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

ANS - SQL (Structured Query Language) is a standardized programming language specifically designed to manage and manipulate relational databases. It provides a powerful and efficient way to interact with data by enabling users to create, read, update, and delete data stored in a database.

-> Why is it essential in database management?

Ans - SQL is a standardized language (by ANSI and ISO), making it widely used and compatible with various database management systems like MySQL, PostgreSQL, Oracle, and SQL Server.

- SQL is optimized for relational databases, making it faster and more efficient than other data manipulation languages for managing large datasets and complex relationships.

- SQL commands can enforce rules and constraints on data, maintaining consistency and ensuring only authorized access.

- SQL is scalable and supports complex operations and large-scale applications, making it suitable for everything from small databases to large enterprise systems.

Q-2 Explain the difference between DBMS and RDBMS.

ANS –

1. DBMS:

- Data is stored as files in a hierarchical or navigational model.

- Rarely uses data normalization techniques.

- Limited support for data integrity constraints.

- DBMS does not apply any security with regards to data manipulation.

- DBMS uses file system to store data, so there will be no relation between the tables.

- DBMS is meant to be for small organization and deal with small data. it supports single user.

- Examples of DBMS are file systems, xml etc.

2. RDBMS:

- RDBMS applications store data in a tabular form.

- Normalization is present in RDBMS.

- Strongly enforces data integrity with primary, unique, and foreign key constraints.

- RDBMS defines the integrity constraint for the purpose of ACID.

- in RDBMS, data values are stored in the form of tables, so a relationship between these data values will be stored in the form of a table as well.

- RDBMS is designed to handle large amount of data. it supports multiple users.

- Example of RDBMS are mysql, postgre, sql server, oracle etc.

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

ANS - SQL defines the structure of the database. Using Data Definition Language (DDL) commands like CREATE, ALTER, and DROP, users can create tables, define relationships, and set constraints.

- SQL enables users to manipulate and manage data within the database. Data Manipulation Language (DML) commands like INSERT, UPDATE, DELETE, and SELECT allow users to add, modify, and retrieve data.

- SQL controls access to the database, helping secure data by specifying who can view or modify it. Data Control Language (DCL) commands like GRANT and REVOKE allow administrators to manage user permissions

- SQL supports managing transactions through Transaction Control Language (TCL) commands such as COMMIT, ROLLBACK, and SAVEPOINT. These commands ensure data integrity by allowing changes to be grouped into transactions.

- SQL supports the enforcement of data integrity by allowing constraints like PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, and CHECK. These constraints ensure data is stored accurately, preventing errors and maintaining relational integrity between tables.

Q-4 What are the key features of SQL?

Ans - Data Retrieval:

- SQL allows users to retrieve data using commands like SELECT, which can specify which columns to retrieve and use filtering (WHERE), grouping (GROUP BY), and sorting (ORDER BY) to refine query results.

- Data Manipulation:

- SQL supports Data Manipulation Language (DML) commands like INSERT, UPDATE, and DELETE to add, modify, and delete data in tables.

- Data Definition:

- SQL provides Data Definition Language (DDL) commands such as CREATE, ALTER, and DROP to create and modify database objects (tables, views, indexes).

- Data Control:

- SQL includes Data Control Language (DCL) commands, such as GRANT and REVOKE, to manage permissions on database objects, controlling access to data for different users.

- Transaction Control:

- SQL supports transaction control commands like BEGIN, COMMIT, and ROLLBACK to manage transactions.

- Built-in Functions:

- SQL includes numerous functions for performing calculations, data formatting, string manipulation, and date handling. Examples include COUNT, SUM, AVG, MIN, MAX, UPPER, LOWER, DATEPART, etc.

- Data Integrity:

- SQL enforces data integrity through constraints like PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, and NOT NULL, ensuring that the data in the database is accurate and reliable.

- Joins and Relationships:

- SQL supports JOIN operations, allowing data from multiple tables to be combined based on related columns, which helps in establishing relationships between tables in a relational database.

- Views:

- SQL allows the creation of virtual tables known as views, which are saved queries that provide a specific perspective of the data without storing it physically, improving security and simplifying data access.

LAB EXERCISES: 1

Lab 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.

ANS - Database created done

Lab 2: Insert five records into the students table and retrieve all records using the SELECT

statement.

ANS - Insert five records into the students table and retrive all records below query.

SELECT * FROM `students`

2. SQL Syntax

Q-1 What are the basic components of SQL syntax?

ANS- Basic Components of SQL:

1. Keywords:

- SQL uses specific reserved words (keywords) to perform various operations. Common keywords include:

  SELECT, FROM, WHERE, INSERT, UPDATE, DELETE, JOIN, ORDER BY, GROUP BY, HAVING, LIMIT.

2. Clauses:

- Clauses define specific actions in SQL statements and are often based on keywords.

  -SELECT clause to specify which columns to retrieve.

  -FROM clause to specify the table(s) from which to retrieve data.

