***Module 4 – Introduction to DBMS***

**1.Introduction to SQL**

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

SQL (Structured Query Language) is a standard programming language used to store, retrieve, manage, and manipulate data in relational databases.  
It is essential in database management because:

* It allows users to query data efficiently.
* It supports data definition, data manipulation, and access control.
* It enables communication with database systems like MySQL, PostgreSQL, Oracle, and SQL Server.

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

| Feature | DBMS | RDBMS |
| --- | --- | --- |
| Full Form | Database Management System | Relational Database Management System |
| Data Storage | Stores data as files or collections | Stores data in tables (rows & columns) |
| Data Relationship | Does not enforce relationships | Enforces relationships via foreign keys |
| Normalization | Not supported or limited | Supports data normalization |
| Examples | File system, XML databases | MySQL, PostgreSQL, Oracle, SQL Server |
|  |  |  |

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

**SQL plays a central role in relational databases by providing:**

* Data Querying: SELECT statements retrieve data from one or more tables.
* Data Manipulation: INSERT, UPDATE, and DELETE manage data records.
* Data Definition: CREATE, ALTER, and DROP define or change table structures.
* Data Control: GRANT and REVOKE control user permissions.
* Transaction Management: Commands like COMMIT, ROLLBACK, and SAVEPOINT ensure data integrity.

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

Key features include:

* Data Querying with SELECT
* Data Manipulation with INSERT, UPDATE, DELETE
* Schema Definition using CREATE and ALTER
* Data Access Control with GRANT and REVOKE
* Transaction Control (BEGIN, COMMIT, ROLLBACK)
* Functions and Expressions for filtering, sorting, and aggregating data
* Standardization – SQL is standardized by ANSI and widely supported

**2. SQL Syntax**

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

The basic components of SQL syntax include:

* **Keywords** – Reserved words like SELECT, FROM, WHERE, INSERT, UPDATE, etc.
* **Identifiers** – Names of tables, columns, databases (e.g., students, name).
* **Expressions** – Combinations of columns, values, and operators (e.g., age > 18).
* **Predicates** – Conditions used in clauses like WHERE, HAVING.
* **Clauses** – Components that build a statement (e.g., SELECT, FROM, WHERE, GROUP BY, ORDER BY).
* **Operators** – Used in conditions (e.g., =, >, <, AND, OR, LIKE, etc.).
* **Literals** – Fixed values (e.g., 'John', 25).

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

SELECT column1, column2, ...

FROM table\_name

WHERE condition

GROUP BY column

HAVING condition

ORDER BY column ASC|DESC;

Note: Not all clauses are mandatory. Only SELECT and FROM are required; others are optional depending on the query needs.

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

Clauses are **building blocks** of SQL statements that define **what data to retrieve, from where, and how to filter or sort it**. Each clause has a specific role:

* **SELECT** – Specifies which columns to return.
* **FROM** – Indicates the table(s) from which to retrieve data.
* **WHERE** – Filters rows based on conditions.
* **GROUP BY** – Groups rows that share the same values in specified columns.
* **HAVING** – Filters grouped data (used after GROUP BY).
* **ORDER BY** – Sorts the result set by specified columns.

**3. SQL Constraints**

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

Constraints in SQL are rules applied to columns in a table to enforce data integrity and consistency. They help ensure that the data entered into the database follows specific rules.

