**Module – 5**

**Introduction To DBMS**

1. **Theory:**
2. **What is SQL, & why it is essential in database management?**

**Ans. SQL (Structured Query Language) is a standardized programming language used to manage and manipulate relational databases. It’s essential in database management for several key reasons:**

* **Create & modify database structures (tables, views, indexes, etc.)**
* **Insert, update, delete, & retrieve data from databases.**
* **Control access to the data & manage user permissions**
* **Ensure data integrity & consistency**

**Why it is essential in database management is as follows.**

* **Universal language for relational databases.**
* **Efficient data handling**
* **Data integrity & security**
* **Scalability and flexibility**
* **Foundation for data analysis and BI**

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

**Ans.**

1. **Definition**

* **DBMS: A software system that enables the creation, organization, storage, and retrieval of data from databases. It does not necessarily use a relational model.**
* **RDBMS: A type of DBMS that stores data in a relational model — i.e., in tables with rows and columns.**

1. **Data Structure**

* **DBMS: Stores data in files, hierarchical forms, or navigational formats (e.g., key-value pairs, XML).**
* **RDBMS: Uses a tabular format (relations or tables), where data is organized into rows and columns.**

1. **Relationships Between Data**

* **DBMS: Relationships between data are not explicitly defined or enforced.**
* **RDBMS: Relationships are established using foreign keys, primary keys, and constraints.**

1. **Data Integrity and Constraints**

* **DBMS: Less support for data integrity rules like entity integrity or referential integrity.**
* **RDBMS: Enforces data integrity, normalization, and constraints (e.g., NOT NULL, UNIQUE, CHECK).**

1. **Example Systems**

* **DBMS: Microsoft Access (basic versions), XML databases, file systems.**
* **RDBMS: MySQL, PostgreSQL, Oracle, Microsoft SQL Server.**

1. **Support for SQL**

* **DBMS: May or may not support SQL.**
* **RDBMS: Uses Structured Query Language (SQL) for defining and manipulating data.**

1. **Multi-user Environment**

* **DBMS: Typically supports single-user environments.**
* **RDBMS: Designed for multi-user environments and concurrency control.**

1. **ACID Properties**

* **DBMS: May not fully support ACID (Atomicity, Consistency, Isolation, Durability).**
* **RDBMS: Fully supports ACID for transaction management.**

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

**Ans. SQL (Structured Query Language) is the standard language used to manage data in Relational Databases (RDBMS). Its main roles are:**

* **DDL (Data Definition Language) – Defines database structure (e.g., CREATE, ALTER).**
* **DML (Data Manipulation Language) – Manages data (e.g., INSERT, UPDATE, DELETE).**
* **DQL (Data Query Language) – Retrieves data (SELECT).**
* **DCL (Data Control Language) – Manages access (GRANT, REVOKE).**
* **TCL (Transaction Control Language) – Controls transactions (COMMIT, ROLLBACK).**

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

**Ans. Key features are as follows.**

1. **Data definition**

* **Allows creating and modifying database structures.**
* **Commands: CREATE, ALTER, DROP.**

1. **Data Manipulation**

* **Supports inserting, updating, and deleting data.**
* **Commands: INSERT, UPDATE, DELETE.**

1. **Data Manipulation**

* **Powerful querying capability to fetch specific data.**
* **Commands: SELECT (with WHERE, GROUP BY, ORDER BY, etc.).**

1. **Security and Access Control**

* **Controls user access to data.**
* **Commands: GRANT, REVOKE.**

1. **Transaction Control**

* **Ensures reliable processing of multiple operations.**
* **Commands: COMMIT, ROLLBACK, SAVEPOINT.**

1. **Data integrity**

* **Enforces rules like primary keys, foreign keys, and constraints to maintain accurate data.**

1. **Standardized language**

* **SQL is standardized by ANSI and ISO, and widely supported by all major RDBMS.**

1. **Supports multiple views**

* **Allows creation of views to simplify complex queries and enhance security.**

1. **Cross-Platform Compatibility**

* **Works across different database systems with minor variations (e.g., MySQL, Oracle, PostgreSQL).**

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

**Ans. Basic components are as follows.**

* **Statements – Commands like SELECT, INSERT, UPDATE, DELETE.**
* **Keywords – Reserved words such as FROM, WHERE, JOIN, GROUP BY.**
* **Clauses – Parts of a statement, e.g., SELECT, FROM, WHERE, ORDER BY.**
* **Expressions – Calculations or values, like price \* quantity.**
* **Predicates – Conditions in WHERE or HAVING, e.g., =, IN, LIKE.**
* **Identifiers – Names for tables, columns, aliases.**
* **Operators – Symbols like +, -, =, AND, OR.**
* **Functions – Built-ins like COUNT(), SUM(), NOW().**
* **Comments – Notes in code: -- or /\* \*/.**

