Module 4 – Introduction to DBMS

Introduction to SQL

Theory Questions:

1. What is SQL, and why is it essential in database management?

Ans. **SQL (Structured Query Language)** is a standard programming language specifically designed for managing and manipulating relational databases. It’s used to interact with databases by performing tasks such as querying data, updating records, inserting new data, and deleting existing data.

It is essential in data management for the following :

1. **Data Querying:**
   * SQL allows users to retrieve specific data from large datasets using powerful **SELECT** statements with filtering, sorting, and grouping.
2. **Data Manipulation:**
   * It provides commands like **INSERT**, **UPDATE**, and **DELETE** to modify data efficiently.
3. **Data Definition:**
   * Through **Data Definition Language (DDL)** commands like **CREATE**, **ALTER**, and **DROP**, SQL defines and modifies the structure of database tables and other objects.
4. **Data Control:**
   * SQL includes **Data Control Language (DCL)** commands such as **GRANT** and **REVOKE**, which manage access permissions and security.
5. **Transactional Control:**
   * SQL supports transactions through commands like **BEGIN**, **COMMIT**, and **ROLLBACK**, which ensure data integrity and consistency.
6. **Standardization:**
   * SQL is widely used and standardized by ANSI and ISO, making it a universal language across many relational database systems (e.g., MySQL, PostgreSQL, SQL Server, Oracle).

In short, SQL is the backbone of relational database management, enabling structured and efficient handling of data, which is crucial for virtually every data-driven application. Want to dive into a specific SQL topic, like queries or indexing?

1. Explain the difference between DBMS and RDBMS.

Ans. A **DBMS** is a software system that allows users to create, retrieve, update, and manage data in databases. It provides a way to interact with data stored in files or non-relational formats.

**Key Characteristics:**

* Stores data as **files or collections of records**.
* Doesn’t require relationships between tables (i.e., can be non-relational).
* Can support single-user environments (e.g., **Microsoft Access**, **DBF**, etc.).
* May lack strict **data integrity** and **security** features.
* Doesn’t enforce **ACID** (Atomicity, Consistency, Isolation, Durability) properties strictly.

**RDBMS (Relational Database Management System):**

An **RDBMS** is a type of DBMS that stores data in a **structured format using rows and columns (tables)** and enforces relationships between data through **keys**.

**Key Characteristics:**

* Stores data in **tables** (i.e., relations).
* Enforces **relationships** between tables using **primary keys** and **foreign keys**.
* Supports **multi-user** environments.
* Strictly follows **ACID** properties for reliable transactions.
* Examples: **MySQL**, **PostgreSQL**, **Oracle**, **SQL Server**, **SQLite**.

|  |  |  |
| --- | --- | --- |
| Feature | DBMS | RDBMS |
| Data Storage | Files or records | Tables (rows and columns) |
| Relationships | Not enforced | Enforced using keys |
| Data Integrity | Basic or none | High (using constraints) |
| Multi-user Support | Often single-user | Multi-user |
| ACID Compliance | Not necessarily enforced | Strictly follows ACID properties |

Think of **DBMS** as a broader category, and **RDBMS** as a specialized, more powerful version that brings structure and relationships to data.

1. Describe the role of SQL in managing relational databases.

Ans. Absolutely! SQL plays a **central role** in managing relational databases — it’s essentially the **language** that lets you talk to and control the database.

**🧩 Role of SQL in Managing Relational Databases**

SQL (Structured Query Language) is used for **creating**, **managing**, and **manipulating** data stored in **relational database tables**. Here’s how it functions across various aspects:

**1. Data Definition (DDL – Data Definition Language)**

SQL helps define the structure of the database:

* CREATE TABLE – creates new tables
* ALTER TABLE – modifies existing table structure
* DROP TABLE – deletes tables

🔧 **Example:**

CREATE TABLE Students (

ID INT PRIMARY KEY,

Name VARCHAR(50),

Age INT

);

**2. Data Manipulation (DML – Data Manipulation Language)**

SQL allows inserting, updating, and deleting actual data:

* INSERT INTO – adds new records
* UPDATE – modifies existing records
* DELETE – removes records

✍️ **Example:**

INSERT INTO Students (ID, Name, Age) VALUES (1, 'Alex', 20);

**3. Data Querying (DQL – Data Query Language)**

SQL retrieves data with flexible queries:

* SELECT – extracts data with options for filtering (WHERE), sorting (ORDER BY), and grouping (GROUP BY)

🔎 **Example:**

SELECT Name FROM Students WHERE Age > 18;

**4. Data Control (DCL – Data Control Language)**

SQL manages **permissions and access control**:

* GRANT – gives access rights
* REVOKE – removes access rights

**5. Transaction Control (TCL – Transaction Control Language)**

Ensures data integrity during transactions:

* BEGIN, COMMIT, ROLLBACK – manage groups of operations as a unit

🔐 Helps maintain **ACID properties** (Atomicity, Consistency, Isolation, Durability)

**🔄 Summary:**

SQL allows users and applications to:

* Define the structure of data (tables, relationships)
* Insert, retrieve, and modify data
* Maintain data integrity and access control
* Execute reliable and secure transactions

It’s the **backbone of relational database interaction**.

