***1. Introduction to SQL***

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

**SQL** (Structured Query Language) is a standard programming language used for managing and manipulating relational databases. It provides a way to interact with databases by allowing users to define, manage, and manipulate data stored in tables.

**Why SQL is Essential in Database Management:**

**1.Data Retrieval**: SQL allows users to query databases to retrieve specific data using SELECT statements. It's essential for retrieving, filtering, and organizing data in meaningful ways.

**2.Data Manipulation**: It provides commands for inserting (INSERT), updating (UPDATE), and deleting (DELETE) data, which is critical for maintaining the accuracy and relevance of the data stored.

**3.Data Definition**: SQL is used to define the structure of a database using commands like CREATE, ALTER, and DROP to create and modify tables, indexes, and relationships between data.

**4.Data Integrity**: SQL allows for enforcing data integrity through constraints like PRIMARY KEY, FOREIGN KEY, NOT NULL, and CHECK, ensuring that data remains consistent and accurate.

**5.Transaction Management**: SQL supports transaction control commands (BEGIN, COMMIT, ROLLBACK) to ensure that data changes are completed successfully and to maintain the consistency of the database.

**6.Security and Permissions**: It provides mechanisms for managing user access and privileges, ensuring that only authorized users can access or modify sensitive data.

**7.Standardization**: SQL is the standard query language for relational databases, making it widely supported across different database systems (like MySQL, PostgreSQL, Oracle, etc.). This standardization allows developers to work with multiple database systems without needing to learn a different query language for each one.

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

|  |  |  |
| --- | --- | --- |
| **Feature** | **DBMS** | **RDBMS** |
| **Data Storage Format** | Various formats (e.g., flat, hierarchical) | Relational (tables with rows and columns) |
| **Normalization** | Not necessarily supported | Supports normalization |
| **Relationships** | No built-in support for relationships | Supports relationships (primary/foreign keys) |
| **Transaction Management** | Basic or none | Full ACID transaction support |
| **Data Integrity** | Limited | Advanced integrity constraints (e.g., keys, checks) |
| **Scalability** | Less scalable | More scalable with optimizations |

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

SQL plays a crucial role in managing relational databases by providing a standardized way to:

**Query Data**: Retrieve specific data from tables using SELECT.

**Manipulate Data**: Insert, update, or delete data with INSERT, UPDATE, and DELETE.

**Define Structure**: Create and modify tables, indexes, and views with CREATE, ALTER, and DROP.

**Ensure Data Integrity**: Enforce data rules using constraints like PRIMARY KEY and FOREIGN KEY.

**Control Access**: Manage user permissions and security with GRANT and REVOKE.

**Manage Transactions**: Control changes to data with transaction commands like COMMIT and ROLLBACK.

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

The key features of SQL are:

**Data Retrieval**: Allows querying and retrieving data from databases using SELECT statements.

**Data Manipulation**: Enables inserting (INSERT), updating (UPDATE), and deleting (DELETE) data.

**Data Definition**: Supports creating, altering, and dropping database structures like tables and views (CREATE, ALTER, DROP).

**Data Control**: Manages access permissions and user roles with GRANT and REVOKE.

**Data Integrity**: Ensures data accuracy with constraints like PRIMARY KEY, FOREIGN KEY, NOT NULL.

**Transaction Management**: Supports transaction control using COMMIT, ROLLBACK, and SAVEPOINT.

**Support for Joins**: Combines data from multiple tables using INNER JOIN, LEFT JOIN, etc.

**Aggregation**: Allows grouping and summarizing data with functions like COUNT, SUM, AVG.

***2. SQL Syntax***

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

The basic components of SQL syntax are:

**Keywords**: Reserved words like SELECT, FROM, WHERE, JOIN, etc., that define the action or operation.

**Identifiers**: Names of tables, columns, and databases.

**Operators**: Symbols like =, >, <, AND, OR, used to perform comparisons and logical operations.

**Literals**: Fixed values like strings ('John'), numbers (100), or dates ('2025-01-01').

**Clauses**: Components like SELECT, FROM, WHERE, etc., that structure the query.

**Expressions**: Combinations of columns, operators, and literals used to produce values.

**Punctuation**: Symbols like commas (,), parentheses (()), and semicolons (;) used to structure the query.

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

The general structure of an SQL SELECT statement is designed to retrieve data from one or more tables in a relational database. The structure allows you to specify which columns you want to retrieve, where to get the data from, and how to filter, sort, or group the results.

