy code

```
REVOKE manager_role FROM user1;
```

Practice Exercises

1. Create a view named high_earners that displays employee names and salaries for employees earning more than 80,000.
2. Grant SELECT privileges on the high_earners view to a user named analyst.
3. Create a role named data_manager and assign privileges to INSERT, UPDATE, and DELETE on the employees table.
4. Assign the data_manager role to a user named manager1.
5. Revoke DELETE privileges from the data_manager role.

Using Set Operators, Datetime Function

→**Using Set Operators**

Set operators combine the results of two or more SELECT queries. These operators help merge, compare, or differentiate datasets from multiple queries.

Common Set Operators

1. **UNION**: Combines the results of two queries and removes duplicates.

sql

Copy code

```
SELECT first_name FROM employees
UNION
SELECT first_name FROM customers;
```

2. **UNION ALL**: Combines the results of two queries, including duplicates.

sql

Copy code

```
SELECT first_name FROM employees
UNION ALL
SELECT first_name FROM customers;
```

3. **INTERSECT**: Returns rows that are common to both queries.

sql

Copy code

```
SELECT first_name FROM employees
INTERSECT
SELECT first_name FROM customers;
```

4. **EXCEPT (or MINUS)**: Returns rows from the first query that are not in the second query.

sql

Copy code

```
SELECT first_name FROM employees
EXCEPT
SELECT first_name FROM customers;
```

Key Rules

- The number of columns in the SELECT statements must be the same.
- Data types of corresponding columns must be compatible.

Datetime Functions

Datetime functions are used to manipulate and retrieve information from date and time data types.

Common Datetime Functions

1. **Current Date and Time**

- CURRENT_DATE: Returns the current date.
- CURRENT_TIMESTAMP: Returns the current date and time.

Example:

sql

Copy code

```
SELECT CURRENT_DATE AS today, CURRENT_TIMESTAMP AS now;
```

2. **Extracting Components**
    - EXTRACT(): Retrieves parts of a date (e.g., year, month, day).

sql

Copy code

```
SELECT EXTRACT(YEAR FROM hire_date) AS year_hired
FROM employees;
```

3. **Adding or Subtracting Intervals**
    - Add or subtract time intervals to/from a date.

sql

Copy code

```
SELECT hire_date + INTERVAL '1 year' AS next_anniversary
FROM employees;
```

4. **Formatting Dates**
    - TO_CHAR(): Converts dates to a specific string format.

sql

Copy code

```
SELECT TO_CHAR(hire_date, 'YYYY-MM-DD') AS formatted_date
FROM employees;
```

5. **Difference Between Dates**
    - Use subtraction to calculate the difference.

sql

Copy code

```
SELECT CURRENT_DATE - hire_date AS days_worked
FROM employees;
```

6. **Converting Strings to Dates**
    - TO_DATE(): Converts a string into a date.

sql

Copy code

```
SELECT TO_DATE('2025-01-01', 'YYYY-MM-DD') AS new_year;
```

7. **Truncating Dates**
    - TRUNC(): Truncates a date to a specific unit (e.g., month, year).

sql

Copy code

```
SELECT TRUNC(hire_date, 'MONTH') AS start_of_month
FROM employees;
```

Examples

Using Set Operators

1. **Combine Employees and Customers**

sql

Copy code

```
SELECT first_name FROM employees
UNION
SELECT first_name FROM customers;
```

2. **Find Employees Also in Customers**

sql

Copy code

```
SELECT first_name FROM employees
```

INTERSECT

SELECT first_name FROM customers;

3. **Find Employees Not in Customers**

sql

Copy code

SELECT first_name FROM employees

EXCEPT

SELECT first_name FROM customers;

Using Datetime Functions

1. **Calculate Age of Employees**

sql

Copy code

SELECT first_name, CURRENT_DATE - birth_date AS age_in_days

FROM employees;

2. **Find Employees Hired in the Last Year**

sql

Copy code

SELECT first_name, hire_date

FROM employees

WHERE hire_date > CURRENT_DATE - INTERVAL '1 year';

3. **Display Hire Date in Custom Format**

sql

Copy code

SELECT first_name, TO_CHAR(hire_date, 'Month DD, YYYY') AS hire_date_formatted

FROM employees;

Practice Exercises

1. Combine the list of job titles from jobs and departments tables using a set operator.
2. Find the intersection of employee IDs from employees and managers tables.
3. Calculate how many days an employee has worked since their hire date.
4. Display the month and year when each employee was hired, formatted as "Month YYYY."
5. Determine which employees have an anniversary this month.