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

**Ans. SELECT column1, column2, ...**

**FROM table\_name**

**[WHERE condition]**

**[GROUP BY column]**

**[HAVING condition]**

**[ORDER BY column [ASC|DESC]]**

**[LIMIT number]; -- (Optional, depends on the SQL dialect)**

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

**Ans. Clauses are as follows.**

* **SELECT – Chooses which columns to display.**
* **FROM – Specifies the table to query.**
* **WHERE – Filters rows by condition.**
* **GROUP BY – Groups rows with the same values.**
* **HAVING – Filters grouped data.**
* **ORDER BY – Sorts the result set.**
* **LIMIT / TOP – Restricts the number of rows returned.**

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

**Ans. Constraints in SQL are rules applied to columns in a table to enforce data integrity and accuracy. They ensure that only valid data is stored in the database.**

* **NOT NULL – Column can't have NULL values.**
* **UNIQUE – All values in the column must be different.**
* **PRIMARY KEY – Uniquely identifies each row (NOT NULL + UNIQUE).**
* **FOREIGN KEY – Links to a primary key in another table.**
* **CHECK – Limits values based on a condition.**
* **DEFAULT – Sets a default value if none is provided.**
* **AUTO\_INCREMENT – Automatically increases numeric values (e.g., IDs).**

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

**Ans. Differences is as follows.**

**Primary key**

* **Uniquely identifies each row in a table**
* **Must be unique and not null**
* **Defined in the current table**
* **It cannot be null.**
* **Ensures row identity**
* **E.g. id INT PRIMARY KEY.**

**Foreign key**

* **Establishes a link between two tables**
* **Can have duplicate values and can be null**
* **Refers to a key in another table**
* **Yes (depending on design)**
* **Ensures referential integrity**
* **E.g. user\_id INT REFERENCES users(id)**

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

**Ans. Role of NOT NULL is as follows.**

* **Purpose: Ensures that a column must have a value (cannot be NULL).**
* **Use Case: When a field is required, like a username or email.**
* **E.g. name VARCHAR(50) NOT NULL**

**Role of UNIQUE is as follows.**

* **Purpose: Ensures all values in a column are distinct (no duplicates).**
* **Use Case: For fields that must be unique, like email or ID numbers.**
* **E.g. email VARCHAR(100) UNIQUE**

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

**Ans. DDL defines and manages the structure of the database, not the data itself. Changes made with DDL are usually permanent and affect the schema.**

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

**Ans.**

**CREATE TABLE table\_name**

**(column1 datatype [constraints],**

**column2 datatype [constraints]);**

* **table\_name: The name of the table you want to create.**
* **column1, column2, ...: The names of the columns in the table.**
* **datatype: The data type of the column (e.g., INT, VARCHAR).**
* **constraints: Optional constraints like NOT NULL, PRIMARY KEY, UNIQUE, etc.**

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

**Ans. When creating a table in SQL, specifying data types and constraints is essential for ensuring data integrity, consistency, and optimal performance. Here’s why they are important:**

**Data Types:**

* **Purpose: Define the type of data that a column can store (e.g., text, numbers, dates).**
* **Benefits:**
* **Data Accuracy: Ensures that only valid data is stored in each column (e.g., INT for integers, VARCHAR for strings, DATE for dates).**
* **Storage Efficiency: Optimizes database storage by allocating the appropriate amount of space for each column (e.g., INT uses less space than VARCHAR).**
* **Performance: Improves query performance and indexing, as the database can efficiently work with specific data types.**
* **E.g. INT: For integer numbers.**
* **VARCHAR(size): For variable-length strings (e.g., names).**
* **DATE: For date values.**

**Constraints:**

* **Purpose: Set rules to enforce the integrity and validity of the data in the table.**
* **Benefits:**
* **Data Integrity: Prevents invalid or inconsistent data (e.g., NOT NULL ensures that columns cannot have empty values).**
* **Enforcing Uniqueness: Ensures no duplicate values (e.g., UNIQUE for email addresses).**
* **Consistency and Relationships: Ensures correct relationships between tables (e.g., FOREIGN KEY enforces referential integrity).**
* **Business Logic: Enforces rules that align with business requirements (e.g., CHECK to limit values).**
* **E.g.**
* **PRIMARY KEY: Ensures the column has unique, non-null values, identifying each row uniquely.** **NOT NULL: Ensures a column cannot have null values (every row must have data for that column).**
* **UNIQUE: Ensures all values in the column are distinct.**

