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 manage and manipulate relational databases. It allows users to create, retrieve, update, and delete data efficiently.

SQL is essential because it provides a structured way to interact with databases, ensuring data consistency, integrity, and security. It is widely used across different database systems like MySQL, PostgreSQL, and SQL Server, making it a crucial tool for database management.

2.Explain the difference between DBMS and RDBMS.

Feature	DBMS (Database Management System)	RDBMS (Relational Database Management System)
Data Storage	Stores data as files or in a hierarchical format	Stores data in tables (rows and columns)
Relation	No relation between data	Data is organized with relationships
Normalization	Does not support normalization	Supports normalization to reduce redundancy
Data Integrity	Integrity constraints not strongly enforced	Enforces primary key, foreign key, and constraints
Examples	File System, XML-based DBs	MySQL, PostgreSQL, Oracle, SQL Server
Multi-user	Usually for single-user systems	Supports multi-user access
Security	Limited security features	Advanced security features and access controls

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

SQL (Structured Query Language) plays a crucial role in managing relational databases by allowing users to:

1. Create and Modify Structures
   - Use DDL (Data Definition Language) commands like CREATE, ALTER, and DROP to define and manage tables, schemas, and relationships.
2. Manipulate Data
   - Use DML (Data Manipulation Language) commands like INSERT, UPDATE, DELETE, and SELECT to manage the data stored in tables.
3. Control Access and Security
   - Use DCL (Data Control Language) commands like GRANT and REVOKE to manage user permissions and secure data access.
4. Ensure Data Integrity
   - Apply constraints (e.g., PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL) to maintain accuracy and consistency.
5. Query and Analyze Data
   - Retrieve specific information using powerful queries with filtering, sorting, grouping, and joins.

4.What are the key features of SQL?

Key Features of SQL:

Data Definition – Create and modify tables.

Data Manipulation – Insert, update, delete, and retrieve data.

Data Retrieval – Powerful querying with SELECT, JOIN, etc.

Data Control – Manage user access and permissions.

Transaction Control – Ensure data consistency with COMMIT, ROLLBACK.

Integrity Constraints – Maintain data accuracy with keys and rules.

Standardized – Supported by all major relational databases.

2. SQL Syntax

1. What are the basic components of SQL syntax?

Keywords: Reserved words used to perform actions, e.g., SELECT, INSERT, UPDATE, DELETE.

Identifiers: Names of database objects like tables, columns, and databases (e.g., students, id, name).

Clauses: Parts of statements that define conditions or limits, e.g., WHERE, GROUP BY, ORDER BY.

Expressions: Used to calculate values or filter data, e.g., age > 18, salary * 0.1.

Predicates: Conditions used to return true/false, often in WHERE clauses, e.g., id = 5.

Operators: Symbols used in conditions, e.g., =, >, <, AND, OR, LIKE.

Functions: Built-in methods like COUNT(), SUM(), AVG(), NOW().

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]
LIMIT number;
```

3. Explain the role of clauses in SQL statements.

Role of Clauses in SQL Statements:

Clauses in SQL are used to define specific operations and conditions within a query. Each clause serves a particular purpose:

1. SELECT: Specifies the columns to retrieve from a table.
2. FROM: Indicates the table from which to fetch data.
3. WHERE: Filters rows based on specified conditions.
4. GROUP BY: Groups rows that have the same values, typically used with aggregate functions like COUNT (), SUM (), etc.
5. HAVING: Filters groups formed by GROUP BY, usually with aggregate functions.
6. ORDER BY: Sorts the result set in ascending or descending order based on one or more columns.
7. LIMIT: Limits the number of rows returned by the query.

3. SQL Constraints

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

SQL Constraints:

1. PRIMARY KEY: Uniquely identifies each record; no NULL values allowed.
2. FOREIGN KEY: Creates a relationship between tables; values must match a primary key in another table.
3. UNIQUE: Ensures all values in a column are unique, allowing NULL.
4. NOT NULL: Ensures a column cannot have NULL values.
5. CHECK: Ensures values meet a specific condition or range.
6. DEFAULT: Sets a default value for a column if no value is provided.
7. INDEX: Improves query performance by indexing a column.

These constraints ensure data integrity, accuracy, and consistency in SQL databases.

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

 PRIMARY KEY:

- Uniquely identifies each record in a table.
- Cannot have NULL values.
- Each table can have only one PRIMARY KEY.

 FOREIGN KEY:

- Creates a relationship between two tables.
- Refers to the PRIMARY KEY of another table.
- Can have NULL values, but must match a value in the referenced table's PRIMARY KEY.

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

 NOT NULL:

- Ensures a column cannot have NULL values, enforcing that a value must be provided.

 UNIQUE:

- Ensures all values in a column are distinct, but allows NULL values (only one NULL is allowed in some databases).

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

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

DDL is used to define and manage the structure of database objects.

Common DDL Commands:

1. CREATE: Creates a new table or object.
2. ALTER: Modifies an existing object.
3. DROP: Deletes an object.
4. TRUNCATE: Removes all records from a table, keeping its structure.

In short, DDL defines and modifies database structures.

2. Explain the CREATE command and its syntax.

The CREATE command in SQL is used to create database objects like tables, views, indexes, and databases.

Syntax for Creating a Table:

```
CREATE TABLE table_name (

column1 datatype [constraint],

column2 datatype [constraint],

...