Database Design: Logical Design, Conceptual Design, Mapping Conceptual to Logical, Pragmatic issues, Physical Design, Integrity and Correctness, Relational Algebra, Relational Calculus

→**Database Design: Logical, Conceptual, and Physical Design**

Database design involves creating a structure that ensures efficient, accurate, and secure storage and retrieval of data. The design process typically proceeds through several stages: conceptual, logical, and physical.

---

1. Conceptual Design

Conceptual design is the process of defining the overall structure of the database at a high level. This stage focuses on identifying the main entities and their relationships without worrying about how the database will be implemented.

- **Entity-Relationship (ER) Model**: The primary tool for conceptual design.
    - **Entities**: Represent objects, things, or concepts in the real world (e.g., customers, orders, employees).
    - **Attributes**: Define the properties of entities (e.g., name, address, date of birth).
    - **Relationships**: Represent the associations between entities (e.g., a customer places an order).
- **ER Diagram**: Visual representation of entities and their relationships.

Example Conceptual Design:

- **Entities**: Customer, Order, Product
- **Relationships**: Customer places Order, Order contains Product

---

2. Logical Design

Logical design is the process of converting the high-level conceptual design into a more detailed structure that can be implemented in a relational database management system (RDBMS). This stage defines the tables, columns, data types, and relationships using a relational model.

- **Normalization**: Organizing data to reduce redundancy and improve integrity. The most common normal forms are:
    - **1st Normal Form (1NF)**: Eliminate repeating groups, make sure each column contains atomic values.
    - **2nd Normal Form (2NF)**: Eliminate partial dependency (i.e., non-prime attributes must depend on the entire primary key).
    - **3rd Normal Form (3NF)**: Eliminate transitive dependency (i.e., non-prime attributes must not depend on other non-prime attributes).
- **Relational Schema**: The logical representation of the database structure. It defines tables, columns, data types, and constraints (primary keys, foreign keys).

Example Logical Design:

- **Tables**: Customer (customer_id, name, address), Order (order_id, order_date, customer_id), Product (product_id, name, price)

---

3. Mapping Conceptual to Logical

Once the conceptual model is designed, it must be mapped into a relational schema. This process involves transforming the ER diagram into relational tables while preserving the relationships and attributes.

Steps:

1. **Entities to Tables**: Each entity becomes a table.
    - Example: Customer becomes a table Customer(customer_id, name, address).
2. **Relationships to Foreign Keys**: Relationships between entities are represented by foreign keys.
    - Example: A relationship Customer places Order becomes customer_id as a foreign key in the Order table.
3. **Handling Multivalued Attributes**: If an entity has attributes that can hold multiple values (e.g., a customer can have multiple phone numbers), create a new table to store those values.

---

4. Pragmatic Issues in Database Design

While logical and conceptual designs focus on correctness and structure, **pragmatic issues** address practical aspects of database design, such as performance, security, and maintainability.

- **Data Redundancy**: Minimizing unnecessary duplication of data to reduce storage costs and potential inconsistencies.
- **Performance Optimization**: Indexing, denormalization, and query optimization techniques for faster data retrieval.
- **Concurrency Control**: Ensuring that multiple users can access the database simultaneously without conflicts.
- **Security**: Implementing access controls, encryption, and data masking to protect sensitive data.

---

5. Physical Design

Physical design refers to how the logical design is implemented physically in the database system. This stage focuses on the performance of the database.

- **Storage and Indexing**: Determining the optimal way to store data and create indexes to improve query performance.
- **Partitioning**: Splitting large tables into smaller, more manageable pieces.
- **Clustering**: Grouping related data together to minimize disk I/O operations.