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

**Ans. ALTER command in SQL are as follows:**

* **Add a column to an existing table:**

**ALTER TABLE table\_name**

**ADD column\_name datatype;**

* **Modify an existing column's datatype or constraints:**

**ALTER TABLE table\_name**

**MODIFY column\_name new\_datatype;**

**-- or in some databases:**

**ALTER TABLE table\_name**

**ALTER COLUMN column\_name TYPE new\_datatype;**

* **Rename a column (syntax depends on the SQL dialect):**

**In PostgreSQL:**

**ALTER TABLE table\_name**

**RENAME COLUMN old\_name TO new\_name;**

**In MySQL:**

**ALTER TABLE table\_name**

**CHANGE old\_name new\_name datatype;**

**Drop a column:**

**ALTER TABLE table\_name**

**DROP COLUMN column\_name;**

* **Add, drop, or modify constraints (like PRIMARY KEY, FOREIGN KEY, UNIQUE, etc.):**

**ALTER TABLE table\_name**

**ADD CONSTRAINT constraint\_name PRIMARY KEY (column\_name);**

* **Rename the table (varies by SQL dialect):**

**ALTER TABLE old\_table\_name**

**RENAME TO new\_table\_name;**

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

**Ans. Add, modify and drop columns using ALTER command is as follows:**

* **Add a Column**

**ALTER TABLE table\_name**

**ADD column\_name datatype;**

* **Modify a Column**

**-- Syntax varies by SQL dialect**

**-- Example for MySQL:**

**ALTER TABLE table\_name**

**MODIFY column\_name new\_datatype;**

* **Drop a Column**

**ALTER TABLE table\_name**

**DROP COLUMN column\_name;**

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

**Ans. Functions of drop command is as follows:**

* **Deletes the object and all its data permanently.**
* **Cannot be undone (use with caution).**

**Examples:**

**Drop a table:**

**DROP TABLE table\_name;**

**Drop a database:**

**DROP DATABASE database\_name;**

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

**Ans. Key implications of drop table:**

* **Permanent Data Loss**
* **All data stored in the table is permanently deleted.**
* **Cannot be recovered unless you have a backup.**
* **Schema Loss**
  + **The structure (columns, constraints, indexes) of the table is also deleted.**
  + **The table no longer exists in the database.**
* **Dependency Issues**
  + **If other database objects (like views, foreign keys, stored procedures) depend on the table, they may break or throw errors.**
* **Security and Access Changes**
  + **Any permissions granted on the table are also removed.**
* **No Rollback in Most Cases**
  + **In many databases, DROP is immediate and irreversible, especially if not used within a transaction block.**

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

**Ans.**

* **INSERT**

**Add new rows (record) to a table.**

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

**VALUES (value1, value2, ...);**

* **UPDATE**

**Modifies existing records in a table.**

**UPDATE table\_name**

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

**WHERE condition;**

* **DELETE**

**Removes existing records from a table.**

**DELETE FROM table\_name**

**WHERE condition;**

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

**Ans. The WHERE clause is crucial in UPDATE and DELETE operations because it specifies which rows should be updated or deleted.**

**Importance of WHERE Clause:**

* **Prevents Unintended Changes**

**Without WHERE, all rows in the table will be affected.**

**UPDATE employees SET salary = 0;**

* **Targets Specific Rows**

**It lets you update or delete only the rows that meet certain conditions.**

**UPDATE employees SET salary = 5000 WHERE department = 'HR';**

**DELETE FROM orders WHERE order\_date < '2024-01-01';**

* **Protects Data Integrity**

**It helps avoid accidental loss or corruption of data.**

**Without WHERE = Risk of Massive Data Loss**

* **UPDATE without WHERE: changes all rows.**
* **DELETE without WHERE: removes all data in the table.**

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

**Ans. The SELECT statement is used to retrieve data from one or more tables in a database. It's the most commonly used SQL command for querying data.**

* **Basic syntax:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE condition;**

