**DBMS THEORY**

**1.Introduction to SQL**

**SQL (Structured Query Language)** is a standardized programming language specifically designed for managing and manipulating relational databases. It allows users to perform tasks such as querying data, updating records, creating and modifying tables, and managing database access.

**Theory Questions**

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

**SQL** is a language used to communicate with databases. It is essential because:

* It allows users to **create**, **read**, **update**, and **delete** data (commonly known as **CRUD** operations).
* It enables **data retrieval** through complex queries.
* It ensures **data integrity** and **security**.
* It supports **transaction management** and **concurrency control**.

**2. Explain the difference between DBMS and RDBMS.**

| **Feature** | **DBMS (Database Management System)** | **RDBMS (Relational Database Management System)** |
| --- | --- | --- |
| **Data Storage** | Stores data as files or in a hierarchical format | Stores data in **tables** with rows and columns |
| **Relationships** | No concept of relationships between data | Supports **relationships** using **foreign keys** |
| **Normalization** | Does not necessarily support normalization | Supports **data normalization** |
| **Examples** | File systems, XML databases | MySQL, PostgreSQL, Oracle, SQL Server |

**3. Describe the role of SQL in managing relational databases.**

SQL plays a central role by:

* **Defining data structures** (using CREATE, ALTER commands)
* **Manipulating data** (using INSERT, UPDATE, DELETE)
* **Querying data** (using SELECT)
* **Controlling access** (using GRANT, REVOKE)
* **Ensuring data consistency and integrity** (using constraints like PRIMARY KEY, FOREIGN KEY, CHECK)

**4. What are the key features of SQL?**

Key features include:

* **Data Querying** with SELECT
* **Data Manipulation** with INSERT, UPDATE, DELETE
* **Data Definition** using DDL commands (CREATE, ALTER, DROP)
* **Data Control** through access permissions (GRANT, REVOKE)
* **Transaction Control** using COMMIT, ROLLBACK, SAVEPOINT
* **Portability** across different relational database systems
* **Integration** with other programming languages

**2. SQL Syntax**

SQL syntax refers to the rules that define the structure and format of SQL statements. Proper syntax ensures that commands are interpreted correctly by the database system.

**Theory Questions**

**1. What are the basic components of SQL syntax?**

The basic components of SQL syntax include:

* **Keywords**: Reserved words like SELECT, FROM, WHERE, INSERT, UPDATE, etc.
* **Identifiers**: Names of database objects like tables (employees), columns (salary), databases, etc.
* **Operators**: Used in conditions, e.g., =, <, >, AND, OR, LIKE, etc.
* **Literals**: Constant values like 'John', 100, '2025-08-03'
* **Clauses**: Logical segments of an SQL statement (e.g., WHERE, ORDER BY, GROUP BY)
* **Expressions**: Combinations of columns, literals, and operators (e.g., salary \* 0.1)
* **Comments**: Explanatory notes ignored by the SQL engine (e.g., -- This is a comment)

**2. Write the general structure of an SQL SELECT statement.**

sql

CopyEdit

SELECT column1, column2, ...

FROM table\_name

WHERE condition

GROUP BY column

HAVING condition

ORDER BY column [ASC|DESC];

**Explanation:**

* SELECT: Specifies the columns to retrieve.
* FROM: Indicates the table to query.
* WHERE: Filters rows based on a condition.
* GROUP BY: Groups rows sharing the same values.
* HAVING: Filters groups based on a condition.
* ORDER BY: Sorts the result.

**3. Explain the role of clauses in SQL statements.**

**Clauses** are essential parts of SQL statements that define their behavior and structure. Each clause has a specific role:

* SELECT: Defines which columns/data to retrieve.
* FROM: Specifies the data source (tables or views).
* WHERE: Filters individual rows before grouping or output.
* GROUP BY: Aggregates rows based on column values.
* HAVING: Filters groups created by GROUP BY.
* ORDER BY: Sorts the result set.
* JOIN clauses: Combine rows from two or more tables based on related columns.

**3. SQL Constraints**

SQL constraints are rules applied to columns in a table to ensure **data accuracy**, **consistency**, and **integrity** within the database.

**Theory Questions**

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

**Constraints** are rules enforced on table columns to limit the type of data that can be inserted. They help maintain the correctness and reliability of the data.