Example:

- **Indexing**: Creating an index on the customer_id field to speed up queries related to customers.

---

6. Integrity and Correctness

Integrity ensures that the data in the database is accurate, consistent, and reliable. The main types of integrity constraints are:

- **Entity Integrity**: Ensures that each row in a table is uniquely identifiable by a primary key.
- **Referential Integrity**: Ensures that foreign key relationships between tables are valid.

- **Domain Integrity**: Ensures that the data in a column is valid and consistent with its defined data type and constraints.
- **User-Defined Integrity**: Custom rules defined by the database designer.

---

7. Relational Algebra

Relational Algebra is a formal language for relational databases that uses a set of operations to manipulate relations (tables).

Common Operations in Relational Algebra:

1. **SELECT (σ)**: Filters rows based on a condition.

text

Copy code

σ(condition)(Relation)

Example: Retrieve employees with a salary greater than 50,000.

text

Copy code

σ(salary > 50000)(employees)

2. **PROJECT (π)**: Selects specific columns from a relation.

text

Copy code

π(column1, column2)(Relation)

Example: Retrieve the names and salaries of employees.

text

Copy code

π(first_name, salary)(employees)

3. **JOIN (⋈)**: Combines two relations based on a common attribute.

text

Copy code

Relation1 ⋈ Relation2

4. **UNION (∪)**: Combines the results of two queries and removes duplicates.

text

Copy code

Relation1 ∪ Relation2

5. **INTERSECT (∩)**: Retrieves the common rows from two relations.

text

Copy code

Relation1 ∩ Relation2

6. **DIFFERENCE (–)**: Retrieves rows from one relation that are not in another.

text

Copy code

Relation1 − Relation2

---

8. Relational Calculus

Relational Calculus is a non-procedural query language used to express queries in relational databases.

There are two types of relational calculus:

1. **Tuple Relational Calculus**: Uses variables that range over tuples (rows).
    - Example: Find employees with a salary greater than 50,000.

text

Copy code

{t | t ∈ employees ∧ t.salary > 50000}

2. **Domain Relational Calculus**: Uses variables that range over domain values (columns).
    - Example: Find employees with a salary greater than 50,000.

text

Copy code

{f, l | ∃s (employees(f, l, s) ∧ s > 50000)}

Summary of Steps in Database Design

1. **Conceptual Design**: Define entities, relationships, and attributes using the ER model.
2. **Logical Design**: Translate the conceptual design into relational tables, applying normalization.
3. **Mapping Conceptual to Logical**: Map entities and relationships to tables and foreign keys.
4. **Pragmatic Issues**: Address data redundancy, performance, concurrency, and security.
5. **Physical Design**: Optimize storage, indexing, and partitioning.
6. **Integrity and Correctness**: Ensure data accuracy using integrity constraints.
7. **Relational Algebra**: Use operations like SELECT, PROJECT, and JOIN to query data.
8. **Relational Calculus**: Use tuple or domain calculus to express queries.

Normalization: 1NF, 2NF, 3NF, BCNF, 4NF, 5NF, DKNF

→**Normalization in Database Design**

Normalization is the process of organizing a database to reduce redundancy and improve data integrity. The goal of normalization is to break down data into smaller, well-structured tables while eliminating unnecessary duplication. Normalization involves applying a series of rules, called **normal forms (NF)**, to achieve a better database design.

Here's a breakdown of the common normal forms, starting from the first and going up to the fifth, as well as **Domain-Key Normal Form (DKNF)**.

---

1. First Normal Form (1NF)

Definition: A relation (table) is in **1NF** if it contains only atomic values (i.e., indivisible values) and each column contains unique data.

- **Key Points**:
    - Eliminate repeating groups or arrays in columns.
    - Each cell in the table must contain only one value.
    - Each column must contain values of a single data type.

Example (Non-1NF):

plaintext

Copy code

StudentID | StudentName | Courses

1001 | John | Math, English

1002 | Jane | Science, History

- In the above table, the Courses column violates 1NF because it contains multiple values in one cell.