* **Select all columns:**

**SELECT \* FROM employees;**

* **Select specific columns:**

**SELECT first\_name, salary FROM employees;**

* **Filter data with where:**

**SELECT \* FROM employees**

**WHERE department = 'Sales';**

* **Sort results with order by:**

**SELECT \* FROM employees**

**ORDER BY salary DESC**

* **Limit number of results:**

**MY SQL:**

**SELECT \* FROM employees LIMIT 5;**

**SQL Server:**

**SELECT TOP 5 \* FROM employees;**

* **Use Functions or Calculations:**

**SELECT AVG(salary) FROM employees;**

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

**Ans. The WHERE clause is used to filter records based on a specific condition. It tells SQL which rows to include in the result.**

**Example:**

**SELECT \* FROM employees**

**WHERE department = 'Sales';**

* **Using both together:**

**You can use WHERE to filter the rows and ORDER BY to sort the filtered results.**

**Example:**

**SELECT \* FROM employees**

**WHERE department = 'Sales'**

**ORDER BY hire\_date ASC;**

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

**Ans. These commands are used to manage user permissions on database objects like tables, views, and procedures.**

**GRANT – Gives permissions**

**Allows a user or role to perform specific actions (like SELECT, INSERT, UPDATE, etc.) on database objects.**

**Example:**

**GRANT SELECT, INSERT ON employees TO user\_name;**

**REVOKE- Removes permissions:**

**Takes back previously granted permissions from a user or role.**

**Example:**

**REVOKE INSERT ON employees FROM user\_name;**

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

**Ans. You manage user access and control security in a database by granting or revoking specific privileges on database objects (like tables, views, or procedures).**

* **Granting Privileges (GRANT)**

**GRANT privilege\_list**

**ON object\_name**

**TO user\_or\_role;**

**Example:**

**GRANT SELECT, INSERT**

**ON employees**

**TO john;**

* **Revoking Privileges (REVOKE)**

**REVOKE privilege\_list**

**ON object\_name**

**FROM user\_or\_role;**

**Example:**

**REVOKE INSERT**

**ON employees**

**FROM john;**

* **Managing with Roles (Optional Advanced Use):**

**You can grant privileges to a role and then assign that role to users. This makes managing multiple users easier.**

**CREATE ROLE sales\_team;**

**GRANT SELECT, UPDATE ON customers TO sales\_team;**

**GRANT sales\_team TO alice;**

* **SELECT – Read data**
* **INSERT – Add data**
* **UPDATE – Modify data**
* **DELETE – Remove data**
* **ALL PRIVILEGES – Full access to the object**

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

**Ans. These commands are used to manage transactions, ensuring that groups of database operations are treated as a single unit of work.**

* **COMMIT – Save Changes**

**Finalizes a transaction by saving all changes made since the last COMMIT or ROLLBACK.**

**Once committed, changes are permanent.**

**BEGIN;**

**UPDATE accounts SET balance = balance - 100 WHERE id = 1;**

**UPDATE accounts SET balance = balance + 100 WHERE id = 2;**

**COMMIT;**

* **ROLLBACK – Undo Changes**

**Cancels a transaction and reverts all changes made since the last COMMIT.**

**Useful for handling errors or canceling unintended changes.**

**BEGIN;**

**UPDATE accounts SET balance = balance - 100 WHERE id = 1;**

**-- Error occurs here**

**ROLLBACK;**

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

**Ans. A transaction in SQL is a sequence of one or more SQL statements that are executed as a single unit of work. Transactions help ensure data integrity, consistency, and error recovery.**

* **Key Properties of Transactions — ACID**
* **Atomicity**
* **All operations in a transaction succeed together or fail together.**
* **If one part fails, the whole transaction is rolled back.**
* **Consistency**
* **The database moves from one valid state to another, maintaining rules and constraints.**
* **Isolation**
* **Transactions are executed independently, so one transaction doesn’t interfere with another.**
* **Durability**
* **Once a transaction is committed, the changes are permanent, even if the system crashes.**
* **BEGIN / START TRANSACTION**

**BEGIN;**

**-- or**

**START TRANSACTION;**

* **COMMIT**

**Saves all changes made during the transaction.**

**COMMIT;**

* **ROLLBACK**

**Cancels the transaction and undoes all changes.**

**ROLLBACK;**

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

**Ans. In SQL, a JOIN is used to combine rows from two or more tables based on a related column between them, usually a foreign key relationship.**

**JOINs allow you to retrieve data that is spread across multiple tables. For example, if you have a Customers table and an Orders table, you might use a JOIN to find out which customer placed which order.**

**Types of JOIN in SQL:**

1. **Inner JOIN**

* **Returns only the matching rows from both tables.**
* **If there is no match, the row is excluded from the result.**
* **Commonly used when you only care about records that exist in both tables.**

