1. What do you understand By Database

A database is a collection of organized data that is stored in a way that allows for efficient retrieval and manipulation. It is a system that enables the storage, modification, and extraction of data from a collection of interrelated data.

A database can be thought of as an electronic filing system that allows you to store, update, and retrieve data as needed. It is designed to handle large amounts of data and provide a way to manage, search, and analyze the data efficiently.

Characteristics of a Database:

Collection of Data: A database is a collection of data that is stored in a single location.

Organized Data: The data in a database is organized in a way that allows for efficient retrieval and manipulation.

Storage and Retrieval: A database provides a way to store and retrieve data as needed.

Data Sharing: A database allows multiple users to access and share the same data.

Data Security: A database provides mechanisms for securing the data from unauthorized access and ensuring data integrity.

Types of Databases:

Relational Database: A relational database organizes data into tables with well-defined relationships between them. Examples: MySQL, PostgreSQL.

NoSQL Database: A NoSQL database stores data in a variety of formats, such as key-value, document, or graph. Examples: MongoDB, Cassandra.

Cloud Database: A cloud database is a database that is hosted and managed in the cloud. Examples: Amazon Aurora, Google Cloud SQL.

Distributed Database: A distributed database is a database that is spread across multiple physical locations. Examples: Google Cloud Spanner, Amazon DynamoDB.

Database Management Systems (DBMS):

A DBMS is a software system that allows you to define, create, maintain, and manipulate databases. Examples of DBMS include:

MySQL

PostgreSQL

Microsoft SQL Server

Oracle Database

IBM DB2

2. What is Normalization?

In SQL and the context of relational databases, normalization is the process of structuring a database in a way that reduces redundancy and dependency. It involves dividing large tables into smaller tables and defining relationships between them to enhance data integrity and reduce the chances of anomalies during data operations like insert, update, and delete.

The primary objectives of normalization in SQL are:

Eliminating Redundant Data: Ensuring that the same piece of data is not stored in multiple places.

Ensuring Data Dependencies: Organizing data so that modifications can be made in just one place.

Normalization typically involves several stages or "normal forms." Each normal form has specific requirements:

1. First Normal Form (1NF)

Rule: Each column must contain atomic (indivisible) values, and each record needs to be unique.

Objective: Eliminate repeating groups of columns.

2. Second Normal Form (2NF)

Rule: Achieve 1NF and ensure that all non-key columns are fully functional dependent on the primary key.

Objective: Eliminate partial dependency (where non-key attributes are dependent on only part of the primary key).

3. Third Normal Form (3NF)

Rule: Achieve 2NF and ensure that all the columns are not only dependent on the primary key but are also non-transitively dependent on it.

Objective: Eliminate transitive dependency (where non-key attributes depend on other non-key attributes).

4. Boyce-Codd Normal Form (BCNF)

Rule: Achieve 3NF, and for every functional dependency, the left-hand side should be a super key.

Objective: Handle situations where 3NF does not suffice in certain complex relational structures.

5. Fourth Normal Form (4NF)

Rule: Achieve BCNF and ensure no multi-valued dependencies other than a candidate key.

Objective: Eliminate multi-valued dependencies.

6. Fifth Normal Form (5NF)

Rule: Achieve 4NF and ensure that every join dependency in the table is implied by the candidate keys.

Objective: Decompose tables to avoid redundancy without losing data integrity.

3. What is Difference between DBMS and RDBMS?



4. What is MF Cod Rule of RDBMS Systems?

E. F. Codd, an English computer scientist, introduced a set of thirteen rules, known as "Codd's 12 Rules" (numbered from 0 to 12), which define what is required for a database management system to be considered a true relational database management system (RDBMS). These rules ensure that a database system adheres to the principles of relational theory.

Here is a summary of Codd's 12 rules:

Rule 0: Foundation Rule

A system must qualify as a relational database management system (RDBMS) by supporting the management of databases entirely through its relational capabilities.

Rule 1: Information Rule

All information in the database is represented in one and only one way: as values within tables (relations).