Example (1NF):

plaintext

Copy code

StudentID | StudentName | Course

1001 | John | Math

1001 | John | English

1002 | Jane | Science

1002 | Jane | History

---

2. Second Normal Form (2NF)

Definition: A relation is in **2NF** if it is in **1NF** and all non-key attributes are fully functionally dependent on the **entire** primary key (i.e., no partial dependencies).

- **Key Points**:
    - Eliminate partial dependencies where non-key attributes depend only on part of a composite primary key.
    - This rule applies only when the primary key is composite (i.e., consists of more than one attribute).

Example (Non-2NF):

plaintext

Copy code

StudentID | CourseID | StudentName | CourseName

1001 | C001 | John | Math

1001 | C002 | John | English

1002 | C001 | Jane | Science

- Here, StudentName depends only on StudentID and not on the whole primary key (StudentID, CourseID), so it's a partial dependency.

Example (2NF):

To fix the above, break the table into two:

plaintext

Copy code

-- Student Table

StudentID | StudentName

1001 | John

1002 | Jane

-- Course Table

StudentID | CourseID | CourseName

1001 | C001 | Math

1001 | C002 | English

1002 | C001 | Science

---

3. Third Normal Form (3NF)

Definition: A relation is in **3NF** if it is in **2NF** and there is no **transitive dependency** between non-key attributes (i.e., non-key attributes depend only on the primary key and not on each other).

- **Key Points**:
    - Eliminate transitive dependencies where one non-key attribute depends on another non-key attribute.

Example (Non-3NF):

plaintext

Copy code

StudentID | StudentName | Department | DepartmentHead

1001 | John | Computer | Dr. Smith

1002 | Jane | Science | Dr. Johnson

- Here, DepartmentHead depends on Department, which is not a key, causing a transitive dependency.

Example (3NF):

To eliminate the transitive dependency, create a separate table for departments:

plaintext

Copy code

-- Student Table

StudentID | StudentName | Department

1001 | John | Computer

1002 | Jane | Science

-- Department Table

Department | DepartmentHead

Computer | Dr. Smith

Science | Dr. Johnson

---

4. Boyce-Codd Normal Form (BCNF)

Definition: A relation is in **BCNF** if it is in **3NF** and, for every non-trivial functional dependency, the determinant (the attribute on the left side of the dependency) is a **superkey**.

- **Key Points**:
    - BCNF is a stricter version of 3NF. While 3NF allows some non-superkey attributes to functionally determine others, BCNF does not.

Example (Non-BCNF):

plaintext

Copy code

StudentID | CourseID | Instructor

1001 | C001 | Dr. Smith

1001 | C002 | Dr. Johnson

1002 | C001 | Dr. Smith

- In this example, Instructor determines CourseID, but Instructor is not a superkey.

Example (BCNF):

To convert to BCNF, split the table:

plaintext

Copy code

-- Course Table

CourseID | Instructor

C001 | Dr. Smith

C002 | Dr. Johnson

-- Enrollment Table

StudentID | CourseID

1001 | C001

1001 | C002

1002 | C001

---

5. Fourth Normal Form (4NF)

Definition: A relation is in **4NF** if it is in **BCNF** and has no **multi-valued dependencies**. A multi-valued dependency occurs when one attribute determines multiple independent attributes.

- **Key Points**:
    - A multi-valued dependency exists when a single attribute is related to multiple independent attributes in a way that each combination of values is valid.

Example (Non-4NF):

plaintext

Copy code

StudentID | CourseID | Language

1001 | C001 | English

1001 | C001 | Spanish

1002 | C002 | French

- In this example, StudentID determines both CourseID and Language, and the two are independent.

Example (4NF):

To eliminate the multi-valued dependency, break the table into two:

plaintext

Copy code

-- Enrollment Table

StudentID | CourseID

1001 | C001

1002 | C002

-- Language Table

StudentID | Language

1001 | English

1001 | Spanish

1002 | French

---

6. Fifth Normal Form (5NF)

Definition: A relation is in **5NF** if it is in **4NF** and cannot be decomposed into any smaller relations without losing information. It deals with **join dependencies** and ensures that there are no redundancies left in the table.

