(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 **communicate with and manage databases**. It helps users perform operations like:

- Inserting new data
- Retrieving existing data
- Updating or deleting data
- Creating or modifying database structures (tables, views, etc.)

Why is SQL essential?

- It **simplifies interaction** with databases.
- Allows **data manipulation** using simple commands.
- Supports **data security** with user permissions.
- Works with almost all **Relational Database Management Systems (RDBMS)** like MySQL, PostgreSQL, Oracle, etc.

2 - Explain the difference between DBMS and RDBMS.

Aspect	DBMS	RDBMS
Full Form	Database Management System	Relational Database Management System
Data Storage	Data stored as files or simple formats	Data stored in **tables (rows & columns)**
Relation Support	No concept of relations	Uses **relations (tables)** with **primary & foreign keys**
Data Integrity	Limited	High — supports constraints, normalization, etc.
Examples	Microsoft Access, file-based DB	MySQL, Oracle, SQL Server, PostgreSQL

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

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

- **Creating and modifying** tables (`CREATE`, `ALTER`, `DROP`)
- **Inserting and updating** data (`INSERT`, `UPDATE`)
- **Querying** data (`SELECT`)
- **Setting rules** for data (`CONSTRAINTS`, `PRIMARY KEY`, `FOREIGN KEY`)
- **Controlling access** (`GRANT`, `REVOKE`)
- **Ensuring accuracy** and **efficiency** of data handling

SQL acts as the **bridge between users and the database system**.

4 - What are the key features of SQL?

Here are the **main features** of SQL:

1. **Data Querying** – Retrieve data using `SELECT` statements.
2. **Data Manipulation** – Add, modify, and delete data (`INSERT`, `UPDATE`, `DELETE`).
3. **Data Definition** – Create and manage tables and schema (`CREATE`, `ALTER`, `DROP`).
4. **Data Control** – Manage user permissions (`GRANT`, `REVOKE`).
5. **Transaction Control** – Manage multiple operations as a single unit (`COMMIT`, `ROLLBACK`).
6. **Security** – Enforces access control and data integrity.
7. **Portable** – Works with many RDBMS (like MySQL, Oracle, etc.).

(SQL Syntax)

1 - What are the basic components of SQL syntax?

SQL syntax consists of keywords and clauses that help perform operations on a database.

Here are the basic components:

Component	Description
Keywords	**Reserved words like `SELECT`, `FROM`, `WHERE`, `INSERT`, `UPDATE`, `DELETE`, etc.**
Identifiers	**Names of tables, columns, databases (like `students`, `name`, `marks`)**
Operators	**Symbols for comparisons (`=`, `>`, `<`, `>=`, `<=`, `!=`, `LIKE`, `IN`)**
Literals	**Fixed values like numbers (`10`), strings (`'John'`), dates**
Clauses	**Parts of the SQL statement like `SELECT`, `FROM`, `WHERE`, `ORDER BY`, `GROUP BY`**
Functions	**Built-in operations like `COUNT()`, `SUM()`, `AVG()`, `NOW()`**
Comments	**Notes in code: `-- single line` or `/* multi-line */`**

2 - general structure

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

Example:-

```
SELECT name, age
FROM students
WHERE age > 18
ORDER BY age DESC;
```

3. Role of clauses in SQL statements.

Clauses are the building blocks of SQL statements. Each clause serves a specific purpose in shaping the query:

Clause	Purpose
SELECT	Chooses which columns to display
FROM	Specifies the table to retrieve data from
WHERE	Filters rows based on a condition
GROUP BY	Groups rows that have the same values in specified columns
HAVING	Filters groups after grouping is done
ORDER BY	Sorts the

```
SELECT department, COUNT(*) AS total
FROM employees
WHERE salary > 30000
GROUP BY department
HAVING COUNT(*) > 5
ORDER BY total DESC;
```

(SQL Constraints)

1 - constraints in SQL

Constraints are rules in SQL that control what kind of data can be stored in a table's columns. They help keep the data accurate and consistent. The main types are:

- NOT NULL (value must be provided),
- UNIQUE (no duplicate values),
- PRIMARY KEY (uniquely identifies each row),
- FOREIGN KEY (links two tables),
- CHECK (ensures specific condition), and
- DEFAULT (sets a default value if none is given).