);
```

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

Purpose of Specifying Data Types and Constraints:

- **DataTypes**:
  Define the type of data (e.g., INT, VARCHAR, DATE) a column can store, ensuring the correct storage format and efficient data handling.
- **Constraints**:
  Enforce rules like **uniqueness**, **non-nullability**, and **relationships** (e.g., PRIMARY KEY, NOT NULL) to maintain data integrity and consistency.

In short, **data types** ensure proper data storage, and **constraints** enforce rules to maintain data accuracy and integrity.

5. ALTER Command

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

The **ALTER** command is used to modify the structure of an existing database object, such as a table. It allows you to add, delete, or modify columns and constraints.

Common Uses:

- Add a new column: ALTER TABLE table_name ADD column_name datatype;
- Modify a column: ALTER TABLE table_name MODIFY column_name datatype;
- Drop a column: ALTER TABLE table_name DROP COLUMN column_name;

In short, **ALTER** is used to change the structure of database objects after they have been created.

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

1.Add a Column:

ALTER TABLE table_name ADD column_name datatype;

2.Modify a Column:

ALTER TABLE table_name MODIFY column_name new_datatype;

3.Drop a Column:

ALTER TABLE table_name DROP COLUMN column_name;

6. DROP Command

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

The **DROP** command is used to delete an existing database object, such as a table, view, or index. It completely removes the object and all its data from the database.

Drop a Table

```
DROP TABLE table_name;
```

Drop a Database:

```
DROP DATABASE database name;
```

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

Implications of Dropping a Table:

1. **DataLoss**:
   All data stored in the table is permanently deleted.
2. **StructureRemoval**:
   The table structure (columns, constraints, etc.) is also removed from the database.
3. **DependentsAffected**:
   Any foreign key relationships or dependent views or indexes may break.

In short, **dropping a table** permanently removes the table and its data, and can affect related objects in the database.

7. Data Manipulation Language (DML)

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

INSERT, UPDATE, and DELETE Commands in SQL:

1.INSERT:
Adds new records (rows) to a table.

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

2.**UPDATE**:
Modifies existing records in a table.

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

3.DELETE:
Removes records from a table.

```
DELETE FROM table_name WHERE condition.
```

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

Importance of the WHERE Clause in UPDATE and DELETE:

- **UPDATE**:
  The **WHERE** clause specifies which records to modify. Without it, all records in the table will be updated.
- **DELETE**:
  The **WHERE** clause specifies which records to delete. Without it, all records in the table will be removed.

In short, the **WHERE** clause ensures that **only specific records** are updated or deleted, preventing unintended changes to the entire table.

8. Data Query Language (DQL)

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

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

Syntax:

```
SELECT column1, column2, ...
FROM table name
WHERE condition;
```

- **SELECT**: Specifies the columns to retrieve.
- **FROM**: Indicates the table to query data from.
- **WHERE**: Filters the results based on conditions.

In short, **SELECT** is used to **query and retrieve specific data** from a database based on given conditions.

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

ORDERBY:
Sorts the result set in ascending (ASC) or descending (DESC) order based on one or more columns.
SELECT * FROM table name ORDER BY column name ASC|DESC;

WHERE:
Filters records based on specified conditions, ensuring that only rows meeting the condition are returned.
SELECT * FROM table name WHERE condition;

In short, **ORDER BY** sorts the data, while **WHERE** filters the data based on given conditions.

9. Data Control Language (DCL)

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

GRANT:
Gives specific privileges (like SELECT, INSERT, UPDATE, etc.) to users or roles on database objects.
GRANT SELECT, INSERT ON table_name TO user_name;

REVOKE:
Removes specific privileges from users or roles.

REVOKE SELECT, INSERT ON table_name FROM user_name;

In short, **GRANT** assigns privileges, while **REVOKE** removes them from users or role

2.How do you manage privileges using these commands?

To manage privileges using **GRANT** and **REVOKE**:

1. **GRANT**:
   Assign specific privileges to a user or role on a table.

   GRANT SELECT, INSERT ON table_name TO user_name;

2. **REVOKE**:
   Remove specific privileges from a user or role.

   REVOKE SELECT, INSERT ON table_name FROM user_name;

In short, **GRANT** gives privileges, and **REVOKE** removes them to control access.

10. Transaction Control Language (TCL)

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

COMMIT:Saves all changes made during the current transaction permanently to the database.