Common Types of SQL Constraints:

| Constraint | Description |
| --- | --- |
| NOT NULL | Ensures that a column cannot have NULL values. |
| UNIQUE | Ensures that all values in a column are distinct (no duplicates). |
| PRIMARY KEY | Uniquely identifies each record in a table. It is a combination of NOT NULL + UNIQUE. |
| FOREIGN KEY | Creates a link between two tables. It ensures the value in one table matches a value in another table’s primary key. |
| CHECK | Ensures that all values in a column meet a specific condition (e.g., age > 18). |
| DEFAULT | Provides a default value for a column when no value is specified during insertion. |

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

| **Feature** | **PRIMARY KEY** | **FOREIGN KEY** |
| --- | --- | --- |
| Purpose | Uniquely identifies records in a table | Links to the **primary key** in another table |
| Uniqueness | Must be **unique** for each row | Can have **duplicate values** |
| NULL allowed? | **Not allowed** | **Allowed**, unless specified otherwise |
| Table | Defined in the **own table** | Refers to a **column in another table** |
| Example | student\_id in Students table | student\_id in Marks table referencing Students |

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

**🔹 NOT NULL:**

* Ensures that a column **must always have a value** (cannot be left empty).
* Used when a field is **mandatory**.
* Example:

CREATE TABLE Students (

id INT NOT NULL,

name VARCHAR(50) NOT NULL

);

**🔹 UNIQUE:**

* Ensures that all values in a column are **different** from one another.
* Can be used on one or more columns.
* Example:

sql

Copy Edit

CREATE TABLE Students (

email VARCHAR(100) UNIQUE

);

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

|  |
| --- |
|  |

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

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

Key features:

* It **creates**, **alters**, and **deletes** database objects.
* Common DDL commands:
  + CREATE
  + ALTER
  + DROP
  + TRUNCATE
  + RENAME

⚠️ DDL commands **automatically commit** changes—they cannot be rolled back in some DBMS.

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

The CREATE command is used to **create a new database object**, like a table or a database.

**✅ Basic Syntax for Creating a Table:**

sql

CopyEdit

CREATE TABLE table\_name (

column1 datatype [constraint],

column2 datatype [constraint],

...

);

**🧾 Example:**

sql

CopyEdit

CREATE TABLE Students (

student\_id INT PRIMARY KEY,

name VARCHAR(50) NOT NULL,

age INT CHECK (age >= 18),

email VARCHAR(100) UNIQUE

);

This command creates a Students table with four columns, data types, and constraints to ensure data validity.

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

Specifying **datatypes** and **constraints** ensures **data integrity**, **accuracy**, and **optimized storage**.

**🔹 Purpose of Datatypes:**

* Define **what kind of data** a column can store (e.g., INT, VARCHAR, DATE).
* Prevent invalid data entries (e.g., storing text in a number field).
* Optimize **memory usage and performance**.

**🔹 Purpose of Constraints:**

* Enforce **rules** on data to maintain **data consistency** and **reliability**.
* Prevent:
  + Duplicate entries (UNIQUE)
  + Null values where not allowed (NOT NULL)
  + Orphaned records (FOREIGN KEY)
  + Invalid values (CHECK)

**5. ALTER Command**

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

The ALTER command in SQL is used to change the structure of an existing table without deleting or recreating it.

✅ It allows you to:

* Add new columns
* Modify existing columns (datatype, constraints, etc.)
* Drop (delete) columns
* Rename columns or tables
* Add or drop constraints (like PRIMARY KEY, FOREIGN KEY)

🔧 It's a powerful tool for updating your database schema as requirements evolve.

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

🔹 To Add a Column:

sql

CopyEdit

ALTER TABLE table\_name

ADD column\_name datatype [constraint];

Example:

sql

CopyEdit

ALTER TABLE Students

ADD gender VARCHAR(10);

🔹 To Modify a Column:

sql

CopyEdit

ALTER TABLE table\_name

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

-- or

ALTER TABLE table\_name

ALTER COLUMN column\_name TYPE new\_datatype; -- PostgreSQL

Example (MySQL):

sql

CopyEdit

ALTER TABLE Students

MODIFY age INT NOT NULL;

Example (PostgreSQL):

sql

CopyEdit

ALTER TABLE Students

ALTER COLUMN age TYPE SMALLINT;

🔹 To Drop a Column:

sql

CopyEdit

ALTER TABLE table\_name

DROP COLUMN column\_name;

Example:

sql

CopyEdit

ALTER TABLE Students

DROP COLUMN gender;

6. DROP Command

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

The DROP command is used to **permanently delete** database objects such as:

* Tables
* Databases
* Views
* Indexes
* Constraints

**✅ Syntax:**

sql

CopyEdit

DROP TABLE table\_name;

🔥 Once executed, the DROP command **removes the object and all of its data permanently**. It cannot be rolled back in many database systems.

