----------------------------Assignment-4 (Interview Questions) ---

(1.) Database Concepts:-

---

1. What is a database?

A **database** is an organized collection of data that can be easily accessed, managed, and updated. It stores information in a structured way to support data retrieval, insertion, deletion, and modification.
Example: A library management system stores book details in a database.

---

2. What is the difference between a database and a DBMS?

- **Database:** A collection of data stored systematically.
- **DBMS (Database Management System):** A software that manages and interacts with the database to perform operations like querying, updating, and managing users.
  Example: MySQL, Oracle, and PostgreSQL are DBMSs that manage databases.

---

3. What are the different types of databases?

- **Relational Database (RDBMS):** Uses tables (MySQL, PostgreSQL).
- **NoSQL Database:** Stores unstructured data (MongoDB, Cassandra).
- **Hierarchical Database:** Tree-like structure (IBM IMS).
- **Network Database:** Graph-based structure (IDMS).
- **Object-Oriented Database:** Stores objects (db4o, ObjectDB).

---

4. What is a relational database?

A **relational database** organizes data into tables (relations) with rows and columns. It follows ACID properties for consistency.
Example: A student database with a table storing student IDs, names, and marks.

---

5. What is normalization? Explain its types.

Normalization is the process of organizing data to remove redundancy and improve integrity.

- **1NF (First Normal Form):** No duplicate columns; each column has atomic values.
- **2NF (Second Normal Form):** No partial dependency (depends on the entire primary key).

- **3NF (Third Normal Form):** No transitive dependency (non-key column should not depend on another non-key column).
  Example: Splitting a table into Students (ID, Name) and Courses (CourseID, StudentID) removes redundancy.

---

6. What is denormalization?

Denormalization is the process of combining tables to improve read performance by reducing joins.
Example: Instead of separate Orders and Customers tables, a denormalized table may include customer details within Orders.

---

7. What is a primary key? How is it different from a unique key?

- **Primary Key:** Uniquely identifies each row and cannot be NULL.
- **Unique Key:** Ensures unique values but allows NULL.
  Example:
  - Primary Key: StudentID (must be unique & not null).
  - Unique Key: Email (must be unique but can be null).

---

8. What is a foreign key?

A **foreign key** is a column that links to the primary key of another table to maintain relationships.
Example: In a Orders table, CustomerID is a foreign key referencing Customers(ID).

---

9. What are indexes? Why are they used?

Indexes are data structures that speed up database searches by allowing quick lookups.
Example: Creating an index on Employee(Name) allows fast searching in an Employee table.

---

10. What is a composite key?

A **composite key** is a primary key made of two or more columns.
Example: A StudentCourse table with a composite key (StudentID, CourseID) ensures unique enrollments.

(2.) MySQL Commands:-

---

11. What is the purpose of the CREATE command?

Definition: The CREATE command is used to create a new database or table in SQL.

Types:

- CREATE DATABASE – Creates a new database.
- CREATE TABLE – Creates a new table.
- CREATE INDEX – Creates an index on a table.

Example:

CREATE DATABASE myDatabase;

CREATE TABLE students (id INT, name VARCHAR(50));

---

12. How do you delete a database in MySQL?

Definition: The DROP DATABASE command is used to delete an existing database permanently.

Example:

DROP DATABASE myDatabase;

---

13. What is the ALTER command used for?

Definition: The ALTER command is used to modify an existing table by adding, deleting, or modifying columns.

Types:

- ADD COLUMN – Adds a new column.
- DROP COLUMN – Deletes a column.
- MODIFY COLUMN – Changes the datatype of a column.

Example:

ALTER TABLE students ADD COLUMN age INT;

ALTER TABLE students DROP COLUMN age;

---

14. How do you create a table in MySQL?

Definition: The CREATE TABLE command is used to define a new table in a database.

Example:

```
CREATE TABLE employees (
  id INT PRIMARY KEY,
  name VARCHAR(50),
    salary DECIMAL(10,2)
);
```

---

15. What is the DROP command?

Definition: The DROP command is used to delete a database or table permanently.

Example:

```
DROP TABLE employees;
```

---

16. How do you insert data into a table?

Definition: The INSERT command is used to add new records into a table.

Example:

```
INSERT INTO students (id, name) VALUES (1, 'Shivam');
```

---

17. What is the syntax for updating records in a table?

Definition: The UPDATE command modifies existing records in a table.