  -WHERE clause to filter data based on specific conditions.

  -ORDER BY clause to sort the results.

3. Expressions:

- Expressions are used to produce scalar values, such as 2 + 2, or evaluate conditions, like salary > 50000.

4. Predicates:

- Predicates are conditions used in SQL statements to filter data or control flow, often found in WHERE, HAVING, and JOIN clauses.

- Common predicates include:

  Comparison (=, >, <, >=, <=, !=)

  Logical operators (AND, OR, NOT)

  Null checks (IS NULL, IS NOT NULL)

5. Identifiers:

- Identifiers name the database elements, such as tables, columns, and aliases.

6. Functions:

- SQL provides various functions to perform calculations, format data, or extract information.

  Examples include COUNT(), SUM(), MAX(), MIN(), AVG().

7. Data Types:

- Data types define the kind of data a column can store, such as INTEGER, VARCHAR, DATE, BOOLEAN, etc.

Q-2 Write the general structure of an SQL SELECT statement.

ANS- SELECT column1, column2,...

FROM table_name

WHERE condition

GROUP BY column

HAVING condition

ORDER BY column ASC|DESC

LIMIT number;

Q-3 Explain the role of clauses in SQL statements.

ANS - 1. SELECT Clause:

- The SELECT clause is the foundation of data retrieval in SQL. It specifies the columns of data that you want to retrieve from a database table or tables.

Example: SELECT student_name, age FROM STUDENTS;

2. FROM Clause:

- The FROM clause indicates the table (or tables) from which to retrieve or modify data.

It works with SELECT, UPDATE, and DELETE statements to define the data source.

Example: SELECT * FROM students;

3. WHERE Clause:

- The WHERE clause filters records, specifying the conditions that must be met for rows to be selected.

- It's used with SELECT, UPDATE, and DELETE statements.

- Example: SELECT * FROM students WHERE id = 1;

4. GROUP BY Clause:

- GROUP BY organizes rows that have the same values in specified columns into summary rows.

- Commonly used with aggregate functions (like SUM, COUNT, AVG).

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

5. HAVING Clause:

- HAVING works similarly to WHERE but is used to filter groups, not individual rows.

Example: SELECT department, COUNT (*) FROM employees GROUP BY department HAVING COUNT(*) > 10;

6. ORDER BY Clause:

- ORDER BY sorts the result set in ascending or descending order based on one or more columns.

7. JOIN Clause:

- The JOIN clause combines rows from two or more tables based on a related column.

Types of joins include INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN.

- Example: SELECT employees.name, departments.name FROM employees INNER JOIN departments ON employees.dept_id = departments.id;

8. LIMIT (or FETCH) Clause:

- LIMIT restricts the number of rows returned in a result set, which is especially useful for large datasets.

- Example: SELECT * FROM orders LIMIT 10;

9. INSERT INTO, UPDATE, DELETE Clauses:

- INSERT INTO is used to add new rows to a table.

- UPDATE modifies existing rows in a table.

- DELETE removes rows from a table.

- Examples: INSERT INTO employees (name, age) VALUES ('John Doe', 30);

    - UPDATE employees SET age = 31 WHERE name = 'John Doe';

    - DELETE FROM employees WHERE name = 'John Doe';

LAB EXERCISES:2

Lab 1: Write SQL queries to retrieve specific columns (student_name and age) from the

students table.

ANS - SELECT `student_name`, `age` FROM `students`

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

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

3. SQL Constraints

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

ANS - Constraints restrict the types of data that can be inserted into a column, prevent duplicate data, and maintain relationships between tables.

- List and explain the different types of constraints.

1. NOT NULL Constraint:

- Ensures that a column cannot contain NULL values.

- For example, if a column is specified as NOT NULL, you must provide a value for that column when inserting or updating data in the table.

2. UNIQUE Constraint:

- Prevents duplicate values from being entered in the specified column(s).

- Unlike the primary key, a table can have multiple UNIQUE constraints.

3. PRIMARY KEY Constraint:

- Uniquely identifies each record in a table.

- A primary key combines the properties of NOT NULL and UNIQUE constraints, meaning it ensures that no duplicate or NULL values can be entered.

4. FOREIGN KEY Constraint:

- Enforces a link between data in two tables, ensuring referential integrity.

- A foreign key in one table points to a primary key in another table.

5. CHECK Constraint:

- This constraint allows you to define a condition that each value in a column must satisfy, for example, CHECK (age >= 18) to enforce a minimum age requirement.

6. DEFAULT Constraint:

- Ensures that a column has a predefined value when an INSERT operation omits it, helping maintain consistency in data entry.

7. INDEX (not a constraint but commonly used):

-Indexes allow faster retrieval of rows by creating pointers to where data is stored, although they can impact the performance of INSERT, UPDATE, and DELETE operations due to added maintenance.

Q-2 How do PRIMARY KEY and FOREIGN KEY constraints differ?