Here’s the general syntax:

SELECT [DISTINCT] column1, column2, ..., column FROM table\_name

[JOIN other\_table ON join\_condition][WHERE condition][GROUP BY column1, column2, ...][HAVING condition][ORDER BY column1 [ASC|DESC], column2 [ASC|DESC], ...][LIMIT number\_of\_records];

Example:-

SELECT DISTINCT FirstName, LastName, Age FROM Employees

WHERE Age > 30 ORDER BY LastName DESC LIMIT 5;

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

SQL clauses are components of a SQL query that define how to interact with the database. Each clause serves a specific purpose:

**SELECT**: Specifies the columns to retrieve.

**FROM**: Defines the table(s) to retrieve data from.

**WHERE**: Filters records based on conditions.

**GROUP BY**: Groups rows for aggregation.

**HAVING**: Filters grouped data after aggregation.

**ORDER BY**: Sorts the result set.

**LIMIT**: Limits the number of rows returned.

**JOIN**: Combines data from multiple tables based on a related column.

**DISTINCT**: Removes duplicate rows.

**AS**: Provides aliases for columns or tables.

***3. SQL Constraints***

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

**Constraints in SQL** are rules applied to columns or tables to ensure the accuracy and integrity of the data in a database. They limit the type of data that can be stored in the database.

**Types of Constraints:**

**NOT NULL**: Ensures that a column cannot have a NULL value.

**Example**: CREATE TABLE Employees (ID INT NOT NULL, Name VARCHAR(50) NOT NULL);

**UNIQUE**: Ensures that all values in a column are unique.

**Example**: CREATE TABLE Employees (ID INT UNIQUE, Name VARCHAR(50));

**PRIMARY KEY**: Uniquely identifies each record in a table. Combines NOT NULL and UNIQUE.

**Example**: CREATE TABLE Employees (ID INT PRIMARY KEY, Name VARCHAR(50));

**FOREIGN KEY**: Ensures referential integrity by linking a column in one table to a PRIMARY KEY or UNIQUE column in another table.

**Example**: CREATE TABLE Orders (OrderID INT PRIMARY KEY, CustomerID INT, FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID));

**CHECK**: Ensures that the values in a column satisfy a specific condition.

**Example**: CREATE TABLE Employees (Age INT CHECK (Age >= 18));

**DEFAULT**: Provides a default value for a column when no value is specified.

**Example**: CREATE TABLE Employees (ID INT, Name VARCHAR(50), JoinDate DATE DEFAULT CURRENT\_DATE);

**INDEX**: Improves the speed of data retrieval by creating an index on one or more columns.

**Example**: CREATE INDEX idx\_name ON Employees(Name);

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

**PRIMARY KEY**:

Uniquely identifies each record in a table.

Ensures values are unique and cannot be NULL.

A table can have only one PRIMARY KEY.

**FOREIGN KEY**:

Creates a relationship between two tables.

Refers to the PRIMARY KEY or UNIQUE column in another table.

Can accept NULL values.

A table can have multiple FOREIGN KEY constraints.

**Key Difference**: A PRIMARY KEY ensures uniqueness within its own table, while a FOREIGN KEY ensures data integrity between two related tables.

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

**NOT NULL** constraint:

Ensures that a column cannot have NULL values.

Used to enforce that data must be provided for that column.

**UNIQUE** constraint:

Ensures that all values in a column (or a combination of columns) are unique across the table.

Allows NULL values, but ensures non-NULL values are distinct.

**Key Difference**: NOT NULL enforces that a column must have a value, while UNIQUE enforces that all values in a column are distinct.

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

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

**SQL Data Definition Language (DDL)** is a subset of SQL used to define and manage database structures. It includes commands to create, alter, and delete tables, schemas, and other database objects.

**Key DDL Commands:**

**CREATE**: Defines new database objects like tables, views, and indexes.

**ALTER**: Modifies existing database objects, such as adding or dropping columns.

**DROP**: Deletes database objects like tables or views.

**TRUNCATE**: Removes all records from a table without deleting the table structure.

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

The **CREATE** command in SQL is used to define and create new database objects like tables, views, and indexes.

**Syntax for Creating a Table:**

sql

CopyEdit

CREATE TABLE table\_name (

column1 datatype [constraints],

column2 datatype [constraints],

...

);

**Example:**

sql

CopyEdit

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR(50) NOT NULL,

Age INT,

Department VARCHAR(50)

);

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

**Data types** and **constraints** are specified during table creation to:

**1.Data Types**: Define the type of data a column can store (e.g., INT, VARCHAR, DATE). This ensures data consistency and optimizes storage.