Example:

```
UPDATE students SET name = 'Amit' WHERE id = 1;
```

---

18. How do you delete records from a table?

Definition: The DELETE command removes specific records from a table.

Example:

```
DELETE FROM students WHERE id = 1;
```

---

19. What is the SELECT statement used for?

Definition: The SELECT statement retrieves data from one or more tables.

Example:

```
SELECT * FROM students;
```

---

20. How do you retrieve unique records from a table?

Definition: The DISTINCT keyword is used to fetch unique values from a column.

Example:

SELECT DISTINCT name FROM students;

(3.) Clauses and Operators:-

---

21. What is the purpose of the WHERE clause?

Definition: The WHERE clause is used to filter records based on a specific condition.

Example:

SELECT * FROM employees WHERE age > 30;

---

22. Explain the ORDER BY clause.

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

Example:

SELECT * FROM employees ORDER BY salary DESC;

---

23. What is the GROUP BY clause used for?

Definition: The GROUP BY clause is used to group rows with the same values in specified columns and aggregate data.

Example:

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

---

24. How do you use the HAVING clause?

Definition: The HAVING clause is used to filter groups after applying the GROUP BY clause.

Example:

SELECT department, COUNT(*) FROM employees GROUP BY department HAVING COUNT(*) > 5;

---

25. What are the different comparison operators in MySQL?

Definition: Comparison operators are used to compare values in SQL queries.

Types:

- = (Equal)
- != or <> (Not Equal)
- > (Greater Than)

- < (Less Than)
- >= (Greater Than or Equal)
- <= (Less Than or Equal)

Example:

SELECT * FROM employees WHERE salary >= 50000;

---

26. What is the BETWEEN operator?

Definition: The BETWEEN operator is used to filter values within a given range.

Example:

SELECT * FROM employees WHERE salary BETWEEN 40000 AND 60000;

---

27. Explain the LIKE operator.

Definition: The LIKE operator is used to search for a specified pattern in a column.

Types:

- % (Matches any number of characters)
- _ (Matches a single character)

Example:

SELECT * FROM employees WHERE name LIKE 'A%';

(This selects names starting with 'A')

---

28. What is the IN operator?

Definition: The IN operator is used to filter results based on multiple specified values.

Example:

SELECT * FROM employees WHERE department IN ('HR', 'IT', 'Finance');

---

29. How do you use the NULL operator?

Definition: The IS NULL and IS NOT NULL operators are used to check for NULL values in a column.

Example:

SELECT * FROM employees WHERE email IS NULL;

---

30. What is the difference between AND and OR operators?

Definition:

- AND: Returns records where all conditions are true.
- OR: Returns records where at least one condition is true.

Example:

SELECT * FROM employees WHERE age > 30 AND salary > 50000;

SELECT * FROM employees WHERE age > 30 OR salary > 50000;

(4.) Predefined Functions:-

---

31. What are aggregate functions?

Aggregate functions perform calculations on a set of values and return a single value.

Examples: COUNT(), SUM(), AVG(), MAX(), MIN()

SELECT AVG(salary) FROM employees;

---

32. What is the COUNT() function?

The COUNT() function returns the number of rows that match a specified condition.

Example:

SELECT COUNT(*) FROM employees WHERE department = 'IT';

---

33. Explain the SUM() function.

The SUM() function returns the total sum of a numeric column.

Example:

SELECT SUM(salary) FROM employees WHERE department = 'Finance';

---

34. What is the AVG() function?

The AVG() function calculates the average value of a numeric column.

Example:

SELECT AVG(salary) FROM employees;

---

35. How does the MAX() function work?

The MAX() function returns the highest value in a column.

Example:

```
SELECT MAX(salary) FROM employees;
```

---

36. What is the MIN() function?

The MIN() function returns the lowest value in a column.

Example:

```
SELECT MIN(salary) FROM employees;
```

---

37. Explain string functions in MySQL.

String functions manipulate text data in MySQL.

Examples:

- CONCAT() – Joins strings
- SUBSTRING() – Extracts a part of a string
- LENGTH() – Returns string length
- UPPER()/LOWER() – Converts case

```
SELECT UPPER(name) FROM employees;
```

---

38. What is the CONCAT() function?

The CONCAT() function joins two or more strings together.