ANS -

1. Primary Key:

- A primary key uniquely identifies each record in a table.

- Each primary key value must be unique and cannot be null.

- Each table can have only one primary key, though it may consist of multiple columns (composite primary key).

- Ensures that each record in the table is unique and acts as the main identifier for that record.

Example: The id column in a STUDENTS table uniquely identifies each students.

2. Foreign Key:

- A foreign key is a column (or set of columns) in one table that creates a relationship between itself and the primary key in another table.

- It enforces that the values in the foreign key column(s) must match values in the referenced primary key column(s), or be null if allowed.

- A table can have multiple foreign keys, each potentially pointing to different tables.

-Example: An order_id column in an Orders table might reference the id in Customer to associate each order with a specific customer.

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

ANS -

1. NOT NULL Constraint:

- The NOT NULL constraint prevents null (empty) values from being inserted into a column.

- This constraint ensures that a column must always have a value; it cannot be left blank.

- It is commonly used when a value is required for every row in the table, such as an ID or name field.

2. UNIQUE Constraint:

- The UNIQUE constraint ensures that all values in a column are different, preventing duplicate entries in that column.

- Unlike the PRIMARY KEY constraint, which also enforces uniqueness, a table can have multiple UNIQUE constraints but only one PRIMARY KEY.

-Differences of both:

- Purpose: NOT NULL ensures a column cannot be left empty, while UNIQUE ensures no two rows can have the same value in the column.

- Combining Constraints: You can use NOT NULL and UNIQUE together on a column to require both a value and uniqueness, often used for fields like emails, usernames, or other identifiers.

LAB EXERCISES: 3

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

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

ANS - DONE

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

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

ANS - DDL statements are primarily used to create, modify, and delete these objects within a database.

- The primary DDL commands are:

1. CREATE: Defines and creates new database objects, such as tables, views, indexes, and schemas.

Example: CREATE TABLE employees (id INT, name VARCHAR (50), age INT);

2. ALTER: Modifies existing database objects, such as adding or removing columns in a table or changing column properties.

Example: ALTER TABLE employees ADD COLUMN salary DECIMAL (10, 2);

3. DROP: Deletes database objects, permanently removing them from the database.

Example: DROP TABLE employees;

4. TRUNCATE: Removes all records from a table without deleting the table itself, and typically does so more quickly than a DELETE statement since it doesn't log each row individually.

Example: TRUNCATE TABLE employees;

5. RENAME: Changes the name of a database object.

Example: ALTER TABLE employees RENAME TO staff;

Q-2 Explain the CREATE command and its syntax.

ANS - CREATE command Defines and creates new database objects, such as tables, views, indexes, and schemas.

Example: CREATE TABLE employees (id INT, name VARCHAR(50), age INT);

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

ANS - Specifying data types and constraints during table creation in a database serves several key purposes, all of which help ensure data integrity, improve performance, and clarify the structure of the data being stored. Here's a breakdown of the main reasons:

1. Data Types: By specifying the data type (e.g., INT, VARCHAR, DATE), you ensure that only the appropriate kind of data can be inserted into each column. For example, if you specify an INT data type for a column, only integer values will be accepted, preventing the insertion of non-numeric data.

2. Constraints: Constraints help enforce rules about how data should behave. For example:

(A) PRIMARY KEY: Ensures that each row in a table has a unique identifier and prevents duplicate values.

(B) FOREIGN KEY: Ensures referential integrity between tables by linking a column to a primary key or unique key in another table.

(C) NOT NULL: Ensures that a column cannot contain NULL values, guaranteeing that each record has meaningful data for that field.

(D) CHECK: Restricts the range or values that can be stored in a column (e.g., ensuring an AGE column only accepts values greater than 0).

LAB EXERCISES:4

Lab 1: Create a table courses with columns: course_id, course_name, and

course_credits. Set the course_id as the primary key.

ANS - DONE

Lab 2: Use the CREATE command to create a database university_db.

ANS - CREATE DATABASE university_db;

5. ALTER Command

Q-1 What is the use of the ALTER command in SQL?

ANS - The ALTER command in SQL is used to modify the structure of an existing database object, such as a table, column, or index. It allows you to change the schema of a database object without affecting the data that is already stored in it.

Add or remove constraints: You can also use ALTER to add or drop constraints, such as primary keys, foreign keys, unique constraints, etc

Syntax : ALTER TABLE table_name

ADD CONSTRAINT/DROP CONSTRAINT constraint_name constraint_definition;

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

Ans - Common uses of the ALTER command:

1. Add New Column:

Syntax : ALTER TABLE table_name

ADD column_name datatype;

2. Modify an existing column:

Syntax : ALTER TABLE table_name

MODIFY column_name new_datatype;

3. Rename a table or column:

Syntax : ALTER TABLE old_table_name

RENAME TO new_table_name;

4. Drop a column:

Syntax : ALTER TABLE table_name

DROP COLUMN column_name;

LAB EXERCISES:5

Lab 1: Modify the courses table by adding a column course_duration using the ALTER

command.