2. Difference between PRIMARY KEY and FOREIGN KEY

A PRIMARY KEY uniquely identifies each row in a table and doesn’t allow nulls or duplicates. A FOREIGN KEY, on the other hand, is used to link two tables — it refers to the PRIMARY KEY of another table and can have duplicates or nulls. The PRIMARY KEY defines uniqueness within the same table, while the FOREIGN KEY ensures relationships between tables.

3. Role of NOT NULL and UNIQUE

The NOT NULL constraint ensures that a column cannot have missing (null) values — it forces data to be entered. The UNIQUE constraint makes sure all values in a column are different, preventing duplicates but allowing one null (in most databases). Together, they help maintain data integrity.

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

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

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

Common DDL commands include:

- CREATE – to create new tables or databases
- ALTER – to modify existing tables
- DROP – to delete tables or databases
- TRUNCATE – to quickly delete all rows from a table

2. Explain the CREATE command and its syntax:

The CREATE command is used to define new database objects like tables, databases, and views.

Syntax:

```
CREATE TABLE table_name (

   column1 datatype constraint,

   column2 datatype constraint,

);
```

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

Data Types:

- Define the kind of data a column can store (e.g., numbers, text, dates).
- Ensure data consistency and proper memory usage.

Constraints:

- Enforce **rules on data** to maintain accuracy and integrity.

- Examples:
    - `NOT NULL` ensures a value must be entered.
    - `UNIQUE` prevents duplicate values.
    - `PRIMARY KEY` uniquely identifies each row.
    - `FOREIGN KEY` maintains relationships between tables.

(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.
It allows you to:

- Add new columns
- Modify existing columns (like data type or size)
- Drop (remove) columns
- Rename columns or the table itself
- Add or remove constraints

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

You can use the `ALTER TABLE` command in different ways depending on the task:

Add a Column:

-> ALTER TABLE table_name ADD column_name datatype;

Modify a Column:

-> ALTER TABLE table_name MODIFY column_name new_datatype;

Drop a Column:

-> ALTER TABLE table_name DROP COLUMN column_name;

(DROP Command)

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

The `DROP` command is used to permanently delete a database object such as a table, view, or even the entire database.

- When you use `DROP`, the object is completely removed from the system.
- All the data and structure are lost.
- Common usage: `DROP TABLE`, `DROP DATABASE`.

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

Dropping a table has the following effects:

- The table is permanently deleted.
- All data stored in the table is lost and cannot be recovered (unless a backup exists).
- All associated constraints, indexes, and relationships are also removed.
- Any application or query that used the table may stop working due to missing references.

(Data Manipulation Language (DML))

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

- INSERT:
  The `INSERT` command is used to add new records (rows) into a table.
- UPDATE:
  The `UPDATE` command is used to change or modify existing records in a table.
- DELETE:
  The `DELETE` command is used to remove records from a table.

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

- The `WHERE` clause is used to specify which rows should be updated or deleted.
- Without a `WHERE` clause:
  - `UPDATE` will change all rows in the table.
  - `DELETE` will remove all rows from the table.

(Data Query Language (DQL))

1. 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 allows you to choose specific columns or all columns.
- You can also apply conditions, sorting, and calculations.

Basic usage:
It helps in viewing the stored data without changing it.

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

- WHERE Clause:
  - Used to filter rows based on a condition.
  - It selects only the rows that meet the condition.
  - Example: Get all students where age > 18.

- ORDER BY Clause:
  - Used to sort the result in ascending (ASC) or descending (DESC) order.
  - You can sort by one or more columns.
  - Example: Sort students by name or marks.