1. What are the key features of SQL?

Ans. Great question! SQL is powerful because it's both **simple to learn** and **robust enough** to handle complex database tasks. Here are the **key features** that make SQL essential in working with relational databases:

**🔑 Key Features of SQL**

**1. Data Querying**

* SQL is best known for its ability to **retrieve data** using the SELECT statement.
* Allows filtering (WHERE), sorting (ORDER BY), grouping (GROUP BY), and joining tables.

Example:

SELECT name, age FROM Students WHERE age > 18 ORDER BY name;

**2. Data Manipulation**

* Includes commands to **insert**, **update**, and **delete** data.
* These are part of the **Data Manipulation Language (DML)**.

Example:

UPDATE Students SET age = 21 WHERE name = 'Alex';

**3. Data Definition**

* SQL defines and modifies the **structure of tables and databases**.
* Uses **Data Definition Language (DDL)** like CREATE, ALTER, DROP.

Example:

CREATE TABLE Courses (CourseID INT, CourseName VARCHAR(50));

**4. Data Control**

* Manages **user permissions** and access to data.
* Uses **Data Control Language (DCL)**: GRANT and REVOKE.

**5. Transaction Control**

* Supports **ACID transactions** using commands like:
  + BEGIN
  + COMMIT
  + ROLLBACK
* Ensures data reliability and consistency.

**6. Relational Data Handling**

* Allows defining **relationships** between tables using:
  + **Primary Keys**
  + **Foreign Keys**
* Supports **joins** to combine data from multiple tables.

Example:

SELECT Students.name, Courses.CourseName

FROM Students

JOIN Enrollments ON Students.ID = Enrollments.StudentID

JOIN Courses ON Enrollments.CourseID = Courses.CourseID;

**7. Standardized Language**

* SQL is **ANSI and ISO standardized**, making it widely compatible across different RDBMS like MySQL, PostgreSQL, Oracle, and SQL Server.

**8. Easy to Learn and Use**

* SQL uses **plain English-like syntax**, making it approachable for beginners and powerful for experts.

**9. Scalable and Efficient**

* SQL can handle everything from **small databases** to **enterprise-level systems** with **millions of records**.

**2. SQL Syntax**

**Theory Questions:**

1. What are the basic components of SQL syntax?

Ans. Great one! Understanding the **basic components of SQL syntax** is like learning the grammar of a new language — it's how you structure commands to communicate with a database.

**🧱 Basic Components of SQL Syntax**

**1. Statements (Commands)**

SQL is made up of statements, which are specific commands like:

* SELECT – retrieve data
* INSERT – add new data
* UPDATE – modify data
* DELETE – remove data
* CREATE – create objects like tables
* DROP – delete objects

Each statement usually ends with a **semicolon (;)**.

**2. Clauses**

Clauses are parts of statements that define what you're doing more specifically.

Examples:

* WHERE – filters rows
* ORDER BY – sorts results
* GROUP BY – groups results
* HAVING – filters grouped results
* FROM – specifies the source table

Example:

SELECT name, age FROM Students WHERE age > 18 ORDER BY age;

**3. Expressions**

Expressions combine **columns, values, operators, and functions** to produce a result.

Example:

SELECT name, (salary \* 0.10) AS Bonus FROM Employees;

**4. Predicates**

Predicates return **TRUE or FALSE** and are used in conditions (e.g., in WHERE, HAVING).