ANS - ALTER TABLE courses ADD course_duration int;

Lab 2: Drop the course_credits column from the courses table.

ANS - ALTER TABLE courses

DROP COLUMN course_credits;

6. DROP Command

Q-1 What is the function of the DROP command in SQL?

ANS - The DROP command in SQL is used to remove database objects permanently. This command completely deletes the specified object and all of its associated data, structures, and definitions from the database.

Common uses of the DROP command:

1. DROP DATABASE: Removes an entire database, including all tables, views, indexes, stored procedures, and data within that database.

2. DROP TABLE: Deletes a specific table and all of its data from the database.

3. DROP VIEW: Removes a view from the database.

4. DROP INDEX: Removes an index from a table.

5. DROP PROCEDURE: Deletes a stored procedure.

6. DROP TRIGGER: Removes a trigger from the database.

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

ANS - Dropping a table from a database has significant implications, as it permanently removes the table and all its associated data and structures. Here are the key implications:

1. Data Loss:

Permanent Data Removal: When a table is dropped, all the data stored within that table is permanently deleted.

No Recovery: Once the table is dropped, the data cannot be recovered through regular SQL commands like ROLLBACK.

2. Loss of Table Structure:

Table Definition Gone: Dropping a table removes not only its data but also its structure, including all columns, data types, and constraints (such as primary keys, foreign keys, unique constraints, etc.).

Associated Indexes Removed: Any indexes created on the table (including primary key or unique indexes) are also dropped.

3. Impact on Dependencies:

Foreign Key Constraints: If other tables have foreign key references to the table being dropped, the database may throw an error if those constraints are not properly handled.

Views, Stored Procedures, and Triggers: If views, stored procedures, or triggers depend on the table, they may fail after the table is dropped.

4. No Undo Option:

Irreversible Action: The DROP TABLE command does not offer an automatic undo feature. Once executed, there is no way to recover the dropped table unless you have a backup.

Some databases provide a CASCADE option that automatically handles dependent objects, but this requires careful management.

LAB EXERCISES:

Lab 1: Drop the teachers table from the school_db database.

ANS - DROP TABLE teachers;

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

ANS - DROP TABLE students;

SELECT * FROM students----but data not retrieve...

7. Data Manipulation Language (DML)

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

ANS - 1. INSERT: The INSERT command is used to add new rows of data into a table. You specify the table and the values to be inserted for each column.

Syntax:

INSERT INTO table_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);

2. UPDATE: The UPDATE command is used to modify existing data in a table. You specify the table, the columns to be updated, and the new values. You also use a WHERE clause to specify which rows should be updated.

Syntax:

UPDATE table_name

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

WHERE condition;

3. DELETE: The DELETE command is used to remove one or more rows from a table. Similar to UPDATE, the DELETE statement typically includes a WHERE clause to specify which rows to delete.

Syntax:

DELETE FROM table_name

WHERE condition;

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

ANS - The WHERE clause allows you to target specific rows for updating or deleting, ensuring that only the intended records are affected. Without it, every record in the table would be modified or deleted, which could result in significant data loss or corruption.

Example:

->UPDATE: Without a WHERE clause, an UPDATE would change all records in the table.

- UPDATE employees SET salary = 50000;

->This would set the salary of every employee in the table to 50,000.

-> DELETE: Without a WHERE clause, a DELETE would remove all records from the table.

- DELETE FROM employees;

-> This would delete every row from the employees table.

The WHERE clause allows you to apply specific filters using conditions (e.g., based on column values, ranges, or comparisons) to select the exact data to update or delete. This provides flexibility and control over the operation.

LAB EXERCISES:

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

ANS - INSERT INTO courses (course_id, course_name, course_duration)

VALUES

(1, 'Introduction to Computer Science', 3),

(2, 'Database Systems', 4),

(3, 'Data Structures', 3);

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

ANS - UPDATE `courses` SET `course_duration`='6' WHERE`course_name`='Database Systems';

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

ANS - DELETE FROM courses

WHERE course_id = 2;

8. Data Query Language (DQL)

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

ANS - The SELECT statement is a fundamental SQL command used to query and retrieve data from one or more tables in a relational database. It allows users to specify which columns of data they want to see and from which table, as well as filter, sort, and manipulate the data according to their needs.

- Syntax:

SELECT column1, column2, ...

FROM table_name;

- If you want to retrieve all data from a table called table name, you can use:

SELECT *FROM tablename;

- If you only want to retrieve specific columns (e.g., employee names and salaries), you can specify them:

SELECT employee_name, salary

FROM employees;

- You can add a WHERE clause to filter the data based on specific conditions:

SELECT employee_name, salary

FROM employees

WHERE department = 'Sales';

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

ANS - 1. WHERE Clause:

The WHERE clause is used to filter the records from the database based on specific conditions. It is placed after the FROM clause in a query and allows you to restrict the rows returned by the query.