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

Dropping a table has **serious and irreversible consequences**:

**⚠️ Key Implications:**

* ✅ **All data in the table is permanently deleted.**
* ✅ **The table structure (columns, constraints, etc.) is removed.**
* ✅ **All associated indexes, triggers, and constraints are also deleted.**
* ❌ **Dependent objects** (like views or foreign keys in other tables) may be invalidated or cause errors if not handled properly.
* 🔁 **Cannot be undone** unless a backup is available.

📌 Example:

sql

CopyEdit

DROP TABLE Employees;

This command will delete the Employees table and everything it contains—**no recovery unless backed up**.

7. Data Manipulation Language(DML)

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

These are **Data Manipulation Language (DML)** commands used to manage the data inside database tables.

**🔹 INSERT – Adds new rows of data to a table.**

**Syntax:**

sql

CopyEdit

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

VALUES (value1, value2, ...);

**Example:**

sql

CopyEdit

INSERT INTO Students (student\_id, name, age)

VALUES (101, 'Alice', 20);

**🔹 UPDATE – Modifies existing data in one or more rows.**

**Syntax:**

sql

CopyEdit

UPDATE table\_name

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

WHERE condition;

**Example:**

sql

CopyEdit

UPDATE Students

SET age = 21

WHERE student\_id = 101;

**🔹 DELETE – Removes one or more rows from a table.**

**Syntax:**

sql

CopyEdit

DELETE FROM table\_name

WHERE condition;

**Example:**

sql

CopyEdit

DELETE FROM Students

WHERE student\_id = 101;

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

The WHERE clause is **crucial** in both UPDATE and DELETE to **target specific rows**.

**✅ Importance:**

| **Without WHERE** | **With WHERE** |
| --- | --- |
| Affects **all rows** | Affects **only selected rows** |
| Risk of accidental data loss or incorrect updates | Ensures **precise and safe** changes |

**🔥 Example (dangerous):**

sql

CopyEdit

DELETE FROM Students;

-- Deletes all rows from the table!

**✅ Safer version:**

sql

CopyEdit

DELETE FROM Students

WHERE student\_id = 101;

⚠️ **Always use the WHERE clause unless you're intentionally modifying or deleting the entire table.**

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

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

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

**✅ Syntax:**

sql

CopyEdit

SELECT column1, column2, ...

FROM table\_name;

**📌 Example:**

sql

CopyEdit

SELECT name, age

FROM Students;

This will display the name and age of all records in the Students table.

**🔎 You can:**

* Retrieve **all columns** using \*
* Use **conditions** with WHERE
* **Sort** results with ORDER BY
* Perform **aggregations**, join tables, filter rows, etc.

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

**🔹 WHERE Clause – Filters records based on specified conditions.**

**Syntax:**

sql

CopyEdit

SELECT column1, column2

FROM table\_name

WHERE condition;

**Example:**

sql

CopyEdit

SELECT name, age

FROM Students

WHERE age > 18;

→ Returns only students older than 18.

**🔹 ORDER BY Clause – Sorts the query result by one or more columns.**

**Syntax:**

sql

CopyEdit

SELECT column1, column2

FROM table\_name

ORDER BY column1 ASC|DESC;

**Example:**

sql

CopyEdit

SELECT name, age

FROM Students

ORDER BY age DESC;

→ Lists students sorted by age in descending order.

**✅ You can combine both:**

sql

CopyEdit

SELECT name, age

FROM Students

WHERE age > 18

ORDER BY name ASC;

→ This query returns students over 18, sorted alphabetically by name.

