Introduction To DBMS

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

➔ SQL (Structured Query Language) is a standard programming language used to communicate with and manage databases.

➔ It allows users to create, retrieve, update, and delete data from relational databases.

★ **Importance in Database Management:**

- It is the primary tool used by database administrators and developers.
- Enables efficient data manipulation and querying.
- Ensures data integrity and security through permissions and constraints.
- Supports transaction management, making it reliable for business applications.

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

Feature	DBMS	RDBMS
Full Form	Database Management System	Relational Database Management System
Data Structure	Stores data as files or collections	Stores data in tables (rows and columns)
Relationships	No relationships between data	Supports relationships using foreign keys
Normalization	Not enforced	Supports normalization to reduce redundancy
Examples	MS Access, XML-based DB	MySQL, Oracle, SQL Server, PostgreSQL

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

➔ SQL is the core language used to interact with relational databases. It allows users to:

- Define the structure of databases (CREATE, ALTER, DROP)
- Manipulate data (INSERT, UPDATE, DELETE)
- Query data (SELECT, JOIN, WHERE, GROUP BY)
- Control access and set permissions (GRANT, REVOKE)
- Maintain integrity through constraints and transactions

➔ In short, SQL provides complete control over how data is stored, retrieved, and maintained.

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

➔ **Data Querying:** Use SELECT to fetch specific data.
➔ **Data Manipulation:** Use INSERT, UPDATE, DELETE to modify data.
➔ **Data Definition:** Use CREATE, ALTER, DROP to manage database structures.
➔ **Data Control:** Use GRANT, REVOKE to manage user permissions.
➔ **Transaction Control:** Use COMMIT, ROLLBACK, SAVEPOINT for reliable transactions.
➔ **Support for Constraints:** Such as PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL.
➔ **Built-in Functions:** Aggregates like SUM(), AVG(), COUNT(), etc.
➔ **Join Operations:** Combine data from multiple tables using JOIN.

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

➔ **Keywords-** Reserved words like SELECT, FROM, WHERE, INSERT, UPDATE, etc.
➔ **Identifiers**- Names of tables, columns, databases (e.g., students, marks).
➔ **Operators**- Symbols used in conditions (=, >, <, LIKE, IN, etc.).
➔ **Clauses**- Components that define parts of a query (WHERE, ORDER BY, GROUP BY).
➔ **Expressions**- Combinations of values and operators that evaluate to a result.
➔ **Literals**- Constant values like 'John', 25, '2025-06-25'.
➔ **Comments**- Lines ignored by the SQL engine for documentation:
    ◆ Single-line: -- comment here
    ◆ Multi-line: /* comment block */

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

★ **Syntax :**

SELECT * FROM Table_name;

OR

```
SELECT column1, column2, ...
FROM table_name
WHERE condition
GROUP BY column
HAVING condition
ORDER BY column ASC|DESC;
```

★ **Example :**

SELECT * FROM students;

OR

```
SELECT department, city
FROM student
WHERE city = 'Delhi'
GROUP BY department, city
HAVING department = 'Science'
ORDER BY department ASC;
```

7. Explain the role of clauses in SQL statements.

Clause	Purpose
SELECT	Specifies the columns to retrieve.
FROM	Indicates the table to retrieve data from.
WHERE	Filters rows based on conditions.
GROUP BY	Groups rows sharing the same values into summary rows.
HAVING	Filters groups based on conditions (used with GROUP BY).
ORDER BY	Sorts the result set by one or more columns.
JOIN	Combines rows from two or more tables based on related columns.

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

➔ Constraints are rules applied to table columns to enforce data integrity and accuracy.

➔ They restrict the type of data that can be stored in a column and help maintain consistent and valid data in the database.

❖ **Types of Constraints in SQL:**

- **PRIMARY KEY :** Uniquely identifies each record in a table. Cannot have NULL values.

- **FOREIGN KEY :** Maintains referential integrity by linking a column to the PRIMARY KEY of another table.

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

- **UNIQUE :** Ensures all values in a column are different.

- **CHECK :** Ensures all values in a column meet a specific condition. Example: age > 18.

- **DEFAULT :** Assigns a default value if no value is specified during insertion.