Example:

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name FROM employees;
```

---

39. How do you use the SUBSTRING() function?

The SUBSTRING() function extracts a part of a string.

Example:

```
SELECT SUBSTRING('Hello World', 1, 5); -- Output: Hello
```

---

40. What is the NOW() function?

The NOW() function returns the current date and time.

Example:

```
SELECT NOW();
```

(5.) User -Defined Functions:-

---

41. What is a user-defined function (UDF) in MySQL?

A **User-Defined Function (UDF)** in MySQL is a custom function created by users to perform specific tasks that are not available in built-in functions. It extends MySQL's functionality.

---

42. How do you create a UDF?

UDFs in MySQL are created using shared libraries (C/C++) and registered using SQL commands.
Steps to create a UDF:

1. Write the function in C/C++.
2. Compile it into a shared library (.so file).
3. Install it in MySQL using CREATE FUNCTION.

Example:

```
CREATE FUNCTION square RETURNS INTEGER SONAME 'square.so';
```

---

43. What is the syntax for calling a UDF?

After a UDF is registered, it can be called like any other MySQL function.

Syntax:

```
SELECT function_name(arguments);
```

Example:

```
SELECT square(5); -- Returns 25
```

---

44. Can UDFs return multiple values?

No, UDFs in MySQL can only return a single scalar value (string, integer, or floating point). To return multiple values, use **stored procedures** instead.

Example of stored procedure returning multiple values:

```
DELIMITER //

CREATE PROCEDURE GetUser(IN userId INT, OUT userName VARCHAR(50), OUT userAge INT)

BEGIN

  SELECT name, age INTO userName, userAge FROM users WHERE id = userId;

END //
```

DELIMITER ;

---

45. What are the advantages of using UDFs?

- **Performance:** Faster execution than stored procedures for simple calculations.
- **Reusability:** Can be used across multiple queries.
- **Extensibility:** Extends MySQL with custom logic.

Example Use Case:
A UDF like square can be used for mathematical operations directly in queries.

SELECT square(4); -- Output: 16

(6.) Views:-

---

46. What is a view in MySQL?

A **view** in MySQL is a virtual table based on the result of a SQL query. It does not store data itself but dynamically fetches data from underlying tables.

Types of Views:

1. **Simple View** – Based on a single table.
2. **Complex View** – Based on multiple tables using joins.

Example:

CREATE VIEW employee_view AS

SELECT id, name, salary FROM employees;

---

47. How do you create a view?

You create a view using the CREATE VIEW statement with a SELECT query.

Syntax:

CREATE VIEW view_name AS

SELECT column1, column2 FROM table_name WHERE condition;

Example:

CREATE VIEW high_salary AS

SELECT name, salary FROM employees WHERE salary > 50000;

---

48. What is the difference between a view and a table?

Feature	Table	View
Storage	Stores data permanently	Does not store data, just retrieves it dynamically
Modification	Can insert, update, and delete	Can be updatable (with conditions)
Performance	Faster as data is stored	Slower as it fetches data dynamically

---

49. Can you update a view? If yes, how?

Yes, a view can be updated if:

- It is based on a single table.
- It does not use aggregate functions, joins, or DISTINCT.

Example (Updating a View):

UPDATE high_salary SET salary = 60000 WHERE name = 'John';

Example (Creating an Updatable View):

CREATE VIEW editable_view AS

SELECT id, name, salary FROM employees WHERE department = 'IT';

---

50. How do you drop a view?

A view can be removed using the DROP VIEW statement.

Syntax:

DROP VIEW view_name;

Example:

DROP VIEW high_salary;

(7.) Common Table Expressions (CTE):-

---

51. What is a Common Table Expression (CTE)?

A **Common Table Expression (CTE)** is a temporary named result set that exists within the execution of a single query. It improves readability and simplifies complex queries.

Types of CTE:

1. **Non-Recursive CTE** – Standard CTE without self-referencing.

2. **Recursive CTE** – A CTE that references itself to perform iterative operations.

---

52. How do you create a CTE?

A **CTE is created using the WITH clause** followed by a temporary result set.

Example:

```
WITH EmployeeCTE AS (
    SELECT EmployeeID, Name, Salary
    FROM Employees
    WHERE Salary > 50000
)
SELECT * FROM EmployeeCTE;
```

This selects employees with salaries above 50,000.

---

53. What is the difference between a CTE and a subquery?

Feature	**CTE**	**Subquery**
Readability	More readable and structured	Can become complex and harder to read
Reusability	Can be used multiple times in a query	Cannot be reused directly
Performance	Optimized in some cases	Might be less optimized

Example of a Subquery:

```
SELECT EmployeeID, Name FROM Employees
WHERE Salary > (SELECT AVG(Salary) FROM Employees);
```

This filters employees with a salary above the average.

---

54. Can you use a CTE recursively?

Yes, **CTEs can be recursive**, meaning they can refer to themselves to generate hierarchical or iterative results.

Example of Recursive CTE:

```
WITH RecursiveCTE (n) AS (
    SELECT 1  -- Base case
    UNION ALL
```

```
   SELECT n + 1 FROM RecursiveCTE WHERE n < 5  -- Recursive case
)
SELECT * FROM RecursiveCTE;
```

This generates numbers from 1 to 5.

---

55. How do you reference a CTE in a query?

A **CTE is referenced just like a table** by using its name in a SELECT, INSERT, UPDATE, or DELETE statement.

Example:

```
WITH HighSalary AS (
   SELECT EmployeeID, Name, Salary FROM Employees WHERE Salary > 70000
)
SELECT Name FROM HighSalary;
```

Here, HighSalary is used like a table in the final SELECT statement.

(8.) Joins:-

---

56. What is a JOIN in SQL?

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

---

57. Different Types of JOINs in SQL

SQL supports the following types of joins:

- **INNER JOIN**
- **LEFT JOIN (LEFT OUTER JOIN)**
- **RIGHT JOIN (RIGHT OUTER JOIN)**
- **FULL JOIN (FULL OUTER JOIN)**
- **CROSS JOIN**
- **SELF JOIN**

---

58. What is an INNER JOIN?

An **INNER JOIN** returns only the matching rows between both tables based on a common column.

Example:

```
SELECT employees.name, departments.department_name

FROM employees

INNER JOIN departments ON employees.department_id = departments.id;
```

---

59. What is a LEFT JOIN?

A **LEFT JOIN (LEFT OUTER JOIN)** returns all records from the left table and the matching records from the right table. If no match is found, NULL is returned for the right table's columns.

Example:

```
SELECT employees.name, departments.department_name

FROM employees

LEFT JOIN departments ON employees.department_id = departments.id;
```

---

60. What is a RIGHT JOIN?

A **RIGHT JOIN (RIGHT OUTER JOIN)** returns all records from the right table and the matching records from the left table. If no match is found, NULL is returned for the left table's columns.

Example:

```
SELECT employees.name, departments.department_name

FROM employees

RIGHT JOIN departments ON employees.department_id = departments.id;
```

---

61. What is a FULL OUTER JOIN?

A **FULL OUTER JOIN** returns all records when there is a match in either the left or right table. If no match is found, NULL is returned in non-matching columns.

Example:

```
SELECT employees.name, departments.department_name

FROM employees

FULL OUTER JOIN departments ON employees.department_id = departments.id;
```

---

62. How to perform a CROSS JOIN?

A **CROSS JOIN** returns the Cartesian product of both tables, meaning every row from the first table is paired with every row from the second table.

Example:

```
SELECT employees.name, departments.department_name
FROM employees
CROSS JOIN departments;
```

---

63. What is a SELF JOIN?

A **SELF JOIN** joins a table with itself, treating it as two separate tables.

Example:

```
SELECT e1.name AS Employee, e2.name AS Manager
FROM employees e1
JOIN employees e2 ON e1.manager_id = e2.id;
```

---

64. How to join multiple tables?

You can join multiple tables using multiple JOIN statements.

Example:

```
SELECT employees.name, departments.department_name, locations.city
FROM employees
JOIN departments ON employees.department_id = departments.id
JOIN locations ON departments.location_id = locations.id;
```

---

65. Difference between JOIN and SUBQUERY?

JOIN	**SUBQUERY**
Combines data from multiple tables	Uses a query inside another query
Typically performs better on large datasets	Can be slower due to nested execution
Requires a relationship between tables	Can be used even without direct relationships

Example of a subquery:

```
SELECT name FROM employees
WHERE department_id = (SELECT id FROM departments WHERE department_name = 'IT');
```

(9.) Subqueries:-

---

66. What is a Subquery?

A **subquery** is a query nested inside another SQL query. It is used to retrieve data that will be used in the main query.

Types of Subqueries:

- **Single-row Subquery** (returns one value)
- **Multi-row Subquery** (returns multiple values)
- **Correlated Subquery** (dependent on the outer query)

Example:

SELECT name FROM students WHERE id = (SELECT MAX(id) FROM students);

---

67. How do you write a subquery in the SELECT statement?

A subquery can be written inside the SELECT clause to retrieve computed values.

Example:

SELECT name, (SELECT AVG(salary) FROM employees) AS avg_salary FROM employees;

Here, the subquery calculates the average salary and displays it with each row.

---

68. Can you use a subquery in the WHERE clause?

Yes, a subquery can be used inside the WHERE clause to filter data based on another query's result.

Example:

SELECT name FROM employees WHERE salary > (SELECT AVG(salary) FROM employees);

This retrieves employees earning more than the average salary.

---

69. What is a Correlated Subquery?

A **correlated subquery** is a subquery that depends on the outer query for its execution. It runs once per row processed in the outer query.

Example:

SELECT e1.name, e1.salary

FROM employees e1

```
WHERE salary > (SELECT AVG(e2.salary) FROM employees e2 WHERE e1.department =
e2.department);
```

Here, the subquery calculates the department-wise average salary dynamically for each employee.

---

70. How do you handle subqueries that return multiple rows?

If a subquery returns multiple rows, you must use operators like IN, ANY, or ALL instead of =.

Example:

```
SELECT name FROM employees WHERE department_id IN (SELECT department_id FROM
departments WHERE location = 'New York');
```

This retrieves employees working in New York departments.

(10.) Stored Procedures:-

---

71. What is a Stored Procedure?

A **stored procedure** is a precompiled set of SQL statements stored in a database that can be executed repeatedly to perform a specific task.

---

72. How do you create a stored procedure in MySQL?

A stored procedure is created using the CREATE PROCEDURE statement.

Example:

```
DELIMITER //

CREATE PROCEDURE GetAllUsers()

BEGIN

   SELECT * FROM users;

END //

DELIMITER ;
```

---

73. What is the syntax for calling a stored procedure?

Stored procedures are executed using the CALL statement.

Example:

```
CALL GetAllUsers();
```

---

74. Can stored procedures accept parameters?

Yes, stored procedures can accept **IN, OUT, and INOUT** parameters.

Example:

```
DELIMITER //

CREATE PROCEDURE GetUserByID(IN userID INT)

BEGIN

   SELECT * FROM users WHERE id = userID;

END //

DELIMITER ;
```

Calling the procedure:

```
CALL GetUserByID(1);
```

---

75. What are the advantages of using stored procedures?

- **Improved Performance**: Reduces query compilation time.
- **Code Reusability**: Eliminates repetitive SQL code.
- **Security**: Limits direct access to the database.
- **Faster Execution**: Optimized execution plan.

Example (Advantage - Security)

Instead of allowing direct access to the users table, a stored procedure can be used:

```
DELIMITER //

CREATE PROCEDURE SecureGetUsers()

BEGIN

   SELECT id, name FROM users; -- Hides sensitive columns like passwords

END //

DELIMITER ;
```

Calling the procedure:

```
CALL SecureGetUsers();
```

Muja Itna He Pata Ha Boss

<u>(11.) Triggers:-</u>

---

76. What is a trigger in MySQL?

A **trigger** in MySQL is a stored procedure that is automatically executed or fired when certain events (such as INSERT, UPDATE, or DELETE) occur on a specified table or view.

Example:

```
CREATE TRIGGER before_insert_trigger

BEFORE INSERT ON employees

FOR EACH ROW

SET NEW.created_at = NOW();
```

---

77. How do you create a trigger?

A **trigger** can be created using the CREATE TRIGGER statement, specifying the event (INSERT, UPDATE, DELETE), the timing (BEFORE, AFTER), and the associated table.

Example:

```
CREATE TRIGGER trigger_name

AFTER INSERT ON table_name

FOR EACH ROW

BEGIN

   -- Trigger logic here

END;
```

---

78. What are the different types of triggers?

There are **3 types of triggers** based on timing and event:

1. **BEFORE Trigger**: Executes before the event.
   - **Example**: BEFORE INSERT, BEFORE UPDATE
2. **AFTER Trigger**: Executes after the event.
   - **Example**: AFTER INSERT, AFTER UPDATE
3. **INSTEAD OF Trigger**: Executes in place of the event (mainly for views).
   - **Example**: INSTEAD OF INSERT (for views)

---

79. Can a trigger call a stored procedure?

Yes, a **trigger** can call a **stored procedure**. This is useful for executing complex logic when the trigger is fired.

Example:

CREATE TRIGGER trigger_name

AFTER INSERT ON employees

FOR EACH ROW

BEGIN

  CALL my_procedure(NEW.employee_id);

END;

---

80. What is the difference between a trigger and a stored procedure?

- A **trigger** is automatically executed in response to certain events on a table (e.g., INSERT, UPDATE, DELETE).
- A **stored procedure** is a set of SQL statements that can be executed explicitly using a CALL statement.

Difference:

- **Trigger**: Runs automatically in response to data changes (cannot be manually invoked).
- **Stored Procedure**: Needs to be manually invoked via CALL.

Example:

- **Trigger**:
- CREATE TRIGGER before_update_employee
- BEFORE UPDATE ON employees
- FOR EACH ROW
- BEGIN
-   -- Trigger action
- END;
- **Stored Procedure**:
- DELIMITER //
- CREATE PROCEDURE my_procedure()
- BEGIN
-   -- Procedure logic
- END;

- //
- DELIMITER ;

.

(12.) Data Control Language (DCL):-

---

81. What is Data Control Language (DCL)?

Definition: DCL is a set of SQL commands used to control access to data stored in a database. It allows administrators to define and manage user permissions. **Types**:

- GRANT: Provides user permissions.
- REVOKE: Removes user permissions.

Example:

GRANT SELECT, INSERT ON database_name.* TO 'user'@'host';

REVOKE SELECT ON database_name.* FROM 'user'@'host';

---

82. What is the purpose of the GRANT command?

Definition: The GRANT command is used to give specific privileges to a user on database objects (tables, views, etc.). **Example**:

GRANT SELECT, INSERT ON employees TO 'john'@'localhost';

This grants the SELECT and INSERT privileges on the employees table to the user john.

---

83. How do you revoke privileges using the REVOKE command?

Definition: The REVOKE command is used to remove privileges that were previously granted to a user. **Example**:

REVOKE SELECT, INSERT ON employees FROM 'john'@'localhost';

This removes the SELECT and INSERT privileges from the user john on the employees table.

---

84. What is the difference between a user and a role in MySQL?

Definition:

- **User**: A specific individual or entity that can connect to a MySQL server and perform actions based on the privileges granted.

- **Role**: A set of privileges that can be assigned to users. Roles help manage multiple users with similar permissions.

Example:

CREATE USER 'john'@'localhost';

CREATE ROLE 'admin_role';

GRANT ALL PRIVILEGES ON *.* TO 'admin_role';

GRANT 'admin_role' TO 'john'@'localhost';

---

85. How do you create a new user in MySQL?

Definition: You can create a new user using the CREATE USER command and assign privileges using GRANT. **Example**:

CREATE USER 'john'@'localhost' IDENTIFIED BY 'password';

GRANT SELECT ON database_name.* TO 'john'@'localhost';

This creates a new user john with the password 'password' and grants SELECT permission on database_name.

<u>(13.) Transaction Control Language (TCL):-</u>

---

86. What is Transaction Control Language (TCL)?

Definition: TCL is a set of SQL commands used to manage changes made by DML (Data Manipulation Language) commands. It ensures that the database remains consistent even in the case of system failures.

Types:

1. **COMMIT**
2. **ROLLBACK**
3. **SAVEPOINT**
4. **SET TRANSACTION**

Example:

-- COMMIT saves all changes made in the current transaction.

COMMIT;

---

87. What is the purpose of the COMMIT command?

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

Example:

-- After inserting a record, the changes are saved permanently.

INSERT INTO students (id, name) VALUES (1, 'John');

COMMIT;

---

88. How do you use the ROLLBACK command?

Definition: The ROLLBACK command undoes all changes made in the current transaction, reverting the database to its state before the transaction began.

Example:

-- Rollback undoes the insert operation.

INSERT INTO students (id, name) VALUES (1, 'John');

ROLLBACK;

