Module 4 : Introduction to DBMS

Introduction to SQL

- **THEORY QUESTION:**

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

→

SQL (Structured Query Language) is a standard programming language used to create, manage, and manipulate relational databases like MySQL, PostgreSQL, Oracle, and SQL Server.

What SQL Does

- Create databases and tables
- Insert, update, delete data
- Retrieve data (queries)
- Set relationships between tables
- Control user access

Why SQL Is Essential

- Standard language for all major databases
- Fast and efficient data retrieval
- Maintains data integrity (keys, constraints)
- Provides security and access control
- Used in websites, banking, apps, and enterprise systems

2) Explain the difference between DBMS and RDBMS.

→

A DBMS (Database Management System) is software used to store, manage, and retrieve data. It does not necessarily follow a relational structure and may store data as files or simple tables.

An RDBMS (Relational Database Management System) is an advanced type of DBMS that stores data in structured tables with relationships between them. It uses keys (primary and foreign keys) to maintain relationships and ensures data integrity through constraints.

In short, DBMS focuses on basic data storage, while RDBMS focuses on structured data with relationships and rules.

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

→

Role of SQL in Managing Relational Databases:-

SQL (Structured Query Language) is the core language used to manage and operate relational databases. It is used in relational database systems such as MySQL, PostgreSQL, Oracle Database, and Microsoft SQL Server.

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

SQL is used to **create and modify database structures**, such as:

- Creating databases and tables
- Defining columns and data types
- Setting constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE)

EXAMPLE:-

```
CREATE TABLE Students (
   id INT PRIMARY KEY,
   name VARCHAR(50),
   age INT
);
```

ii. Data Manipulation (DML – Data Manipulation Language)

SQL allows users to insert, update, delete, and retrieve data.

Main commands:

- INSERT
- SELECT
- UPDATE
- DELETE

EXAMPLE:-

SELECT * FROM Students WHERE age > 18;

iii. Managing Relationships

In relational databases, tables are connected using primary keys and foreign keys.SQL enforces these to maintain data integrity and consistency.

EXAMPLE:-

FOREIGN KEY (student_id) REFERENCES Students(id)

iv. Data Control (DCL – Data Control Language)

SQL controls user access and permissions, such as:

- GRANT
- REVOKE

This ensures database security.

v. Transaction Management (TCL – Transaction Control Language)

SQL ensures reliable transactions using commands like:

- COMMIT
- ROLLBACK
- SAVEPOINT

4) What are the key features of SQL?

→

- It is easy to learn and use due to its simple syntax.
- It supports data definition, data manipulation, and data control operations.
- It allows complex queries using conditions, joins, and functions.
- It ensures data integrity through constraints like primary keys and foreign keys.
- It provides security by controlling user access and permissions.
- It is platform-independent and works with most relational database systems.

- **LAB EXERCISES:**

1) Create a new database named school_db and a table called students with the following columns: student_id, student_name, age, class, and address.

→

```
CREATE DATABASE school_db;


USE school_db;


CREATE TABLE students (
   student_id INT PRIMARY KEY,
   student_name VARCHAR(50) NOT NULL,
   age INT,
   class VARCHAR(50),
   address VARCHAR(50)
);
```

2) Insert five records into the students table and retrieve all records using the SELECT statement.

→

```
INSERT INTO students (student_id, student_name, age, class, address)
VALUES
(1, 'Rahul Sharma', 15, '10A', 'Delhi'),
(2, 'Anita Verma', 14, '9B', 'Mumbai'),
(3, 'Karan Singh', 16, '11A', 'Chandigarh'),
(4, 'Priya Mehta', 15, '10B', 'Jaipur'),
(5, 'Arjun Patel', 14, '9A', 'Ahmedabad');


SELECT * FROM students;
```

2. SQL Syntax

- **THEORY QUESTION:**

1] What are the basic components of SQL syntax?

→

SQL syntax defines how SQL statements are written. The main components are:

1. Keywords – Reserved words like SELECT, FROM, WHERE, INSERT, UPDATE, DELETE, CREATE, DROP.
2. Identifiers – Names of database objects (table names, column names).
3. Clauses – Add conditions or details (WHERE, ORDER BY, GROUP BY, HAVING).
4. Expressions – Combine values, columns, and operators (e.g., age + 1).
5. Operators – Arithmetic (+, −), Comparison (=, >), Logical (AND, OR).
6. Data Types – Define type of data (INT, VARCHAR, DATE).
7. Statements – Complete commands (DDL, DML, DCL, TCL).

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

→

General Structure of an SQL SELECT Statement

The basic syntax of a SELECT statement in SQL is:

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

Explanation of Each Clause:

- **SELECT** → Specifies the columns to retrieve
- **FROM** → Specifies the table to fetch data from
- **WHERE** → Filters records based on a condition
- **GROUP BY** → Groups rows with the same values
- **HAVING** → Filters grouped records
- **ORDER BY** → Sorts the result (Ascending or Descending)

Simplest Form:

SELECT * FROM table_name;

This retrieves all columns from the specified table.

3] Explain the role of clauses in SQL statements.

→ Role of Clauses in SQL Statements

In SQL, clauses are parts of a statement that perform specific functions. They define conditions, organize data, and control how results are returned. Clauses make SQL statements more powerful and flexible.

i. FROM Clause

- Specifies the table(s) from which data is retrieved.
- It is the starting point of most SELECT queries.

SELECT * FROM students;

ii. WHERE Clause

- Filters records based on given conditions.
- Returns only the rows that satisfy the condition.

```
SELECT * FROM students WHERE age > 14;
```

iii. GROUP BY Clause

- Groups rows that have the same values in specified columns.
- Often used with aggregate functions like COUNT, SUM, AVG, etc.

```
SELECT class, COUNT(*)
FROM students
GROUP BY class;
```

iv. HAVING Clause

- Filters grouped records (used with GROUP BY).
- Applied after grouping.

```
SELECT class, COUNT(*)
FROM students
GROUP BY class
HAVING COUNT(*) > 2;
```

v. ORDER BY Clause

- Sorts the result set in ascending (ASC) or descending (DESC) order.

```
SELECT * FROM students
ORDER BY age DESC;
```

vi. LIMIT Clause (in systems like MySQL)

- Restricts the number of rows returned.

```
SELECT * FROM students
LIMIT 5;
```

- **LAB EXERCISES:**

1] Write SQL queries to retrieve specific columns (student_name and age) from the students table.

→

```
SELECT student_name, age
FROM students;
```

✅ **Explanation:**

- **SELECT student_name, age** → Retrieves only these two columns
- **FROM students** → Specifies the table from which data is fetched

2] Write SQL queries to retrieve all students whose age is greater than 10.

→

```
SELECT *
FROM students
WHERE age > 10;
```

✅ **Explanation:**

- **SELECT *** → Retrieves all columns
- **FROM students** → Specifies the table
- **WHERE age > 10** → Filters records where age is greater than 10

3. SQL Constraints

- **THEORY QUESTION:**

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

→

Constraints in SQL are rules applied to table columns to ensure the accuracy, validity, and integrity of data stored in a database.
They prevent invalid data from being inserted into tables.