(Data Control Language (DCL))

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

- GRANT:
  The GRANT command is used to give permissions to users so they can access or perform actions on database objects (like tables, views, etc.).

- REVOKE:
  The REVOKE command is used to take back permissions that were previously given using GRANT.

Note :-These commands help in controlling user access and security in the database.

2. How do you manage privileges using these commands?

- You can manage who can do what in a database using GRANT and REVOKE.
- With GRANT, you can allow users to:
    - Read data (`SELECT`)
    - Insert or delete data (`INSERT`, `DELETE`)
    - Modify structure (`ALTER`, `DROP`)
- With REVOKE, you can remove those permissions at any time.

(Transaction Control Language (TCL))

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

- COMMIT:
  The `COMMIT` command is used to save all the changes made in the current transaction permanently to the database.
- ROLLBACK:
  The `ROLLBACK` command is used to undo all changes made in the current transaction and return the database to its previous state.

2. Explain how transactions are managed in SQL databases:

- A transaction is a group of one or more SQL operations that are treated as a single unit of work.
- Transactions follow the ACID properties:
    - Atomicity – All or nothing
    - Consistency – Keeps database in valid state
    - Isolation – Transactions are independent
    - Durability – Once committed, changes are permanent
- You start a transaction → perform changes → then use either:
    - `COMMIT` to save the changes, or
    - `ROLLBACK` to cancel the changes.

(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?

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

  Types of JOINs:

- INNER JOIN:
  Returns only the matching rows from both tables.
  → No match = no result.

- LEFT JOIN (or LEFT OUTER JOIN):
  Returns all rows from the left table and the matching rows from the right table.
  → If no match, right side shows NULL.

- RIGHT JOIN (or RIGHT OUTER JOIN):
  Returns all rows from the right table and the matching rows from the left table.
  → If no match, left side shows NULL.

- FULL OUTER JOIN:
  Returns all rows from both tables, with NULLs where there's no match.

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

- Joins are used to gather related information that is stored in different tables.
- They are based on common columns, usually a primary key in one table and a foreign key in the other.
- This allows you to:
  - Fetch data from multiple tables in a single query.
  - Build complete reports without repeating data.

Example use case:
Joining `Students` and `Courses` tables to get a list of students with their enrolled course names.

(SQL Group By)

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 into summary rows, like “total sales by category” or “average salary by department”. It is commonly used with aggregate functions like:

- COUNT() – total number of items
- SUM() – total value
- AVG() – average value
- MAX() / MIN() – highest/lowest value

Syntax -

SELECT department, AVG(salary)

FROM employees

GROUP BY department;

2 - difference between GROUP BY and ORDER BY.

The GROUP BY and ORDER BY clauses in SQL serve different purposes. GROUP BY is used to group rows that have the same values in specified columns, often for the purpose of performing aggregate calculations like sums, averages, or counts. It reduces multiple rows into summary rows based on grouping. On the other hand, ORDER BY is used to sort the result set in either ascending or descending order based on one or more columns. While GROUP BY organizes data into logical groups, ORDER BY controls the sequence in which those results are displayed. Importantly, GROUP BY is typically used before ORDER BY in the SQL query execution process.

(SQL Stored Procedure)

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

A stored procedure is a precompiled set of SQL statements that are stored in the database and can be executed whenever needed. It can include SQL queries, logic (IF/ELSE), loops, and input/output parameters. Unlike a standard SQL query, which runs one-time and performs a specific task (like SELECT * FROM employees), a stored procedure can perform multiple operations, reuse logic, and accept parameters to be more dynamic.

```
CREATE PROCEDURE GetEmployeeByDept

  @DeptName VARCHAR(50)

AS

BEGIN

  SELECT * FROM employees WHERE department = @DeptName;

END;
```

2. Explain the advantages of using stored procedures.

Stored procedures offer several advantages:

- **Reusability**: You write it once and use it multiple times with different parameters.
- **Performance**: They are **precompiled**, so they often execute faster than regular SQL queries.
- **Security**: Users can be given access to execute the procedure **without accessing the underlying tables** directly.
- **Maintainability**: Logic is kept in one place in the database, making changes easier and centralized.
- **Reduced network traffic**: Since multiple operations can be performed in one call, it reduces the need to send multiple queries over the network.