**2.Constraints**: Enforce rules on the data to ensure integrity and correctness, such as:

**NOT NULL**: Ensures a column cannot have NULL values.

**PRIMARY KEY**: Ensures uniqueness and identifies records.

**FOREIGN KEY**: Maintains relationships between tables.

**UNIQUE**: Ensures all values in a column are distinct.

***5. ALTER Command***

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

The **ALTER** command in SQL is used to modify an existing database object, such as a table. It allows changes like adding, deleting, or modifying columns, or altering the structure of the table.

**Common uses of the ALTER command:**

**1.Add a Column**:

ALTER TABLE table\_name ADD column\_name datatype;

**2.Modify a Column**:

ALTER TABLE table\_name MODIFY column\_name new\_datatype;

**3.Drop a Column**:

ALTER TABLE table\_name DROP COLUMN column\_name;

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

You can use the **ALTER** command to add, modify, and drop columns from a table as follows:

**1.Add a Column**:

ALTER TABLE table\_name ADD column\_name datatype;

**2.Modify a Column**:

ALTER TABLE table\_name MODIFY column\_name new\_datatype;

**3.Drop a Column**:

ALTER TABLE table\_name DROP COLUMN column\_name;

***6. DROP Command***

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

The **DROP** command in SQL is used to delete an existing database object, such as a table, view, or index, permanently from the database.

**Syntax:**

**1.Drop a Table**:

DROP TABLE table\_name;

**2.Drop a View**:

DROP VIEW view\_name;

**3.Drop an Index**:

DROP INDEX index\_name;

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

Dropping a table from a database has the following implications:

**1.Data Loss**: All data stored in the table is permanently deleted and cannot be recovered.

**2.Structure Removal**: The table's structure, including columns, constraints, and indexes, is also removed.

**3.Impact on Relationships**: If the table is referenced by foreign keys in other tables, those relationships may be broken, leading to referential integrity issues.

**4.Irreversible Action**: The DROP command is permanent and cannot be undone, so it's important to ensure that data is backed up if needed.

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

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

**INSERT:**

**Purpose**: Adds new records (rows) to a table.

**Syntax**:

INSERT INTO table\_name (column1, column2, ...) VALUES (value1, value2, ...);

**UPDATE:**

**Purpose**: Modifies existing records in a table.

**Syntax**:

UPDATE table\_name SET column1 = value1, column2 = value2 WHERE condition;

**DELETE:**

**Purpose**: Removes records from a table.

**Syntax**:

DELETE FROM table\_name WHERE condition;

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

The **WHERE** clause is crucial in **UPDATE** and **DELETE** operations because it specifies the condition that determines which records to modify or delete.

**In UPDATE**: The WHERE clause ensures that only specific rows are updated, preventing unintended changes to all records.

Example: UPDATE Employees SET Salary = 5000 WHERE EmployeeID = 1;

**In DELETE**: The WHERE clause ensures that only specific rows are deleted, preventing the deletion of all records in the table.

Example: DELETE FROM Employees WHERE EmployeeID = 1;

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

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

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

**Syntax:**

SELECT column1, column2, ... FROM table\_name WHERE condition;

**SELECT**: Specifies the columns you want to retrieve.

**FROM**: Defines the table from which to query data.

**WHERE**: (Optional) Filters the data based on a condition.

**Example:**

SELECT Name, Age FROM Employees WHERE Age > 30;

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

**ORDER BY Clause:**

**Purpose**: Sorts the result set of a query based on one or more columns, either in ascending (ASC) or descending (DESC) order.

**Syntax**:

SELECT column1, column2 FROM table\_name ORDER BY column1 [ASC|DESC];

**WHERE Clause:**

**Purpose**: Filters the rows returned by a query based on a specified condition.

**Syntax**:

SELECT column1, column2 FROM table\_name WHERE condition;

**Example:**

SELECT Name, Age FROM Employees WHERE Age > 30 ORDER BY Name ASC;

**WHERE** filters employees older than 30.

**ORDER BY** sorts the result by Name in ascending order.

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

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

**GRANT** and **REVOKE** are used to manage permissions in SQL:

**GRANT:**

**Purpose**: Grants specific privileges (like SELECT, INSERT, UPDATE, etc.) to users or roles for database objects (tables, views, etc.).