Constraints are defined at the time of table creation using the CREATE TABLE statement or later using ALTER TABLE.

Types of Constraints in SQL

i. NOT NULL

- Ensures that a column cannot have a NULL (empty) value.
- Every record must contain a value in that column.

student_name VARCHAR(50) NOT NULL

ii. UNIQUE

- Ensures that all values in a column are different.
- No duplicate values are allowed.

email VARCHAR(100) UNIQUE

iii. PRIMARY KEY

- Uniquely identifies each record in a table.
- Cannot contain NULL values.
- Combines **NOT NULL + UNIQUE**.

student_id INT PRIMARY KEY

iv. FOREIGN KEY

- Creates a link between two tables.
- Ensures referential integrity.
- The value must match a value in the referenced table.

FOREIGN KEY (class_id) REFERENCES classes(class_id)

v. CHECK

- Ensures that values in a column satisfy a specific condition.

age INT CHECK (age >= 5)

vi. DEFAULT

- Assigns a default value if no value is provided.

city VARCHAR(50) DEFAULT 'Delhi'

Constraints are rules applied to table columns to maintain data accuracy and integrity.
The main types are:
NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, and DEFAULT.

2] How do PRIMARY KEY and FOREIGN KEY constraints differ?

→

i. PRIMARY KEY

- Uniquely identifies each record in a table
- Cannot contain **NULL** values
- Must contain **unique** values
- Only **one primary key** per table (can be composite)
- Ensures **entity integrity**

Example:

```
CREATE TABLE students (
   student_id INT PRIMARY KEY,
   student_name VARCHAR(50)
);
```

ii. FOREIGN KEY

- Creates a relationship between two tables
- Refers to the **PRIMARY KEY** in another table
- Can contain duplicate values
- Can contain NULL (unless restricted)
- Ensures **referential integrity**

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

→

i. NOT NULL

- Ensures that a column cannot have a NULL (empty) value.
- Every record must contain a value in that column.

student_name VARCHAR(50) NOT NULL

ii. UNIQUE

- Ensures that all values in a column are different.
- No duplicate values are allowed.

email VARCHAR(100) UNIQUE

- **LAB EXERCISES:**

1] Create a table teachers with the following columns: teacher_id (Primary Key), teacher_name (NOT NULL), subject (NOT NULL), and email (UNIQUE).

→

```
CREATE TABLE teachers (
    teacher_id INT PRIMARY KEY,
    teacher_name VARCHAR(100) NOT NULL,
    subject VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE
);
```

2] Implement a FOREIGN KEY constraint to relate the teacher_id from the teachers table with the students table.

→

```
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(50) NOT NULL,
    age INT,
    class VARCHAR(50),
    address VARCHAR(50),
    teacher_id INT,
    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id));
```

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

- **THEORY QUESTION:**

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

→

SQL Data Definition Language (DDL) is a set of SQL commands used to **define, modify, and manage the structure of database objects** such as tables, databases, indexes, and constraints.

DDL commands deal with the **schema (structure)** of the database, not the actual data stored inside it.

Command	Purpose
CREATE	Creates database objects (database, table, index, etc.)
ALTER	Modifies existing database objects
DROP	Deletes database objects
TRUNCATE	Removes all records from a table (structure remains)
RENAME	Renames a database object

Key Points

- DDL changes the **structure** of the database.
- DDL commands are usually **auto-committed** (cannot be rolled back in most DBMS).
- It defines objects like:
  - Databases
  - Tables
  - Constraints
  - Views
  - Indexes

2] Explain the CREATE command and its syntax.

→

The **CREATE** command is a **DDL (Data Definition Language)** command used to **create new database objects** such as databases, tables, views, indexes, and constraints.

It defines the **structure** of the object in the database.

📌 **Purpose of CREATE**

- To create a new **database**
- To create a new **table**
- To create **views**
- To create **indexes**
- To create other database objects

✅ Syntax of CREATE Command

i. Create Database

CREATE DATABASE database_name;

Example:

CREATE DATABASE school_db;

ii. Create Table

CREATE TABLE table_name (

column1 datatype constraint,

column2 datatype constraint,

column3 datatype constraint,

...

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

→ When creating a table in SQL, **data types** and **constraints** are defined to ensure the database stores data correctly, efficiently, and securely.

i. Purpose of Data Types

A **data type** defines the kind of data a column can store.

❖ **Why Data Types Are Important:**

- Ensure correct type of data is stored (e.g., numbers, text, dates)
- Prevent invalid data entry
- Improve storage efficiency
- Improve query performance

Example:

age INT

student_name VARCHAR(50)

- INT → Stores only numbers
- VARCHAR(50) → Stores text up to 50 characters

ii. Purpose of Constraints

Constraints are rules applied to columns to maintain **data accuracy and integrity**.

❖ **Why Constraints Are Important:**

- Prevent duplicate values (PRIMARY KEY, UNIQUE)
- Prevent null (empty) values (NOT NULL)
- Maintain relationships between tables (FOREIGN KEY)
- Enforce valid data conditions (CHECK)
- Provide default values (DEFAULT)

- **LAB EXERCISES:**

1] Create a table courses with columns: course_id, course_name, and course_credits. Set the course_id as the primary key.

→

```
CREATE TABLE courses (
   course_id INT PRIMARY KEY,
   course_name VARCHAR(100) NOT NULL,
   course_credits INT
);
```

2] Use the CREATE command to create a database university_db.

→

```
CREATE DATABASE university_db;
```

🔎 **Explanation:**

- CREATE DATABASE → DDL command used to create a new database.
- university_db → Name of the database being created.

5. ALTER Command

- **THEORY QUESTION:**

1] What is the use of the ALTER command in SQL?

→ The **ALTER** command is a **DDL (Data Definition Language)** command used to **modify the structure of an existing database object**, such as a table.

It allows you to change the design of a table **after it has already been created**.

📌 **Why We Use ALTER**

You use ALTER when you need to:

- Add a new column
- Delete an existing column
- Modify a column's data type or size
- Add or remove constraints
- Rename a column or table

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

→

i. **Add a Column**

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

ii. **Modify a Column**

```
ALTER TABLE table_name
MODIFY column_name datatype;
```

iii. **Drop a Column**

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

- **LAB EXERCISES:**

1] Modify the courses table by adding a column course_duration using the ALTER command.

→

```
ALTER TABLE courses
ADD course_duration VARCHAR(50);
```

Explanation:

- ALTER TABLE courses → Specifies the table you want to modify.
- ADD course_duration VARCHAR(50) → Adds a new column named course_duration with a data type of VARCHAR(50).

2] Drop the course_credits column from the courses table.

→

```
ALTER TABLE courses
DROP COLUMN course_credits;
```

6. DROP Command

- **THEORY QUESTION:**

1] What is the function of the DROP command in SQL?

