ASSIGNMENT 1 MODULE 4 :- Basics of Database

Q1. What do you understand By Database?

A1. A **database** is an organized collection of data that is stored, managed, and accessed electronically. It provides a systematic way to store, retrieve, and manipulate data, making it easier to manage large amounts of information efficiently and securely.

Q2. What is Normalization?

A2. **Normalization** is a systematic process in database design used to organize data in a way that reduces redundancy and dependency. The primary goal of normalization is to divide large tables into smaller, more manageable ones and to establish relationships between them to ensure data integrity.

Q3. What is Difference between DBMS and RDBMS?

A3.

| **Aspect** | **DBMS (Database Management System)** | **RDBMS (Relational Database Management System)** |
| --- | --- | --- |
| **Definition** | A software system for storing and managing data. | A DBMS that organizes data in a tabular (relational) format. |
| **Data Structure** | Stores data in files, hierarchies, or non-tabular forms. | Stores data in tables with rows and columns. |
| **Normalization** | Does not support or enforce normalization. | Supports and enforces normalization to reduce redundancy. |
| **Relationships** | Does not inherently support relationships between data. | Supports relationships through keys (primary, foreign). |
| **Key Constraints** | No support for primary or foreign key constraints. | Enforces constraints like primary key, foreign key, etc. |
|  |  |  |
|  |  |  |
| **Query Language** | Typically uses proprietary query languages. | Uses SQL (Structured Query Language) for data operations. |
| **Example Databases** | Microsoft Access, FoxPro. | MySQL, PostgreSQL, Oracle, SQL Server. |
| **Scalability** | Suitable for small to medium-sized databases. | Suitable for large-scale databases. |
|  |  |  |
| **Data Redundancy** | Data redundancy is high as it lacks normalization. | Reduces redundancy through normalization techniques. |
| **Performance** | May be slower for complex queries due to lack of optimization. | Optimized for complex queries due to relational design. |
| **Usage** | Used in standalone applications. | Used in web-based, enterprise, and distributed applications. |

**Summary:**

1. **DBMS** is a general system for managing databases without rigid relational structures.
2. **RDBMS** is a specific type of DBMS that organizes data in tables with defined relationships, offering better performance, scalability, and integrity for large systems.

Q4. What is MF Cod Rule of RDBMS Systems?

A4. The **E. F. Codd Rules** are a set of 12 rules proposed by Dr. Edgar F. Codd, the inventor of the relational model, to define what is required for a database management system (DBMS) to qualify as a **relational database management system (RDBMS)**. These rules provide a benchmark to differentiate a true RDBMS from other database systems.

Here’s a summary of **Codd's 12 rules**:

**Rule 0: The Foundation Rule**

A system must qualify as a relational database management system by supporting relational rules and all its features.

**Rule 1: The Information Rule**

All information in a relational database is represented explicitly in tables (rows and columns).

**Rule 2: Guaranteed Access Rule**

Every piece of data is accessible using a unique identifier (a combination of a table name, primary key, and column name).

**Rule 3: Systematic Treatment of NULL Values**

The system must handle NULL values (unknown or inapplicable data) systematically and distinctly from other data.

**Rule 4: Active Online Catalog**

The database's structure and metadata must be stored as relational tables, making them accessible via the same query language used for the data.

**Rule 5: Comprehensive Data Sublanguage Rule**

The system must support at least one relational query language (like SQL) that allows users to define, query, and manipulate data and structures.

**Rule 6: View Updating Rule**

Any view (a virtual table) that can theoretically be updated must be updatable by the system.

**Rule 7: High-Level Insert, Update, and Delete**

The system must support set-based operations for inserting, updating, and deleting data, not just single-row operations.

**Rule 8: Physical Data Independence**

Changes in the physical storage of data (e.g., hardware, file structures) should not affect the logical structure or how users interact with the data.

**Rule 9: Logical Data Independence**

Changes in the logical structure of the database (e.g., adding new columns or tables) should not affect the applications or queries using the data.

**Rule 10: Integrity Independence**

The system must support integrity constraints (like primary keys, foreign keys, and domain constraints) and allow users to define them at the database level rather than in applications.

**Rule 11: Distribution Independence**

The system must work seamlessly regardless of whether the data is distributed across multiple locations or centralized.

**Rule 12: Non-Subversion Rule**

If the system provides a low-level access mechanism, it must not allow users to bypass relational integrity constraints through this mechanism.

Q5. What do you understand By Data Redundancy?

A5. **Data Redundancy** refers to the unnecessary duplication of data within a database or storage system. It occurs when the same piece of information is stored in multiple places, either within the same database or across different databases. While some level of redundancy is sometimes intentional for backup or performance purposes, excessive redundancy can lead to problems such as inefficiency, increased storage requirements, and data inconsistency.

Q6. What is DDL Interpreter?

A6. A **DDL Interpreter** is a component of a database management system (DBMS) that processes **Data Definition Language (DDL)** statements. DDL commands are used to define and modify the structure of database objects like tables, indexes, views, schemas, etc.

