MODULE – 5
INTRODUCTION TO DBMS

1. What do you understand By Database ?

➔A **database** is an organized collection of data that is stored and managed electronically. It is designed to efficiently store, retrieve, and manage large amounts of information in a structured way. Databases can be used to store data for various applications, such as websites, software systems, and enterprise-level platforms.

■ Here are some key points about databases:

➔1. **Data Organization**: Data in a database is typically structured in tables, which consist of rows (records) and columns (fields). Each column holds a specific type of data, like text, numbers, or dates.

➔2. **Database Management System (DBMS)**: A DBMS is software that facilitates the creation, management, and manipulation of a database. Popular DBMS examples include MySQL, PostgreSQL, Oracle, and Microsoft SQL Server.

➔3. **Types of Databases**:
- **Relational Databases (RDBMS)**: These store data in tables that can be related to each other using keys (e.g., primary and foreign keys). Examples include MySQL and PostgreSQL.
- **NoSQL Databases**: These store data in formats other than tables, such as key-value pairs, documents, or graphs. Examples include MongoDB, Redis, and Cassandra.

➔4. **Operations**: Databases allow for various operations, such as:

MODULE – 5

INTRODUCTION TO DBMS

- **CRUD**: Create, Read, Update, and Delete operations.
- **Querying**: Extracting data using languages like SQL (Structured Query Language).
- **Transactions**: Ensuring that database operations are executed reliably (with features like ACID properties – Atomicity, Consistency, Isolation, Durability).

➔5. **Purpose**: Databases are used in almost all fields like banking systems, e-commerce websites, social media platforms, and more, to handle large volumes of data while ensuring that it is safe, easily accessible, and efficiently managed.

In short, databases are a fundamental tool for managing data in a way that makes it easy to store, retrieve, update, and maintain information.

2. What is Normalization?

→**Normalization** is the process of organizing data in a database to reduce redundancy and improve efficiency.

→It involves breaking down large tables into smaller, related ones and ensuring data dependencies are properly structured.

→The goal is to make the database more efficient and prevent issues like data duplication and inconsistencies. This is done through applying **normal forms** (1NF, 2NF, 3NF, etc.), which establish rules to eliminate unnecessary data repetition and ensure integrity.

3. What is Difference between DBMS and RDBMS?

MODULE – 5

INTRODUCTION TO DBMS

→

Feature	DBMS	RDBMS
Data Storage	Can store data in any model	Stores data in tables (relational)
Relationships	No inherent support for relationships	Supports relationships (via primary and foreign keys)
Normalization	Not mandatory	Encourages normalization
ACID Compliance	Not guaranteed	Fully supports ACID properties
Data Integrity	Minimal support for integrity	Strong support for data integrity
Example	File systems, XML databases	MySQL, PostgreSQL, Oracle, SQL Server
Scalability	Limited scalability	Better scalability and performance
Operations	Basic data manipulation	Advanced querying (joins, subqueries, transactions)
Redundancy	Allows data redundancy	Minimizes data redundancy

4.What is MF Cod Rule of RDBMS Systems?

→The **MF Cod Rule** in the context of **RDBMS (Relational Database Management Systems)** is not a widely recognized or standard term in database management literature. However, it's possible you're referring to a specific concept or rule within a certain context, or there might be a misunderstanding or typo in the term.

1. **Data stored in tables**: All information should be in tables (also called relations).

MODULE – 5

INTRODUCTION TO DBMS

2. **Data accessed using keys**: Data should be accessed through a combination of table names, primary keys, and attributes.

3. **Null values handled**: The system should handle missing or unknown data (null values) consistently.

4. **Catalog is relational**: The database's structure (metadata) should also follow the relational model.

5. **Support for relational language**: The database should support a language like SQL to perform all database operations.

6. **Views are updatable**: Virtual tables (views) should be able to be updated if possible.

7. **High-level operations**: Inserting, updating, and deleting data should be simple and not require programming.

8. **Independence from physical storage**: The way data is stored should not affect how it is accessed.