**SELECT \***

**FROM Customers**

**INNER JOIN Orders**

**ON Customers.CustomerID = Orders.CustomerID;**

1. **LEFT JOIN (or LEFT OUTER JOIN)**

* **Returns all rows from the left table, and the matching rows from the right table.**
* **If there's no match, the result will contain NULLs for columns from the right table.**

**SELECT \***

**FROM Customers**

**LEFT JOIN Orders**

**ON Customers.CustomerID = Orders.CustomerID;**

1. **RIGHT JOIN (or RIGHT OUTER JOIN).**

* **Returns all rows from the right table, and the matching rows from the left table.**
* **If there's no match, the result will contain NULLs for columns from the left table.**

**SELECT \***

**FROM Customers**

**RIGHT JOIN Orders**

**ON Customers.CustomerID = Orders.CustomerID;**

1. **FULL OUTER JOIN**

* **Returns all rows from both tables.**
* **If there is no match, the missing side will contain NULLs.**
* **Combines the results of both LEFT and RIGHT JOINs.**

**SELECT \***

**FROM Customers**

**FULL OUTER JOIN Orders**

**ON Customers.CustomerID = Orders.CustomerID;**

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

**Ans. JOINs are used in SQL to combine data from multiple tables by matching rows based on a related column (usually a key like ID).**

* **Identify the related columns (e.g., CustomerID in both Customers and Orders).**
* **Choose the JOIN type based on what data you want (only matches, all from one table, etc.).**
* **Write the SQL statement to link the tables using ON condition.**

**Example:**

**Imagine two tables:**

* **Customers**
  + **CustomerID | Name**
* **Orders**
  + **OrderID | CustomerID | OrderDate**

**SELECT Customers.Name, Orders.OrderDate**

**FROM Customers**

**INNER JOIN Orders**

**ON Customers.CustomerID = Orders.CustomerID;**

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

**Ans. The GROUP BY clause is used to group rows that have the same values in specified columns. It's often used with aggregate functions (like SUM, COUNT, AVG, MAX, MIN) to summarize data.**

**Purpose:**

* **To summarize or aggregate data by one or more columns.**
* **Example: Get total sales per customer, average salary per department, etc.**

**SELECT column\_name, AGGREGATE\_FUNCTION(column\_name)**

**FROM table\_name**

**GROUP BY column\_name;**

**SELECT Customer, SUM(Amount)**

**FROM Sales**

**GROUP BY Customer;**

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

**Ans. The difference between GROUP BY & ORDER BY is as follows:**

**GROUP BY:**

* **Groups rows with the same values**
* **Used with Aggregate functions (e.g., SUM, COUNT)**
* **Returns one row per group**
* **Comes before ORDER BY in a query**

**ORDER BY:**

* **Sorts the result set**
* **Used with Any query**
* **Keeps all rows, just changes order**
* **Comes last in the query**

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

**Ans. A stored procedure is a precompiled set of SQL statements that is saved in the database and can be executed repeatedly.**

**Think of it like a function in programming: it can take inputs, perform operations (like queries, updates), and return results or perform actions.**

**How it differ from a standard SQL Query:**

**Stored Procedure:**

* **Predefined and stored in the database**
* **Can be reused many times**
* **Can accept input/output parameters**
* **Often faster due to precompilation**
* **Supports conditional logic, loops, etc.**
* **Can encapsulate logic, restrict direct access**

**Standard Procedure:**

* **Written and executed as needed**
* **Typically one-time use or copied manually**
* **Usually has hardcoded values or simple inputs**
* **Parsed and compiled at runtime**
* **Limited to one query or statement block**
* **Users see the raw SQL and data access**

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

**Ans. Advantages are as follows:**

* **Reusable – Write once, use many times.**
* **Faster – Precompiled and optimized by the database.**
* **Secure – Hide SQL logic and control access.**
* **Maintainable – Centralized logic, easy to update.**
* **Supports logic – Allows conditions, loops, and transactions.**
* **Parameterized – Accepts inputs for dynamic use.**
* **Less network load – Fewer commands sent between app and database.**-=-9

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

**Ans. A view is a virtual table that contains the result of a query. It doesn't store data itself but displays data stored in one or more underlying tables. A view can be used like a table in SELECT queries, but it doesn't store data — it dynamically retrieves it when the view is queried.**

