MODULE 5

BASICS OF DATABASE

1. What do you understand by Database ?

Ans. A database is a structured collection of data that is organized and stored electronically in a computer system. It is designed to efficiently manage and manipulate large volumes of information, allowing for easy retrieval, modification, and deletion of data as needed. Databases typically consist of tables, which contain rows and columns where each row represents a record and each column represents a specific attribute or field of the data.

2. What is Normalization ?

Ans. Database normalization is a process of organizing data in a relational database to minimize redundancy and improve data integrity. Redundancy means having the same data stored in multiple places. This can be inefficient and lead to errors. Normalization helps to ensure that data is stored consistently and accurately.

There are different levels of normalization, called "normal forms." Each level has a set of rules that a database table must follow. The most common normal forms are:

- **First normal form (1NF) :** This is the basic level of normalization. It ensures that each cell in a table contains a single value, and there are no repeating groups of columns.
- **Second normal form (2NF) :** In addition to the rules of 1NF, 2NF ensures that all columns in a table are dependent on the table's primary key, which is a unique identifier for each row in the table.
- **Third normal form (3NF) :** This builds on 2NF by eliminating any columns that are dependent on other non-key columns.

Normalization is an important concept in database design. By following the principles of normalization, you can create databases that are more efficient, reliable, and easier to maintain.

3. What is Difference between DBMS and RDBMS?

Ans. DBMS (Database Management System) and RDBMS (Relational Database Management System) are both involved in data management, but they have some key differences :

Data Storage :

- **DBMS :** Stores data in various formats like files or hierarchical structures. There's no direct relationship between different pieces of data.
- **RDBMS :** Stores data in tables with rows and columns. These tables can be linked together to show relationships between different data points.

Data Access and Manipulation :

- **DBMS :** Data access methods depend on the specific storage format used.
- **RDBMS :** Uses a standardized query language (like SQL) to access and manipulate data in a structured way.

Data Integrity :

- **DBMS :** May have limited data integrity features, leading to data redundancy and inconsistencies.
- **RDBMS :** Enforces data integrity through features like normalization to minimize redundancy and ensure data accuracy.

Security :

- **DBMS :** Security features may vary depending on the system.
- **RDBMS :** Offers robust security features like user access controls and data encryption.

Scalability :

- **DBMS :** Generally less scalable for large datasets.
- **RDBMS :** More scalable and can handle growing data volumes efficiently.

Use Cases :

- **DBMS :** Suitable for small, simple applications or storing unstructured data. (Ex: Microsoft Access)
- **RDBMS :** Ideal for storing large amounts of structured data with complex relationships. (Ex: MySQL, Oracle)

In simpler terms, an RDBMS is a more advanced type of DBMS that offers a structured and relational way to manage data. It's the industry standard for most database applications.

4. What is MF Cod Rule of RDBMS Systems ?

Ans. There actually isn't a single "MF Cod Rule" in RDBMS systems. It likely refers to Codd's Rules, a set of thirteen principles developed by Edgar F. Codd that define a true Relational Database Management System (RDBMS). These rules ensure data integrity, consistency, and usability in relational databases.

Here are some of the key Codd's Rules :

- **The Information Rule :** All data in the database is stored in tables with rows and columns.
- **The Guaranteed Access Rule :** Every single data point can be accessed using a combination of table name, primary key, and column name.
- **The Systematic Treatment of Null Values :** A consistent way to handle missing information (null values) is defined.
- **The Active Data Catalog Rule :** The database structure (metadata) is stored within the database itself and accessible through the same relational language.

Following all thirteen rules ensures a database adheres strictly to the relational model. In practice, some modern RDBMS systems may deviate from a few of the less critical rules while still offering core relational functionality.

5. What do you understand By Data Redundancy ?

Ans. In a Relational Database Management System (RDBMS), data redundancy refers to the situation where the same piece of data is stored in multiple places. This can happen intentionally for specific purposes, or unintentionally due to poor database design.

Here's a breakdown of data redundancy :

- **What it is :** Duplicate copies of the same data existing in different tables or parts of the database.
- **Why it's a concern :** Redundancy can lead to several issues :
  - **Data inconsistencies :** If one copy of the data is updated but not all others, the database becomes inconsistent and unreliable.
  - **Wasted storage space :** Storing the same data multiple times uses up unnecessary storage capacity.
  - **Increased maintenance :** Keeping all redundant data synchronized requires more effort and can be error-prone.

Here are some examples of data redundancy :

- Storing a customer's address in both the customer table and the order table.

- Including a department name in every employee record, even though there's a separate department table.

Data redundancy can be minimized through techniques like :

- **Normalization :** This process involves restructuring the database to eliminate unnecessary duplication and establish relationships between tables.
- **Master data management :** Maintaining a central source for specific types of data (e.g., customer information) ensures consistency across the entire system.

6. What is DDL Interpreter ?

Ans. A Data Definition Language (DDL) interpreter is a component of a database management system (DBMS) responsible for processing and executing DDL statements. DDL statements are used to define, modify, and manage the structure and organization of database objects such as tables, views, indexes, and constraints**.**

The DDL interpreter parses DDL statements issued by users or applications and translates them into internal commands or instructions that the DBMS can understand and execute. These instructions typically involve actions such as creating, altering, or dropping database objects, as well as defining constraints and permissions.