(SQL View)

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

A view in SQL is a virtual table that is based on the result of a SELECT query. It does not store data physically like a table; instead, it shows data from one or more tables dynamically whenever the view is accessed. A table is a real, physical structure in the database that stores data in rows and columns.

Example

```
CREATE VIEW HighSalaryEmployees AS

SELECT name, salary FROM employees WHERE salary > 50000;
```

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

Views offer many advantages:

- **Simplifies complex queries**: You can write a complex SQL query once and save it as a view. Later, you can just SELECT from the view instead of writing the full query again.
- **Improved security**: You can restrict access to specific columns or rows in a table by creating a view, so users see **only the data they are allowed to see**.
- **Logical data independence**: If the structure of underlying tables changes, you can modify the view without changing the application code.
- **Data abstraction**: Views help hide the complexity of the database structure from end-users and provide a cleaner interface.
- **Reusability**: A view can be used in multiple queries, reports, or procedures without repeating the logic.

(SQL Triggers)

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

A **trigger** in SQL is a **special stored procedure** that **automatically executes** in response to certain events on a table or view. These events can be **INSERT**, **UPDATE**, or **DELETE** operations. Triggers are mainly used to **enforce business rules**, **maintain data integrity**, or **automate system tasks** like logging or validation.

Types of Triggers:

1. **BEFORE Trigger**
    - Executes **before** the triggering event (like before an INSERT, UPDATE, or DELETE).
    - Used for **validation or modification** before data is changed.

2. **AFTER Trigger**
    - Executes **after** the triggering event.
    - Commonly used for **logging**, **auditing**, or **related table updates**.

3. **INSTEAD OF Trigger**
    - Replaces the triggering operation.
    - Mostly used with **views** where direct INSERT/UPDATE/DELETE is not allowed.

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

Trigger Type	When It Fires	Example Use Case
INSERT	When a new row is added to a table	Automatically log new user registration
UPDATE	When an existing row is modified	Track changes in salary or address
DELETE	When a row is removed from a table	Backup deleted data to another table

(Introduction to PL/SQL)

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

PL/SQL (Procedural Language/Structured Query Language) is Oracle's extension to SQL that adds programming features like loops, conditions (IF...ELSE), variables, and exception handling to standard SQL. While SQL is used mainly for querying and manipulating data, PL/SQL allows you to write procedural code that can include multiple SQL statements along with logic and control flow.

- Stored procedures and functions
- Triggers
- Packages
- Cursors for row-by-row processing

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

Procedural Capabilities
Allows the use of **loops, conditions, and control structures**, which SQL alone doesn't support.

Improved Performance
Multiple SQL statements can be sent to the server as a **single block**, reducing network traffic.

Reusability
You can write **procedures, functions, and packages** once and use them repeatedly in different programs.

Maintainability
PL/SQL code is stored in the database. So if you need to change logic, you can update it in one place without changing application code.

Exception Handling
It provides robust **error handling** features using `EXCEPTION` blocks, which makes code safer and more reliable.

Security
You can **hide the implementation details** and give users access to only the procedure/function — not the actual data.

Tight Integration with SQL
PL/SQL is **fully compatible with SQL**, so you can easily combine them to create powerful database applications.

(PL/SQL Control Structures)

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

Control structures in PL/SQL are programming elements used to control the flow of execution. They allow PL/SQL programs to make decisions, perform iterations, and repeat tasks, which is not possible with standard SQL.