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

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

The GRANT and REVOKE commands are part of Data Control Language (DCL) in SQL.  
They are used to manage access and permissions for users in a database system.

🔹 GRANT

* Gives specific privileges (permissions) to users or roles.
* Example uses: read/write access to tables, the ability to create tables, or run certain commands.

🔹 REVOKE

* Removes privileges that were previously granted to users or roles.

2. How do you manage privileges using these commands?

✅ Using GRANT:

Syntax:

sql

CopyEdit

GRANT privilege\_list

ON object\_name

TO user\_name;

Example:

sql

CopyEdit

GRANT SELECT, INSERT

ON Students

TO user1;

→ This allows user1 to view and add records to the Students table.

✅ Using REVOKE:

Syntax:

sql

CopyEdit

REVOKE privilege\_list

ON object\_name

FROM user\_name;

Example:

sql

CopyEdit

REVOKE INSERT

ON Students

FROM user1;

→ This removes the INSERT permission from user1 on the Students table.

🔐 Common Privileges You Can Grant/Revoke:

* SELECT – read data
* INSERT – add data
* UPDATE – modify data
* DELETE – remove data
* ALL PRIVILEGES – grant all rights

✅ Managing privileges with GRANT and REVOKE ensures security, controlled access, and data integrity in a multi-user environment.

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

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

These two commands are part of **transaction control** in SQL. They help manage changes to data in a safe and consistent way.

**🔹 COMMIT**

* **Permanently saves** all changes made during the current transaction.
* After COMMIT, changes **cannot be undone**.

**Example:**

sql

CopyEdit

UPDATE Students

SET age = 21

WHERE student\_id = 101;

COMMIT;

**🔹 ROLLBACK**

* **Undoes** all changes made during the current transaction.
* Used to revert the database back to its previous state if something goes wrong.

**Example:**

sql

CopyEdit

UPDATE Students

SET age = 21

WHERE student\_id = 101;

ROLLBACK;

→ This will **cancel** the update and restore the original data.

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

**✅ What is a transaction?**

A **transaction** is a **sequence of one or more SQL operations** that are executed as a **single unit of work**.

The transaction must follow the **ACID properties**:

* **A**tomicity – All or nothing.
* **C**onsistency – Maintains database rules.
* **I**solation – Transactions are independent.
* **D**urability – Once committed, changes are permanent.

**🔁 Managing Transactions:**

**Basic transaction flow:**

sql

CopyEdit

BEGIN TRANSACTION; -- (Optional in some DBMS)

-- SQL operations here

COMMIT; -- or ROLLBACK;

* In many DBMS, changes are **not final** until you explicitly use COMMIT.
* If there's an error or interruption, you can use ROLLBACK to cancel the operations.

**✅ Example Use Case:**

Suppose you transfer money between accounts:

sql

CopyEdit

BEGIN TRANSACTION;

UPDATE Accounts SET balance = balance - 500 WHERE account\_id = 1;

UPDATE Accounts SET balance = balance + 500 WHERE account\_id = 2;

COMMIT; -- Ensures both updates succeed together

If any of the updates fail, you can do:

sql

CopyEdit

ROLLBACK;

→ This cancels the whole transaction and prevents data inconsistency.

**11. SQL Joins**

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

**✅ What is a JOIN in SQL?**

A **JOIN** is used to **combine rows from two or more tables** based on a related column (usually a **foreign key**).