- **Key Points**:
    - 5NF is concerned with eliminating redundancy caused by **join dependencies**.
    - A table is in 5NF when every join dependency is implied by the candidate keys.

Example (Non-5NF):

plaintext

Copy code

ProjectID | EmployeeID | Skill

P001 | E001 | Java

P001 | E001 | Python

P002 | E002 | Java

- Here, the table can be decomposed in a way that would avoid redundancy. In practice, this would be quite rare, but this is where 5NF comes into play.

Example (5NF):

Decompose the table into separate relations:

plaintext

Copy code

-- Project Employee Table

ProjectID | EmployeeID

P001 | E001

P002 | E002

-- Employee Skill Table

EmployeeID | Skill

E001 | Java

E001 | Python

E002 | Java

---

7. Domain-Key Normal Form (DKNF)

Definition: A relation is in **DKNF** if it is free from all modification anomalies, i.e., no integrity constraints are violated.

- **Key Points**:
    - DKNF is the most restrictive normal form. It is concerned with the domain of the data and key constraints, ensuring that the design eliminates all possible anomalies.

Example:

In practice, a relation in DKNF would have no restrictions apart from the inherent constraints, making it the highest level of normalization.

Summary of Normal Forms:

- **1NF**: Eliminate repeating groups; ensure atomicity.
- **2NF**: Eliminate partial dependencies (no partial key dependency).
- **3NF**: Eliminate transitive dependencies (non-key attributes should not depend on other non-key attributes).
- **BCNF**: Every determinant must be a superkey.
- **4NF**: Eliminate multi-valued dependencies.
- **5NF**: Eliminate join dependencies and ensure data is stored in the smallest possible relations.
- **DKNF**: No modification anomalies, respecting all integrity constraints.

Practice Exercises

1. Normalize the following table to 2NF:

plaintext

Copy code

OrderID | ProductID | ProductName | OrderDate | CustomerName

2. Normalize a table with multiple values in a single column (multi-valued dependency) to 4NF.

Architecture of DBMS: Client-server, Open Architectures, Transaction Processing, Multi-User & Concurrency, and Backup & Recovery Database

→**Architecture of DBMS (Database Management System)**

The architecture of a DBMS defines how various components of the system interact with each other to manage the storage, retrieval, and manipulation of data. Below are the main architectural models and components of a DBMS, along with an explanation of related topics such as client-server architecture, transaction processing, and concurrency control.

---

1. Client-Server Architecture

Client-Server Architecture refers to the way the DBMS system is organized to separate tasks between clients (users or applications) and servers (where the database is stored and managed).

- **Client**: This is the user interface or application that requests data from the DBMS. Clients can be remote systems or applications accessing the database over a network. Clients may send SQL queries, request reports, or perform updates.
- **Server**: The DBMS server processes client requests, manages database storage, and performs operations such as query execution, data retrieval, transaction management, and security enforcement.
- **Two-Tier Architecture**:
    - In a two-tier architecture, there are two layers: the client (front-end) and the DBMS server (back-end).
    - Clients directly interact with the server. The client sends requests to the server, which processes them and sends back the results.
    - **Example**: A user runs a desktop application (client) that queries a database running on a centralized server.
- **Three-Tier Architecture**:
    - In a three-tier architecture, an additional layer is added between the client and the DBMS server, known as the **application server** or **middleware**.
    - The client communicates with the application server, which processes the business logic and communicates with the DBMS server.
    - **Example**: A web-based application where the user interface (client) interacts with an application server, which in turn interacts with the database server.

Advantages of Client-Server Architecture:

- Centralized management of data.
- Easier maintenance and updates.
- Supports large-scale database applications.

---

2. Open Architectures

Open Architecture refers to a DBMS that uses an open system design and adheres to standards that ensure interoperability with other systems, platforms, and software.

- **Open Database Connectivity (ODBC)** and **JDBC (Java Database Connectivity)** are common standards for connecting DBMS to applications, ensuring platform independence.
- **Data independence** is a crucial feature in open architectures, meaning that changes in the database schema should not affect application software.