→

The **DROP** command is a **DDL (Data Definition Language)** command used to **permanently remove database objects** such as **tables, databases, views, or indexes** from the database.

It **deletes the object and all its data**, so it cannot be recovered unless you have a backup.

📌 **Uses of DROP**

- Delete an entire **table**
- Delete a **database**
- Delete a **view**
- Delete an **index**

Syntax Examples

i. Drop a Table

DROP TABLE table_name;

Example:

DROP TABLE courses;

This deletes the courses table and all its data permanently.

ii. Drop a Database

DROP DATABASE database_name;

Example:

DROP DATABASE university_db;

This deletes the university_db and all its tables permanently.

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

→

Dropping a table from a database using SQL's DROP TABLE command has significant and often irreversible consequences.

i. Permanent Data Loss

- All rows in the table are **deleted permanently**.
- There is **no automatic undo** unless you restore from a backup.

DROP TABLE students;

ii. Table Structure is Deleted

- The table's columns, data types, constraints, and indexes are removed.
- Recreating the table requires defining its structure from scratch.

iii. Dependent Objects May Break

- Any views, triggers, stored procedures, or foreign keys that reference the table may fail.
- Example: If a teachers table references students via a foreign key, dropping students can cause errors.

iv. Cannot Usually Be Rolled Back

- Most DBMS treat DROP TABLE as an auto-commit operation, so it cannot be undone in a transaction.
- Recovery requires restoring from a backup.

- **LAB EXERCISES:**

1] Drop the teachers table from the school_db database.

→

-- Select the database

USE school_db;

-- Drop the teachers table

DROP TABLE teachers;

2] Drop the students table from the school_db database and verify that the table has been removed.

→

Step 1: Select the Database

USE school_db;

Step 2: Drop the students Table

DROP TABLE students;

Step 3: Verify the Table Has Been Removed

SHOW TABLES;

If students no longer appears in the list, it has been successfully dropped.

7. Data Manipulation Language (DML)

- **THEORY QUESTION:**

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

→

i. INSERT Command

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

✅ **Syntax:**

```
INSERT INTO table_name (column1, column2, column3)
VALUES (value1, value2, value3);
```

✅ **Example:**

```
INSERT INTO students (student_id, student_name, age)
VALUES (6, 'Aman Khan', 15);
```

ii. UPDATE Command

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

✅ **Syntax:**

```
UPDATE table_name
SET column_name = value
WHERE condition;
```

iii. DELETE Command

The **DELETE** command is used to remove 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 very important in UPDATE and DELETE statements because it specifies which records should be modified or removed.

Without the WHERE clause, all records in the table will be affected, which can cause serious data loss or unwanted changes.

🔹 **Importance of WHERE in UPDATE**

The WHERE clause ensures that only specific rows are updated.

✅ **Example With WHERE:**

UPDATE students

SET age = 16

WHERE student_id = 3;

Only the student with student_id = 3 will be updated.

✅ **Example With WHERE:**

DELETE FROM students

WHERE student_id = 3;

Only one student record is deleted.

- **LAB EXERCISES:**

1] Insert three records into the courses table using the INSERT command.

→

To add new records into the courses table, we use the **INSERT** command.

```
INSERT INTO courses (course_id, course_name, course_duration)
VALUES
(101, 'Mathematics', '6 Months'),
(102, 'Computer Science', '1 Year'),
(103, 'Physics', '6 Months');
```

2] Update the course duration of a specific course using the UPDATE command.

→

To modify the duration of a particular course, we use the UPDATE command with a WHERE clause.

```
UPDATE courses
SET course_duration = '1 Year'
WHERE course_id = 101;
```

3] Delete a course with a specific course_id from the courses table using the DELETE command.

→

To remove a particular course from the courses table, we use the DELETE command with a WHERE clause.

```
DELETE FROM courses
WHERE course_id = 102;
```

8. Data Query Language (DQL)

- **THEORY QUESTION:**

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

→

The **SELECT** statement is a DML (Data Manipulation Language) command used to **retrieve data from a database table**.

It is the most commonly used SQL command because it allows users to **query, filter, sort, and analyze data**.

- **Basic Syntax**

SELECT column1, column2

FROM table_name;

i. **Retrieve All Columns**

SELECT * FROM courses;

ii. **Retrieve Specific Columns**

SELECT course_name, course_duration

FROM courses;

iii. **Using WHERE Clause (Filtering)**

SELECT *

FROM courses

WHERE course_id = 101;

iv. Using ORDER BY (Sorting)

```
SELECT *
FROM courses
ORDER BY course_name ASC;
```

v. Using GROUP BY (Grouping)

```
SELECT course_duration, COUNT(*)
FROM courses
GROUP BY course_duration;
```

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

→

Both **WHERE** and **ORDER BY** clauses are used with the **SELECT** statement to control how data is retrieved and displayed.

i. WHERE Clause

The **WHERE clause** is used to **filter records** based on a specified condition.

It returns only the rows that satisfy the given condition.

✅ **Syntax:**

```
SELECT column_name
FROM table_name
WHERE condition;
```

✅ **Example:**

```
SELECT *
FROM students
WHERE age > 15;
```

ii. ORDER BY Clause

The **ORDER BY clause** is used to **sort the result set** in ascending (ASC) or descending (DESC) order.

✅ **Syntax:**

```
SELECT column_name
FROM table_name
ORDER BY column_name ASC | DESC;
```

✅ **Example:**

```
SELECT *
FROM students
ORDER BY age DESC;
```

- **LAB EXERCISES:**

1] Retrieve all courses from the courses table using the SELECT statement.

→

To retrieve all records from the courses table, we use the SELECT statement with *.

SELECT * FROM courses;

2] Sort the courses based on course_duration in descending order using ORDER BY.

→

To sort records in descending order, we use the ORDER BY clause with DESC.

SELECT *

FROM courses

ORDER BY course_duration DESC;

3] Limit the results of the SELECT query to show only the top two courses using LIMIT.

→ To restrict the number of rows returned in a query, we use the LIMIT clause.

SELECT *

FROM courses

LIMIT 2;

9. Data Control Language (DCL)

- **THEORY QUESTION:**

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

→

GRANT and **REVOKE** are SQL commands used to **control user access and permissions** in a database.

They belong to **DCL (Data Control Language)**.

These commands help maintain **database security and access control**.

i. GRANT Command

The **GRANT** command is used to give specific permissions to users.

✅ **Syntax:**

```
GRANT permission
ON table_name
TO user_name;
```

ii. REVOKE Command

The **REVOKE** command is used to remove previously granted permissions from a user.

✅ **Syntax:**

```
REVOKE permission
ON table_name
FROM user_name;
```

2] How do you manage privileges using these commands?

→

Privileges in SQL are managed using the **GRANT** and **REVOKE** commands.
These commands control **who can access what data** and **what actions they can perform**.

i. **Granting Privileges (Using GRANT)**

To give specific permissions to a user.