**🔄 Types of Joins:**

| **JOIN Type** | **Description** | **Visual Concept** |
| --- | --- | --- |
| **INNER JOIN** | Returns **only matching rows** from both tables. | 🟢 A ∩ B (common area) |
| **LEFT JOIN** | Returns **all rows from the left table**, and matching rows from the right. | 🟢 All A + A ∩ B |
| **RIGHT JOIN** | Returns **all rows from the right table**, and matching rows from the left. | 🟢 All B + A ∩ B |
| **FULL OUTER JOIN** | Returns **all rows** from both tables, with NULL for unmatched values. | 🟢 A ∪ B (complete area) |

**✅ Syntax Examples:**

Assume two tables:

sql

CopyEdit

Students (student\_id, name)

Marks (student\_id, score)

**🔹 INNER JOIN:**

sql

CopyEdit

SELECT Students.name, Marks.score

FROM Students

INNER JOIN Marks ON Students.student\_id = Marks.student\_id;

→ Returns students **who have marks** (matched in both tables).

**🔹 LEFT JOIN:**

sql

CopyEdit

SELECT Students.name, Marks.score

FROM Students

LEFT JOIN Marks ON Students.student\_id = Marks.student\_id;

→ Returns **all students**, even those **without marks** (score will be NULL).

**🔹 RIGHT JOIN:**

sql

CopyEdit

SELECT Students.name, Marks.score

FROM Students

RIGHT JOIN Marks ON Students.student\_id = Marks.student\_id;

→ Returns **all marks**, even if a student **is missing** in the Students table.

**🔹 FULL OUTER JOIN (Not supported in MySQL, but works in PostgreSQL, SQL Server):**

sql

CopyEdit

SELECT Students.name, Marks.score

FROM Students

FULL OUTER JOIN Marks ON Students.student\_id = Marks.student\_id;

→ Returns **all students and all marks**, matching where possible, filling NULLs where not matched.

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

Joins are used to:

* Merge related data spread across different tables.
* Perform complex queries involving relationships (like one-to-many, many-to-many).
* Retrieve meaningful information without data duplication.

**✅ Example Use Case:**

If Orders and Customers are in separate tables:

sql

CopyEdit

SELECT Orders.order\_id, Customers.name

FROM Orders

JOIN Customers ON Orders.customer\_id = Customers.customer\_id;

→ Combines order data with customer details in **one query result**.

**12. SQL Group By**

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

**✅ What is GROUP BY?**

The GROUP BY clause is used to **group rows that have the same values** in one or more columns into summary rows.

**🔹 Purpose:**

* To **aggregate data** across groups (e.g., total sales per region).
* Used **with aggregate functions** like COUNT(), SUM(), AVG(), MAX(), and MIN().

**🔹 Example:**

sql

CopyEdit

SELECT department, COUNT(\*) AS employee\_count

FROM Employees

GROUP BY department;

* This query counts employees **in each department** by grouping rows with the same department value.

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

| **Feature** | **GROUP BY** | **ORDER BY** |
| --- | --- | --- |
| Purpose | Groups rows with the same values into summary rows | Sorts the result set by specified columns |
| Used with | Aggregate functions (SUM(), COUNT(), etc.) | Any column (aggregate or not) |
| Result | Returns one row per group | Returns all rows sorted |
| Example Use | Total sales per category | List sales records sorted by date or amount |

**Summary Example:**

sql

CopyEdit

SELECT department, AVG(salary) AS avg\_salary

FROM Employees

GROUP BY department

ORDER BY avg\_salary DESC;

* Groups employees by department,
* Calculates average salary per department,
* Sorts departments by average salary in descending order.

**13. SQL Stored Procedure**

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

**🔹 Stored Procedure:**

* A **stored procedure** is a **precompiled set of SQL statements** stored in the database.
* It can include control-of-flow logic (like loops, conditions) and can accept input parameters.
* Stored procedures are **executed by calling their name** with optional parameters.