- 🔹 Types of Control Structures:
  - Conditional Control: `IF`, `IF-THEN`, `IF-THEN-ELSE`, `CASE`
  - Looping Control: `LOOP`, `WHILE LOOP`, `FOR LOOP`
  - Sequential Control: `GOTO`, `NULL`

IF-THEN Structure:

sed for decision-making — to execute certain code only if a condition is true.

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

```
IF salary > 50000 THEN
   bonus := 1000;
END IF;
```

LOOP Structure:

Used to **repeat a block of code** multiple times. You must use `EXIT` condition to stop the loop manually.

(i) LOOP

```
  -- statements

  EXIT WHEN condition;

END LOOP;
```

(ii) i := 1;

```
LOOP

  DBMS_OUTPUT.PUT_LINE(i);

  i := i + 1;

  EXIT WHEN i > 5;

END LOOP;
```

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

Control structures make PL/SQL powerful by allowing logic-based programming inside the database. Here's how they help:

- **Decision Making**: Using `IF` or `CASE`, you can perform different actions based on conditions (e.g., apply discounts if order value > ₹10,000).
- **Repetition**: With `LOOP`, `FOR`, and `WHILE`, you can handle **repetitive tasks** like processing rows one by one (e.g., for generating reports).
- **Automation of Complex Logic**: Control structures let you build **business rules, workflows**, and **dynamic operations** in stored procedures or triggers.
- **Improved Readability & Maintenance**: Makes the code **more modular and organized**, which helps when logic needs to be updated later.

(SQL Cursors)

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

A cursor in PL/SQL is a pointer to the result set of a SQL query. It allows row-by-row processing of query results, which is useful when you need to handle each row individually.

PL/SQL provides two types of cursors:

Implicit Cursor:

- Automatically created by PL/SQL when a DML statement (like `INSERT`, `UPDATE`, `DELETE`, or `SELECT INTO`) is executed.
- You don't need to declare or manage it.
- Good for queries that return only one row.

Example

```
SELECT salary INTO emp_salary FROM employees WHERE emp_id = 101;
```

Explicit Cursor:

- Created **manually** by the programmer for handling **queries that return multiple rows**.
- You must **declare, open, fetch, and close** the cursor.

Example:

```
CURSOR emp_cursor IS
  SELECT name, salary FROM employees;

OPEN emp_cursor;

FETCH emp_cursor INTO emp_name, emp_salary;

CLOSE emp_cursor;
```

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

You use an **explicit cursor** when:

- You need to **process multiple rows** one at a time (e.g., printing each employee's salary).
- You want **more control** over the fetching process.
- You need to **loop through records** with custom logic.
- You want to **handle each row individually** (e.g., applying different calculations per row).

(Rollback and Commit Savepoint)

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

A SAVEPOINT in SQL is a marker within a transaction that lets you set a point to which you can roll back later, without rolling back the entire transaction. It helps manage partial rollbacks within a transaction, giving more control and flexibility.

SAVEPOINT savepoint_name;

ROLLBACK TO savepoint_name;

Interaction:

- ROLLBACK TO savepoint_name: Undoes only the changes made after that savepoint.
- COMMIT: Saves all changes made in the transaction and removes all savepoints.
- ROLLBACK (without TO): Undoes everything in the transaction.

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

Savepoints are useful when:

- You want to test part of a transaction, and if something goes wrong, undo just that part.
- You are executing multiple DML operations, and you want the option to partially undo a few without losing the rest.
- In complex transactions, they help prevent full rollbacks, saving time and data.
- They're very helpful in error handling, where you can roll back only to the point where things were stable.

Example

```
BEGIN;

INSERT INTO orders VALUES (1, 'Laptop');

SAVEPOINT sp1;

INSERT INTO orders VALUES (2, 'Tablet'); -- Something goes wrong here

ROLLBACK TO sp1; -- Only 'Tablet' insert is undone

COMMIT; -- 'Laptop' insert is saved
```