**Syntax**:

GRANT privilege\_type ON object TO user;

**REVOKE:**

**Purpose**: Removes previously granted privileges from users or roles.

**Syntax**:

REVOKE privilege\_type ON object FROM user;

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

You manage privileges using the **GRANT** and **REVOKE** commands to control user access to database objects:

**Using GRANT:**

**Purpose**: Assign specific privileges (like SELECT, INSERT, UPDATE, DELETE) to users or roles for database objects.

**Example**:

GRANT SELECT, INSERT ON Employees TO user1;

This grants user1 permission to SELECT and INSERT data in the Employees table.

**Using REVOKE:**

**Purpose**: Remove previously granted privileges from users or roles.

**Example**:

REVOKE SELECT ON Employees FROM user1;

This removes user1's permission to SELECT data from the Employees table.

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

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

The **COMMIT** and **ROLLBACK** commands in SQL are used to manage transactions:

**COMMIT:**

**Purpose**: Saves all changes made during the current transaction to the database permanently.

**Usage**: Ensures that all modifications (like INSERT, UPDATE, DELETE) are finalized and stored.

COMMIT;

**ROLLBACK:**

**Purpose**: Undoes all changes made during the current transaction, reverting the database to its previous state.

**Usage**: Used to cancel a transaction if an error occurs or if changes should not be saved.

ROLLBACK;

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

Transactions in SQL databases are managed using **ACID properties** (Atomicity, Consistency, Isolation, Durability) to ensure reliable and consistent data operations.

**Transaction Management Process:**

**Begin Transaction**: The transaction starts automatically when the first SQL operation is executed, or you can explicitly use BEGIN TRANSACTION.

**Perform Operations**: SQL operations like INSERT, UPDATE, and DELETE are performed within the transaction.

**COMMIT**: If all operations are successful, use COMMIT to make the changes permanent.

**ROLLBACK**: If an error occurs, use ROLLBACK to undo all changes made during the transaction.

**Example:**

BEGIN TRANSACTION;

UPDATE Employees SET Salary = 5000 WHERE EmployeeID = 1;

COMMIT; -- or ROLLBACK in case of an error

***11. SQL Joins***

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

**JOIN in SQL:**

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

**Types of Joins:**

**1.INNER JOIN**:

**Purpose**: Returns rows that have matching values in both tables.

**Example**:

SELECT \* FROM table1 INNER JOIN table2 ON table1.id = table2.id;

**2.LEFT JOIN (or LEFT OUTER JOIN)**:

**Purpose**: Returns all rows from the left table and matching rows from the right table. If no match, NULL is returned for the right table.

**Example**:

SELECT \* FROM table1 LEFT JOIN table2 ON table1.id = table2.id;

**3.RIGHT JOIN (or RIGHT OUTER JOIN)**:

**Purpose**: Returns all rows from the right table and matching rows from the left table. If no match, NULL is returned for the left table.

**Example**:

SELECT \* FROM table1 RIGHT JOIN table2 ON table1.id = table2.id;

**4.FULL OUTER JOIN**:

**Purpose**: Returns all rows when there is a match in either the left or the right table. If no match, NULL is returned for the non-matching side.

**Example**:

SELECT \* FROM table1 FULL OUTER JOIN table2 ON table1.id = table2.id;

**Key Differences:**

**INNER JOIN**: Only matching rows.

**LEFT JOIN**: All rows from the left table, matched with right table.

**RIGHT JOIN**: All rows from the right table, matched with left table.

**FULL OUTER JOIN**: All rows from both tables, with NULL where no match exists.

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

Joins are used to combine data from multiple tables based on a related column, such as a foreign key or primary key. The relationship between the tables is defined using the **ON** clause.

**Steps to Combine Data:**

**1.Define Relationship**: Specify the common column between the tables (e.g., foreign key in one table, primary key in another).

**2.Use JOIN Type**: Choose the appropriate join type (INNER JOIN, LEFT JOIN, RIGHT JOIN, or FULL OUTER JOIN) based on the desired result.

**3.Write the Query**: Use SQL syntax to fetch data from the joined tables.

**Example:**

SELECT Employees.Name, Departments.DepartmentName FROM Employees

INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;

This query combines data from Employees and Departments by matching the DepartmentID column in both tables.

***12. SQL Group By***

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

The **GROUP BY** clause in SQL is used to group rows that have the same values in specified columns into summary rows, often with aggregate functions (like COUNT, SUM, AVG, MIN, MAX) applied to each group.