**Common types of SQL constraints:**

1. **PRIMARY KEY**
   * Uniquely identifies each row in a table.
   * It **cannot contain NULL** values and must be **unique**.
2. **FOREIGN KEY**
   * Links two tables by referencing the **primary key** in another table.
   * Ensures **referential integrity** between records.
3. **NOT NULL**
   * Prevents a column from having NULL values.
   * Ensures that data is always present in that column.
4. **UNIQUE**
   * Ensures that all values in a column are **distinct** (but can include one NULL).
5. **CHECK**
   * Validates that values in a column meet a specific **condition**.
   * Example: CHECK (age >= 18)
6. **DEFAULT**
   * Sets a **default value** for a column if no value is provided during insert.

**2. How do PRIMARY KEY and FOREIGN KEY constraints differ?**

| **Aspect** | **PRIMARY KEY** | **FOREIGN KEY** |
| --- | --- | --- |
| **Purpose** | Uniquely identifies rows in the same table | Creates a relationship between two tables |
| **Uniqueness** | Must be **unique** | Can contain duplicate values |
| **NULL values** | **Not allowed** | **Allowed**, unless explicitly restricted |
| **Referenced** | Can’t reference another table | **References** the primary key of another table |
| **Example** | id INT PRIMARY KEY | user\_id INT FOREIGN KEY REFERENCES users(id) |

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

* **NOT NULL**
  + Ensures that a column **must contain a value**.
  + Used when data is **mandatory** and cannot be left empty.
* **UNIQUE**
  + Ensures that all values in a column are **distinct**.
  + Used to prevent **duplicate entries** (e.g., email addresses).
  + Unlike PRIMARY KEY, it **allows one NULL** value unless combined with NOT NULL.

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

**Theory Questions**

**1. Define the SQL Data Definition Language (DDL).**

**DDL (Data Definition Language)** is a subset of SQL used to **define and manage database structure** such as tables, schemas, and indexes. It involves commands that **create**, **modify**, or **delete** database objects.

**Common DDL commands include:**

* CREATE: To create new tables, views, or databases
* ALTER: To modify existing structures
* DROP: To delete tables or databases
* TRUNCATE: To remove all records from a table quickly, without logging each row deletion

DDL changes the structure of the database and is usually **auto-committed**, meaning changes are permanent.

**2. Explain the CREATE command and its syntax.**

The CREATE command is used to **create new database objects**, most commonly **tables**.

**Syntax for creating a table:**

sql

CopyEdit

CREATE TABLE table\_name (

column1 datatype [constraint],

column2 datatype [constraint],

...

);

**Example:**

sql

CopyEdit

CREATE TABLE employees (

emp\_id INT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE,

hire\_date DATE DEFAULT CURRENT\_DATE,

salary DECIMAL(10, 2) CHECK (salary > 0)

);

**3. What is the purpose of specifying data types and constraints during table creation?**

**Specifying data types and constraints serves several important purposes:**

* **Data types:**
  + Define the **kind of data** that can be stored (e.g., INT, VARCHAR, DATE)
  + Ensure **data consistency** (e.g., storing numbers where numbers are expected)
  + Improve **storage efficiency** and performance
* **Constraints:**
  + Enforce **rules and data integrity**
  + Prevent **invalid data entry** (e.g., NOT NULL, CHECK)
  + Ensure **uniqueness** (UNIQUE, PRIMARY KEY)
  + Maintain **relationships** between tables (FOREIGN KEY)

**5. ALTER Command**

**Theory Questions**

**1. What is the use of the ALTER command in SQL?**

The **ALTER** command in SQL is used to **modify the structure** of an existing table **without losing the data**.

**Key uses of ALTER:**

* Add new columns
* Modify existing columns (data type, constraints)
* Drop (delete) columns
* Add or drop constraints (e.g., PRIMARY KEY, UNIQUE)
* Rename tables or columns (in some DBMSs)

ALTER helps in **adapting the table structure** as requirements change over time.

**2. How can you add, modify, and drop columns from a table using ALTER?**

Here are the main operations using ALTER:

**To ADD a new column:**

sql

CopyEdit

ALTER TABLE table\_name

ADD column\_name datatype [constraint];

**Example:**

sql

CopyEdit

ALTER TABLE employees

ADD department VARCHAR(50);

**To MODIFY an existing column:**

sql

CopyEdit

ALTER TABLE table\_name

MODIFY column\_name new\_datatype [new\_constraint];

**Example (MySQL syntax):**

sql

CopyEdit

ALTER TABLE employees

MODIFY salary DECIMAL(12, 2) NOT NULL;

Note: In SQL Server, use ALTER COLUMN instead of MODIFY.