```
COMMIT;
```

ROLLBACK:
Undoes all changes made during the current transaction, reverting to the state before the transaction started.

```
ROLLBACK;
```

COMMIT saves changes, and **ROLLBACK** undoes changes in a transaction.

2.Explain how transactions are managed in SQL databases.

Managing Transactions in SQL:

Transactions in SQL are managed using the following commands:

1. **BEGINTRANSACTION**:
   Starts a new transaction.

   ```
   BEGIN TRANSACTION;
   ```

2. **COMMIT**:
   Saves all changes made during the transaction permanently.

   ```
   COMMIT;
   ```

3. **ROLLBACK**:
   Undoes all changes made during the transaction, reverting to the previous state.

   ```
   ROLLBACK;
   ```

In short, transactions ensure that all changes are completed successfully (via **COMMIT**) or not applied at all (via **ROLLBACK**), ensuring data integrity.

11. SQL Joins

Theory Questions:

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** combines data from two or more tables based on a related column.

Types of Joins:

1. **INNERJOIN**:
   Returns only matching rows from both tables.

   SELECT * FROM table1 INNER JOIN table2 ON table1.id = table2.id;

2. **LEFTJOIN**:
   Returns all rows from the left table and matching rows from the right table. Non-matching rows from the right table are NULL.

   SELECT * FROM table1 LEFT JOIN table2 ON table1.id = table2.id;

3. **RIGHTJOIN**:
   Returns all rows from the right table and matching rows from the left table. Non-matching rows from the left table are NULL.

   SELECT * FROM table1 RIGHT JOIN table2 ON table1.id = table2.id;

4. **FULLOUTERJOIN**:
   Returns all rows when there's a match in either table, with NULL where there's no match.

   SELECT * FROM table1 FULL OUTER JOIN table2 ON table1.id = table2.id;

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

Joins combine data from multiple tables by matching rows using a common column (like a key).

```
SELECT *

FROM orders

JOIN customers ON orders.customer_id = customers.id;
```

his returns a result with data from both **orders** and **customers** where the customer_id matches.

In short, **joins link rows from different tables** based on related columns to fetch combined results.

12. SQL Group By

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

- The **GROUP BY** clause groups rows that have the same values in specified columns. It’s commonly used with **aggregate functions** like SUM(), AVG(), COUNT(), etc., to perform calculations on each group.
- **Example:**
- SELECT department, COUNT(*)
- FROM employees
- GROUP BY department;
- In short, **GROUP BY** organizes data into groups and applies aggregate functions to each group.

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

- **GROUPBY**:
  Groups rows based on column values to apply aggregate functions like SUM(), COUNT(), etc.
- **ORDERBY**:
  Sorts the final result in ascending or descending order based on one or more columns.

Example:

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

-- ORDER BY
SELECT * FROM employees ORDER BY salary DESC;
```

⍰ **GROUP BY** = Grouping data

⍰ **ORDER BY** = Sorting data

13. SQL Stored Procedure

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

Stored Procedure vs Standard SQL Query (Shortly):

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

Difference:

- **Stored Procedure**: Reusable, accepts parameters, can include logic (loops, conditions).
- **Standard SQL Query**: One-time command executed directly.

2.Explain the advantages of using stored procedures.

- Reusability – Write once, use many times.
- Performance – Precompiled for faster execution.
- Security – Restrict direct table access; control via procedures.
- Maintainability – Easier to update logic in one place.
- Modularity – Organize complex logic into manageable units.

In short: Stored procedures improve performance, security, and code manageability

14. SQL View

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

A **view** is a virtual table based on a SQL query. It shows data from one or more tables but doesn’t store data itself.

Difference:

- **Table**: Physically stores data.
- **View**: Stores a query, shows data dynamically from tables.

Inshort:
A **view** is a saved SQL query acting like a table but without storing data

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

Advantages of Views in SQL (Shortly):

1. **Simplifies complex queries**
2. **Enhances security** by hiding sensitive data
3. **Improves readability** and maintenance
4. **Provides data abstraction**
5. **Allows customized data representation**

In short: Views make data access easier, safer, and more organized.

15. SQL Triggers

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

Trigger in SQL is a set of instructions that automatically executes (fires) in response to certain events on a table or view, such as **INSERT**, **UPDATE**, or **DELETE**.

Types of Triggers:

1. **BEFORE Trigger** – Executes **before** the operation (INSERT/UPDATE/DELETE).
   Used for validation or modifying values before saving.
2. **AFTER Trigger** – Executes **after** the operation.
   Used for logging, sending notifications, or maintaining audit tables.
3. **INSTEAD OF Trigger** – Replaces the triggering action (mainly used on views).
   Used when you want to customize the data modification logic on a view.