✅ **Syntax:**

GRANT privilege

ON object_name

TO user_name;

ii. **Revoking Privileges (Using REVOKE)**

To remove previously granted permissions.

✅ **Syntax:**

REVOKE privilege

ON object_name

FROM user_name;

iii. **Granting All Privileges**

GRANT ALL PRIVILEGES

ON courses

TO user1;

Gives full control over the table.

iv. **Revoking All Privileges**

REVOKE ALL PRIVILEGES

ON courses

FROM user1;

Removes all permissions from the user.

- **LAB EXERCISES:**

1] Create two new users user1 and user2 and grant user1 permission to SELECT from the courses table.

→

Step 1️⃣ Create Users

✅ **Syntax:**

CREATE USER 'username'@'host' IDENTIFIED BY 'password';

✅ **Example (MySQL):**

CREATE USER 'user1'@'localhost' IDENTIFIED BY 'pass123';

CREATE USER 'user2'@'localhost' IDENTIFIED BY 'pass123';

Step 2️⃣ Grant SELECT Permission to user1

✅ **Syntax:**

GRANT SELECT

ON table_name

TO 'username'@'host';

✅ **Query:**

GRANT SELECT

ON school_db.courses

TO 'user1'@'localhost';

Step 3️⃣ Apply Changes (MySQL Only)

FLUSH PRIVILEGES;

✓ Reloads privilege settings.

2] Revoke the INSERT permission from user1 and give it to user2.

→

1️⃣ Remove INSERT permission from user1
2️⃣ Grant INSERT permission to user2

Step 1️⃣ Revoke INSERT from user1

✅ **Query:**

```
REVOKE INSERT
ON school_db.courses
FROM 'user1'@'localhost';
```

✓ user1 can no longer insert records into the courses table.

Step 2️⃣ Grant INSERT to user2

✅ **Query:**

```
GRANT INSERT
ON school_db.courses
TO 'user2'@'localhost';
```

✓ user2 can now insert records into the courses table.

Step 3️⃣ (MySQL Only)

```
FLUSH PRIVILEGES;
```

✓ Applies the updated privileges.

10. Transaction Control Language (TCL)

- **THEORY QUESTION:**

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

→

COMMIT and **ROLLBACK** are part of **TCL (Transaction Control Language)**.

They are used to manage **transactions** in a database.

A **transaction** is a group of SQL operations treated as a single unit of work.

i. COMMIT Command

The **COMMIT** command is used to permanently save all changes made during a transaction.

Once committed:

- Changes are saved in the database
- They cannot be undone

✅ **Syntax:**

```
COMMIT;
```

ii. ROLLBACK Command

The **ROLLBACK** command is used to undo changes made during a transaction.

It restores the database to its previous state before the transaction began.

✅ **Syntax:**

```
ROLLBACK;
```

2] Explain how transactions are managed in SQL databases.

→

A **transaction** in SQL is a group of one or more SQL statements executed as a single unit of work.

Transactions ensure that database operations are completed **safely and consistently**.

1️⃣ What is a Transaction?

A transaction may include:

- INSERT
- UPDATE
- DELETE

i. BEGIN / START TRANSACTION

Starts a transaction.

START TRANSACTION;

ii. COMMIT

Permanently saves all changes.

COMMIT;

iii. ROLLBACK

Cancels all changes made during the transaction.

ROLLBACK;

iv. SAVEPOINT

Creates a point inside a transaction to roll back to.

SAVEPOINT sp1;

Rollback to savepoint:

ROLLBACK TO sp1;

- **LAB EXERCISES:**

1] Insert a few rows into the courses table and use COMMIT to save the changes.

→ START TRANSACTION;

INSERT INTO courses (course_id, course_name, course_duration)

VALUES

(201, 'Chemistry', '6 Months'),

(202, 'Biology', '1 Year'),

(203, 'English', '3 Months');

COMMIT;

2] Insert additional rows, then use ROLLBACK to undo the last insert operation.

→ START TRANSACTION;

INSERT INTO courses (course_id, course_name, course_duration)

VALUES

(204, 'History', '6 Months'),

(205, 'Geography', '1 Year');

ROLLBACK;

3] Create a SAVEPOINT before updating the courses table, and use it to roll back specific changes.

→. START TRANSACTION;

SAVEPOINT before_update;

UPDATE courses

SET course_duration = '2 Years'

WHERE course_id = 201;

ROLLBACK TO before_update;

COMMIT;

11. SQL Joins

- **THEORY QUESTION:**

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

→

A **JOIN** in SQL is used to combine rows from **two or more tables** based on a related column between them.

It is mainly used when data is stored in multiple tables and we need to retrieve related information together.

i. INNER JOIN

Returns only the rows that have matching values in both tables.

✅ **Syntax:**

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

ii. LEFT JOIN (LEFT OUTER JOIN)

Returns all records from the left table, and matching records from the right table.
If no match, NULL is returned.

iii. RIGHT JOIN (RIGHT OUTER JOIN)

Returns all records from the right table, and matching records from the left table.
If no match, NULL is returned.

iv. FULL OUTER JOIN

Returns all records from both tables.
If there is no match, NULL is returned on the missing side.

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

→

In a relational database, data is stored in **separate tables** to reduce redundancy.
To retrieve related data from these tables, we use **JOIN**.

A JOIN combines rows from two or more tables based on a related column (usually a **Primary Key** and **Foreign Key**).

- **Using JOIN to Combine Data**

```
SELECT students.student_name, enrollments.course_name
FROM students
INNER JOIN enrollments
ON students.student_id = enrollments.student_id;
```

- **Joining More Than Two Tables**

```
SELECT s.student_name, e.course_name, c.course_duration
FROM students s
JOIN enrollments e ON s.student_id = e.student_id
JOIN courses c ON e.course_name = c.course_name;
```

- **LAB EXERCISES:**

1] Create two tables: departments and employees. Perform an INNER JOIN to display employees along with their respective departments.

→

✅ **Step 1: Create departments Table**

```
CREATE TABLE departments (
   department_id INT PRIMARY KEY,
   department_name VARCHAR(50)
);
```

✅ **Step 2: Create employees Table**

```
CREATE TABLE employees (
   employee_id INT PRIMARY KEY,
   employee_name VARCHAR(50),
   department_id INT,
   FOREIGN KEY (department_id) REFERENCES departments(department_id)
);
```

✅ **Step 3: Insert Sample Data**

```
INSERT INTO departments VALUES
(1, 'HR'),
(2, 'IT'),
(3, 'Finance');
INSERT INTO employees VALUES
(101, 'Rahul', 2),
(102, 'Anita', 1),
(103, 'Karan', 2),
(104, 'Meena', 3);
```

✅ **Step 4: Perform INNER JOIN**

```
SELECT employees.employee_name, departments.department_name
FROM employees
INNER JOIN departments
ON employees.department_id = departments.department_id;
```

2] Use a LEFT JOIN to show all departments, even those without employees.