**To DROP a column:**

sql

CopyEdit

ALTER TABLE table\_name

DROP COLUMN column\_name;

**Example:**

sql

CopyEdit

ALTER TABLE employees

DROP COLUMN department;

**6. DROP Command**

**Theory Questions**

**1. What is the function of the DROP command in SQL?**

The **DROP** command is used to **completely remove** a database object such as a:

* Table
* View
* Index
* Database
* Stored Procedure (depending on the RDBMS)

**Key point:**  
Once an object is dropped, **all associated data and structure are permanently deleted**.

**Syntax Example:**

sql

CopyEdit

DROP TABLE table\_name;

**2. What are the implications of dropping a table from a database?**

Dropping a table has **significant consequences**, including:

* **Permanent deletion** of:
  + The table structure
  + All rows (data) stored in the table
  + Associated constraints (e.g., primary key, foreign keys)
  + Indexes and triggers related to the table
* **Irreversible**: The action **cannot be undone** unless you have a backup.
* **Dependency issues**: If other tables or views reference the dropped table (e.g., through foreign keys), the **DROP command may fail** unless you first remove or update those dependencies.
* **Storage freed**: Dropping a table **releases storage space** used by that table.

**7. Data Manipulation Language (DML)**

**Theory Questions**

**1. Define the INSERT, UPDATE, and DELETE commands in SQL.**

**INSERT**  
The INSERT command is used to **add new records (rows)** into a table.

**Syntax:**

sql

CopyEdit

INSERT INTO table\_name (column1, column2, ...)

VALUES (value1, value2, ...);

**Example:**

sql

CopyEdit

INSERT INTO employees (emp\_id, name, salary)

VALUES (101, 'Alice', 50000);

**UPDATE**  
The UPDATE command is used to **modify existing records** in a table.

**Syntax:**

sql

CopyEdit

UPDATE table\_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

**Example:**

sql

CopyEdit

UPDATE employees

SET salary = 55000

WHERE emp\_id = 101;

**DELETE**  
The DELETE command is used to **remove existing records** from a table.

**Syntax:**

sql

CopyEdit

DELETE FROM table\_name

WHERE condition;

**Example:**

sql

CopyEdit

DELETE FROM employees

WHERE emp\_id = 101;

Be careful: Without a WHERE clause, DELETE will remove **all rows** in the table.

**2. What is the importance of the WHERE clause in UPDATE and DELETE operations?**

The **WHERE clause** is **crucial** in UPDATE and DELETE statements because it **filters** which rows are affected by the operation.

**Importance:**

* Prevents **accidental updates or deletions** of all rows
* Allows **targeted changes** based on conditions (e.g., a specific emp\_id)
* Helps maintain **data integrity** and avoids irreversible data loss

**Examples:**

Without WHERE (dangerous!):

sql

CopyEdit

DELETE FROM employees;

-- Deletes all rows!

With WHERE (safe):

sql

CopyEdit

DELETE FROM employees

WHERE emp\_id = 101;

**8. Data Query Language (DQL)**

**Theory Questions**

**1. What is the SELECT statement, and how is it used to query data?**

The **SELECT** statement is used to **retrieve data** from one or more tables in a database.

**Basic Syntax:**

sql

CopyEdit

SELECT column1, column2, ...

FROM table\_name;

**Key features:**

* Can return **specific columns** or use \* to return **all columns**
* Can include **conditions, sorting, grouping, and joins**
* Results are displayed as a **result set** (table format)

**Example:**

sql

CopyEdit

SELECT name, salary

FROM employees;

Retrieves the name and salary columns from the employees table.

**2. Explain the use of the ORDER BY and WHERE clauses in SQL queries.**

**WHERE Clause**

* Filters the rows **based on a condition**.
* Only rows that satisfy the condition are returned.
* Used with SELECT, UPDATE, and DELETE.

**Syntax:**

sql

CopyEdit

SELECT column1, column2

FROM table\_name

WHERE condition;

**Example:**

sql

CopyEdit

SELECT name

FROM employees

WHERE salary > 50000;

Returns only employees whose salary is greater than 50,000.

**ORDER BY Clause**

* Sorts the result set **in ascending (ASC) or descending (DESC) order**.
* Default is ascending.

**Syntax:**

sql

CopyEdit

SELECT column1, column2

FROM table\_name

ORDER BY column1 [ASC|DESC];

**Example:**

sql

CopyEdit

SELECT name, salary

FROM employees

ORDER BY salary DESC;

**9. Data Control Language (DCL)**

**Theory Questions**

**1. What is the purpose of GRANT and REVOKE in SQL?**

**GRANT**  
The GRANT command is used to **give specific privileges** to users or roles on database objects like tables, views, or procedures.