Rule 2: Guaranteed Access Rule

Every piece of data must be logically accessible by using a combination of table name, primary key (row identifier), and column name.

Rule 3: Systematic Treatment of Null Values

Null values (distinct from empty character strings or a zero value) must be uniformly treated as "missing information" and must be handled in a systematic way, independent of data type.

Rule 4: Dynamic Online Catalog Based on the Relational Model

The database's description, also known as metadata, must be stored within the database itself, and it should be accessible using the same relational query language used to access the database data.

Rule 5: Comprehensive Data Sub-language Rule

A relational system may support several languages and various modes of terminal use (e.g., interactive, batch, application program), but there must be at least one language whose statements can express all of the following:

Data definition

Data manipulation

Data integrity

Transaction management

Rule 6: View Updating Rule

All views that are theoretically updatable must also be updatable by the system.

Rule 7: High-level Insert, Update, and Delete

The system must support set-based insertion, update, and deletion at the table level, not just row-level operations.

Rule 8: Physical Data Independence

Changes to the physical level (how data is stored) must not require a change to an application that has been designed at the logical level (how data is presented to users).

Rule 9: Logical Data Independence

Changes to the logical level (tables, columns, rows) must not require a change to the application programs that access the data.

Rule 10: Integrity Independence

Integrity constraints must be specified separately from application programs and stored in the catalog. It must be possible to change such constraints without affecting existing applications.

Rule 11: Distribution Independence

The data manipulation language should enable application programs to treat the database as a single logical entity, regardless of its distribution across various locations.

Rule 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 high-level (multiple-records-at-a-time) relational language.

5. What do you understand By Data Redundancy?

Data redundancy refers to the unnecessary duplication of data within a database or storage system. This can occur when the same piece of data is stored in multiple places. While some level of redundancy can be beneficial for backup and recovery purposes, excessive redundancy can lead to various issues and inefficiencies.

Types of Data Redundancy

Unintentional Redundancy

Occurs due to poor database design.

Leads to inconsistencies and anomalies in data.

Example: Storing the same customer information in multiple tables without synchronization.

Intentional Redundancy

Implemented for performance optimization or data recovery.

Example: Maintaining duplicate copies of data in different locations for faster access or disaster recovery.

Problems Caused by Data Redundancy

Data Inconsistency

When redundant data is not updated simultaneously, it can result in discrepancies.

Example: If a customer's address is updated in one table but not in another, the database will contain conflicting information.

Increased Storage Costs

Storing the same data multiple times consumes additional storage space, leading to higher costs.

Example: Duplicating large datasets unnecessarily inflates storage requirements.

Data Integrity Issues

Redundant data can compromise the integrity of the database, making it difficult to ensure that all data is accurate and up-to-date.

Example: Ensuring referential integrity becomes challenging when redundant data exists.

Maintenance Overhead

Managing and maintaining redundant data requires additional effort and resources.

Example: Additional logic is required to keep redundant data synchronized.

Complexity in Database Design

Redundancy can complicate the design and structure of a database, making it harder to understand and manage.

Example: More complex queries and update procedures are needed to handle redundant data.

Preventing Data Redundancy

Normalization

Organizing data into separate tables and defining relationships between them to minimize redundancy.

Example: Applying normalization forms (1NF, 2NF, 3NF, etc.) to structure data efficiently.

Database Design Principles

Following best practices in database design to avoid unnecessary duplication.

Example: Using foreign keys to link related data instead of duplicating information.

Data Integrity Constraints

Implementing constraints like primary keys, foreign keys, and unique constraints to enforce data integrity.

Example: Ensuring that each record in a table is unique and related records are consistent.

Use of Views

Creating database views to present data from multiple tables without storing redundant copies.

Example: A view can combine data from multiple tables to present a consolidated view without duplicating data.

6. What is DDL Interpreter?

A DDL (Data Definition Language) interpreter is a component of a database management system (DBMS) that processes and executes DDL statements. DDL statements are used to define, modify, and manage the structure of database objects such as tables, indexes, views, schemas, and other schema-related constructs. The DDL interpreter parses these statements, interprets their meanings, and carries out the necessary actions to modify the database schema.