Syntax:

SELECT column1, column2, ...

FROM table_name

WHERE condition;

2. ORDER BY Clause:

The ORDER BY clause is used to sort the results of a query in either ascending or descending order. It is placed after the WHERE clause (if both are used) and before the LIMIT or OFFSET clauses (if applicable).

Syntax:

SELECT column1, column2, ...

FROM table_name

ORDER BY column1 [ASC|DESC], column2 [ASC|DESC], ...;

LAB EXERCISES:

Lab 1: Retrieve all courses from the courses table using the SELECT statement.

ANS - SELECT * FROM `courses`;

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

ANS - SELECT * FROM courses

ORDER BY course_duration DESC;

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

ANS - SELECT *

FROM courses

LIMIT 2;

9. Data Control Language (DCL)

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

ANS - In SQL, GRANT and REVOKE are commands used to manage permissions or privileges on database objects such as tables, views, and procedures.

1. GRANT:

- The main purpose of the GRANT command is to provide access to users or roles, enabling them to interact with database objects.

2. REVOKE

- The REVOKE command ensures that a user or role no longer has access to certain database actions, effectively removing their permissions.

Q- 2 How do you manage privileges using these commanREVOKE INSERT ON employees FROM user1;

ds?

ANS - 1. GRANT: The GRANT statement is used to assign privileges to users or roles, allowing them to perform specific actions (e.g., SELECT, INSERT, UPDATE, DELETE) on database objects.

Syntax Example:

GRANT SELECT, INSERT ON employees TO user1;

- This example gives the user user1 permission to SELECT and INSERT data from the employees table.

2. REVOKE: The REVOKE statement is used to remove or restrict privileges that were previously granted to users or roles.

Syntax Example:

REVOKE INSERT ON employees FROM user1;

LAB EXERCISES:

Lab 1: Create two new users user1 and user2 and grant user1 permission to SELECT

from the courses table.

ANS -

```
-- Create user1

CREATE USER 'user1'@'localhost' IDENTIFIED BY 'password1';

-- Create user2

CREATE USER 'user2'@'localhost' IDENTIFIED BY 'password2';

-- Grant SELECT permission to user1 on the courses table

GRANT SELECT ON school_db.courses TO 'user1'@'localhost';
```

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

```
-- Revoke INSERT permission from user1 on the courses table
REVOKE INSERT ON school_db.courses FROM 'user1'@'localhost';
-- Grant INSERT permission to user2 on the courses table
GRANT INSERT ON school_db.courses TO 'user2'@'localhost';
```

10. Transaction Control Language (TCL)

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

ANS - 1

1. COMMIT:

The COMMIT command is used to save all the changes made during the current transaction to the database.

Example:

```
BEGIN TRANSACTION;

UPDATE employees SET salary = 5000 WHERE employee_id = 123;

COMMIT;
```

2. ROLLBACK:

The ROLLBACK command is used to undo the changes made during the current transaction.

Example:

```
BEGIN TRANSACTION;

UPDATE employees SET salary = 5000 WHERE employee_id = 123;

ROLLBACK;
```

Q-2 Explain how transactions are managed in SQL databases.

ANS - In SQL databases, transactions are used to manage a series of operations that must be executed as a single unit of work. Transactions ensure the integrity, consistency, and reliability of the database, even in the face of errors, system crashes, or concurrent access by multiple users.

A transaction is a sequence of one or more SQL operations (such as INSERT, UPDATE, DELETE) executed as a single unit. A transaction is started, and its changes are either committed (saved) or rolled back (undone) based on whether the operations are successful or not.

- Starting a Transaction: In many databases, a transaction starts automatically when a data-modifying operation (like INSERT, UPDATE, or DELETE) is performed.

- Performing Operations: After starting a transaction, you can perform various SQL operations (e.g., INSERT, UPDATE, DELETE). These operations are part of the ongoing transaction and will not be saved until a COMMIT is issued.

- COMMIT: When all operations within a transaction are successful and you want to save the changes permanently, you issue the COMMIT command.

- ROLLBACK: If an error occurs or if you decide that you do not want to keep the changes made in a transaction, you can use ROLLBACK to undo all the changes made since the last commit.

11. SQL Joins

Q-1 Explain the concept of JOIN in SQL. What is the difference between INNER JOIN, LEFT

JOIN, RIGHT JOIN, and FULL OUTER JOIN?

ANS - In SQL, the JOIN operation is used to combine data from two or more tables based on a related column between them. This allows you to retrieve data from multiple tables in a single query.

1. INNER JOIN: The INNER JOIN returns rows from both tables where there is a match on the specified condition (usually, a common column).

It only returns the rows where the condition is true in both tables. If no match is found in either table, that row is excluded from the result.

2. LEFT JOIN (or LEFT OUTER JOIN): The LEFT JOIN returns all rows from the left table (the first table) and the matched rows from the right table (the second table). If no match is found, the result is NULL for columns from the right table.