**REVOKE**  
The REVOKE command is used to **take back previously granted privileges** from users or roles.

These commands are essential for implementing **database security and access control**.

**2. How do you manage privileges using these commands?**

**Using GRANT:**

**Syntax:**

sql

CopyEdit

GRANT privilege(s)

ON object\_name

TO user\_or\_role;

**Example:**

sql

CopyEdit

GRANT SELECT, INSERT

ON employees

TO user1;

This gives user1 permission to select and insert data into the employees table.

**Using REVOKE:**

**Syntax:**

sql

CopyEdit

REVOKE privilege(s)

ON object\_name

FROM user\_or\_role;

**Example:**

sql

CopyEdit

REVOKE INSERT

ON employees

FROM user1;

This removes the INSERT privilege from user1 on the employees table.

**Common Privileges You Can Grant/Revoke:**

* SELECT: Read data
* INSERT: Add new data
* UPDATE: Modify data
* DELETE: Remove data
* ALL PRIVILEGES: Grants all possible permissions

**Summary:**

| **Command** | **Purpose** | **Used To...** |
| --- | --- | --- |
| GRANT | Give permissions | Allow users to access/modify database data |
| REVOKE | Remove permissions | Restrict users from accessing/modifying data |

**10. Transaction Control Language (TCL)**

**Theory Questions**

**1. What is the purpose of the COMMIT and ROLLBACK commands in SQL?**

**COMMIT**

* **Saves** all the changes made by the current transaction **permanently** to the database.
* After a COMMIT, changes cannot be undone.
* Typically used after successful operations.

**Example:**

sql

CopyEdit

UPDATE accounts SET balance = balance - 100 WHERE account\_id = 1;

UPDATE accounts SET balance = balance + 100 WHERE account\_id = 2;

COMMIT;

**ROLLBACK**

* **Undoes** all changes made in the current transaction.
* Returns the database to the last committed state.
* Used when an error or issue occurs during a transaction.

**Example:**

sql

CopyEdit

UPDATE accounts SET balance = balance - 100 WHERE account\_id = 1;

-- Error occurs here

ROLLBACK;

These commands help maintain **data accuracy** by preventing partial or inconsistent updates.

**2. Explain how transactions are managed in SQL databases.**

Transactions in SQL are managed based on the **ACID properties**:

| **Property** | **Description** |
| --- | --- |
| **Atomicity** | A transaction is **all or nothing** — either all operations succeed or none. |
| **Consistency** | Transactions take the database from one **valid state to another**. |
| **Isolation** | Concurrent transactions are **independent** and don’t interfere with each other. |
| **Durability** | Once committed, changes are **permanent**, even if a system crash occurs. |

**Transaction Lifecycle:**

1. **BEGIN** a transaction (implicitly or using BEGIN TRANSACTION)
2. Perform operations (INSERT, UPDATE, DELETE)
3. If successful → **COMMIT**
4. If error → **ROLLBACK**

**Optional TCL Commands:**

**SAVEPOINT**

* Sets a savepoint within a transaction to allow partial rollback.

sql

CopyEdit

SAVEPOINT save1;

**ROLLBACK TO savepoint\_name**

* Undoes actions back to a specific savepoint.

**11. SQL Joins**

**Theory Questions**

**1. Explain the concept of JOIN in SQL. What is the difference between INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN?**

**What is a JOIN?**  
A **JOIN** in SQL is used to **combine rows from two or more tables** based on a **related column**, usually a primary key in one table and a foreign key in another.

**Types of SQL Joins:**

| **Join Type** | **Description** |
| --- | --- |
| **INNER JOIN** | Returns **only matching rows** from both tables based on the join condition. |
| **LEFT JOIN** | Returns **all rows from the left table** and matching rows from the right table. |
| **RIGHT JOIN** | Returns **all rows from the right table** and matching rows from the left table. |
| **FULL OUTER JOIN** | Returns **all rows when there is a match in either table**; unmatched rows contain NULLs. |

**Example:**

Consider two tables:

**Employees**

| **emp\_id** | **name** | **dept\_id** |
| --- | --- | --- |
| 1 | Alice | 10 |
| 2 | Bob | 20 |
| 3 | Carol | NULL |

**Departments**

| **dept\_id** | **dept\_name** |
| --- | --- |
| 10 | HR |
| 30 | IT |

**INNER JOIN**:

sql

CopyEdit

SELECT e.name, d.dept\_name

FROM employees e

INNER JOIN departments d ON e.dept\_id = d.dept\_id;