→SELECT departments.department_name, employees.employee_name
FROM departments
LEFT JOIN employees
ON departments.department_id = employees.department_id;

department_name	employee_name
HR	Anita
IT	Rahul
IT	Karan
Finance	Meena
Marketing	NULL

12. SQL Group By

- **THEORY QUESTION:**

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

→

The **GROUP BY** clause is used to **group rows that have the same values** in specified columns.

It is mainly used with **aggregate functions** to perform calculations on each group of data.

Suppose we have an employees table:

employee_id	employee_name	department	salary
1	Rahul	IT	50000
2	Anita	HR	40000
3	Karan	IT	60000
4	Meena	HR	45000

If we want:

- Total salary of each department
- Average salary per department
- Number of employees in each department

◆ **Syntax**

SELECT column_name, aggregate_function(column_name)

FROM table_name

GROUP BY column_name;

🔹 **Common Aggregate Functions**

Function	Purpose
COUNT()	Counts rows
SUM()	Adds values
AVG()	Calculates average
MAX()	Finds maximum
MIN()	Finds minimum

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

→

Both clauses are used in SELECT queries, but they serve **different purposes**.

🔹 1️⃣ **GROUP BY**

Used to **group rows with similar values** into summary rows.

✓ **Mainly used with:**

Aggregate functions like:

- COUNT()
- SUM()
- AVG()
- MAX()
- MIN()

🔎 **Example:**

```
SELECT department, COUNT(*) AS total_employees
FROM employees
GROUP BY department;
```

🔹 2️⃣ **ORDER BY**

📌 **Purpose:**

Used to **sort the result** in ascending or descending order.

🔎 **Example:**

SELECT employee_name, salary

FROM employees

ORDER BY salary DESC;

- **LAB EXERCISES:**

1] Group employees by department and count the number of employees in each department using GROUP BY.

→

SELECT department_id, COUNT(*) AS total_employees

FROM employees

GROUP BY department_id;

2] Use the AVG aggregate function to find the average salary of employees in each department.

→

SELECT department_id, AVG(salary) AS average_salary

FROM employees

GROUP BY department_id;

What This Query Does

1. Groups employees by department_id

2. Calculates the average salary inside each group
3. Returns one row per department

🔎 **Example**

Suppose employees table contains:

employee_id	employee_name	department_id	salary
101	Rahul	2	50000
102	Anita	1	40000
103	Karan	2	60000
104	Meena	3	45000
105	Arjun	2	55000

✅ **Output Will Be**

department_id	average_salary
1	40000
2	55000
3	45000

13. SQL Stored Procedure

- **THEORY QUESTION:**

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

→

A **Stored Procedure** is a **pre-written set of SQL statements** that is saved inside the database and can be executed whenever needed.

🔹 **Simple Example of Stored Procedure**

```
DELIMITER //

CREATE PROCEDURE GetAllEmployees()
BEGIN
   SELECT * FROM employees;
END //

DELIMITER ;
```

🔹 **Execute It**

```
CALL GetAllEmployees();
```

✅ **What is a Standard SQL Query?**

A standard SQL query is a **single SQL statement** written and executed directly.

```
SELECT * FROM employees;
```

✓ It runs immediately
✓ It is not saved permanently in the database

2] Explain the advantages of using stored procedures.

→

Advantages of Stored Procedure

- **Reusability** – Write once, use many times.
- **Better Performance** – Precompiled, so faster execution.
- **Improved Security** – Users can execute it without direct table access.
- **Reduced Network Traffic** – Only one procedure call is sent instead of many queries.
- **Centralized Logic** – Business rules stored in one place.
- **Programming Features** – Supports variables, conditions, loops, and error handling.
- **Easy Maintenance** – Organized and easier to manage complex tasks.

- **LAB EXERCISES:**

1] Write a stored procedure to retrieve all employees from the employees table based on department.

→

```
DELIMITER //

CREATE PROCEDURE GetEmployeesByDepartment(IN dept_id INT)
BEGIN
   SELECT *
   FROM employees
   WHERE department_id = dept_id;
END //

DELIMITER ;
```

- **How to Execute the Procedure**

CALL GetEmployeesByDepartment(2);

2] Write a stored procedure that accepts course_id as input and returns the course details.

→

This procedure accepts a course_id as input and returns the details of that course from the courses table.

```
DELIMITER //

CREATE PROCEDURE GetCourseById(IN p_course_id INT)
BEGIN
    SELECT *
    FROM courses
    WHERE course_id = p_course_id;
END //

DELIMITER ;
```

- **How to Execute It**

CALL GetCourseById(101);

This will return the course whose course_id = 101.

14. SQL View

- **THEORY QUESTION:**

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

→

A View in SQL is a virtual table created from a SELECT query.

It does not store data physically.
Instead, it displays data from one or more tables whenever you query it.

- **Syntax to Create a View**

CREATE VIEW view_name AS

SELECT column1, column2

FROM table_name

WHERE condition;

- **Example**

Suppose we have an employees table:

employee_id	employee_name	salary	department_id

Create a View

CREATE VIEW IT_Employees AS

SELECT employee_id, employee_name, salary

FROM employees

WHERE department_id = 2;

Now you can use:

SELECT * FROM IT_Employees;

✓ It will show only IT department employees
✓ Data comes from original table

🔥 **Key Differences Explained**

i. **Storage**

- Table → Stores actual data.
- View → Does not store data, just shows data.

ii. **Data Modification**

- Table → You can insert, update, delete freely.
- View → Sometimes updatable (depends on complexity).

iii. **Purpose**

- Table → Permanent data storage.
- View → Data presentation, filtering, security.

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

Advantages of Views

- **Improved Security** – Restricts access to sensitive columns or rows (e.g., hide salary).
- **Simplifies Complex Queries** – Saves complex JOINs; users can query the view directly.
- **Data Abstraction** – Hides internal table structure from users.
- **Reusability** – Write once, use multiple times.
- **Consistency** – Ensures standardized data format for all users.
- **Logical Data Independence** – Changes in tables won't affect users if the view remains the same.

- **LAB EXERCISES:**

1] Create a view to show all employees along with their department names.

→

To display employees along with their department names, we create a **VIEW** using a JOIN between the employees and departments tables.

```
CREATE VIEW Employee_Department_View AS
SELECT e.employee_id,
    e.employee_name,
    d.department_name
FROM employees e
JOIN departments d
ON e.department_id = d.department_id;
```

🔹 **How to Use the View**

```
SELECT * FROM Employee_Department_View;
```

🔎 **What This View Does**

- Combines data from employees and departments
- Matches records using department_id
- Shows employee details along with department names
- Saves the JOIN query for reuse

✅ **Example Output**

employee_id	employee_name	department_name
101	Rahul	IT
102	Anita	HR
103	Karan	IT
104	Meena	Finance

2] Modify the view to exclude employees whose salaries are below $50,000.