The DDL interpreter plays a critical role in database administration by ensuring that changes to the database schema are carried out accurately and efficiently. It enforces data integrity constraints, manages access permissions, and maintains the overall consistency and integrity of the database structure.

7. What is DML Compiler in SQL ?

Ans. In SQL (Structured Query Language), a Data Manipulation Language (DML) compiler is a component responsible for processing and executing DML statements. DML statements are used to manipulate the data stored in the database, such as inserting, updating, deleting, and querying records.

The DML compiler parses DML statements issued by users or applications and generates an execution plan or set of instructions for the database engine to carry out the requested data manipulation operations. This execution plan includes details on how the data should be accessed, modified, or retrieved from the underlying database tables.

The DML compiler plays a crucial role in query optimization and performance tuning by determining the most efficient way to execute DML statements based on factors such as indexing, data distribution, and query complexity. It aims to minimize the execution time and resource consumption while ensuring the correctness and consistency of the data manipulation operations.

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

Ans. SQL key constraints are special rules you can define within a table to enforce data integrity. They ensure your data follows specific guidelines, making it more reliable and easier to manage. There are three main types of key constraints in SQL :

I. **Primary Key :** This constraint acts as a unique identifier for each row in a table. It can be a single column or a combination of columns that cannot contain duplicate values and must not allow NULL values.

II. **Unique Key :** Similar to the primary key, a unique key constraint ensures all values in a specified column (or set of columns) are distinct within the table. However, a table can have multiple unique constraints, unlike the primary key which is singular. Unlike the primary key, unique keys can allow NULL values.

III. **Foreign Key :** This constraint establishes a link between two tables, referencing a primary key in another table. It ensures data consistency by referencing a valid existing record in the referenced table.

Example :

```
CREATE TABLE Orders (
order_id INT PRIMARY KEY,
 customer_id INT NOT NULL,
order_number VARCHAR(20) UNIQUE,
FOREIGN KEY (customer_id) REFERENCES Customers(customer_id) );
```

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

Ans. Savepoint is a command in SQL that is used with the rollback command.

- It is a command in Transaction Control Language that is used to mark the transaction in a table.

- Consider you are making a very long table, and you want to roll back only to a certain position in a table then; this can be achieved using the savepoint.

- If you made a transaction in a table, you could mark the transaction as a certain name, and later on, if you want to roll back to that point, you can do it easily by using the transaction's name.
- Savepoint is helpful when we want to roll back only a small part of a table and not the whole table. In simple words, we can say savepoint is a bookmark in SQL.

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

Ans. In SQL, a trigger is a special type of stored procedure that automatically executes a set of SQL statements when a specific event occurs in a database. These events typically involve data manipulation in a table, such as inserting, updating, or deleting rows. Triggers are essentially rule enforcers or automators that act upon these events.

Here's a breakdown of the key aspects of triggers :

- **Functionality :** Triggers are like mini-programs that reside within the database. They are inactive until the designated event triggers them. Once triggered, they execute the embedded SQL statements, which can perform various tasks like updating other tables, logging changes, or sending notifications.

- **Types of Triggers :** There are generally three main types of triggers based on timing:

  - **BEFORE triggers :** These execute **before** the main INSERT, UPDATE, or DELETE operation occurs. They can be used for data validation or to set up the database for the operation.

  - **AFTER triggers :** These execute **after** the main INSERT, UPDATE, or DELETE operation is successfully completed. They are commonly used for data auditing, logging changes, or performing post-processing tasks.

  - **INSTEAD OF triggers :** Less frequently used, these triggers act as a substitute for the original INSERT, UPDATE, or DELETE operation. They completely replace the standard operation with custom logic defined within the trigger.

- **Benefits of Triggers :** Triggers offer several advantages :
    - **Enforcing Data Integrity :** They can ensure data consistency by validating changes or enforcing referential constraints that might not be possible with standard SQL statements.
    - **Automating Tasks :** Triggers can automate repetitive tasks that would otherwise require manual intervention, improving database efficiency.
    - **Data Auditing :** Triggers can be used to track changes made to the database, creating a log of insertions, updates, and deletions.

Creating a Trigger in SQL

The specific syntax for creating triggers can vary slightly between different SQL platforms (e.g., MySQL, SQL Server, Oracle). However, the general structure typically involves the following :

```
CREATE TRIGGER trigger_name
<trigger_timing> <event> ON table_name
FOR EACH ROW
BEGIN
  -- SQL statements to be executed
END;
```

Here's an explanation of the parts involved :

- CREATE TRIGGER trigger_name : This defines that you are creating a trigger and assigns a unique name to it.
- <trigger_timing> : This specifies the timing of the trigger execution, such as BEFORE, AFTER, or INSTEAD OF.
- <event> : This indicates the specific event that triggers the code execution, like INSERT, UPDATE, or DELETE.
- ON table_name : This defines the table on which the trigger is created.
- FOR EACH ROW : This indicates that the trigger logic applies to each row affected by the triggering event (insert, update, or delete).

- BEGIN : This marks the beginning of the trigger body where you write the SQL statements to be executed.

- END : This marks the end of the trigger body.