**Difference between view and table are as follows:**

**View:**

* **Does not store data, just a query result**
* **Defined by a query (dynamic data)**
* **Can be read-only or updatable, depending on the complexity of the view**
* **Can have performance overhead (since it's based on a query)**
* **Simplifies complex queries or hides complexity.**

**Table:**

* **Stores actual data in the database**
* **Defined by columns and data types**
* **Always updatable (unless restricted)**
* **Direct access to stored data**
* **Holds raw, persistent data**

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

**Ans. Advantages are as follows:**

* **Simplify complex queries – Encapsulate complex logic in one reusable object.**
* **Enhance security – Hide sensitive data or restrict access.**
* **Data abstraction – Hide database structure from users.**
* **Reusability & maintainability – Use the same logic across multiple queries.**
* **Improve readability – Make queries more understandable with clear names.**
* **Consistency – Ensure consistent data retrieval across applications.**
* **Reduce redundancy – Avoid repeating the same code.**

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

**Ans. A trigger is a special kind of stored procedure that is automatically executed or fired when a specific event occurs on a particular table or view. Triggers are used to enforce business rules, maintain data integrity, or automatically perform actions in the database when certain changes (INSERT, UPDATE, DELETE) happen**

**Types:**

* **BEFORE: Executes before the event (e.g., data validation).**
* **AFTER: Executes after the event (e.g., logging or updating related data).**
* **INSTEAD OF: Executes instead of the event (e.g., custom behaviour for views).**

**Uses:**

* **Enforce data integrity.**
* **Audit changes to data..**
* **Implement business rules.**
* **Update related tables automatically.**

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

**Ans. These are the three most common types of triggers in SQL, each designed to respond to different types of data modification actions.**

**INSERT Trigger:**

* **Fires when a new record is inserted into a table.**

**Commonly used to:**

* **Automatically populate or modify values in other columns or tables.**
* **Enforce rules before inserting data (e.g., ensure valid data before insertion).**

**UPDATE Trigger:**

* **Fires when an existing record is updated (via UPDATE).**

**Commonly used to:**

* **Track changes made to the data.**
* **Enforce additional logic when a record is modified (e.g., prevent updates under certain conditions).**

**DELETE Trigger**

* **Fires when a record is deleted from a table.**

**Commonly used to:**

* **Prevent deletion of important records (e.g., soft delete by marking the record instead).**
* **Automatically delete or archive related data in other tables (cascading actions).**

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

**Ans. PL/SQL (Procedural Language/SQL) is Oracle's procedural extension to SQL. It combines the power of SQL (for querying and manipulating data) with the features of procedural programming (such as loops, conditions, and variables). This allows for more complex, flexible, and efficient database operations than plain SQL.**

**How PL/SQL Extends SQL's Capabilities:**

**Procedural Features:**

* **PL/SQL allows you to write procedures, functions, and packages, which can execute multiple SQL statements in a logical sequence.**
* **You can use variables, loops, conditional statements (IF-THEN-ELSE), and exception handling in PL/SQL, which are not available in standard SQL.**

**Control Structures:**

* **PL/SQL adds control structures like loops (FOR, WHILE), conditional logic (IF, CASE), and exceptions to handle errors gracefully.**

**Error Handling:**

* **PL/SQL provides a robust exception-handling mechanism to catch and manage runtime errors, unlike SQL, where errors are usually returned as results.**

**Block Structure:**

* **Declaration section: For declaring variables.**
* **Executable section: For writing logic and SQL queries.**
* **Exception section: For handling errors.**

**Modularity:**

* **You can bundle related SQL operations into reusable units like procedures, functions, and packages, improving code reusability and maintainability.**

**Performance:**

* **Bulk operations and the ability to execute multiple SQL statements in one block make PL/SQL more efficient than executing individual SQL queries separately.**

**Triggers:**

* **PL/SQL is used to create complex database triggers, allowing automatic actions in response to DML events (INSERT, UPDATE, DELETE) on tables.**

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

**Ans. Enhanced Performance:**

**Reduced Network Traffic: PL/SQL allows you to bundle multiple SQL statements into a single block. Instead of sending multiple queries over the network, a single PL/SQL block can execute everything at once, improving performance.**

**Efficient Bulk Operations: PL/SQL supports bulk processing, where you can process multiple rows of data at once, reducing the need for multiple round trips to the database.**

**Error Handling and Exception Management:**

**PL/SQL provides a structured way to catch and handle errors with its exception handling mechanism. This allows for more robust and reliable applications, as you can anticipate errors and manage them in a controlled way (e.g., rollbacks, logging, custom error messages).**