3. RIGHT JOIN (or RIGHT OUTER JOIN): The RIGHT JOIN is the opposite of the LEFT JOIN. It returns all rows from the right table and the matched rows from the left table. If there's no match in the left table, the result is NULL for columns from the left table.

4. FULL OUTER JOIN: The FULL OUTER JOIN returns all rows when there is a match in either the left or the right table. If there's no match in one of the tables, the result will include NULL for the columns from that table.

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

ANS - Joins are used in relational databases to combine data from multiple tables based on related columns, typically keys. They allow you to retrieve data from more than one table in a single query, aligning records that are logically related. Joins are essential in relational database systems to work with normalized data.

- An inner join combines rows from two or more tables where there is a match based on a specified condition (often involving foreign keys).

- A left join (or left outer join) returns all rows from the left table and the matching rows from the right table.

- A right join (or right outer join) works like a left join, but it returns all rows from the right table and the matching rows from the left table.

- A full join (or full outer join) returns all rows when there is a match in either the left or right table.

- A cross join returns the Cartesian product of two tables. It combines each row from the first table with every row from the second table, which can result in a large number of rows.

- A self join is a join where a table is joined with itself.

- A natural join automatically joins tables based on all columns that have the same name and compatible data types.

LAB EXERCISES:

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

ANS -1. CREATE TABLE departments (

```
    department_id INT PRIMARY KEY,
    department_name VARCHAR(100)
);
```

2. CREATE TABLE employees (

```
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES departments(department_id)
);
```

3. INSERT INTO departments (department_id, department_name)

VALUES

(1, 'Human Resources'),

(2, 'Engineering'),

(3, 'Sales');

4. INSERT INTO employees (employee_id, first_name, last_name, department_id)

VALUES

(101, 'John', 'Doe', 1),

(102, 'Jane', 'Smith', 2),

(103, 'Alice', 'Johnson', 3),

(104, 'Bob', 'Brown', 2);

5. SELECT

```
    e.employee_id,
    e.first_name,
    e.last_name,
    d.department_name
FROM
    employees e
```

```
INNER JOIN
   departments d
ON
   e.department_id = d.department_id;
```

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

ANS -

```
SELECT
   d.department_id,
   d.department_name,
   e.employee_id
FROM
   departments d
LEFT JOIN
   employees e
ON
   d.department_id = e.department_id;
```

12. SQL Group By

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

ANS - The GROUP BY clause in SQL is used to group rows that have the same values in specified columns into summary rows. This is typically used when we want to perform aggregate operations (like counting, summing, averaging, etc.) on data grouped by certain criteria. It is often used in conjunction with aggregate functions like COUNT(), SUM(), AVG(), MIN(), and MAX().

- Aggregate functions are used to perform calculations on grouped data.

- COUNT(): Counts the number of rows in each group.

- SUM(): Sums up the values of a column in each group.

- AVG(): Computes the average value of a column in each group.

- MIN() and MAX(): Return the minimum or maximum value of a column in each group.

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

ANS - 1. GROUP BY: The GROUP BY clause is used to group rows that have the same values in specified columns into summary rows. It is often used in conjunction with aggregate functions like COUNT(), SUM(), AVG(), MIN(), and MAX() to perform calculations on each group of rows.

After grouping the rows based on one or more columns, you can apply aggregate functions to calculate summary values for each group.

Example Use Case: To calculate the total sales per product category.

```
SELECT category, SUM(sales)

FROM products

GROUP BY category;
```

2. ORDER BY: The ORDER BY clause is used to sort the result set by one or more columns in ascending or descending order.

It organizes the query results in a specific order (either ascending or descending) based on the values in one or more columns.

Example Use Case: To display the products ordered by price in descending order.

```
SELECT product_name, price

FROM products

ORDER BY price DESC;
```

LAB EXERCISES:

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

ANS - SELECT department, COUNT(*) AS employee_count

FROM employees

GROUP BY department;

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

ANS - SELECT department_id, AVG(salary) AS average_salary

FROM employees

GROUP BY department_id;

13. SQL Stored Procedure

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

ANS - A stored procedure in SQL is a precompiled set of one or more SQL statements that can be executed as a single unit. It is stored in the database and can be invoked by a client application or another SQL query. Stored procedures are used to perform operations such as data retrieval, data modification, and complex business logic without needing to repeatedly write the same SQL code.

- A standard SQL query is a single, standalone statement executed directly by a user or application to retrieve or manipulate data, such as SELECT, INSERT, UPDATE, or DELETE.

- A stored procedure, on the other hand, is a predefined collection of SQL statements that are stored and executed together, often with input parameters.

- A standard SQL query needs to be written and executed each time it's needed.

- A stored procedure can be executed multiple times without re-writing the SQL logic. Once defined, it can be invoked by its name, and it can also be parameterized for different inputs.