Key Points:

- Open architectures allow the DBMS to work in a heterogeneous environment (different hardware, operating systems, or databases).
- **Modular design** allows components like storage management, query processing, and transaction management to be replaced or upgraded without affecting the overall system.

---

3. Transaction Processing

Transaction Processing is a core feature of any DBMS, ensuring that database operations are carried out in a consistent, reliable, and efficient manner.

- **Transaction**: A transaction is a logical unit of work, such as a group of SQL queries or operations that must be executed together.
- A transaction can involve operations like reading, writing, or updating data in the database.

ACID Properties:

A transaction must follow the ACID properties to ensure reliable processing:

1. **Atomicity**: Ensures that all operations within a transaction are completed successfully; otherwise, the transaction is aborted and the database is reverted to its previous state.
2. **Consistency**: The database must remain in a valid state after the transaction, adhering to all integrity constraints.
3. **Isolation**: Transactions must execute independently of each other, even if they occur simultaneously. Intermediate states should not be visible to other transactions.
4. **Durability**: Once a transaction is committed, its changes are permanent, even in the event of a system failure.

Types of Transaction Processing:

- **Online Transaction Processing (OLTP)**: Deals with real-time transaction data, typically used in banking, retail, and other business systems.

- **Batch Processing**: A system that processes large volumes of data in batches, often used for payroll or report generation.

---

4. Multi-User & Concurrency Control

Multi-User Environment: DBMS is often used in environments where multiple users access and manipulate the database simultaneously. Managing simultaneous access is essential to maintain the integrity and consistency of the database.

Concurrency Control: Mechanisms to ensure that multiple transactions can occur concurrently without conflicting with each other and causing data inconsistencies.

Concurrency Control Techniques:

1. **Locking**:
    - **Locking** is a method used to prevent conflicting operations by locking the data items being used by a transaction.
    - **Exclusive Lock**: Prevents other transactions from accessing the data.
    - **Shared Lock**: Allows multiple transactions to read the data but not modify it.
2. **Timestamp Ordering**: Each transaction is given a unique timestamp, and the DBMS ensures that transactions are processed in timestamp order to avoid conflicts.
3. **Optimistic Concurrency Control**: Transactions proceed without locking data, but before committing, the DBMS checks whether any conflict has occurred. If a conflict is detected, the transaction is rolled back.

Challenges in Multi-User Systems:

- **Lost Updates**: When two transactions overwrite each other's changes.
- **Uncommitted Data**: A transaction may read data that has been modified but not yet committed, leading to inconsistency.
- **Inconsistent Retrievals**: A transaction may see data in an inconsistent state if it reads values that are in the process of being updated by another transaction.

---

5. Backup & Recovery in DBMS

Backup and Recovery are critical components of DBMS to ensure data is protected and can be restored in case of failures, such as system crashes or data corruption.

Backup:

- **Full Backup**: A complete copy of the database is taken, capturing all data, schema, and objects.
- **Incremental Backup**: Only changes made since the last backup are recorded.
- **Differential Backup**: Captures changes since the last full backup.

Recovery:

The process of restoring the database to its previous state in case of a failure, ensuring that no transactions are lost or corrupted.

Techniques for Recovery:

1. **Transaction Log**: A log that records every transaction and its operations. In case of failure, the DBMS can use the log to replay transactions and restore the database to a consistent state.
2. **Checkpointing**: Periodically saving the state of the database to minimize recovery time in case of failure.
3. **Shadow Paging**: Uses backup pages to maintain consistency. The DBMS keeps track of "shadow" copies of data, ensuring that if a failure occurs, the system can roll back to the last consistent state.

Types of Failures:

1. **System Failure**: The hardware or software crashes, causing the DBMS to stop functioning.
2. **Transaction Failure**: A transaction fails due to errors, such as violations of integrity constraints or deadlocks.
3. **Media Failure**: A failure in the storage media, such as hard drive corruption or data loss.