Usage:

- Enforcing business rules
- Automatic auditing/logging
- Preventing invalid transactions
- Cascading actions

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

- **INSERT Trigger**:
  Fires **when a new record is added** to a table.
  Used to validate or log inserted data.
- **UPDATE Trigger**:
  Fires **when existing data is modified**.
  Used to track changes or prevent unwanted updates.
- **DELETE Trigger**:
  Fires **when a record is removed** from a table.
  Used to archive deleted data or restrict deletion.

Each trigger type can be defined as **BEFORE**, **AFTER**, or **INSTEAD OF** based on when you want it to execute.

16. Introduction to PL/SQL:

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

PL/SQL (Procedural Language/SQL) is Oracle's procedural extension to SQL, designed for writing full programs to interact with databases.

How PL/SQL extends SQL's capabilities:

1. **Procedural Logic** – Adds programming constructs like loops, conditions (IF, CASE), and variables.
2. **Modularity** – Supports functions, procedures, and packages for reusability.
3. **Error Handling** – Allows handling of runtime errors using EXCEPTION blocks.
4. **Improved Performance** – Executes multiple SQL statements in a single block, reducing server-client communication.
5. **Triggers and Cursors** – Enables complex data manipulation and control over query results.

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

1. **Block Structure**: Code is organized into logical blocks (DECLARE, BEGIN, EXCEPTION, END) for better readability and management.
2. **Procedural Capabilities**: Supports loops, conditions, and variables—making it more powerful than plain SQL.
3. **Improved Performance**: Reduces network traffic by sending a block of SQL statements to the database at once.
4. **Error Handling**: Built-in exception handling allows you to catch and manage errors gracefully.
5. **Code Reusability**: Functions, procedures, and packages can be reused across applications.
6. **Tight Integration with SQL**: Seamlessly combines procedural code with SQL for efficient data manipulation.
7. **Security**: Can restrict direct access to tables and allow controlled access through procedures.

PL/SQL is ideal for writing complex business logic within the database.

17. PL/SQL Control Structures

1.What are control structures in PL/SQL? Explain the IF-THEN and LOOP control structures.
Control structures are used to control the **flow of execution** in a PL/SQL program. They help in making decisions and repeating actions.

1. IF-THEN Structure

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

Syntax:

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

Example:

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

2. LOOP Structure

Used to **repeat** a block of code **multiple times**.

Basic LOOP Syntax:

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

Example:

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

These structures help in adding logic and repetition to PL/SQL programs.

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

Control structures like **IF-THEN**, **LOOP**, **CASE**, and **WHILE** make PL/SQL more powerful than SQL alone by enabling:

1. **Conditional Logic**

   - Execute different SQL statements based on conditions (e.g., IF salary > 50000 THEN ...).

2. **Iteration**
   - Repeat tasks using loops (e.g., processing rows one by one using a CURSOR and LOOP).
3. **Dynamic Decisions**
   - Handle multiple scenarios in real-time using IF-ELSE or CASE.
4. **Efficient Error Handling**
   - Skip or manage errors during execution using EXCEPTION blocks inside control structures.
5. **Modular Code**
   - Break down large logic into smaller, manageable, and reusable blocks.

In short, control structures add flexibility, logic, and flow control, enabling developers to write smarter and more complex PL/SQL programs.

18. SQL Cursors

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

A **cursor** is a pointer to the **context area** that stores the result of a SQL query. It allows row-by-row processing of query results in PL/SQL.

Types of Cursors:

1. Implicit Cursor

- Automatically created by Oracle for **single-row queries** like INSERT, UPDATE, DELETE, or SELECT INTO.
- No need to declare or open it manually.

Example:

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

2. Explicit Cursor

- **Manually declared and controlled** by the programmer for queries returning **multiple rows**.
- Requires OPEN, FETCH, and CLOSE.

Example:

```
CURSOR emp_cur IS SELECT * FROM employees;
OPEN emp_cur;
FETCH emp_cur INTO var1, var2;
CLOSE emp_cur;
```

Key Differences:

Feature	**Implicit Cursor**	**Explicit Cursor**
Declaration	Automatic	Manual
Use	Single-row operations	Multi-row queries
Control	Managed by Oracle	Controlled by user (OPEN/FETCH/CLOSE)
Flexibility	Limited	High (can loop through rows)

In short, implicit cursors are simple and automatic, while explicit cursors offer more control for handling multiple rows.

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

You would use an **explicit cursor** when:

- The query **returns multiple rows**.
- You need to **loop through each row** and apply custom logic.
- You want to **control the fetch process** (OPEN, FETCH, CLOSE).
- You need to **handle complex business rules** per row.