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

Feature	PRIMARY KEY	FOREIGN KEY
Purpose	Uniquely identifies a record in a table	Links records between two tables
Uniqueness	Must be unique and not null	Can have duplicate values
NULL values	Not allowed	Can be NULL (if not mandatory)
Table Role	Defined in the own table	Refers to another table's primary key
Example	student_id in Students table	student_id in Marks table referencing Students(student_id)

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

❖ **NOT NULL :**

➔ Prevents a column from storing NULL values.

➔ **Use :** When a column must always have a value (e.g., name, email).

★ **Example :**

```
CREATE TABLE Employees (
   EmpID INT PRIMARY KEY,
   Name VARCHAR(100) NOT NULL,
   Department VARCHAR(50)
);
```

❖ **UNIQUE :**

➔ Ensures that all values in a column are different.

➔ **Use :** To avoid duplicates (e.g., email, username).

★ **Example :**

```
CREATE TABLE Users (
   UserID INT PRIMARY KEY,
   Username VARCHAR(50) UNIQUE,
   Email VARCHAR(100)
);
```

11. Define the SQL Data Definition Language (DDL).

➔ DDL (Data Definition Language) is a subset of SQL used to define and manage the structure of database objects such as tables, views, indexes, schemas, etc.

❖ **Key DDL commands:**

- CREATE – to create a new table or database object.
- ALTER – to modify an existing object.
- DROP – to delete an object.
- TRUNCATE – to delete all data from a table quickly.
- RENAME – to rename a table or object.

12. Explain the CREATE command and its syntax.

➔ The CREATE command is used to create new database objects, such as tables, databases, views, indexes, etc.

★ Syntax

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    ...
);
```

★ Example:

```
CREATE TABLE Students (
    RollNo INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Age INT CHECK (Age >= 18),
    Email VARCHAR(100) UNIQUE
);
```

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

❖ **Purpose of Data Types:**

- Define the kind of data that can be stored in a column (e.g., INT, VARCHAR, DATE).
- Helps the database engine allocate memory and optimize performance.
- Prevents invalid data entry (e.g., entering text in a numeric field).

❖ **Purpose of Constraints:**

- Enforce rules for data integrity and accuracy.
- Prevent errors like duplicate entries or missing mandatory values.
- Maintain relationships between tables (e.g., using FOREIGN KEY).
- Support business logic (e.g., age must be above 18).

14. What is the use of the ALTER command in SQL?

➔ The ALTER command is used to change the structure of an existing table in a database.

➔ The changes made by ALTER are permanent and directly affect the table structure.

★ Common uses of ALTER:

- Add a new column.
- Modify the data type or constraints of a column.
- Rename a column (in some SQL versions).
- Delete (drop) an existing column.
- Add or drop constraints (like PRIMARY KEY, UNIQUE, etc.).

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

A. Add a new column

★ **Syntax**

```
ALTER TABLE table_name
ADD column_name datatype constraint;
```

★ **Example**

```
ALTER TABLE Students
ADD Gender VARCHAR(10);
```

B. Modify an existing column

★ **Syntax**

```
ALTER TABLE table_name
MODIFY column_name new_datatype constraint;
```

★ **Example**

```
ALTER TABLE Students
MODIFY Name VARCHAR(150) NOT NULL;
```

C. Drop a column

★ **Syntax**

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

★ **Example**

```
ALTER TABLE Students
DROP COLUMN Gender;
```

16. 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
- Database
- View
- Index
- Procedure, etc.

★ **Syntax:**

DROP TABLE table_name;

★ **Example:**

DROP TABLE Student;

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

Implication	Explanation
Data Loss	All data in the table is permanently deleted.
Loss of Structure	Table definition (columns, data types, constraints) is removed.
Cascade Effects (if FOREIGN KEYS)	If other tables reference this table using FOREIGN KEY, dropping it may fail or break referential integrity.
Cannot Rollback (in most cases)	DROP is a DDL command and typically cannot be undone.
Dependent Objects are Affected	Views, indexes, or procedures depending on the table may also be invalidated.

18. Define the INSERT, UPDATE, and DELETE commands in SQL

➔ These are Data Manipulation Language (DML) commands used to manage data inside tables:

❖ **INSERT**

➔ Used to add new rows into a table.

★ **Syntax:**

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

★ **Example:**