9. **Schema changes should not affect applications**: Changes to the database structure should not break programs that use the data.

MODULE – 5

INTRODUCTION TO DBMS

10. **Integrity rules**: The database should enforce rules to ensure data integrity (e.g., constraints).

11. **Data distribution independence**: The database should support data being spread across multiple locations without affecting access.

12. **No bypassing relational model**: The relational model should not be ignored or bypassed.

These rules were designed by **Edgar F. Codd** to ensure that RDBMS systems are efficient, flexible, and maintain data integrity.

5. What do you understand By Data Redundancy?

→**Data redundancy** refers to the unnecessary duplication of data in a database or system. This happens when the same piece of data is stored in multiple places, which can lead to several issues:

😊 Key points about **data redundancy**:

1. **Unnecessary duplication**: The same data is repeated across different tables or parts of a database.
2. **Storage waste**: It takes up extra space, which is inefficient and increases storage costs.

3. **Inconsistent data**: If one copy of the data is updated but others aren't, it can lead to data inconsistency or errors.

4. **Maintenance challenges**: When data is duplicated, it becomes harder to manage, as updates need to be made in multiple places.

Example:

If a company's database stores the address of a customer in several different tables, any changes to the customer's address would need to be updated in every table. Failing to do so could result in different addresses being listed for the same customer, causing confusion or errors.

In relational database design, reducing data redundancy is important, and techniques like **normalization** are used to organize the data efficiently.

6. What is DDL Interpreter?

→A **DDL Interpreter** is a component of a **Database Management System (DBMS)** that interprets and executes **Data Definition Language (DDL)** commands.

→ In simple terms, the **DDL Interpreter** acts as a translator and executor for the **commands that define the structure** of the database.

MODULE – 5

INTRODUCTION TO DBMS

- **DDL** is a subset of SQL (Structured Query Language) used to define or modify the structure of database objects, such as tables, schemas, indexes, and views.

- The **DDL Interpreter** processes these commands and translates them into a format that the DBMS can understand, allowing it to make changes to the database structure.

→ Common DDL Commands:

1. **CREATE**: Used to create a new table, index, or database.

2. **ALTER**: Used to modify an existing database object (like adding a new column to a table).

3. **DROP**: Used to delete an existing database object (like a table or index).

4. **TRUNCATE**: Used to remove all records from a table but keep its structure.

→ Role of the DDL Interpreter:

- **Parsing**: The interpreter first parses the DDL statement, checking its syntax to ensure it's correct.

- **Execution**: Once parsed, it sends instructions to the underlying DBMS to modify the database structure accordingly.

- **Updating Metadata**: The DBMS maintains metadata (information about database objects). The DDL Interpreter ensures that the metadata is updated whenever a change to the database structure is made.

MODULE – 5
INTRODUCTION TO DBMS

7. What is DML Compiler in SQL?

→ A **DML Compiler** in SQL is a component of a **Database Management System (DBMS)** that processes and executes **Data Manipulation Language (DML)** statements. DML commands are used to manipulate data in the database, such as retrieving, inserting, updating, or deleting data.

- A **DML Compiler** in SQL is a part of a **DBMS** (Database Management System) that handles commands used to work with the data in the database, such as **SELECT**, **INSERT**, **UPDATE**, and **DELETE**.

- When you write a DML query, the compiler first **checks** if the query is written correctly. Then, it tries to **optimize** the query to make sure it runs as fast as possible. After that, it **translates** the query into a format the database can understand and execute.

- Once the query is ready, the compiler tells the **database engine** to run it, which could involve pulling data from the database or updating it. The results (if it's a `SELECT` query) or the changes (for `INSERT`, `UPDATE`, or `DELETE`) are then returned to the user.

In short, a DML compiler helps make sure the SQL queries are efficient and run correctly on the database.

MODULE – 5
INTRODUCTION TO DBMS

8. What is SQL Key Constraints writing an Example of SQL Key Constraints?

→ SQL Key Constraints are rules that help maintain data integrity and ensure that the relationships between tables in a database are consistent. These constraints are applied to the columns in a table to ensure the correctness and validity of the data.