**Syntax:**

SELECT column1, AGGREGATE\_FUNCTION(column2)FROM table\_name GROUP BY column1;

**Example:**

SELECT DepartmentID, COUNT(\*) FROM Employees GROUP BY DepartmentID;

This query groups employees by DepartmentID and counts the number of employees in each department.

**How It Works with Aggregate Functions:**

**COUNT()**: Counts the number of rows in each group.

**SUM()**: Sums the values in each group.

**AVG()**: Calculates the average of values in each group.

**MIN() and MAX()**: Find the minimum and maximum values in each group.

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

**GROUP BY:**

**Purpose**: Groups rows that have the same values in specified columns into summary rows.

**Usage**: Often used with aggregate functions like COUNT, SUM, AVG, etc.

**Example**:

SELECT DepartmentID, COUNT(\*) FROM Employees GROUP BY DepartmentID;

Groups employees by DepartmentID and counts the number of employees in each department.

**ORDER BY:**

**Purpose**: Sorts the result set by one or more columns, in ascending (ASC) or descending (DESC) order.

**Usage**: Used to arrange the results of a query in a specific order.

**Example**:

SELECT \* FROM Employees ORDER BY Salary DESC;

Sorts employees by their salary in descending order.

**Key Difference:**

**GROUP BY** groups data into summary rows, while **ORDER BY** sorts the result set.

***13. SQL Stored Procedure***

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

A **stored procedure** in SQL is a precompiled collection of one or more SQL statements that can be executed as a single unit. It is stored in the database and can be called to perform specific operations like data manipulation or complex queries.

**Key Differences:**

**1.Stored Procedure**:

Predefined and saved in the database.

Can contain multiple SQL statements.

Allows for parameters to pass values into the procedure.

Can include logic (like loops, conditions, etc.).

**2.Standard SQL Query**:

Written and executed on the fly.

Typically performs a single operation (e.g., SELECT, INSERT, UPDATE).

Does not have stored logic or reusable parameters.

**Example of a Stored Procedure:**

CREATE PROCEDURE GetEmployeeDetails(IN emp\_id INT)

BEGIN

SELECT \* FROM Employees WHERE EmployeeID = emp\_id;

END;

Stored procedures are useful for encapsulating repetitive tasks, improving security, and enhancing performance by reducing the need to send multiple queries from the application.

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

**Advantages of Using Stored Procedures:**

**1.Performance**:

Stored procedures are precompiled, meaning they are executed faster than standard SQL queries.

Reduces network traffic by allowing multiple operations to be executed in a single call.

**2.Reusability**:

Can be reused multiple times, reducing code duplication in applications.

**3.Security**:

Restrict direct access to tables, providing controlled access through the procedure, enhancing data security.

**4.Maintainability**:

Easier to manage and update logic in one place rather than modifying code in multiple applications.

**5.Encapsulation**:

Business logic can be encapsulated within the procedure, making it easier to maintain and modify without affecting application code.

**6.Error Handling**:

Provides better error handling capabilities (e.g., using TRY...CATCH blocks).

***14. SQL View***

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

A **view** in SQL is a virtual table created by querying one or more tables. It does not store data physically but rather presents data from underlying tables based on a defined query.

**Key Differences Between a View and a Table:**

**Data Storage**:

**Table**: Physically stores data in the database.

**View**: Does not store data; it is a dynamic result set generated from a query on one or more tables.

**Modification**:

**Table**: Data can be directly inserted, updated, or deleted.

**View**: Cannot directly modify data (unless it's updatable). It's typically read-only.

**Purpose**:

**Table**: Stores raw data.

**View**: Provides a simplified or specific perspective on data, often used for complex queries, security, or reporting.

**Example:**

CREATE VIEW EmployeeView ASSELECT Name, Department FROM Employees WHERE Salary > 50000;

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

**Advantages of Using Views in SQL:**

**1.Simplification**:

Views simplify complex queries by encapsulating them, making it easier to retrieve data without writing repetitive SQL statements.

**2.Security**:

Views can restrict access to specific columns or rows, providing controlled access to sensitive data while hiding underlying table details.

**3.Data Abstraction**:

Views provide an abstraction layer, allowing users to interact with a simplified representation of the data without needing to understand the underlying schema.

**4.Reusability**:

Once created, views can be reused across multiple queries, saving time and effort in writing complex SQL each time.

**5.Consistency**:

Views ensure consistent data presentation, as they always return the same result set, regardless of how the underlying tables are modified.

**6.Maintainability**:

Views make it easier to manage changes to complex queries or aggregations in one place without modifying application code.