---

89. What is the SAVEPOINT command?

Definition: The SAVEPOINT command sets a point within a transaction to which you can later roll back, without affecting the entire transaction.

Example:

-- Set a savepoint in a transaction.

SAVEPOINT my_savepoint;

-- Rollback to the savepoint.

ROLLBACK TO my_savepoint;

---

90. How do you set the transaction isolation level?

Definition: The transaction isolation level determines the visibility of the changes made by one transaction to other transactions. It helps control concurrency.

Types:

1. **READ UNCOMMITTED**
2. **READ COMMITTED**
3. **REPEATABLE READ**

4. **SERIALIZABLE**

Example:

-- Set the isolation level to SERIALIZABLE.

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

(14.) Types of Databases:-

---

91. What are the different types of databases?
Databases are classified into different types based on their data structure and usage.

- **Relational Databases (RDBMS):** Stores data in tables with rows and columns (e.g., MySQL, PostgreSQL).
- **NoSQL Databases:** Used for unstructured data, scalable horizontally (e.g., MongoDB, Cassandra).
- **Graph Databases:** Stores data as nodes and edges, representing relationships (e.g., Neo4j).
- **Object-Oriented Databases:** Stores objects as data (e.g., db4o).
- **Distributed Databases:** Data is spread across multiple locations or servers (e.g., Apache Cassandra).
- **Cloud Databases:** Hosted on cloud platforms (e.g., Amazon RDS, Google Firestore).

---

92. What is the difference between SQL and NoSQL databases?

- **SQL (Structured Query Language)**: Used in relational databases, stores data in tables, and requires a fixed schema (e.g., MySQL, PostgreSQL).
- **NoSQL (Not Only SQL)**: Used in non-relational databases, stores unstructured data, and can have flexible schemas (e.g., MongoDB, Cassandra).

Key Difference: SQL databases are better for structured data and complex queries, while NoSQL is more flexible and better for large volumes of unstructured or semi-structured data.

---

93. What are some examples of NoSQL databases?

- **MongoDB**: Document-based database, stores data in JSON-like format.
- **Cassandra**: Column-store database, highly scalable for distributed systems.
- **Redis**: In-memory key-value store, suitable for caching.
- **CouchDB**: Document store that uses JSON to store data.
- **Neo4j**: Graph database for representing and querying relationships between data.

94. What is a distributed database?
A distributed database is a database that is spread across multiple physical locations, either within the same data center or across multiple data centers. This setup ensures better scalability, availability, and fault tolerance.

- **Example:** Apache Cassandra, MongoDB in a sharded setup.

---

95. What is a cloud database?
A cloud database is a database that runs on a cloud computing platform. These databases are hosted and managed by cloud service providers, offering scalability, high availability, and remote access.

- **Example:** Amazon RDS, Google Cloud SQL, Azure SQL Database.

(15.) Database Management Systems (DBMS):-

---

96. What is a Database Management System (DBMS)?

- A DBMS is software that allows users to store, manage, and manipulate data in a structured format.
- **Example:** MySQL is a DBMS used to store and manage data for web applications.

---

97. What are the functions of a DBMS?

- Functions of DBMS include data storage, data retrieval, data manipulation, security management, and backup.
- **Example:** In a library system, DBMS stores book information, retrieves details, and helps with issuing and returning books.

---

98. What is the difference between a DBMS and a RDBMS?

- DBMS stores data in a single file or format, whereas RDBMS stores data in tables with relationships between them (i.e., relational data).
- **Example:** DBMS is like a flat file system, while RDBMS (like MySQL) uses tables to store data with primary and foreign keys.

99. What are some popular DBMS software?

- Some popular DBMS software includes:
  - **DBMS:** MS Access, SQLite
  - **RDBMS:** MySQL, PostgreSQL, Oracle, SQL Server
- **Example:** MySQL is a widely used RDBMS for building web applications.

100. What is data integrity, and how does a DBMS ensure it?

- Data integrity refers to the accuracy and consistency of data. A DBMS ensures it by enforcing rules like constraints, validation, and ACID properties (Atomicity, Consistency, Isolation, Durability).
- **Example:** In a school database, data integrity ensures that student grades cannot exceed a maximum value or that each student ID is unique.