😊 Here are the main types of SQL Key Constraints:

1. **PRIMARY KEY**: Ensures that the column (or combination of columns) has unique values and cannot contain `NULL` values. A table can only have one primary key.

2. **FOREIGN KEY**: Ensures that a column (or combination of columns) refers to a valid row in another table. This enforces referential integrity between tables.

3. **UNIQUE**: Ensures that all values in a column are distinct, meaning no duplicate values are allowed.

4. **NOT NULL**: Ensures that a column cannot have a `NULL` value. This is often used for columns that must have data.

MODULE – 5

INTRODUCTION TO DBMS

5. **CHECK**: Ensures that all values in a column satisfy a specific condition.

- Explanation of Constraints:

1. **PRIMARY KEY**: The `student_id` column in the `Students` table and the `course_id` column in the `Courses` table are both primary keys. This ensures that every student and course has a unique identifier.

2. **FOREIGN KEY**: The `student_id` in the `Courses` table is a foreign key referencing the `student_id` in the `Students` table. This ensures that a course is assigned to a valid student.

3. **UNIQUE**: The `email` column in the `Students` table has a `UNIQUE` constraint, which ensures no two students can have the same email address.

4. **NOT NULL**: The `first_name`, `last_name`, and `email` columns in the `Students` table cannot be `NULL`, ensuring every student has this essential information.

- This structure enforces relationships between tables, ensures data consistency, and prevents issues like duplicate or missing information.

MODULE – 5
INTRODUCTION TO DBMS

9. What is save Point? How to create a save Point write a Query?

→ A **SAVEPOINT** in SQL is a marker or a point within a transaction that allows you to partially roll back a transaction to that specific point without affecting the entire transaction. This is useful when you're working with complex transactions and want to be able to roll back to a specific state if needed, rather than undoing all changes made within the transaction.

■ **Key Points:**

- **SAVEPOINT** is used to mark specific points in a transaction, and you can roll back to those points if needed.
- **ROLLBACK TO SAVEPOINT** will undo changes made after a specific savepoint but keep changes made before it.
- `COMMIT` makes all changes permanent.
- `ROLLBACK` without specifying a savepoint undoes all changes made in the current transaction.

10. What is trigger and how to create a Trigger in SQL?

→ A **Trigger** in SQL is a special type of stored procedure that automatically executes or "fires" when certain events (such as an `INSERT`, `UPDATE`, or `DELETE` operation) occur on a specified table or view. Triggers help automate actions in response to changes in the database and can be used to enforce business rules, maintain data integrity, or log changes.

😊 Types of Triggers:

1. **BEFORE Trigger**: Executes before the specified operation (INSERT, UPDATE, DELETE) on a table.

2. **AFTER Trigger**: Executes after the specified operation has been performed.

3. **INSTEAD OF Trigger**: Executes instead of the specified operation (typically used in views).

😊Trigger Events:

-- `INSERT`: Trigger fires when new data is inserted into a table.

-- `UPDATE`: Trigger fires when data in a table is modified.

-- `DELETE`: Trigger fires when data is deleted from a table.

😊Syntax to Create a Trigger:

The general syntax for creating a trigger in SQL is:

```sql
CREATE TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF}
{INSERT | UPDATE | DELETE}
ON table_name
FOR EACH ROW
```

MODULE – 5

INTRODUCTION TO DBMS

```
BEGIN

  -- Trigger body (SQL statements to execute)

END;

```

Parameters:

- `trigger_name`: The name of the trigger.

- `BEFORE`, `AFTER`, `INSTEAD OF`: Specifies when the trigger should fire (before or after the operation).

- `INSERT`, `UPDATE`, `DELETE`: The event that causes the trigger to fire.

- `table_name`: The name of the table on which the trigger is defined.

- `FOR EACH ROW`: Ensures the trigger fires for each row affected by the operation.