**Modularity and Reusability:**

**You can create procedures, functions, and packages, making your code modular and reusable. This improves maintainability because you can change the logic in one place without affecting other parts of the application**

**Once a procedure or function is created, you can reuse it in multiple places without rewriting the same logic.**

**Control Structures:**

**PL/SQL supports procedural logic such as loops (FOR, WHILE), conditional statements (IF-THEN-ELSE), and case statements, which standard SQL doesn't offer. This gives you more flexibility to handle complex logic directly inside the database.**

**Improved Security:**

**PL/SQL allows you to encapsulate sensitive logic inside procedures and functions. By hiding complex SQL logic, users and applications don’t have direct access to the underlying data or table structures, enhancing security.**

**You can also control user permissions at the procedure level, granting access to specific actions without exposing raw data.**

**Triggers and Automation:**

**PL/SQL is essential for creating database triggers, which are automatically executed when certain actions (like INSERT, UPDATE, DELETE) happen on tables. This allows for automatic actions like logging, auditing, or cascading changes, all within the database.**

**Transaction Control:**

**PL/SQL allows for transaction control with commands like COMMIT, ROLLBACK, and SAVEPOINT. You can ensure that a series of SQL operations are performed atomically, meaning that if one part of the transaction fails, the entire set of operations can be rolled back.**

**Portability:**

**PL/SQL code is portable within Oracle databases. Once written, your PL/SQL procedures, functions, and triggers will work across all Oracle-based systems, making it easier to maintain and transfer applications.**

**Better Integration with Oracle Tools:**

**PL/SQL integrates seamlessly with other Oracle technologies like Oracle Forms, Oracle Reports, and Oracle Application Express (APEX). This tight integration allows you to build complex applications directly on top of the Oracle database.**

**Scalability and Maintainability:**

**With PL/SQL, you can manage large datasets and complex database logic, ensuring that applications remain scalable as the database grows.**

**The modularity and reusability of PL/SQL code also make it easier to maintain over time. For example, updating a function or procedure doesn’t require changes to all the applications that rely on it.**

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

**Ans. Control structures in PL/SQL allow you to define the flow of execution within your programs. These structures include conditional logic (like IF-THEN) and loops (like FOR, WHILE) that enable complex decision-making and repetitive tasks.**

**Common Control Structures in PL/SQL:**

**Conditional Statements:**

* **IF-THEN**
* **IF-THEN-ELSE**
* **CASE statement**

**Loops:**

* **LOOP**
* **FOR LOOP**
* **WHILE LOOP**

**Exception Handling:**

* **EXCEPTION block to handle errors.**

**IF-THEN Control Structure:** **The IF-THEN structure allows you to execute a block of code only if a specific condition is true.**

**DECLARE**

**v\_salary NUMBER := 5000;**

**BEGIN**

**IF v\_salary > 3000 THEN**

**DBMS\_OUTPUT.PUT\_LINE('Salary is above average.');**

**END IF;**

**END;**

**IF-THEN-ELSE: This allows you to define an alternative set of statements if the condition is not met.**

**DECLARE**

**v\_salary NUMBER := 2500;**

**BEGIN**

**IF v\_salary > 3000 THEN**

**DBMS\_OUTPUT.PUT\_LINE('Salary is above average.');**

**ELSE**

**DBMS\_OUTPUT.PUT\_LINE('Salary is below average.');**

**END IF;**

**END;**

**LOOP Control Structure: Loops are used to repeat a block of code multiple times as long as a specific condition is met.**

**Basic LOOP: This loop will continue executing until a EXIT condition is met.**

**DECLARE**

**v\_counter NUMBER := 1;**

**BEGIN**

**LOOP**

**DBMS\_OUTPUT.PUT\_LINE('Counter value: ' || v\_counter);**

**v\_counter := v\_counter + 1;**

**EXIT WHEN v\_counter > 5;**

**END LOOP;**

**END;**

**FOR LOOP: The FOR LOOP is used when you know the number of iterations in advance. It automatically handles the increment of the loop counter.**

**BEGIN**

**FOR i IN 1..5 LOOP**

**DBMS\_OUTPUT.PUT\_LINE('Iteration: ' || i);**

**END LOOP;**

**END;**

**WHILE LOOP: The WHILE LOOP continues to execute the block of code as long as the condition remains true. If the condition is false initially, the loop won't execute at all.**