```
INSERT INTO Students (RollNo, Name, Age)
VALUES (1, 'Keval', 20);
```

❖ **UPDATE**

➔ Used to modify existing records in a table.

★ **Syntax:**

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

★ **Example:**

```
UPDATE Students
SET Age = 21
WHERE RollNo = 1;
```

❖ **DELETE**

➔ Used to remove rows from a table.

★ **Syntax:**

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

★ **Example:**

```
DELETE FROM Students
WHERE RollNo = 1;
```

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

➔ The WHERE clause is crucial in UPDATE and DELETE statements to:

➢ **Target specific rows**

- Prevents updating or deleting all rows unintentionally.

➢ **Ensure accuracy**

- Helps apply changes only to the relevant records.

★ **Without WHERE clause:**

```
-- Updates all records!
UPDATE Students SET Age = 22;

-- Deletes all records!
DELETE FROM Students;
```

20. 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.

➔ It is the most commonly used SQL command for querying data.

★ **Syntax:**

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

★ **Example**:

```
SELECT Name, Age
FROM Students;
```

❖ **To select all columns:**

```
SELECT * FROM Students;
```

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

A. WHERE Clause

➔ The WHERE clause is used to filter rows based on a condition.

★ **Syntax:**

```
SELECT column1, column2
FROM table_name
WHERE condition;
```

★ **Example:**

```
SELECT Name, Age
FROM Students
WHERE Age > 18;
```

B. ORDER BY Clause

➔ The ORDER BY clause is used to sort the result set by one or more columns in ascending (ASC) or descending (DESC) order.

★ **Syntax:**

```
SELECT column1, column2
FROM table_name
ORDER BY column1 ASC|DESC;
```

★ **Example:**

```
SELECT Name, Age
FROM Students
ORDER BY Age DESC;
```

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

➔ **GRANT :-** Gives privileges to users or roles to perform actions (e.g., SELECT, INSERT)

➔ **REVOKE :-** Removes previously granted privileges

23. How do you manage privileges using these commands?

➔ **GRANT – Assign Privileges**

★ Syntax:

```
GRANT privilege_list
ON object_name
TO user_or_role;
```

★ Example:

➔ Grant SELECT and INSERT on the Students table to user keval

```
-> GRANT SELECT, INSERT

ON Students

TO keval;
```

➔ **REVOKE – Remove Privileges**

★ Syntax**:**

```
REVOKE privilege_list

ON object_name

FROM user_or_role;
```

★ Example**:**

➔ Remove INSERT privilege from user keval on Students

```
->REVOKE INSERT

ON Students

FROM keval;
```

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

➔ **COMMIT :-**

- **Purpose:** Save all the changes made during the transaction permanently to the database.
- Once you COMMIT, changes cannot be undone.

★ **Example:**

```
UPDATE Accounts
SET Balance = Balance - 500
WHERE AccountNo = 1001;

COMMIT;  -- Save the deduction
```

➔ **ROLLBACK :-**

- **Purpose:** Undo all changes made during the current transaction and restore the previous state.
- Used when there is an error or you want to discard your work.

★ **Example:**

```
UPDATE Accounts
SET Balance = Balance - 500
WHERE AccountNo = 1001;

ROLLBACK;  -- Cancel the deduction
```

25. Explain how transactions are managed in SQL databases.

➔ A transaction is a sequence of SQL statements that are treated as a single unit of work.

➔ It must be all-or-nothing: either all changes are saved, or none are.

❖ **Transaction Control Workflow**

➔ **Start a Transaction (implicitly):-** Most databases automatically start a transaction when you run a DML statement (INSERT, UPDATE, DELETE).

➔ **Perform Operations:-** Make your changes.

➔ **End the Transaction:**

- Use COMMIT to save changes.
- Use ROLLBACK to undo changes.

➔ **Optional SAVEPOINT:-** Create intermediate points to partially roll back.

★ **Example**

```
BEGIN;

UPDATE Accounts

SET Balance = Balance - 500

WHERE AccountNo = 1001;


UPDATE Accounts

SET Balance = Balance + 500

WHERE AccountNo = 2002;