**🔹 Difference from a standard SQL query:**

| **Aspect** | **Stored Procedure** | **Standard SQL Query** |
| --- | --- | --- |
| Precompilation | Yes, compiled and stored in DB | No, parsed and executed at runtime |
| Complexity | Can include multiple statements & logic | Usually a single statement |
| Reusability | Reusable and modular | Typically ad-hoc or one-time use |
| Performance | Faster execution due to precompilation | Slower for repetitive tasks |
| Security | Can restrict direct table access | Runs as-is |

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

**✅ Advantages:**

* **Improved performance:** Precompiled and optimized by the database engine.
* **Code reuse:** Write once, call multiple times from different applications.
* **Reduced network traffic:** Execute complex logic on the server, sending fewer commands over the network.
* **Enhanced security:** Users can be granted permission to execute procedures without direct access to underlying tables.
* **Maintainability:** Centralized logic makes it easier to update and manage.
* **Consistency:** Ensures consistent execution of complex operations.

**14. SQL View**

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

**🔹 View:**

* A **view** is a **virtual table** in SQL.
* It is created by a **query that selects data from one or more tables**.
* The view itself **does not store data physically**; it just **displays data stored in the underlying tables** dynamically when queried.

**🔹 Difference between a view and a table:**

| **Aspect** | **Table** | **View** |
| --- | --- | --- |
| Data Storage | Stores data physically | Does **not** store data physically |
| Definition | Actual database object | Defined by a SQL query |
| Updates | Can be updated directly | May or may not be updatable (depends on DBMS and complexity) |
| Usage | Base for storing data | Used for simplifying complex queries, security, or abstraction |
| Structure | Has its own schema and data | Schema based on query result |

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

**✅ Advantages:**

* **Simplify complex queries:** Users can query the view without writing complex joins or calculations.
* **Data security:** Restrict access to specific columns or rows by controlling what the view exposes.
* **Data abstraction:** Hide the complexity of the underlying database schema from users.
* **Consistency:** Provide a consistent, customized presentation of data.
* **Reusability:** Reuse the same logic in multiple queries without rewriting the code.
* **Maintainability:** If the underlying table structure changes, you can update the view without changing all client queries.

**15. SQL Triggers**

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

**🔹 Trigger:**

A **trigger** is a special kind of stored procedure that **automatically executes (fires)** in response to certain events on a table or view, such as INSERT, UPDATE, or DELETE operations.

* Triggers help enforce **business rules**, **data integrity**, and **audit changes**.
* They run **automatically**, so no manual execution is needed.

**🔹 Types of Triggers:**

| **Type** | **When It Fires** | **Use Case Example** |
| --- | --- | --- |
| **BEFORE Trigger** | Executes **before** the triggering event (insert/update/delete) happens | Validate or modify data before changes |
| **AFTER Trigger** | Executes **after** the triggering event has completed successfully | Log changes, update related tables |
| **INSTEAD OF Trigger** | Executes **instead of** the triggering event, mainly used with views | Enable updates on complex views |

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

| **Trigger Type** | **What It Responds To** | **Typical Use Case** |
| --- | --- | --- |
| **INSERT Trigger** | Fires when a new row is **inserted** into the table | Validate or modify inserted data; audit inserts |
| **UPDATE Trigger** | Fires when an existing row is **updated** | Track changes; enforce complex validations |
| **DELETE Trigger** | Fires when a row is **deleted** | Maintain referential integrity; log deletions |

**Summary Example:**

sql

CopyEdit

CREATE TRIGGER trg\_after\_insert

AFTER INSERT ON Employees

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (EmployeeID, Action, ActionDate)

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

END;

This example logs every new employee added.

**16. Introduction to PL/SQL**

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

**🔹 PL/SQL (Procedural Language/SQL):**

* PL/SQL is **Oracle’s procedural extension** to SQL.
* It allows you to **combine SQL with programming logic** like variables, loops, conditions, and error handling.