→ **To modify an existing view, we use:**

- CREATE OR REPLACE VIEW (MySQL / PostgreSQL)
  OR
- ALTER VIEW (SQL Server)

```
CREATE OR REPLACE VIEW Employee_Department_View AS
SELECT e.employee_id,
    e.employee_name,
    e.salary,
    d.department_name
FROM employees e
JOIN departments d
ON e.department_id = d.department_id
WHERE e.salary >= 50000;
```

🔎 **What This Does**

- Joins employees and departments
- Filters employees with salary **$50,000 or more**
- Excludes employees earning less than 50,000
- Updates the existing view

🔹 **How to Check the View**

```
SELECT * FROM Employee_Department_View;
```

✅ **Example Output**

employee_id	employee_name	salary	department_name
101	Rahul	55000	IT
103	Karan	60000	IT

Employees earning less than 50,000 will not appear.

15. SQL Triggers

- **THEORY QUESTION:**

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

→

A **Trigger** in SQL is a special type of stored program that **automatically executes (fires)** when a specific event occurs on a table.

It is attached to a table and runs when:

- INSERT
- UPDATE
- DELETE

operations happen.

👉 Triggers are used to enforce rules, maintain logs, and ensure data integrity automatically.

🔹 **Basic Syntax (MySQL)**

```
CREATE TRIGGER trigger_name
BEFORE INSERT
ON table_name
FOR EACH ROW
BEGIN
   -- SQL statements
END;
```

🔹 **Types Combined**

So we can have:

- BEFORE INSERT
- AFTER INSERT

- BEFORE UPDATE
- AFTER UPDATE
- BEFORE DELETE
- AFTER DELETE

✅ **Example: BEFORE INSERT Trigger**

```
CREATE TRIGGER check_salary
BEFORE INSERT
ON employees
FOR EACH ROW
BEGIN
   IF NEW.salary < 0 THEN
      SET NEW.salary = 0;
   END IF;
END;
```

Ensures salary cannot be negative.

✅ **Example: AFTER DELETE Trigger**

```
CREATE TRIGGER log_delete
AFTER DELETE
ON employees
FOR EACH ROW
BEGIN
   INSERT INTO employee_log(employee_id, action)
   VALUES (OLD.employee_id, 'Deleted');
END;
```

Logs deleted records.

When Are Triggers Used?

1. Enforcing business rules
2. Maintaining audit logs
3. Automatic calculations
4. Data validation
5. Preventing invalid operations

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

→

1️⃣ INSERT Trigger

Fires: When a new row is inserted.
Used for: Validating data, setting default values, logging new records.
Keyword: NEW → Refers to the newly inserted row.

2️⃣ UPDATE Trigger

Fires: When a row is modified.
Used for: Tracking changes, logging old/new values, preventing invalid updates.
Keywords:

- OLD → Previous value
- NEW → Updated value

3️⃣ DELETE Trigger

Fires: When a row is deleted.
Used for: Audit records, backup of deleted data, enforcing rules.
Keyword: OLD → Refers to deleted row data.

- **LAB EXERCISES:**

1] Create a trigger to automatically log changes to the employees table when a new employee is added.

→

Create a trigger to automatically log changes to the employees table when a new employee is added.

- **Step 1: Create Log Table**

```
CREATE TABLE employee_log (
   log_id INT AUTO_INCREMENT PRIMARY KEY,
   employee_id INT,
   employee_name VARCHAR(50),
   action VARCHAR(50),
   log_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- **Step 2: Create INSERT Trigger**

```
DELIMITER //
CREATE TRIGGER after_employee_insert
AFTER INSERT
ON employees
FOR EACH ROW
BEGIN
   INSERT INTO employee_log (employee_id, employee_name, action)
   VALUES (NEW.employee_id, NEW.employee_name, 'Employee Added');
END //
DELIMITER ;
```

🔹 **Step 3: Test the Trigger**

INSERT INTO employees (employee_id, employee_name, department_id, salary)

VALUES (110, 'Arjun', 2, 55000);

Now check:

SELECT * FROM employee_log;

✅ Example Output in employee_log

log_id	employee_id	employee_name	action	log_time
1	110	Arjun	Employee Added	2026-02-21 10:30:00

2] Create a trigger to update the last_modified timestamp whenever an employee record is updated.

→

1️⃣ Make sure the employees table has a last_modified column
2️⃣ Create a BEFORE UPDATE trigger to automatically update it

🔹 **Step 1: Add last_modified Column (If Not Exists)**

ALTER TABLE employees

ADD COLUMN last_modified TIMESTAMP DEFAULT CURRENT_TIMESTAMP;

🔹 **Step 2: Create UPDATE Trigger**

DELIMITER //

```
CREATE TRIGGER before_employee_update
BEFORE UPDATE
ON employees
FOR EACH ROW
BEGIN
   SET NEW.last_modified = CURRENT_TIMESTAMP;
END //

DELIMITER ;
```

- **Step 3: Test the Trigger**

```
UPDATE employees
SET salary = 60000
WHERE employee_id = 101;
```

Now check:

```
SELECT employee_id, salary, last_modified
FROM employees
WHERE employee_id = 101;
```

16. Introduction to PL/SQL

- **THEORY QUESTION:**

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

→ PL/SQL (Procedural Language/SQL) is a procedural extension of SQL developed by Oracle.
SQL is used to query and modify data, while PL/SQL adds programming features to SQL.

How PL/SQL Extends SQL

PL/SQL adds:

- Variables – Store values inside programs.
- Conditional Statements (IF–ELSE) – Make decisions in code.
- Loops – Repeat statements multiple times.
- Exception Handling – Handle errors properly.
- Stored Programs – Create Procedures, Functions, Triggers, and Packages.

Structure of PL/SQL Block

```
DECLARE
  -- variable declaration
BEGIN
  -- executable statements
EXCEPTION
  -- error handling
END;
```

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

→

PL/SQL (Procedural Language/SQL) is a procedural extension of SQL developed by **Oracle Corporation** for the **Oracle Database**.
It combines SQL with programming features to build powerful database applications.

- Better Performance – Executes multiple statements together and reduces network traffic.

- Programming Features – Supports variables, loops, IF–ELSE, and exception handling.

- Error Handling – Can handle errors like NO_DATA_FOUND and ZERO_DIVIDE.

- Code Reusability – Create procedures, functions, and packages (write once, use many times).

- Improved Security – Users can execute programs without direct table access.

- Modular Programming – Large programs can be divided into smaller blocks.

- Data Integrity – Enforce business rules using triggers and procedures.

- SQL Integration – Fully supports SELECT, INSERT, UPDATE, and DELETE.

- **LAB EXERCISES:**

1] Write a PL/SQL block to print the total number of employees from the employees table.

→

This PL/SQL block:

- Declares a variable
- Counts total employees
- Prints the result

🔹 **PL/SQL Code**

```
SET SERVEROUTPUT ON;


DECLARE
   total_employees NUMBER;
BEGIN
   -- Count total employees
   SELECT COUNT(*)
   INTO total_employees
   FROM employees;
   -- Print the result
   DBMS_OUTPUT.PUT_LINE('Total Number of Employees: ' ||
total_employees);