- A standard SQL query is compiled and optimized each time it's executed.

- A stored procedure is precompiled and stored in the database, which can lead to better performance when called multiple times.

- A standard SQL query typically performs a single operation.

- A stored procedure can include complex logic (e.g., loops, conditions, transactions), making it suitable for multi-step processes.

Q-2 Explain the advantages of using stored procedures.

ANS - Stored procedures are executed on the database server, reducing the need to send multiple individual queries over the network. This minimizes network latency, especially when multiple operations need to be performed in sequence.

- Stored procedures are compiled once and stored in the database. This reduces the overhead of parsing and optimizing SQL queries every time they are executed, leading to faster execution times.

- Stored procedures allow developers to define a block of logic that can be reused multiple times throughout an application. This avoids redundancy in SQL code and reduces maintenance effort.

- Stored procedures allow access to the underlying database without exposing the actual SQL code or tables directly to the application layer. This reduces the chances of SQL injection attacks.

- Specific permissions can be set on stored procedures, restricting who can execute them. This enables fine-grained control over database access.

- Business logic and complex queries are stored within the database, making it easier to maintain and update them. Modifying a stored procedure in the database doesn't require changes to the application code, making the system more flexible.

- Since stored procedures centralize the SQL code, they help in maintaining consistent behavior across different parts of an application.

- Stored procedures can encapsulate complex transactional operations, making it easier to handle commits, rollbacks, and error handling within a single unit.

- stored procedures are precompiled, the database server can optimize the execution plan for frequently used queries, potentially reducing the load and resource consumption compared to ad-hoc queries.

- Stored procedures abstract the underlying database schema and structure from the client application. This means that the client doesn't need to know about the specific tables or columns being queried.

- Since stored procedures are stored within the database, they are optimized for the specific database system.

LAB EXERCISES:

Lab 1: Write a stored procedure to retrieve all employees from the employees table based

on department.

ANS - DELIMITER $$

```
CREATE PROCEDURE GetEmployeesByDepartment(IN dept_name VARCHAR(100))
BEGIN
   SELECT employee_id, first_name, last_name, department
   FROM employees
   WHERE department = IT;
END $$
DELIMITER ;
```

Lab 2: Write a stored procedure that accepts course_id as input and returns the course

details.

ANS - DELIMITER $$

```
CREATE PROCEDURE GetCourseDetailsById(IN course_id INT)
BEGIN
   SELECT course_id, course_name, course_description, department
   FROM courses
   WHERE course_id = course_id;
END $$

DELIMITER ;
```

14. SQL View

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

ANS - In SQL, a view is essentially a virtual table that is based on the result of a query. It does not store data itself but presents data from one or more tables in a structured way. When you query a view, it dynamically fetches data from the underlying tables according to the query defined when the view was created.

- Differences Between a View and a Table:

- Table: A table is a physical structure that stores actual data in the database. It has its own storage space on the disk and maintains persistent data.

- View: A view does not store data. It is a stored SQL query that, when queried, pulls data from the underlying tables. The data is not physically stored in the view itself, but it is calculated each time the view is queried.

- Table: A table is designed to hold raw data that represents entities and relationships in a database.

- View: A view is typically used to present data in a specific format or to simplify complex queries.

- Table: Data in a table can be directly modified using INSERT, UPDATE, or DELETE statements.

- View: Generally, data cannot be modified directly through a view (though in some cases, views can be updatable under certain conditions).

- Table: Since tables store data physically, querying a table can be faster when compared to querying a view, especially if indexes are present.

- View: Querying a view can be slower, since the underlying query must be executed every time the view is referenced.

- Table: Used for data storage in a database.

- View: Used to simplify complex queries, provide a customized presentation of data, aggregate results, or restrict access to specific columns or rows.

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

ANS - Views allow users to encapsulate complex queries into a single object. This means that users do not have to write long and complex SQL statements repeatedly.

- By using views, users can perform operations on a "virtual table" that abstracts the complexity of the underlying data, such as joins, aggregations, or subqueries.

- Views can be used to restrict access to sensitive data. By providing users with access only to specific columns or rows via views, sensitive data (such as social security numbers or salary information) can be hidden from certain users.

- You can grant permissions on views without giving users direct access to the underlying base tables, enhancing data security.

- Views provide an abstraction layer between the user and the database schema. Users can interact with a view as though it were a regular table, even though the data is dynamically retrieved from one or more base tables.

- Views allow for central management of business logic, ensuring that the same logic is applied consistently whenever the view is accessed.

- Views can standardize frequently used queries. If a query is frequently used across multiple parts of the application, it can be encapsulated into a view and reused, ensuring consistent results across the system.

- Views can sometimes improve performance by simplifying complex queries. For example, pre-aggregating data in a view can save processing time for complex reporting queries.

- Views can help integrate data from multiple tables or even different databases by abstracting the complexity of how the data is related.