Returns: Only rows where dept\_id matches in both tables (e.g., Alice → HR)

**LEFT JOIN**:

sql

CopyEdit

SELECT e.name, d.dept\_name

FROM employees e

LEFT JOIN departments d ON e.dept\_id = d.dept\_id;

Returns: All employees + matching departments  
 Carol will show NULL for dept\_name

**RIGHT JOIN**:

sql

CopyEdit

SELECT e.name, d.dept\_name

FROM employees e

RIGHT JOIN departments d ON e.dept\_id = d.dept\_id;

Returns: All departments + matching employees  
 IT department will show NULL for employee if no match

**FULL OUTER JOIN** *(if supported by your DBMS)*:

sql

CopyEdit

SELECT e.name, d.dept\_name

FROM employees e

FULL OUTER JOIN departments d ON e.dept\_id = d.dept\_id;

Returns: All data from both tables  
 Missing matches will appear as NULLs

**2. How are joins used to combine data from multiple tables?**

Joins are used to **logically link related data** across tables by:

* Using **primary key–foreign key relationships**
* Allowing you to retrieve **comprehensive information** that’s distributed across tables
* Reducing data redundancy by normalizing data into multiple tables

**Example Use Case:**  
To find the **employee names and their department names**, you'd join the employees and departments tables using dept\_id.

sql

CopyEdit

SELECT e.name, d.dept\_name

FROM employees e

JOIN departments d ON e.dept\_id = d.dept\_id;

**12. SQL GROUP BY**

**Theory Questions**

**1. What is the GROUP BY clause in SQL? How is it used with aggregate functions?**

**GROUP BY** is a clause in SQL used to **group rows that have the same values** in specified columns.  
It is most commonly used **with aggregate functions** like:

* COUNT()
* SUM()
* AVG()
* MAX()
* MIN()

**🔹 Purpose:**

* Summarizes data **per group** (e.g., total sales per region, average salary per department)

**🔹 Syntax:**

sql

CopyEdit

SELECT column\_name, AGGREGATE\_FUNCTION(column)

FROM table\_name

GROUP BY column\_name;

**Example:**

sql

CopyEdit

SELECT dept\_id, AVG(salary)

FROM employees

GROUP BY dept\_id;

This query returns the **average salary for each department**.

**🔹 HAVING Clause (Optional):**

* Used to filter **groups**, not individual rows (unlike WHERE)

**Example with HAVING:**

sql

CopyEdit

SELECT dept\_id, COUNT(\*) AS num\_employees

FROM employees

GROUP BY dept\_id

HAVING COUNT(\*) > 5;

Returns only departments with **more than 5 employees**.

**2. Explain the difference between GROUP BY and ORDER BY.**

| **Feature** | **GROUP BY** | **ORDER BY** |
| --- | --- | --- |
| **Purpose** | Groups rows with the same values | Sorts the result set |
| **Used With** | Often used with **aggregate functions** | Used for **sorting**, with or without aggregates |
| **Affects Output** | Summarizes data into **fewer rows** (grouped results) | Does **not change row content**, just order |
| **Clause Order** | Appears **before** ORDER BY in the query | Appears **last** in the query |

**🔹 Example showing both:**

sql

CopyEdit

SELECT dept\_id, COUNT(\*) AS total

FROM employees

GROUP BY dept\_id

ORDER BY total DESC;

**13. SQL Stored Procedure**

**Theory Questions**

**1. What is a stored procedure in SQL, and how does it differ from a standard SQL query?**

**Stored Procedure**  
A **stored procedure** is a **named block of SQL code** that is **stored in the database** and can be **executed (called) as needed**. It may contain SQL statements like SELECT, INSERT, UPDATE, control-of-flow logic (IF, WHILE), variables, parameters, and error handling.

**🔹 Key Differences:**

| **Feature** | **Stored Procedure** | **Standard SQL Query** |
| --- | --- | --- |
| **Persistence** | Stored **in the database** for reuse | Typed and executed **manually or ad hoc** |
| **Reusability** | Can be called **multiple times** | Executed once unless manually repeated |
| **Modularity** | Supports **parameters, logic, loops, and conditions** | Usually a single-purpose statement |
| **Security** | Access can be controlled via **permissions** | Each query must be authorized separately |

**Simple Example:**

sql

CopyEdit

CREATE PROCEDURE GetEmployeeByDept (@DeptID INT)

AS

BEGIN

SELECT name, salary FROM employees

WHERE dept\_id = @DeptID;

END;

To execute:

sql

CopyEdit

EXEC GetEmployeeByDept 10;

**2. Explain the advantages of using stored procedures**

Stored procedures offer several important **advantages**:

**🔹 1. Reusability**

* Code is written once and reused many times.
* Easy to maintain and update in one place.