COMMIT;  -- Save both updates together
```

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

➔ JOIN is used to combine rows from two or more tables based on a related column between them.

➔ This is powerful because it lets you:

- Query data spread across multiple tables
- Build relationships between tables (e.g., customers and orders)

JOIN Type	Description
INNER JOIN	Returns only rows with matching values in both tables.
LEFT JOIN *(LEFT OUTER JOIN)*	Returns all rows from the left table, plus matching rows from the right table. If there's no match, NULLs are returned for the right table's columns.
RIGHT JOIN *(RIGHT OUTER JOIN)*	Returns all rows from the right table, plus matching rows from the left table. If there's no match, NULLs are returned for the left table's columns.
FULL OUTER JOIN	Returns all rows when there is a match in one of the tables. Non-matching rows get NULLs on the side that didn't match.

★ **INNER JOIN**

```
SELECT Customers.Name, Orders.Amount
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

★ **LEFT JOIN**

```
SELECT Customers.Name, Orders.Amount
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

★ **RIGHT JOIN**

```
SELECT Customers.Name, Orders.Amount
FROM Customers
RIGHT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

★ **FULL OUTER JOIN**

```
SELECT Customers.Name, Orders.Amount
FROM Customers
FULL OUTER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

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

➔ Pull together related information from different tables in a single query.
➔ Avoid duplicating data in multiple tables (good normalization).
➔ Create composite views for reporting and analysis.

★ **Example: Combining Customers and Orders**

```
SELECT
   Customers.Name,
   Orders.OrderID,
   Orders.Amount
FROM
   Customers
INNER JOIN Orders
ON
   Customers.CustomerID = Orders.CustomerID;
```

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

➔ GROUP BY is a clause that groups rows having the same values in specified columns into summary rows.

➔ It is commonly used with aggregate functions (like COUNT(), SUM(), AVG(), MIN(), MAX()) to perform calculations for each group of rows.

★ **Syntax:**

```
SELECT column1, AGGREGATE_FUNCTION(column2)
FROM table_name
GROUP BY column1;
```

★ **Example:**

```
SELECT Product, SUM(Quantity) AS TotalQuantity
FROM Sales
GROUP BY Product;
```

29. Explain the difference between GROUP BY and ORDER BY.

Feature	GROUP BY	ORDER BY
Purpose	Groups rows to perform aggregates.	Sorts the result rows in specified order.
Use With	Must be used with aggregate functions to summarize data.	Can be used alone or with GROUP BY.
Effect	Reduces rows into one per group.	Changes display order; does not group rows.
Example	GROUP BY Product to get totals per product.	ORDER BY Product DESC to list products in reverse order.

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

➔ A stored procedure is a precompiled collection of SQL statements (and optional control-of-flow logic) stored in the database under a name.

➔ You can think of it as a program saved inside the database that you can call repeatedly.

Aspect	Stored Procedure	Standard SQL Query
Persistence	Stored in the database and reused.	Written and executed each time manually.
Reusability	Can be called multiple times by applications or users.	Must be re-written or re-executed.
Parameters	Can accept input/output parameters to be dynamic.	Usually static unless you build queries dynamically.
Precompiled	Parsed and optimized once, improving performance.	Parsed and optimized each time you run it.
Logic	Can contain complex logic (loops, IF/ELSE, transactions).	Generally a single command or simple batch.

31. Explain the advantages of using stored procedures.

➔ **Performance Improvement :-** Stored procedures are precompiled, so they execute faster.

➔ **Reusability :-** You can call them many times without rewriting SQL.

➔ **Maintainability :-** Business logic is centralized in one place—easier to update or fix.

➔ **Security :-** You can grant permission to execute the procedure without giving direct access to the underlying tables.

➔ **Reduced Network Traffic :-** Instead of sending many SQL statements from an application, you send a single call to the procedure.

➔ **Modularity :-** Complex operations can be broken into manageable units.

➔ **Support for Transactions :-** Procedures can contain multiple statements with BEGIN TRANSACTION, COMMIT, and ROLLBACK.

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

➔ A view is a virtual table based on the result of an SQL query.

- It does not store data physically.
- It shows data dynamically derived from one or more tables.

★ **Syntax:**