END;
/
```

✅ **Example Output**

```
Total Number of Employees: 25
```

2] Create a PL/SQL block that calculates the total sales from an orders table.

→

```
SET SERVEROUTPUT ON;


DECLARE
   total_sales NUMBER;
BEGIN
   -- Calculate total sales
   SELECT SUM(order_amount)
   INTO total_sales
   FROM orders;


   -- Handle NULL case (if table is empty)
   IF total_sales IS NULL THEN
      total_sales := 0;
   END IF;


   -- Print result
   DBMS_OUTPUT.PUT_LINE('Total Sales: ' || total_sales);


END;
/
```

✅ **Example Output**

Total Sales: 125000

17. PL/SQL Control Structures

- **THEORY QUESTION:**

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

→

Control structures in **PL/SQL** are programming statements that control the **flow of execution** in a PL/SQL block.

They allow you to:

- Make decisions
- Repeat actions
- Control program logic

🔹 **Types of Control Structures in PL/SQL**

1️⃣ Conditional Control
2️⃣ Iterative (Looping) Control
3️⃣ Sequential Control

1️⃣ **IF–THEN Control Structure**

Used to execute statements **only if a condition is true**.

🔹 **Syntax**

```
IF condition THEN
  statements;
END IF;
```

- **Example**

```
DECLARE
  salary NUMBER := 60000;
BEGIN
  IF salary > 50000 THEN
    DBMS_OUTPUT.PUT_LINE('High Salary');
  END IF;
END;
/
```

Output:

High Salary

- **IF–THEN–ELSE Example**

```
IF salary > 50000 THEN
  DBMS_OUTPUT.PUT_LINE('High Salary');
ELSE
  DBMS_OUTPUT.PUT_LINE('Normal Salary');
END IF;
```

- **IF–THEN–ELSIF–ELSE Example**

```
IF salary > 70000 THEN
  DBMS_OUTPUT.PUT_LINE('Very High Salary');
ELSIF salary > 50000 THEN
  DBMS_OUTPUT.PUT_LINE('High Salary');
ELSE
```

```
  DBMS_OUTPUT.PUT_LINE('Average Salary');
END IF;
```

2️⃣ LOOP Control Structure

📌 Purpose:

Used to repeat a block of code multiple times.

PL/SQL supports three types of loops:

- Basic LOOP
- WHILE LOOP
- FOR LOOP

🔹 (A) Basic LOOP

Runs until EXIT condition is met.

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

Output:

1

2

3

4

5

- **(B) WHILE LOOP**

Runs while condition is true.

```
DECLARE
  i NUMBER := 1;
BEGIN
  WHILE i <= 5 LOOP
    DBMS_OUTPUT.PUT_LINE(i);
    i := i + 1;
  END LOOP;
END;
/
```

- **(C) FOR LOOP**

Automatically controls loop variable.

```
BEGIN
  FOR i IN 1..5 LOOP
    DBMS_OUTPUT.PUT_LINE(i);
  END LOOP;
END;
/
```

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

→

Control structures in **PL/SQL** allow you to add **logic and decision-making ability** to SQL statements.

While SQL only retrieves or modifies data, PL/SQL (used in **Oracle Database**) allows you to process that data using programming logic.

🔹 **Why SQL Alone Is Limited**

SQL:

- Executes one statement at a time
- Cannot use loops
- Cannot perform complex conditional logic
- Cannot handle errors properly

PL/SQL solves these problems using control structures.

🔥 **How Control Structures Help**

i. **Decision Making (IF–THEN)**

Allows queries to behave differently based on conditions.

Example:

- Give bonus only if salary > 50,000
- Update status if order amount exceeds limit

```
IF total_sales > 100000 THEN
  UPDATE employees
  SET bonus = 5000;
END IF;
```

ii. Looping Through Records

Loops allow processing row-by-row.

Example:

- Calculate total salary manually
- Update multiple rows with custom logic

iii. Error Handling (EXCEPTION)

Helps manage runtime errors safely.

```
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No record found');
```

iv. Conditional Query Execution

Execute different SQL statements based on logic.

Example:

- Insert if record doesn't exist
- Update if record exists

```
IF employee_exists THEN
  UPDATE employees ...
ELSE
  INSERT INTO employees ...
END IF;
```

v. Nested Logic

PL/SQL allows:

- IF inside LOOP
- LOOP inside IF

This enables very complex processing.

- **LAB EXERCISES:**

1] Write a PL/SQL block using an IF-THEN condition to check the department of an employee.

→

```
DECLARE
   v_dept_name  departments.department_name%TYPE;
BEGIN
   -- Get department name of employee
   SELECT d.department_name
   INTO v_dept_name
   FROM employees e
   JOIN departments d
   ON e.department_id = d.department_id
   WHERE e.employee_id = 101;

   -- Check department using IF-THEN
   IF v_dept_name = 'IT' THEN
      DBMS_OUTPUT.PUT_LINE('Employee works in IT Department');
   ELSE
      DBMS_OUTPUT.PUT_LINE('Employee works in ' || v_dept_name || '
Department');
   END IF;

EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Employee not found.');
END;
```

/

Output Example

If employee 101 is in IT:

Employee works in IT Department

If in HR:

Employee works in HR Department

2] Use a FOR LOOP to iterate through employee records and display their names.

→

```
BEGIN
   -- Cursor FOR LOOP to fetch employee records
   FOR emp_rec IN (SELECT first_name, last_name FROM employees)
   LOOP
      DBMS_OUTPUT.PUT_LINE('Employee Name: ' ||
                           emp_rec.first_name || ' ' ||
                           emp_rec.last_name);
   END LOOP;
END;
/
```

18. SQL Cursors

- **THEORY QUESTION:**

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 SQL query.
It allows you to **fetch and process rows one by one** from a query result.

i. **Implicit Cursor**

Automatically created by PL/SQL when you execute:

- SELECT INTO
- INSERT
- UPDATE
- DELETE

You don't declare it manually.

◆ **Example (Implicit Cursor)**

```
DECLARE
  v_name employees.first_name%TYPE;
BEGIN
  SELECT first_name
  INTO v_name
  FROM employees
  WHERE employee_id = 101;

  DBMS_OUTPUT.PUT_LINE(v_name);
END;
/
```

ii. Explicit Cursor

A cursor that is **declared manually** by the programmer to handle queries that return multiple rows.

Used when:

- Query returns many rows
- You want full control over fetching

🔹 Steps in Explicit Cursor

1. Declare cursor
2. Open cursor
3. Fetch rows
4. Close cursor

🔹 Example (Explicit Cursor)

```
DECLARE
  CURSOR emp_cursor IS
    SELECT first_name FROM employees;

  v_name employees.first_name%TYPE;

BEGIN
  OPEN emp_cursor;

  LOOP
    FETCH emp_cursor INTO v_name;
    EXIT WHEN emp_cursor%NOTFOUND;
```

```
    DBMS_OUTPUT.PUT_LINE(v_name);
  END LOOP;