**🔹 2. Performance**

* Stored procedures are **precompiled and cached** by the database engine, resulting in faster execution compared to sending individual queries every time.

**🔹 3. Security**

* Can restrict user access to underlying tables by granting access to procedures instead.
* Users execute the logic without needing full table permissions.

**🔹 4. Maintainability**

* Complex operations can be organized into manageable, modular blocks.
* Easier to debug and audit.

**🔹 5. Reduced Network Traffic**

* Multiple SQL statements can be bundled and executed in one call, reducing round-trips between application and server.

**14. SQL View**

**Theory Questions**

**1. What is a view in SQL, and how is it different from a table?**

**View**  
A **view** in SQL is a **virtual table** that is based on the result of a **SELECT query**. It does **not store data** itself but displays data stored in one or more underlying tables.

**🔹 Key Differences:**

| **Feature** | **View** | **Table** |
| --- | --- | --- |
| **Storage** | Virtual, stores **no data** | Physically stores data |
| **Creation** | Created using a SELECT statement (CREATE VIEW) | Created with CREATE TABLE |
| **Modifiability** | Sometimes read-only, especially if complex | Fully updatable |
| **Performance** | Executes query at runtime | Faster for frequent access |
| **Use Case** | Simplify queries, restrict access, abstract logic | Store and manage persistent data |

**Example:**

sql

CopyEdit

CREATE VIEW high\_salary\_employees AS

SELECT name, salary

FROM employees

WHERE salary > 60000;

To use the view:

sql

CopyEdit

SELECT \* FROM high\_salary\_employees;

**2. Explain the advantages of using views in SQL databases**

**🔹 1. Simplifies Complex Queries**

* Views encapsulate complex joins or filtering logic into a single query interface.
* Makes SQL easier to read and write.

**🔹 2. Enhances Security**

* Restricts user access to specific columns or rows.
* You can allow users to query a view **without giving them access to the base table**.

**🔹 3. Provides Data Abstraction**

* Users interact with a logical representation of the data, without needing to know the underlying structure.

**🔹 4. Encourages Reusability and Maintainability**

* Centralizes business logic in one place.
* Changes in the view definition automatically reflect wherever the view is used.

**🔹 5. Supports Logical Independence**

* Applications can rely on views instead of tables, so if the base table changes, the view can be updated to preserve compatibility.

**15. SQL Triggers**

**Theory Questions**

**1. What is a trigger in SQL? Describe its types and when they are used.**

**Trigger**  
A **trigger** is a special type of stored procedure in SQL that **automatically executes** (or “fires”) **in response to specific events** on a table or view — such as INSERT, UPDATE, or DELETE.

Triggers are used to:

* Enforce business rules automatically
* Maintain audit trails (e.g., logging changes)
* Prevent invalid transactions
* Automatically update or validate related data

**🔹 Types of Triggers (based on timing and event):**

| **Type** | **Description** |
| --- | --- |
| **BEFORE Trigger** | Executes **before** the triggering event (not supported in all DBMS) |
| **AFTER Trigger** | Executes **after** the triggering event is completed |
| **INSTEAD OF Trigger** | Replaces the standard action with custom logic (mainly used on views) |

**🔹 Types by event:**

* **INSERT Trigger** – Executes when a new row is inserted
* **UPDATE Trigger** – Executes when a row is modified
* **DELETE Trigger** – Executes when a row is deleted

Not all DBMSs support BEFORE or INSTEAD OF triggers. For example, SQL Server supports AFTER and INSTEAD OF, but not BEFORE.

**2. Explain the difference between INSERT, UPDATE, and DELETE triggers.**

| **Trigger Type** | **When It Fires** | **Use Case Example** |
| --- | --- | --- |
| **INSERT** | When a new record is added to a table | Log new user sign-ups to an audit table |
| **UPDATE** | When an existing record is modified | Validate new values or store old vs. new values |
| **DELETE** | When a record is removed from a table | Archive deleted data or prevent deletion if restricted |

**Example of an AFTER INSERT trigger (MySQL/PostgreSQL style):**

sql

CopyEdit

CREATE TRIGGER log\_new\_employee

AFTER INSERT ON employees

FOR EACH ROW

BEGIN

INSERT INTO audit\_log (action, emp\_name, action\_time)

VALUES ('INSERT', NEW.name, NOW());

END;

This trigger logs every new employee added to the employees table.