```
CREATE VIEW view_name AS
SELECT column1, column2
FROM table_name
WHERE condition;
```

★ **Example:**

```
CREATE VIEW ActiveCustomers AS
SELECT CustomerID, Name, Email
FROM Customers
WHERE Status = 'Active';
```

Aspect	Table	View
Storage	Physically stores data.	Does not store data—retrieves dynamically.
Data persistence	Data is permanently saved.	Data is generated when you query the view.
Modification	You can insert, update, delete data directly.	Some views are read-only; updates are limited.
Definition	Created with CREATE TABLE.	Created with CREATE VIEW.

33. Explain the advantages of using views in SQL databases.

➔ **Simplify Complex Queries :-** Encapsulate complex joins and calculations into a single object.

➔ **Enhance Security :-** You can restrict access to sensitive columns by exposing only specific data.

➔ **Improve Maintainability :-** Centralize business logic in a view. If logic changes, you only update the view definition.

➔ **Provide Logical Data Independence :-** Underlying table structures can change without affecting applications that use the view.

➔ **Enable Data Abstraction :-** Present data in a user-friendly format, hiding complexity.

➔ **Reusability :-** Use the same view across multiple queries and applications.

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

➔ A trigger is a special stored procedure that automatically executes (fires) in response to a specific event occurring in a table or view.

➔ Triggers are often used to enforce business rules, maintain audit logs, validate data, or synchronize tables.

❖ **Types of Triggers**

Trigger Type	Description
INSERT Trigger	Fires after a new row is inserted into the table.
UPDATE Trigger	Fires after existing data is modified.
DELETE Trigger	Fires after a row is removed from the table.

❖ **When are triggers used?**

- **Auditing:** Record changes automatically (who changed what and when).
- **Validation:** Enforce complex business rules not handled by constraints.
- **Cascading Actions:** Automatically update or delete related data.
- **Preventing Invalid Transactions:** Reject or modify changes before they happen.

35. Explain the difference between INSERT, UPDATE, and DELETE triggers.

Trigger Event	When it fires	Typical Use Cases
INSERT Trigger	After new rows are added.	Log inserted data, auto-create related records, validate inserts.
UPDATE Trigger	After rows are modified.	Track changes, enforce data consistency, prevent certain updates.
DELETE Trigger	After rows are removed.	Archive deleted data, restrict deletion, cascade clean-up.

Example Scenarios:

INSERT Trigger:- Automatically insert a record into an audit table when a new customer is added.

UPDATE Trigger:- Log old and new values of salary changes in an employees table.
DELETE Trigger:- Copy the deleted row to an archive table before deletion.

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

➔ PL/SQL (Procedural Language/SQL) is Oracle's procedural extension to SQL.

➔ While SQL is designed for querying and manipulating data, PL/SQL adds programming features like:

- Variables
- Loops
- Conditions (IF-THEN-ELSE)
- Exception handling
- Procedures and functions

SQL	PL/SQL
Declarative: *What data you want*	Procedural: *How to process the data*
Executes a single statement at a time	Can execute multiple SQL statements together in blocks
No control structures (e.g., IF, LOOP)	Has full programming logic
No error handling within SQL	Has exception handling

37. List and explain the benefits of using PL/SQL.

➔ **Tight Integration with SQL:-** You can seamlessly mix SQL statements with procedural logic.

➔ **Improved Performance :-** Multiple SQL operations are sent to the database in a single block, reducing network traffic.

➔ **Modularity :-** Code can be organized into procedures, functions, and packages for reusability.

➔ **Portability :-** PL/SQL code runs on any Oracle database without changes.

➔ **Error Handling :-** Robust exception handling lets you gracefully manage runtime errors.

➔ **Security :-** Sensitive operations can be encapsulated inside stored procedures or packages, hiding logic from users.

➔ **Maintainability :-** Business rules and processing logic are centralized in the database, making updates easier.

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

➔ Control structures are programming constructs that control the flow of execution of PL/SQL code.

❖ **Types of control structures:**

1. **Conditional control**
    - IF-THEN
    - IF-THEN-ELSE
    - IF-THEN-ELSIF
    - CASE

2. **Iterative control (loops)**
    - LOOP
    - WHILE LOOP
    - FOR LOOP

3. **Sequential control**
    - GOTO
    - EXIT

❖ **IF-THEN Control Structure**

➔ Used to execute statements conditionally.

★ **Syntax:**

