**JIGNESH KUMAR**

**MODULE: 5**

**Database**

**Q.1** **What do you understand By Database.**

**Ans:**

A database is an organized collection of data, typically stored and accessed electronically from a computer system. It is designed to efficiently manage, store, and retrieve data according to the needs of the users or applications. Databases are structured in a way that allows for easy insertion, modification, and deletion of data while maintaining data integrity and security. They are essential components of modern information systems, used in various applications such as websites, enterprise software, mobile apps, and more. Databases can range from simple, single-user systems to complex, distributed databases that serve large organizations or even the entire internet.

**Q.2** **What is Normalization?**

**Ans:**

Normalization is a process used in database design to organize data in a relational database efficiently. It involves breaking down a database into smaller, more manageable tables and defining relationships between them. The primary goal of normalization is to minimize redundancy and dependency in the data, which helps improve data integrity, reduce anomalies, and make the database more efficient in terms of storage and querying.

Normalization typically involves several steps, each represented by a normal form (e.g., First Normal Form, Second Normal Form, Third Normal Form, etc.). These normal forms define specific criteria that the database must meet to be considered normalized. For example:

1. First Normal Form (1NF): Ensures that each table has a primary key and that each column contains atomic values (i.e., no repeating groups or arrays).
2. Second Normal Form (2NF): Requires that each non-key attribute is fully functionally dependent on the primary key, meaning there are no partial dependencies.
3. Third Normal Form (3NF): Ensures that there are no transitive dependencies, meaning that non-key attributes are not dependent on other non-key attributes.

Top of Form

Bottom of Form

**Q.3 What is Difference between DBMS and RDBMS?**

**Ans:**

1. **DBMS (Database Management System)**:
   * A DBMS is a software system that enables users to define, create, maintain, and control access to databases.
   * It provides various functionalities such as data storage, retrieval, manipulation, security, and integrity.
   * DBMS may or may not support the relational model. Examples include MongoDB (a NoSQL database), Microsoft Access, and Apache Cassandra.
2. **RDBMS (Relational Database Management System)**:
   * An RDBMS is a specific type of DBMS that manages data based on the relational model, introduced by E.F. Codd.
   * In an RDBMS, data is organized into tables (relations) consisting of rows (tuples) and columns (attributes).
   * RDBMSs enforce relationships between tables through the use of primary keys, foreign keys, and relational algebra operations.
   * SQL (Structured Query Language) is the standard language used to interact with RDBMSs.
   * Examples of RDBMSs include MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server, and SQLite.

Q.4.What is MF Cod Rule of RDBMS Systems?

Ans:

The MF Cod Rule, more commonly known as Codd's rules, are a set of thirteen rules proposed by Edgar F. Codd, the inventor of the relational model for databases. These rules define what is required from a database management system (DBMS) to be considered relational, i.e., a Relational Database Management System (RDBMS).

Here is a summary of Codd's 12 rules (the numbering starts at zero):

1. Information Rule: All information in a relational database is represented explicitly at the logical level and in exactly one way – by values in tables.

2. Guaranteed Access Rule: Each and every datum (atomic value) is guaranteed to be logically accessible by resorting to a combination of table name, primary key value, and column name.

3. Systematic Treatment of Null Values: Null values (distinct from empty character string or a string of blank characters and distinct from zero or any other number) are supported for representing missing or inapplicable information in a systematic way, independent of data type.

4. Dynamic Online Catalog Based on the Relational Model: The database description is represented at the logical level in the same way as ordinary data, so authorized users can apply the same relational language to its interrogation as they apply to the regular data.

5. Comprehensive Data Sublanguage Rule: A relational system may support several languages and various modes of terminal use (for example, the fill-in-the-blanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and that is comprehensive in supporting all the following items: data definition; view definition; data manipulation (interactive and by program); integrity constraints; and authorization.

6. View Updating Rule: All views that are theoretically updateable are also updateable by the system.

7. High-Level Insert, Update, and Delete: The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update, and deletion of data.

8. Physical Data Independence: Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods.

9. Logical Data Independence: Application programs and terminal activities remain logically unimpaired when information-preserving changes of any kind that theoretically permit unimpairment are made to the base tables.

10. Integrity Independence: Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

11. Distribution Independence: The data manipulation sublanguage of a relational DBMS must enable application programs to remain logically unimpaired whether and wherever data is distributed or reorganized.