**16. Introduction to PL/SQL**

**Theory Questions**

**1. What is PL/SQL, and how does it extend SQL's capabilities?**

**PL/SQL** (Procedural Language/SQL) is **Oracle's procedural extension to SQL**.  
It combines the power of **SQL for data manipulation** with the **control structures** of a **procedural language** (like loops, conditionals, and variables).

**🔹 How PL/SQL extends SQL:**

| **Feature** | **SQL** | **PL/SQL Extension** |
| --- | --- | --- |
| Only supports queries/commands | Basic data access and DML | Adds programming logic |
| No procedural flow | Cannot use loops or conditions | Supports IF, LOOP, CASE, etc. |
| No error handling | Fails on runtime errors | Supports structured exception handling |
| No modularization | No blocks, functions, or procedures | Supports procedures, functions, packages |

**Example: A simple PL/SQL block:**

sql

CopyEdit

BEGIN

UPDATE employees

SET salary = salary \* 1.10

WHERE dept\_id = 10;

END;

✔ This block can be wrapped with error handling, stored as a procedure, and reused.

**2. List and explain the benefits of using PL/SQL.**

PL/SQL offers several important benefits:

**🔹 1. Procedural Capabilities**

* Allows use of **variables, loops, conditional logic** (IF, WHILE, FOR)
* Enables writing more **complex and dynamic logic**

**🔹 2. Modular Programming**

* Supports **procedures, functions, packages**, and **triggers**
* Encourages **code reuse and maintainability**

**🔹 3. Improved Performance**

* Reduces **network traffic** by sending entire logic blocks to the database server
* **SQL and procedural code** run together on the server

**🔹 4. Better Error Handling**

* Offers structured **exception handling** using BEGIN ... EXCEPTION ... END
* Helps handle runtime errors gracefully

**🔹 5. Tight Integration with SQL**

* You can embed SQL directly within PL/SQL, and it supports **SQL features natively**

**🔹 6. Security & Maintainability**

* Logic can be stored in **compiled form** as packages and procedures
* Access can be controlled via permissions, without exposing the raw SQL

**17. PL/SQL Control Structures**

**Theory Questions**

**1. What are control structures in PL/SQL? Explain the IF-THEN and LOOP control structures.**

**Control structures** in PL/SQL are programming constructs that allow developers to **control the flow of execution** based on conditions or repetitions. They make PL/SQL more powerful and flexible than standard SQL.

There are **three main categories** of control structures:

1. **Conditional statements** (IF, CASE)
2. **Loops** (LOOP, WHILE, FOR)
3. **Sequential control** (GOTO, EXIT, CONTINUE)

**🔹 IF-THEN Structure**

Used to execute a block of code **based on a condition**.

**Syntax:**

plsql

CopyEdit

IF condition THEN

-- statements

END IF;

**With ELSE and ELSIF:**

plsql

CopyEdit

IF condition1 THEN

-- statements

ELSIF condition2 THEN

-- statements

ELSE

-- statements

END IF;

**Example:**

plsql

CopyEdit

IF salary > 50000 THEN

bonus := 1000;

ELSE

bonus := 500;

END IF;

**🔹 LOOP Structures**

Used to **repeat** a block of code multiple times.

**BASIC LOOP** (runs indefinitely unless EXIT is used)

plsql

CopyEdit

LOOP

-- statements

EXIT WHEN condition;

END LOOP;

**WHILE LOOP**

plsql

CopyEdit

WHILE condition LOOP

-- statements

END LOOP;

**FOR LOOP**

plsql

CopyEdit

FOR i IN 1..5 LOOP

-- statements

END LOOP;

**2. How do control structures in PL/SQL help in writing complex queries?**

Control structures help write **more dynamic, intelligent, and maintainable logic** in PL/SQL by:

**🔹 Adding Decision Logic**

* Use IF-THEN or CASE to **branch logic** based on data values or business rules.
* Enables conditional execution of SQL operations.

**🔹 Automating Repetitive Tasks**

* Use LOOP, WHILE, and FOR to **process multiple rows or operations** efficiently.
* Ideal for batch updates, validations, or calculations.

**🔹 Error Handling Integration**

* Combine control structures with **exception handling** for safer, more robust code.

**🔹 Enhancing Readability & Modularity**

* Structured control makes procedural code easier to understand, debug, and maintain.