**DECLARE**

**v\_counter NUMBER := 1;**

**BEGIN**

**WHILE v\_counter <= 5 LOOP**

**DBMS\_OUTPUT.PUT\_LINE('Counter value: ' || v\_counter);**

**v\_counter := v\_counter + 1;**

**END LOOP;**

**END;**

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

**Ans. Control structures in PL/SQL help write complex queries by:**

* **Adding logic to execute queries conditionally (e.g., IF-THEN).**
* **Automating repetitive tasks using loops.**
* **Handling multi-step processes with FOR and WHILE loops.**
* **Managing error handling and exception processing for robustness.**
* **Transaction control ensures operations are atomic and consistent.**

**These features make PL/SQL a powerful tool for creating efficient, maintainable, and flexible database applications.**

**Control structures in PL/SQL help write complex queries by adding logic and repetition to SQL:**

* **IF-THEN: Executes code based on conditions, allowing for conditional updates or actions (e.g., updating prices only if stock is low).**
* **LOOP: Repeats code for multiple rows or operations, useful for bulk processing (e.g., processing all employees).**
* **FOR LOOP: Loops through a specified range, ideal for tasks like calculating totals for each month.**
* **WHILE LOOP: Continues execution as long as a condition is true, useful for dynamic or variable-length tasks.**
* **CASE: Handles multiple conditions in a single statement, simplifying complex logic (e.g., categorizing customers based on spending).**
* **Exception Handling: Catches and manages errors within complex operations to ensure smooth execution.**
* **Transaction Control: Uses COMMIT and ROLLBACK to ensure multiple operations succeed or fail together.**

**These structures make PL/SQL more powerful for automating tasks, handling errors, and processing complex logic within the database.**

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

**Ans. A cursor in PL/SQL is a pointer to a result set (a set of rows returned by a SQL query). It allows you to fetch and process individual rows of data from a result set one at a time. Cursors are particularly useful when you need to handle row-by-row processing, which standard SQL cannot do.**

**Difference between implicit cursor and explicit cursor is as follows:**

**Implicit Cursor:**

* **Automatically created by Oracle for DML operations or single queries.**
* **No direct control (Oracle handles it automatically).**
* **For single SQL queries that return a result set.**
* **Suitable for simple, single-result queries.**
* **UPDATE employees SET salary = salary + 1000 WHERE department\_id = 10;**

**Explicit Cursor:**

* **Manually declared and controlled by the developer.**
* **Full control: you manage opening, fetching, and closing.**
* **For multi-row queries and when you need to process each row.**
* **More efficient for complex queries or handling large result sets.**
* **DECLARE CURSOR c\_emp IS SELECT \* FROM employees; ...**

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

**Ans.**

* **Multi-row queries: You need to process more than one row.**
* **More control: You need explicit control over the fetch process (e.g., skipping rows, changing the flow).**
* **Parameterized queries: You want to pass dynamic input to the query.**
* **Specific columns: You need to fetch specific columns and store them in variables.**
* **Complex queries: Complex SQL queries (like joins and aggregates) that require row-by-row processing.**

**In short, use explicit cursors when you need more control over row-by-row processing, dynamic SQL, or complex data handling, and use implicit cursors for simpler operations like single-row queries and basic DML.**

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

**Ans. SAVEPOINT marks an intermediate point in the transaction.**

**ROLLBACK TO SAVEPOINT undoes changes made after a specific savepoint but keeps earlier changes intact.**

**COMMIT finalizes the entire transaction, making all changes permanent and releasing any defined savepoints.**

**A ROLLBACK without specifying a savepoint undoes all changes made during the transaction.**

**Use Cases for SAVEPOINT:**

**Error Handling: You can set savepoints before critical operations. If an error occurs, you can rollback to the savepoint and avoid losing all progress.**

**Partial Rollback: In complex transactions, savepoints allow you to undo only part of a transaction without affecting the entire operation.**

**Complex Transactions: Savepoints make multi-step transactions safer by allowing you to backtrack to a known point of consistency in case something goes wrong.**

**In short, SAVEPOINT offers a way to rollback selectively within a transaction, giving you more control over what parts of the transaction to undo, while COMMIT finalizes and ROLLBACK undoes all changes if necessary.**

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

**Ans. Savepoints are useful in database transactions when you want to:**

1. **Rollback part of a transaction without canceling the whole thing.**
2. **Handle errors gracefully in specific steps.**
3. **Simulate nested transactions in databases that don’t support them.**
4. **Try optional operations and undo them if needed.**
5. **Test and debug complex transaction logic safely.**