12. Non-Subversion Rule: If a relational system has a low-level (single-record-at-a-time) language, that low-level language cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher-level (multiple-records-at-a-time) relational language.

These rules are meant to ensure that a database management system is truly relational and can efficiently manage data in a consistent, logical manner.

**Q.** **5. What do you understand By Data Redundancy?**

**Ans:**

Data redundancy refers to the duplication of data within a database or information system. In other words, it occurs when the same piece of data is stored in multiple locations or multiple times within a system. This duplication can occur intentionally or unintentionally.

Intentional redundancy may be implemented for various reasons, such as improving data availability, enhancing system performance, or ensuring data integrity through backup copies. However, excessive redundancy can lead to inefficiency in storage usage, increased maintenance complexity, and inconsistency issues if the redundant data is not properly synchronized.

Unintentional redundancy often occurs due to poor database design or application architecture, resulting in duplicate records or information being stored unnecessarily. This can lead to inconsistencies and discrepancies in data, making it challenging to maintain data accuracy and integrity.

Overall, managing data redundancy effectively is crucial in database management to balance the benefits of redundancy with the associated costs and risks.

**Q.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.

**Q.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.

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

**Ans:**

SQL key constraints are used to enforce rules on columns in a database table, ensuring data integrity and consistency. There are several types of key constraints in SQL, including:

1. **Primary Key Constraint**: A primary key uniquely identifies each record in a table. It ensures that the values in the specified column(s) are unique and not null. Only one primary key constraint can be defined per table.
2. **Unique Constraint**: A unique constraint ensures that the values in the specified column(s) are unique, but unlike a primary key, it allows null values.
3. **Foreign Key Constraint**: A foreign key constraint establishes a relationship between two tables. It ensures that the values in the specified column(s) in one table match the values in another table's primary key or unique constraint.

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

**Ans:**

A save point in SQL is a point within a transaction where you can roll back to, if necessary, without rolling back the entire transaction. It allows you to create intermediate points in a transaction to which you can later roll back if needed, providing finer control over transaction management.

To create a savepoint in SQL, you use the **SAVEPOINT** statement followed by the name of the savepoint. Here's the syntax:

SAVEPOINT savepoint\_name;

Here's an example of how to create a savepoint within a transaction:

BEGIN TRANSACTION;

-- Some SQL statements here

SAVEPOINT my\_savepoint;

-- More SQL statements here

COMMIT;

Top of Form

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

**Ans:**

A trigger in SQL is a special type of stored procedure that automatically executes in response to certain database events. These events can include actions like inserting, updating, or deleting records in a table.

Here's how you can create a trigger in SQL:

**CREATE TRIGGER trigger name**

**{BEFORE | AFTER} {INSERT | UPDATE | DELETE} ON table name**

**FOR EACH ROW**

**BEGIN**

**-- SQL statements to be executed when the trigger fires**

**END;**

Let's break down the components of this syntax:

* **CREATE TRIGGER**: This keyword is used to start the creation of a new trigger.
* **trigger name**: This is the name you give to the trigger.
* **{BEFORE | AFTER}**: This specifies whether the trigger should execute before or after the specified event (**INSERT**, **UPDATE**, or **DELETE**) occurs.
* **{INSERT | UPDATE | DELETE}**: This specifies the event that will cause the trigger to execute.
* **ON table name**: This specifies the table on which the trigger will be created.
* **FOR EACH ROW**: This specifies that the trigger will execute once for each row affected by the triggering event.
* **BEGIN** and **END**: These keywords enclose the SQL statements that will be executed when the trigger fires.

Here's an example of creating a trigger that automatically updates a timestamp column **last updated** whenever a record in the **employees** table is updated:

CREATE TRIGGER update\_last\_updated

AFTER UPDATE ON employees

FOR EACH ROW

BEGIN

UPDATE employees

SET last\_updated = CURRENT\_TIMESTAMP

WHERE employee\_id = NEW.employee\_id;

END;

In this example:

* The trigger is named **update\_last\_updated**.
* It is set to execute **AFTER** an **UPDATE** occurs on the **employee’s** table.
* The trigger fires **FOR EACH ROW** that is updated.
* Inside the trigger's body, an **UPDATE** statement is executed to set the **last\_updated** column to the current timestamp (**CURRENT\_TIMESTAMP**) for the row being updated (**NEW.employee\_id**).

Top of Form