Summary of Key DBMS Concepts:

1. **Client-Server Architecture**: Separates the client (user or application) from the server (DBMS) for centralized data management.
2. **Open Architecture**: Ensures interoperability and flexibility by using open standards.
3. **Transaction Processing**: Ensures that transactions follow the ACID properties to maintain data consistency.
4. **Concurrency Control**: Manages simultaneous transactions in multi-user environments to prevent conflicts and maintain integrity.

5. **Backup & Recovery**: Ensures data protection and restoration in case of system failure, using methods like transaction logs, checkpoints, and backup strategies.

Basic Concept of major RDBMS products: Oracle, Sybase, DB2, SQL Server and other Databases

→**Basic Concepts of Major RDBMS Products**

Relational Database Management Systems (RDBMS) are designed to manage data that is structured in tables, where relationships between data elements are established using keys. Here's an overview of some major RDBMS products: **Oracle**, **Sybase**, **DB2**, **SQL Server**, and others.

---

1. Oracle Database

Overview: Oracle Database is one of the most widely used RDBMS products. It is known for its reliability, scalability, and security features. Oracle Database supports both traditional on-premise databases and cloud databases.

Key Features:

- **Architecture**: Oracle uses a **multi-layer architecture** with an instance (in-memory structures) and database (physical storage files). An Oracle database includes several background processes for managing memory, recovery, and transaction management.
- **SQL Support**: Oracle uses a powerful implementation of SQL and PL/SQL (Procedural Language for SQL) for programming and query optimization.
- **Concurrency Control**: Oracle uses **Read Consistency** and **Undo Segments** to manage multi-user access and provide consistent views of data.
- **Backup & Recovery**: Oracle's **RMAN (Recovery Manager)** is used for backup and recovery operations. It also provides tools like **Data Guard** for disaster recovery.
- **Scalability**: Oracle supports **Real Application Clusters (RAC)**, which allows for multiple database instances on different servers to share the same physical database, providing high availability and scalability.
- **Cloud Services**: Oracle has expanded into the cloud with **Oracle Autonomous Database**, offering automated performance tuning, scaling, and patching.

---

2. Sybase (SAP ASE - Adaptive Server Enterprise)

Overview: Sybase, now owned by SAP, is a widely used RDBMS for business applications, particularly for industries needing high-performance transaction processing. Sybase ASE is particularly popular in financial institutions.

Key Features:

- **Architecture**: Sybase ASE follows a **client-server** model, where clients connect to a Sybase server that manages data stored in databases.
- **Transaction Management**: Sybase has robust transaction management using **locks** to handle concurrent access to data.
- **Replication**: Sybase includes **Replication Server**, which is used to replicate data between databases for backup and high availability.
- **Performance**: It is known for its high performance in handling transactional workloads, particularly for **OLTP (Online Transaction Processing)**.
- **Security**: Sybase offers **role-based access control**, encryption, and auditing capabilities to secure the database.
- **Backup & Recovery**: Sybase provides several tools for backup and restore, including **ASE Backup Server**.

---

3. IBM DB2

Overview: IBM DB2 is an RDBMS developed by IBM. It is known for its scalability, reliability, and strong support for both transactional and analytical processing. DB2 is available on multiple platforms, including IBM mainframes, UNIX, Linux, and Windows.

Key Features:

- **Architecture**: DB2 supports both **single-node** and **multi-node configurations** for high availability and scalability. It supports different **data models** (relational, XML, and JSON).
- **SQL Support**: DB2 uses a rich implementation of **SQL** and is highly optimized for large-scale transactional and analytical queries.
- **Concurrency Control**: DB2 uses **multi-version concurrency control (MVCC)** to ensure that multiple users can access the database without interfering with each other's work.
- **Backup & Recovery**: DB2 has **online backup and recovery** tools that allow for continuous database operation during backup processes.
- **Optimization**: DB2 features an **Automatic Storage Management** system and **query optimization** to improve performance.
- **Integration with Big Data**: DB2 supports integration with Big Data systems and is capable of handling large-scale analytics through the **DB2 Big Data capabilities**.

---

4. Microsoft SQL Server

Overview: SQL Server is a relational database developed by Microsoft. It is one of the most widely used RDBMS products, particularly in enterprises that rely on Microsoft technologies.