**🔧 How it extends SQL’s capabilities:**

| **Feature** | **SQL** | **PL/SQL** |
| --- | --- | --- |
| Control structures | ❌ Not supported | ✅ Supports IF, LOOP, WHILE, FOR, etc. |
| Variables | ❌ Cannot declare | ✅ You can declare and use variables |
| Exception handling | ❌ No error control | ✅ Built-in error handling with EXCEPTION blocks |
| Procedures/functions | ❌ Not supported | ✅ Create reusable blocks with parameters |
| Complex logic | ❌ Limited | ✅ Perform full programming logic |

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

**✅ Benefits of PL/SQL:**

1. **Modularity**
   * Code can be organized into **procedures, functions, packages**, and triggers for better structure and reuse.
2. **Performance**
   * Reduces **network traffic** by executing multiple SQL statements in a single block on the server.
3. **Maintainability**
   * Easier to manage and update logic since business rules are **centralized in stored procedures**.
4. **Error Handling**
   * Provides robust **exception handling** to gracefully manage runtime errors.
5. **Integration with SQL**
   * Seamlessly blends **SQL and procedural logic**, allowing data manipulation and control logic in one place.
6. **Security**
   * You can grant users access to execute procedures **without giving direct access** to the underlying tables.
7. **Portability**
   * PL/SQL programs can run on any **Oracle database**, making them portable within Oracle environments.

**🔁 Simple Example of PL/SQL Block:**

sql

CopyEdit

DECLARE

v\_name VARCHAR2(50);

BEGIN

SELECT name INTO v\_name FROM Students WHERE student\_id = 101;

DBMS\_OUTPUT.PUT\_LINE('Student Name: ' || v\_name);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Student not found');

END;

**17. PL/SQL Control Structures**

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

**🔹 Control Structures in PL/SQL:**

Control structures allow you to **control the flow of execution** within a PL/SQL block. These are similar to those in other programming languages and are used to implement decision-making and repetition logic.

**✅ Types of Control Structures in PL/SQL:**

* **Conditional Control** – IF-THEN, IF-THEN-ELSE, IF-THEN-ELSIF
* **Looping Control** – LOOP, WHILE LOOP, FOR LOOP
* **Sequential Control** – GOTO, EXIT, CONTINUE

**🔹 IF-THEN Structure (Conditional)**

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

**Syntax:**

sql

CopyEdit

IF condition THEN

-- statements

END IF;

**Example:**

sql

CopyEdit

IF salary > 50000 THEN

bonus := 1000;

END IF;

You can also use:

sql

CopyEdit

IF condition THEN

-- statements

ELSE

-- alternative statements

END IF;

**🔹 LOOP Structure (Repetition)**

Used to **repeat a block of code** until a specific condition is met using EXIT.

**Syntax:**

sql

CopyEdit

LOOP

-- statements

EXIT WHEN condition;

END LOOP;

**Example:**

sql

CopyEdit

i := 1;

LOOP

DBMS\_OUTPUT.PUT\_LINE(i);

i := i + 1;

EXIT WHEN i > 5;

END LOOP;

Other types of loops:

* WHILE condition LOOP ... END LOOP;
* FOR i IN 1..5 LOOP ... END LOOP;

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

Control structures make PL/SQL **more powerful than standard SQL** by allowing:

**✅ Key Benefits:**

| **Feature** | **Benefit** |
| --- | --- |
| **Conditional execution** | Handle different business rules (e.g., IF salary > threshold) |
| **Iteration** | Automate repetitive tasks (e.g., looping through rows or values) |
| **Decision-making logic** | Execute different SQL operations based on conditions |
| **Error handling** | Control flow during exceptions using EXCEPTION blocks |

**📌 Example:**

A control structure can loop through employee records and give bonuses only to those who meet performance criteria—this kind of logic is **not possible with plain SQL alone**.

**18. SQL Cursors**

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

**🔹 What is a Cursor?**

A **cursor** in PL/SQL is a **pointer to a result set** of a SQL query. It allows row-by-row processing of query results, especially when multiple rows are returned.