Examples:

* age > 18
* salary BETWEEN 30000 AND 50000
* name LIKE 'A%'

**5. Identifiers**

These are the **names of database objects**, like:

* Table names (Students)
* Column names (ID, Name)
* Aliases (AS s or AS StudentName)

**6. Operators**

Used in conditions and expressions:

* **Arithmetic:** +, -, \*, /
* **Comparison:** =, <>, >, <, >=, <=
* **Logical:** AND, OR, NOT
* **Set:** IN, BETWEEN, LIKE

**7. Functions**

Built-in operations for calculations or formatting:

* **Aggregate:** SUM(), COUNT(), AVG()
* **String:** CONCAT(), UPPER()
* **Date/Time:** NOW(), DATE()

**8. Comments**

Used to explain or disable parts of SQL code:

* Single-line: -- comment here
* Multi-line: /\* comment block \*/

1. Write the general structure of an SQL SELECT statement.

Ans. Sure! The **general structure of an SQL SELECT statement** is one of the most important things to understand in SQL. It’s used to **retrieve data** from one or more tables in a database.

**🧱 General Structure of a SELECT Statement:**

SELECT column1, column2, ...

FROM table\_name

[WHERE condition]

[GROUP BY column]

[HAVING group\_condition]

[ORDER BY column [ASC|DESC]]

[LIMIT number]; -- (or TOP / FETCH depending on the database)

**🔍 Explanation of Each Part:**

| **Clause** | **Description** |
| --- | --- |
| SELECT | Specifies the columns to be returned. Use \* to select all columns. |
| FROM | Specifies the table(s) to query data from. |
| WHERE | Filters rows based on specified conditions. |
| GROUP BY | Groups rows that have the same values in specified columns. |
| HAVING | Filters grouped rows (like WHERE but for aggregates). |
| ORDER BY | Sorts the result set by one or more columns (default is ASC). |
| LIMIT | Restricts the number of rows returned (used in MySQL/PostgreSQL). |

**✅ Example Query:**

SELECT name, age, COUNT(\*) AS course\_count

FROM Students

JOIN Enrollments ON Students.ID = Enrollments.StudentID

WHERE age > 18

GROUP BY name, age

HAVING COUNT(\*) > 1

ORDER BY age DESC

LIMIT 10;

This would:

* Select name, age, and number of enrollments
* From students joined with enrollments
* Where age is above 18
* Group results by name and age
* Only include those with more than one course
* Sort them by age (descending)
* And return only the first 10 results

1. Explain the role of clauses in SQL statements.

Ans. Absolutely! Clauses are the **building blocks** of SQL statements. They define **what data** to retrieve or manipulate, **where to get it from**, **how to filter it**, **how to sort or group it**, and so on.

**🎯 What Is a Clause in SQL?**

A **clause** is a **part of an SQL statement** that performs a specific task — kind of like a sentence part in grammar (subject, verb, etc.). Clauses can’t usually stand alone but work together in a full SQL statement.

**🔑 Common SQL Clauses and Their Roles:**

| **Clause** | **Role / Purpose** |
| --- | --- |
| SELECT | Specifies the columns (fields) to return from a query. |
| FROM | Indicates the table(s) from which to retrieve the data. |
| WHERE | Filters rows based on a condition. |
| GROUP BY | Groups rows that have the same values in specified columns. |
| HAVING | Filters groups created by GROUP BY (used with aggregate functions). |
| ORDER BY | Sorts the result set by one or more columns, ascending or descending. |
| LIMIT / TOP | Restricts the number of rows returned (depends on SQL dialect). |
| JOIN | Combines rows from two or more tables based on a related column. |

**📘 Example: Putting Clauses Together**

SELECT department, COUNT(\*) AS employee\_count

FROM Employees

WHERE status = 'Active'

GROUP BY department

HAVING COUNT(\*) > 5

ORDER BY employee\_count DESC

LIMIT 3;

**✅ This does:**

* SELECT: Choose department and count
* FROM: Look in the Employees table
* WHERE: Only include "Active" employees
* GROUP BY: Group data by department
* HAVING: Only keep groups with more than 5 people
* ORDER BY: Sort by number of employees, highest first
* LIMIT: Show top 3 results only

**3 . SQL Constraints**

**Theory Questions:**

1. What are constraints in SQL? List and explain the different types of constraints.

**Ans.** Great topic! **Constraints in SQL** are rules enforced on data in a table to ensure **accuracy, consistency, and integrity** of the data. They prevent invalid data from being entered into the database.

**🔐 What Are Constraints in SQL?**

Constraints are set at the **column** or **table** level and are used to:

* Restrict the type of data that can be stored
* Enforce relationships between tables
* Maintain data validity

**📋 Types of SQL Constraints**

Here’s a list of common constraints, along with what they do:

**1. PRIMARY KEY**

* Ensures each row in a table is uniquely identifiable.
* Combines **NOT NULL** and **UNIQUE**.
* A table can have only **one primary key**.