Key Features:

- **Architecture**: SQL Server uses a **client-server** architecture and includes several components such as the **SQL Server engine**, **SQL Server Agent**, and **SQL Server Reporting Services (SSRS)**.
- **T-SQL**: SQL Server uses **Transact-SQL (T-SQL)**, an extended version of SQL, for writing queries and stored procedures.
- **Concurrency Control**: SQL Server uses **locking**, **row-level versioning**, and **transaction isolation levels** to manage concurrency.
- **High Availability**: SQL Server offers several high availability features, including **Always On Availability Groups**, **Replication**, and **Database Mirroring**.
- **Backup & Recovery**: SQL Server has comprehensive **backup and recovery** options, including **point-in-time recovery** and **transaction log backups**.
- **Integration**: SQL Server is highly integrated with other Microsoft tools, such as **Power BI**, **Azure**, and **.NET** development environment.
- **Cloud Services**: SQL Server is also available as part of **Microsoft Azure SQL Database**.

---

5. Other Databases

MySQL

- **Overview**: MySQL is an open-source RDBMS popular for web applications. It is widely used with PHP and is the "M" in the LAMP stack (Linux, Apache, MySQL, PHP).
- **Key Features**:
    - Simple architecture and low-cost licensing.
    - Excellent performance for web applications with support for **replication** and **clustered environments**.
    - Uses **InnoDB** as its default storage engine, providing ACID compliance and support for foreign keys.

PostgreSQL

- **Overview**: PostgreSQL is an advanced open-source RDBMS that emphasizes extensibility, standards compliance, and data integrity.
- **Key Features**:
    - Supports advanced data types like JSON, XML, and user-defined types.
    - Known for **ACID compliance**, **MVCC**, and **robust concurrency control**.
    - Provides **full-text search** and **geospatial data** handling (PostGIS).
    - Extensible through user-defined functions and custom extensions.

SQLite

- **Overview**: SQLite is a lightweight, serverless, self-contained RDBMS used in embedded systems, mobile devices, and small-scale applications.
- **Key Features**:
    - Zero-configuration: No need for a server process or complex setup.
    - ACID-compliant transactions.
    - Frequently used in **mobile apps**, **IoT**, and **local storage** in applications.

Amazon Aurora

- **Overview**: Amazon Aurora is a fully managed relational database service by AWS that is compatible with MySQL and PostgreSQL.
- **Key Features**:
    - Highly available with **automatic failover** and **replication** across multiple availability zones.
    - Scalable and cost-effective, with features like **auto-scaling** and **parallel query execution**.
    - **Integration with AWS services**, including Lambda and S3.

Comparison of Key Features

Feature	Oracle	Sybase	DB2	SQL Server	MySQL	PostgreSQL	SQLite
License	Commercial	Commercial	Commercial	Commercial	Open-source	Open-source	Open-source

Feature	Oracle	Sybase	DB2	SQL Server	MySQL	PostgreSQL	SQLite
Platform	Cross-platform	Cross-platform	Cross-platform	Windows, Linux	Cross-platform	Cross-platform	Cross-platform
ACID Compliance	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SQL Support	Advanced	Advanced	Advanced	Advanced	Basic	Advanced	Basic
Concurrency Control	MVCC	Locks	MVCC	MVCC	Locks	MVCC	Basic
High Availability	RAC, Data Guard	Replication	HADR, DB2 Replication	Always On	Replication	Replication	No
Backup & Recovery	RMAN, Data Guard	ASE Backup	Online Backup	SQL Server Backup	mysqldump	pg_dump	Simple Backup
Cloud Support	Oracle Cloud	SAP Cloud	IBM Cloud	Azure SQL Database	Amazon RDS	Amazon RDS	None

---

Conclusion

Each of these RDBMS products offers unique features and strengths, and the choice of system largely depends on the specific needs of an organization or project. Oracle, DB2, and SQL Server are powerful, enterprise-grade solutions, while MySQL, PostgreSQL, and SQLite provide strong open-source alternatives suitable for smaller-scale applications, web apps, and development environments.