**🔁 Types of Cursors in PL/SQL:**

| **Cursor Type** | **Description** |
| --- | --- |
| **Implicit Cursor** | Automatically created by PL/SQL **for single-row queries** like SELECT INTO, INSERT, UPDATE, or DELETE. |
| **Explicit Cursor** | Manually declared by the programmer to **fetch multiple rows** from a query. Provides more control over row-by-row processing. |

**🔹 Implicit Cursor Example:**

sql

CopyEdit

DECLARE

v\_name Students.name%TYPE;

BEGIN

SELECT name INTO v\_name FROM Students WHERE student\_id = 101;

DBMS\_OUTPUT.PUT\_LINE('Name: ' || v\_name);

END;

* Used for simple, **single-row operations**
* Cursor is handled automatically by PL/SQL

**🔹 Explicit Cursor Example:**

sql

CopyEdit

DECLARE

CURSOR c\_students IS

SELECT name FROM Students WHERE age > 18;

v\_name Students.name%TYPE;

BEGIN

OPEN c\_students;

LOOP

FETCH c\_students INTO v\_name;

EXIT WHEN c\_students%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Name: ' || v\_name);

END LOOP;

CLOSE c\_students;

END;

* Used for **multi-row result sets**
* You manually **OPEN**, **FETCH**, and **CLOSE** the cursor

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

**✅ Use an explicit cursor when:**

* You need to **process multiple rows one at a time**
* You want **fine-grained control** over fetching, looping, or exiting
* You want to check **cursor attributes** like %ROWCOUNT, %ISOPEN, or %FOUND
* You need to **loop through records** and apply logic to each row individually

**❌ Implicit cursors:**

* Are limited to **single-row results**
* Do **not** allow row-by-row operations
* Are **automatic**, but offer **less control**

**🔑 Summary:**

| **Feature** | **Implicit Cursor** | **Explicit Cursor** |
| --- | --- | --- |
| Creation | Automatic by PL/SQL | Manually declared |
| Row Handling | One row only | Multiple rows |
| Control | Limited | Full control over operation |
| Use Case | Simple queries | Looping or complex logic |

**19. Rollback and Commit Save point**

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

**🔹 What is a SAVEPOINT?**

A **SAVEPOINT** is a marker within a transaction that allows you to **partially roll back** to a specific point without canceling the entire transaction.

It helps manage complex transactions by dividing them into smaller, recoverable parts.

**✅ How ROLLBACK and COMMIT interact with SAVEPOINTs:**

| **Command** | **Behavior** |
| --- | --- |
| **SAVEPOINT savepoint\_name** | Sets a **named checkpoint** within the transaction. |
| **ROLLBACK TO savepoint\_name** | **Undoes** all operations after the savepoint, but keeps the transaction active. |
| **COMMIT** | **Saves all changes** made in the transaction **and removes all savepoints**. |

**🔹 Example:**

sql

CopyEdit

BEGIN;

UPDATE Accounts SET balance = balance - 500 WHERE account\_id = 1;

SAVEPOINT after\_withdraw;

UPDATE Accounts SET balance = balance + 500 WHERE account\_id = 2;

-- Suppose this update fails

ROLLBACK TO after\_withdraw;

-- Continue with other operations

COMMIT;

* If something goes wrong after the withdrawal, we can **roll back to the savepoint**, without losing the entire transaction.

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

**✅ Use cases for SAVEPOINTs:**

1. **Partial error handling:**  
   When you want to undo **only a portion** of a transaction without canceling the entire operation.
2. **Complex transactions:**  
   In large transactions involving multiple steps, savepoints let you **control which steps to undo or keep**.
3. **Improved debugging:**  
   Useful during development/testing to isolate problems and **test rollback scenarios**.
4. **User-driven operations:**  
   In applications where the user may **confirm or cancel parts** of a multi-step process.

**🔑 Summary:**

| **Feature** | **Description** |
| --- | --- |
| **SAVEPOINT** | Sets a marker inside a transaction |
| **ROLLBACK TO** | Undoes changes after the savepoint only |
| **COMMIT** | Finalizes the whole transaction and clears all savepoints |