- When refactoring or migrating a database schema, views can act as a buffer between applications and the underlying schema. This means that if you need to restructure the tables or change relationships, the views can remain consistent, reducing the impact on the application layer.

- Views can help when accessing and combining data from multiple databases or systems. In cases where a database federates data from other systems, views can be used to create a unified interface to that data.

LAB EXERCISES:

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

ANS - CREATE VIEW employee_department_view AS

```
SELECT
   e.employee_id,
   e.first_name,
   e.last_name,
   d.department_name
FROM
   employees e
JOIN
   departments d
ON
   e.department_id = d.department_id;
```

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

```
CREATE OR REPLACE VIEW employee_department_view AS
SELECT
   e.employee_id,
   e.first_name,
   e.last_name,
   d.department_name
FROM
```

```
    employees e
JOIN
    departments d
ON
    e.department_id = d.department_id
WHERE
    e.salary >= 50000;
```

15. SQL Triggers

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

ANS - a trigger is a special kind of stored procedure that is automatically executed (or "triggered") when certain events occur in a database. Triggers are used to enforce data integrity, automate system tasks, or capture events that happen within the database. They are associated with a particular table or view and are activated by DML (Data Manipulation Language) statements like INSERT, UPDATE, or DELETE.

- Types of Triggers:

1. BEFORE Trigger: A BEFORE trigger is executed before the actual INSERT, UPDATE, or DELETE operation takes place.

- Validation of data before an insert or update.

- Modifying data before it is inserted (e.g., automatically setting a value for a field).

2. AFTER Trigger: An AFTER trigger is executed after the data modification (INSERT, UPDATE, or DELETE) has occurred. It is useful for logging changes, updating other tables, or performing complex operations after the changes are made.

- Logging changes to a table.

- Updating related tables after a change.

3. INSTEAD OF Trigger:An INSTEAD OF trigger is typically used on views rather than tables. It overrides the default action of an INSERT, UPDATE, or DELETE on the view and instead executes the trigger's action.

- Performing complex INSERT, UPDATE, or DELETE operations on views when direct operations are not allowed.

- Redirecting the operations to underlying tables.

4. Compound Trigger (Specific to Oracle):A compound trigger allows you to combine multiple triggers that might fire in response to the same event into a single trigger. It reduces the risk of trigger conflicts and ensures a clean flow of control in complex cases.

- Handling multiple actions that would otherwise be handled by multiple separate triggers.

* Triggers Are Used:

- Ensure that data adheres to business rules before it is committed to the database.

- Capture changes to critical data for auditing purposes, such as logging every insert, update, or delete operation performed on important tables.

- Ensure that actions taken on parent records propagate appropriately to related child records. For example, automatically updating or deleting related records in child tables when the parent record is updated or deleted.

- Perform complex operations that involve multiple tables, fields, or business logic after a data manipulation operation.

- Prevent certain types of operations on the database, such as disallowing updates to certain critical fields or preventing deletes on records that are still in use.

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

ANS - 1. INSERT Trigger : This type of trigger is fired before or after a new row of data is inserted into a table.

- An INSERT trigger is triggered when a new record is added to the table via an INSERT statement.

- Automatically updating audit tables when a new record is inserted.

- Validating or modifying the new data before it's actually inserted into the table.

- Enforcing business rules or logging information.

2. UPDATE Trigger: This type of trigger is fired before or after an existing row is modified via an UPDATE statement.

- An UPDATE trigger is triggered when one or more columns in an existing record are modified in the table.

- Tracking changes to records, like logging when a user updates information.

- Enforcing integrity or business rules, such as automatically adjusting related records in other tables when one table's data changes.

3. DELETE Trigger: This type of trigger is fired before or after a row is deleted from a table.

- A DELETE trigger is triggered when a DELETE statement is executed, removing one or more rows from a table.

- Preventing deletions under certain conditions (for example, if foreign key constraints are violated).

- Performing cleanup or maintenance tasks when a record is deleted

- INSERT Trigger: Executes when a new row is inserted.

- UPDATE Trigger: Executes when an existing row is modified.

- DELETE Trigger: Executes when a row is removed.

LAB EXERCISES:

Lab 1: Create a trigger to automatically log changes to the employees table when a new

employee is added.

ANS -
```
CREATE TABLE employee_log (
   log_id INT PRIMARY KEY AUTO_INCREMENT,
   employee_id INT,
   action VARCHAR(50),
   action_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
DELIMITER //

CREATE TRIGGER log_employee_insert
```

```
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
   INSERT INTO employee_log (employee_id, action)
   VALUES (NEW.employee_id, 'INSERT');
END//

DELIMITER ;
```

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

ANS - 1. ALTER TABLE employees
ADD COLUMN last_modified TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP;

2. DELIMITER //

```
CREATE TRIGGER update_last_modified
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
   -- Update the last_modified timestamp to the current time
   SET NEW.last_modified = CURRENT_TIMESTAMP;
END//

DELIMITER ;
```