***15. SQL Triggers***

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

A **trigger** in SQL is a set of automated actions or SQL statements that are executed in response to specific events on a table or view (e.g., INSERT, UPDATE, or DELETE).

**Types of Triggers:**

**1.BEFORE Trigger**:

**Purpose**: Executes before an insert, update, or delete operation on a table.

**Use Case**: Used for data validation or modification before the operation.

Example: Checking data integrity before inserting records.

**2.AFTER Trigger**:

**Purpose**: Executes after an insert, update, or delete operation.

**Use Case**: Used for actions that should occur after data is modified (e.g., updating audit logs, cascading changes).

Example: Logging changes to a record after an update.

**3.INSTEAD OF Trigger**:

**Purpose**: Executes instead of the triggering action (commonly used with views).

**Use Case**: Allows custom actions instead of the default insert, update, or delete operation.

Example: Inserting into a view rather than directly manipulating underlying tables.

**Use Cases:**

**Data Integrity**: Ensure valid data is inserted or updated (e.g., checking for constraints).

**Auditing**: Track changes to data for security or record-keeping.

**Cascading Actions**: Automatically update or delete related data in other tables.

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

**INSERT Trigger:**

**Purpose**: Executes when a new row is added to a table.

**Use Case**: Used to perform actions after inserting data, such as logging the insertion or updating related tables.

**Example**: Automatically updating an audit table when a new record is inserted.

**UPDATE Trigger:**

**Purpose**: Executes when an existing row is modified in a table.

**Use Case**: Used to track changes to records, validate updated data, or update related records in other tables.

**Example**: Logging changes or automatically updating timestamps when a record is updated.

**DELETE Trigger:**

**Purpose**: Executes when a row is deleted from a table.

**Use Case**: Used for cascading actions, maintaining data integrity, or logging deletions.

**Example**: Automatically removing related data from other tables when a record is deleted.

**Key Difference:**

**INSERT** triggers activate on new data entry, **UPDATE** triggers activate on modifying existing data, and **DELETE** triggers activate when data is removed.

***16. Introduction to PL/SQL***

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

PL/SQL (Procedural Language/Structured Query Language) is Oracle's procedural extension to SQL. It enhances SQL by adding programming constructs like loops, conditions, variables, and exception handling, enabling the creation of complex scripts, stored procedures, functions, and triggers within the database.

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

Here are key benefits of using PL/SQL:

1. **Improved Performance** – Reduces network traffic by bundling multiple SQL statements into a single block.
2. **Procedural Capabilities** – Supports loops, conditions, and variables, enabling complex logic.
3. **Modularity** – Allows creation of reusable procedures, functions, and packages.
4. **Error Handling** – Built-in exception handling helps manage runtime errors effectively.
5. **Security** – Access can be controlled through stored procedures, reducing direct access to data.
6. **Portability** – Code runs on any Oracle database with minimal changes.

***17. PL/SQL Control Structures***

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

Control structures in PL/SQL are constructs that manage the flow of execution in a program. The main types are **conditional**, **iterative**, and **sequential**.

* **IF-THEN**: Used for decision-making.
* IF condition THEN
* -- statements
* END IF;
* **LOOP**: Used for repeated execution.
* LOOP
* -- statements
* EXIT WHEN condition;
* END LOOP;

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

Control structures in PL/SQL help write complex queries by allowing **conditional logic (IF)** and **repetition (LOOPs)**, enabling dynamic decision-making, iterative processing of data, and handling different scenarios within a single PL/SQL block—making programs more flexible and powerful.

***18. SQL Cursors***

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

A **cursor** in PL/SQL is a pointer to the result set of a query, used to process rows one at a time.

* **Implicit Cursor**: Automatically created by PL/SQL for single-row queries (e.g., SELECT INTO).
* **Explicit Cursor**: Defined by the programmer for queries that return multiple rows, requiring manual control (OPEN, FETCH, CLOSE).

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

You use an **explicit cursor** when a query returns **multiple rows**, and you need to **process each row individually** (e.g., in a loop). Implicit cursors are for single-row queries.

***19. Rollback and Commit Savepoint***

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

A **SAVEPOINT** marks a point within a transaction to which you can **partially roll back**.

* **COMMIT**: Makes all changes permanent and removes all savepoints.
* **ROLLBACK TO SAVEPOINT**: Undoes changes made after the savepoint, keeping earlier changes.

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

Savepoints are useful when you want to **undo part of a transaction** without rolling back the entire transaction—especially in complex operations where some steps might fail but others should be retained.