**🔹 Examples of Use Cases:**

* Automatically adjust employee salaries based on department and performance
* Loop through a list of customers and send personalized emails
* Perform conditional logging or validation before inserting data

**18. SQL Cursors**

**Theory Questions**

**1. What is a cursor in PL/SQL? Explain the difference between implicit and explicit cursors.**

**What is a Cursor?**  
A **cursor** in PL/SQL is a pointer that **allows row-by-row processing** of a result set returned by a query. It is used when you need to work with **multiple rows** returned by a SQL statement, especially in procedural logic.

**🔹 Types of Cursors:**

| **Cursor Type** | **Description** |
| --- | --- |
| **Implicit Cursor** | Automatically created by PL/SQL for single-row SQL operations like SELECT INTO, INSERT, UPDATE, or DELETE. |
| **Explicit Cursor** | Manually declared by the programmer to process **multiple rows** in a SELECT statement. Gives more control. |

**Implicit Cursor Example:**

plsql

CopyEdit

BEGIN

UPDATE employees

SET salary = salary \* 1.1

WHERE dept\_id = 10;

-- Uses SQL%ROWCOUNT to get affected rows

DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT || ' rows updated.');

END;

🔹 Implicit cursors are used by default and managed automatically by PL/SQL.

**Explicit Cursor Example:**

plsql

CopyEdit

DECLARE

CURSOR emp\_cursor IS

SELECT name, salary FROM employees WHERE dept\_id = 10;

emp\_name employees.name%TYPE;

emp\_salary employees.salary%TYPE;

BEGIN

OPEN emp\_cursor;

LOOP

FETCH emp\_cursor INTO emp\_name, emp\_salary;

EXIT WHEN emp\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(emp\_name || ' earns ' || emp\_salary);

END LOOP;

CLOSE emp\_cursor;

END;

🔹 Explicit cursors are ideal when you need to process each row **individually** with custom logic.

**2. When would you use an explicit cursor over an implicit one?**

Use an **explicit cursor** when:

* You need to **process multiple rows** one at a time.
* You want **fine control** over row retrieval (e.g., fetch, skip, conditional logic).
* You require **looping and conditional logic** for each row in a result set.
* You want to **use cursor attributes** (e.g., %FOUND, %NOTFOUND, %ROWCOUNT) in your logic.

**🔹 Summary:**

| **Feature** | **Implicit Cursor** | **Explicit Cursor** |
| --- | --- | --- |
| Control Level | Minimal (automatic) | Full control (manual open/fetch/close) |
| Row Processing | One row at a time (usually) | Multiple rows, one at a time |
| Use Case | Simple queries | Complex logic on multiple rows |
| Performance | Slightly faster for simple queries | More readable for complex, row-based logic |

**19. Rollback and Commit Savepoint**

**Theory Questions**

**1. Explain the concept of SAVEPOINT in transaction management. How do ROLLBACK and COMMIT interact with savepoints?**

**SAVEPOINT**  
A **SAVEPOINT** is a **marker or checkpoint within a database transaction** that allows partial rollback of the transaction to that point without rolling back the entire transaction.

**🔹 How it works:**

* During a transaction, you can set multiple **savepoints** to mark points where you might want to undo part of the work.
* If an error occurs after a savepoint, you can **ROLLBACK TO SAVEPOINT** to undo only part of the transaction since that savepoint.
* A **COMMIT** operation will finalize the entire transaction, including all savepoints.
* Rolling back without specifying a savepoint rolls back the **entire transaction**.

**🔹 Example:**

sql

CopyEdit

BEGIN TRANSACTION;

-- Step 1

UPDATE accounts SET balance = balance - 100 WHERE account\_id = 1;

SAVEPOINT deduct\_money;

-- Step 2

UPDATE accounts SET balance = balance + 100 WHERE account\_id = 2;

-- Oops! An error detected here

ROLLBACK TO SAVEPOINT deduct\_money; -- Undo step 2 but keep step 1

COMMIT; -- Commit changes made up to this point

**2. When is it useful to use savepoints in a database transaction?**

Savepoints are useful when:

* You have **complex transactions** with multiple steps and want the ability to **undo part of the work without aborting the entire transaction**.
* You want to **handle errors gracefully**, rolling back only the problematic part.
* You need **fine-grained control** over the transaction flow for **partial recovery**.
* You want to **improve efficiency** by avoiding unnecessary rollbacks and commits.

**🔹 Summary:**

| **Command** | **Effect** |
| --- | --- |
| SAVEPOINT name | Sets a savepoint within the transaction |
| ROLLBACK TO name | Rolls back to the savepoint, undoing changes after it |
| ROLLBACK | Rolls back the entire transaction |
| COMMIT | Finalizes all changes, savepoints become irrelevant |