  CLOSE emp_cursor;
END;
/
```

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

→

An **explicit cursor** is used when a query returns multiple rows and you need more control over processing them.

Use Explicit Cursor When:

1. **Query returns multiple rows**
   SELECT INTO causes TOO_MANY_ROWS error.
2. **Row-by-row processing is required**
   Apply business logic separately to each row (e.g., bonus calculation).
3. **More control is needed**
   You can manually **OPEN, FETCH, EXIT, and CLOSE** the cursor.
4. **Using cursor attributes**
   - %FOUND
   - %NOTFOUND
   - %ROWCOUNT
   - %ISOPEN
5. **Using parameterized cursors**
   Explicit cursors can accept parameters (e.g., department-wise employees).

- **LAB EXERCISES:**

1] Write a PL/SQL block using an explicit cursor to retrieve and display employee details.

→

```
DECLARE
   -- 1️⃣ Declare Cursor
   CURSOR emp_cursor IS
      SELECT employee_id, first_name, last_name, salary
      FROM employees;

   -- 2️⃣ Declare Variables
   v_emp_id    employees.employee_id%TYPE;
   v_fname     employees.first_name%TYPE;
   v_lname     employees.last_name%TYPE;
   v_salary    employees.salary%TYPE;

BEGIN
   -- 3️⃣ Open Cursor
   OPEN emp_cursor;

   -- 4️⃣ Fetch Rows
   LOOP
      FETCH emp_cursor INTO v_emp_id, v_fname, v_lname, v_salary;
      EXIT WHEN emp_cursor%NOTFOUND;

      DBMS_OUTPUT.PUT_LINE(
```

```
        'ID: ' || v_emp_id ||
        ' | Name: ' || v_fname || ' ' || v_lname ||
        ' | Salary: ' || v_salary
      );
  END LOOP;

  -- 5️⃣ Close Cursor
  CLOSE emp_cursor;
END;
/
```

Example Output

ID: 101 | Name: John Smith | Salary: 60000

ID: 102 | Name: Sarah Johnson | Salary: 55000

ID: 103 | Name: Michael Brown | Salary: 70000

...

2] Create a cursor to retrieve all courses and display them one by one.

→

```
DECLARE
  -- 1️⃣ Declare Cursor
  CURSOR course_cursor IS
    SELECT course_id, course_name, duration, fees
    FROM courses;
```

```
  -- 2️⃣ Declare Variables
  v_course_id   courses.course_id%TYPE;
  v_course_name courses.course_name%TYPE;
  v_duration    courses.duration%TYPE;
  v_fees        courses.fees%TYPE;

BEGIN
  -- 3️⃣ Open Cursor
  OPEN course_cursor;

  -- 4️⃣ Fetch and Display Each Row
  LOOP
    FETCH course_cursor
    INTO v_course_id, v_course_name, v_duration, v_fees;

    EXIT WHEN course_cursor%NOTFOUND;

    DBMS_OUTPUT.PUT_LINE(
      'Course ID: ' || v_course_id ||
      ' | Name: ' || v_course_name ||
      ' | Duration: ' || v_duration ||
      ' | Fees: ' || v_fees
    );
  END LOOP;

  -- 5️⃣ Close Cursor
```

```
    CLOSE course_cursor;
END;
/
```

Example Output

Course ID: 1 | Name: SQL Basics | Duration: 3 Months | Fees: 10000

Course ID: 2 | Name: PL/SQL | Duration: 2 Months | Fees: 8000

Course ID: 3 | Name: Java | Duration: 4 Months | Fees: 12000

...

19. Rollback and Commit Savepoint

- **THEORY QUESTION:**

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

→ A **SAVEPOINT** is a point within a transaction that allows you to **roll back part of the transaction** instead of undoing everything.

In database systems like **Oracle Database**, transactions ensure data consistency using:

- COMMIT
- ROLLBACK
- SAVEPOINT

✅ **Syntax of SAVEPOINT**

SAVEPOINT savepoint_name;

🔹 **Example**

INSERT INTO courses VALUES (1, 'SQL', 3, 10000);

SAVEPOINT sp1;

INSERT INTO courses VALUES (2, 'PLSQL', 2, 8000);

ROLLBACK TO sp1;

How ROLLBACK Interacts with SAVEPOINT

There are two types:

i. **Full ROLLBACK**

ROLLBACK;

Cancels the entire transaction
All savepoints are removed

ii. Partial ROLLBACK

ROLLBACK TO savepoint_name;

Cancels changes after that savepoint
Savepoints created before it remain

How COMMIT Interacts with SAVEPOINT

COMMIT;

Saves all changes permanently
Removes all savepoints
Ends the transaction

After COMMIT, you cannot rollback to any savepoint.

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

→

A SAVEPOINT allows partial rollback in a transaction instead of cancelling the entire transaction.

When SAVEPOINT Is Useful

- Large Transactions – Undo specific steps (INSERT, UPDATE, DELETE) without losing all changes.
- Complex Business Logic – Roll back only the failed part (e.g., stock update) and keep other data.
- Error Handling in PL/SQL – Roll back to a specific point if an error occurs.
- Testing During Development – Try operations and roll back if results are wrong.
- Long-Running Transactions – Create checkpoints and return to the last safe stage if needed.

- **LAB EXERCISES:**

1] Perform a transaction where you create a savepoint, insert records, then rollback to the savepoint.

→

We insert some records into the courses table, create a savepoint, insert more records, then roll back to the savepoint.

- **SQL Transaction**

-- Step 1: Insert first record

INSERT INTO courses (course_id, course_name, duration, fees)

VALUES (101, 'SQL Basics', '3 Months', 10000);

-- Step 2: Create Savepoint

SAVEPOINT sp_courses;

-- Step 3: Insert additional records

INSERT INTO courses (course_id, course_name, duration, fees)

VALUES (102, 'PLSQL', '2 Months', 8000);

INSERT INTO courses (course_id, course_name, duration, fees)

VALUES (103, 'Java', '4 Months', 12000);

-- Step 4: Rollback to Savepoint

ROLLBACK TO sp_courses;

-- Step 5: Commit remaining changes

COMMIT;

Final Result in Table

course_id	course_name
101	SQL Basics

Courses 102 and 103 will NOT be saved.

2] Commit part of a transaction after using a savepoint and then rollback the remaining changes.

→

```
-- Transaction 1
INSERT INTO courses VALUES (301, 'SQL', '3 Months', 10000);
SAVEPOINT sp1;
INSERT INTO courses VALUES (302, 'PLSQL', '2 Months', 8000);

COMMIT;   -- 301 and 302 are permanently saved


-- Transaction 2
INSERT INTO courses VALUES (303, 'Java', '4 Months', 12000);
INSERT INTO courses VALUES (304, 'Python', '3 Months', 15000);

ROLLBACK;  -- 303 and 304 are undone
```

Records 301 and 302 remain.
Records 303 and 304 are rolled back.