```
IF condition THEN

  -- statements

END IF;
```

Extended:

```
IF condition THEN

  -- statements

ELSIF other_condition THEN
```

```
  -- statements
ELSE
  -- statements
END IF;
```

★ **Example:**

```
DECLARE
  v_salary NUMBER := 3000;
BEGIN
  IF v_salary < 5000 THEN
    DBMS_OUTPUT.PUT_LINE('Salary below target.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Salary meets or exceeds target.');
  END IF;
END;
/
```

❖ **LOOP Control Structure**

➔ Used to execute statements repeatedly.

★ **Basic LOOP syntax:**

```
LOOP
  -- statements
  EXIT WHEN condition;
END LOOP;
```

★ **Example:**
Print numbers 1 to 5:

```
DECLARE
  v_counter NUMBER := 1;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE('Counter = ' || v_counter);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 5;
  END LOOP;
END;
/
```

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

➔ Control structures allow you to add intelligence and procedural logic to SQL operations.

- **Dynamic Decision-Making:-** Use IF-THEN to validate conditions and decide which SQL statements to execute.
- **Automated Repetition:-** Use LOOP to process multiple records, perform calculations, or apply logic repeatedly.
- **Error Handling:-** Combine control structures with EXCEPTION blocks to manage errors safely.
- **Complex Business Logic:-** Implement logic like:
    - Conditional updates
    - Batch processing
    - Dynamic data transformations

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

➔ A cursor is a pointer to the context area in memory where Oracle processes and stores the result set of a query.

Feature	Implicit Cursor	Explicit Cursor
Declaration	Automatic by Oracle	Manually declared by the developer
Usage	For single-row queries and DML statements	For multi-row SELECT queries
Control	No need to OPEN, FETCH, CLOSE	Must OPEN, FETCH, CLOSE explicitly
Attributes	Limited (SQL%ROWCOUNT, SQL%FOUND, etc.)	Richer control and more flexibility

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

❖ **You should use an explicit cursor when:**

★ Your query returns multiple rows, and you want to:

- Fetch and process each row individually.
- Control the flow of fetching (e.g., skip some rows, stop early).
- Handle large result sets more efficiently.

❖ **Examples of when explicit cursors are preferred:**

- Generating reports row by row.
- Performing row-wise calculations or updates.
- Building complex procedural logic around data retrieval.

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

➔ SAVEPOINT is a command in SQL that creates a named point within a transaction to which you can later ROLLBACK *partially* without affecting the entire transaction.

❖ **How ROLLBACK and COMMIT interact with SAVEPOINT:**

Command	Effect on Savepoints
SAVEPOINT	Creates a marker (e.g., SAVEPOINT A)
ROLLBACK TO SAVEPOINT	Undoes all changes after the savepoint but keeps the transaction active
ROLLBACK (without TO)	Undoes the entire transaction and removes all savepoints
COMMIT	Finalizes all changes; all savepoints are removed

★ **Example**

```
BEGIN TRANSACTION;

UPDATE Accounts
SET Balance = Balance - 500
WHERE AccountID = 1;

SAVEPOINT deduct_done;

UPDATE Accounts
SET Balance = Balance + 500
WHERE AccountID = 2;

-- Oops, error occurs here or you change your mind:
ROLLBACK TO deduct_done;

-- Only the second update is undone. The first remains.
COMMIT;
```

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

➔ SAVEPOINT is especially useful when you have complex transactions with multiple steps and you want flexible error recovery.

❖ **When to use SAVEPOINT:**

- **Partial Undo:-** If only part of your transaction might fail (e.g., inserting into several related tables), you can undo just that part.

- **Error Handling:-** In PL/SQL blocks or procedures, you can handle errors gracefully by rolling back only specific operations.

- **Complex Business Logic:-** When operations depend on several conditions, savepoints let you roll back selectively.

- **Long Transactions:-** In long-running transactions, savepoints help you avoid losing all progress if something fails late.