Example:

CREATE TABLE Students (

ID INT PRIMARY KEY,

Name VARCHAR(50)

);

**2. FOREIGN KEY**

* Enforces a link between two tables.
* Ensures the value in one table matches a value in another (referential integrity).

Example:

CREATE TABLE Enrollments (

StudentID INT,

FOREIGN KEY (StudentID) REFERENCES Students(ID)

);

**3. UNIQUE**

* Ensures all values in a column are different (but can be NULL).
* Can be applied to more than one column.

Example:

CREATE TABLE Users (

Email VARCHAR(100) UNIQUE

);

**4. NOT NULL**

* Prevents a column from having a NULL value.

Example:

CREATE TABLE Employees (

Name VARCHAR(50) NOT NULL

);

**5. CHECK**

* Ensures values meet a specific condition.

Example:

CREATE TABLE Products (

Price DECIMAL CHECK (Price > 0)

);

**6. DEFAULT**

* Sets a default value for a column if no value is provided.

Example:

CREATE TABLE Orders (

Status VARCHAR(20) DEFAULT 'Pending'

);

**⚖️ Why Use Constraints?**

* To **enforce business rules** at the database level.
* To **prevent data corruption** or mistakes.
* To ensure **data relationships** are consistent.

1. How do PRIMARY KEY and FOREIGN KEY constraints differ?

Ans. Awesome question — this is key (pun intended 😉) to understanding **relational databases**!

**🔑 PRIMARY KEY vs. FOREIGN KEY: Key Differences**

| **Feature** | **PRIMARY KEY** | **FOREIGN KEY** |
| --- | --- | --- |
| **Purpose** | Uniquely identifies each record in a table | Establishes a link between two tables |
| **Uniqueness** | Must be unique and **cannot contain NULLs** | Can contain duplicate values and **NULLs** (unless restricted) |
| **Table Location** | Defined **in the same table** it applies to | Refers to the **primary key in another table** |
| **Enforces** | **Entity integrity** (each row is unique) | **Referential integrity** (valid links between tables) |
| **Number Allowed** | Only **one** primary key per table (can be composite) | Can have **multiple foreign keys** in a table |

**🔹 PRIMARY KEY Example:**

CREATE TABLE Students (

StudentID INT PRIMARY KEY,

Name VARCHAR(100)

);

* StudentID uniquely identifies each student.
* No two students can have the same ID.
* The value **cannot be NULL**.

**🔹 FOREIGN KEY Example:**

CREATE TABLE Enrollments (

EnrollmentID INT PRIMARY KEY,

StudentID INT,

FOREIGN KEY (StudentID) REFERENCES Students(StudentID)

);

* StudentID in Enrollments links to StudentID in Students.
* It ensures every StudentID in Enrollments exists in Students.

**🎯 Analogy:**

Think of **PRIMARY KEY** as the **ID card** of a person (unique and mandatory), while a **FOREIGN KEY** is like referencing someone’s ID number to look them up in a database — it links the record but doesn't define it.

1. What is the role of NOT NULL and UNIQUE constraints?

Ans. Great question again! Both **NOT NULL** and **UNIQUE** are **column-level constraints** in SQL, and they play important roles in maintaining **data integrity** — but in different ways.

**🔒 1. NOT NULL Constraint**

**✅ Purpose:**

* Ensures that a **column cannot have a NULL value**.
* Forces users to **provide a value** when inserting or updating data.

**📌 Use Case:**

Use NOT NULL when a field is **mandatory**, like an email, username, or date of birth.

**🧪 Example:**

CREATE TABLE Employees (

ID INT,

Name VARCHAR(50) NOT NULL

);

* Now, every employee **must** have a name.

**🧠 2. UNIQUE Constraint**

**✅ Purpose:**

* Ensures that **all values in a column are different** (no duplicates).
* Can apply to **one or more columns**.
* Allows **one NULL** (depending on DBMS).

**📌 Use Case:**

Use UNIQUE when a field needs to be **distinct**, like an email address or employee number.

**🧪 Example:**

CREATE TABLE Users (

UserID INT PRIMARY KEY,

Email VARCHAR(100) UNIQUE

);

* No two users can have the same email.

**⚖️ Comparison Table:**

| **Constraint** | **Prevents** | **Allows NULLs?** | **Enforces Uniqueness?** |
| --- | --- | --- | --- |
| NOT NULL | Missing values | ❌ No | ❌ No |
| UNIQUE | Duplicate values | ✅ Yes (usually 1) | ✅ Yes |

**4. Main SQL Commands and Sub-commands (DDL)**

**Theory Questions:**