The DDL Interpreter interprets these commands and converts them into a format that the database can execute to make changes to the database schema. It ensures that the database's metadata (data about the structure of the database) is updated and maintained properly.

The DDL Interpreter is part of the **query processor** in a DBMS. It works alongside other components, such as the **query optimizer** and **execution engine**, to handle database commands efficiently.

By translating high-level DDL commands into operations that the DBMS can execute, the DDL Interpreter plays a critical role in managing the database schema and ensuring that the structure is accurate and up to date.

Q7. What is DML Compiler in SQL?

A7. A **DML Compiler** is a component of a Database Management System (DBMS) that processes **Data Manipulation Language (DML)** statements. DML commands are used to manipulate data stored in the database, such as inserting, updating, deleting, or retrieving data.

The DML Compiler translates high-level DML queries (e.g., SQL statements like INSERT, SELECT, UPDATE, DELETE) into low-level instructions or commands that the DBMS query execution engine can execute. These commands typically interact with the physical database storage.

The DML Compiler is an essential component of a DBMS's **Query Processor**. It bridges the gap between the high-level user queries written in SQL and the low-level database operations required to manipulate data. By doing so, it ensures efficient data processing while maintaining abstraction for the end-user.

Q8. What is SQL Key Constraints writing an Example of SQL Key Constraints?

A8. SQL Key Constraints are rules enforced on a database table's columns to maintain the integrity, accuracy, and consistency of data. They help ensure the validity of data entered into the table and define relationships between tables.

**Types of SQL Key Constraints:**

1. **Primary Key**:
   * Ensures that each row in a table has a unique and non-null value.
   * A table can have only one primary key.
2. **Foreign Key**:
   * Establishes a relationship between two tables by linking a column in one table to the primary key in another.
   * Ensures referential integrity.
3. **Unique Key**:
   * Ensures all values in a column or combination of columns are unique.
   * A table can have multiple unique keys.
4. **Not Null**:
   * Ensures that a column cannot have a null value.
5. **Check**:
   * Ensures that all values in a column satisfy a specific condition.
6. **Default**:
   * Assigns a default value to a column if no value is specified during insertion.

**Example Combining Multiple Constraints:**

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY, -- Primary Key

Name VARCHAR(100) NOT NULL, -- Not Null

Email VARCHAR(100) UNIQUE, -- Unique

Age INT CHECK (Age >= 18), -- Check

DepartmentID INT,

FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID), -- Foreign Key

JoiningDate DATE DEFAULT CURRENT\_DATE -- Default

);

Q9. What is save Point? How to create a save Point write a Query?

A9. A **savepoint** in SQL is a marker within a transaction that allows you to roll back part of the transaction instead of the entire transaction. Savepoints are particularly useful in large transactions when you want to ensure that some parts of the transaction remain intact while undoing others.

* **Key Features**:
  + A savepoint is created using the SAVEPOINT statement.
  + You can roll back to a specific savepoint using ROLLBACK TO SAVEPOINT.
  + Savepoints do not commit a transaction; they merely set a point for potential rollback.
  + Savepoints are valid only within the current transaction.

**How to Create a Savepoint**

**Syntax**:

SAVEPOINT savepoint\_name;

**Roll Back to a Savepoint**:

ROLLBACK TO SAVEPOINT savepoint\_name;

**Release a Savepoint** (Optional):

RELEASE SAVEPOINT savepoint\_name;

Releasing a savepoint removes it from the transaction, and you cannot roll back to it afterward.

Q10. What is trigger and how to create a Trigger in SQL?

A10. A **trigger** in SQL is a stored procedure that is automatically executed (or "triggered") in response to specific events on a table or view. Triggers are typically used to enforce business rules, validate data, maintain audit trails, and automate tasks in the database.

**Key Characteristics of Triggers:**

1. **Automatically Executed**: Triggers run automatically when a specified event occurs.
2. **Attached to Tables or Views**: They are associated with a table or a view.
3. **Types of Triggers**:
   * **Before Triggers**: Executed before the event (e.g., BEFORE INSERT).
   * **After Triggers**: Executed after the event (e.g., AFTER INSERT).
   * **Instead of Triggers**: Replace the standard action (common in views).
4. **Events That Can Trigger**:
   * INSERT: Triggered when a new record is added.
   * UPDATE: Triggered when a record is updated.
   * DELETE: Triggered when a record is deleted.

**How to Create a Trigger in SQL**

**General Syntax**:

CREATE TRIGGER trigger\_name

AFTER | BEFORE event\_name

ON table\_name

FOR EACH ROW

BEGIN

-- SQL statements

END;

-- Create the trigger

CREATE TRIGGER After\_Insert\_Trigger

AFTER INSERT

ON Employees

FOR EACH ROW

BEGIN

INSERT INTO Audit\_Log (EmployeeID, ActionPerformed, ActionTime)

VALUES (NEW.EmployeeID, 'INSERT', NOW());

END;