Key Functions of a DDL Interpreter

Parsing

The DDL interpreter parses the DDL statements to ensure they are syntactically correct.

Example: Parsing a CREATE TABLE statement to ensure proper syntax and valid data types.

Schema Management

It manages the creation, modification, and deletion of database objects.

Example: Creating a new table, altering an existing table to add a new column, or dropping a table.

Enforcing Constraints

Ensures that constraints defined in DDL statements are enforced.

Example: Enforcing primary key, foreign key, unique, and check constraints during table creation or modification.

Catalog Updates

The DDL interpreter updates the system catalog or data dictionary to reflect the changes made to the database schema.

Example: Updating metadata to include new tables, columns, and constraints.

Error Handling

Provides feedback and error messages when DDL statements are incorrect or violate database rules.

Example: Returning an error if a CREATE TABLE statement includes an invalid data type or violates a foreign key constraint.

7. What is DML Compiler in SQL?

A DML (Data Manipulation Language) compiler in SQL is a component of a database management system (DBMS) that processes and optimizes DML statements. DML statements are used to perform operations on the data within the database, including inserting, updating, deleting, and querying data. The DML compiler translates these high-level SQL statements into a lower-level language or machine code that the database engine can execute efficiently.

Key Functions of a DML Compiler

Parsing

The DML compiler parses the DML statements to ensure they are syntactically correct.

Example: Parsing an INSERT, UPDATE, DELETE, or SELECT statement to check for correct SQL syntax.

Validation

It validates the DML statements to ensure they refer to existing database objects (such as tables and columns) and that the user has the necessary permissions.

Example: Checking if a table mentioned in a SELECT statement exists and whether the user has SELECT privileges on that table.

Query Optimization

The DML compiler optimizes the query to ensure efficient execution. This may involve choosing the best execution plan based on indexes, joins, and other factors.

Example: Rewriting a query to use an index scan instead of a full table scan for faster retrieval.

Execution Plan Generation

It generates an execution plan, which is a step-by-step procedure that the database engine will follow to execute the DML statement.

Example: Determining the order in which tables should be joined and which indexes to use.

Code Generation

The DML compiler converts the optimized execution plan into a lower-level code or machine instructions that the database engine can execute.

Example: Generating bytecode or machine code that performs the specified data manipulation operations.

Caching and Reuse

The compiled execution plans may be cached for reuse to improve performance on repeated queries.

Example: Storing the execution plan for a frequently executed SELECT statement so it can be reused without recompiling.

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

SQL key constraints are rules and restrictions applied to database tables to enforce data integrity and ensure the accuracy and consistency of the data. Key constraints are essential for maintaining relationships between tables and ensuring that the data stored in a database adheres to certain rules. The primary key constraints include primary keys, foreign keys, unique keys, and composite keys.

**Types of Key Constraints**

* **Primary Key Constraint**
* Ensures that each row in a table is unique and not null.
* Example: A table with an ID column set as the primary key.
* **Foreign Key Constraint**
* Establishes a relationship between columns in two tables, ensuring referential integrity.
* Example: A CustomerID column in an Orders table that references the ID column in a Customers table.
* **Unique Key Constraint**
* Ensures that all values in a column or a set of columns are unique across the rows in the table.
* Example: An Email column in a Users table that must be unique for each user.
* **Composite Key**
* A primary key composed of two or more columns.
* Example: A StudentCourse table where the combination of StudentID and CourseID is unique.

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

A savepoint in SQL is a way to set a point within a transaction to which you can later roll back. Savepoints allow you to partially roll back a transaction, undoing some of the changes while keeping the rest. This is particularly useful for complex transactions where you want to commit some parts but retain the ability to revert others.

**Creating and Using Savepoints**

*1. Setting a Savepoint*

You can create a savepoint within a transaction using the SAVEPOINT statement. This names the savepoint so you can reference it later.

*2. Rolling Back to a Savepoint*

If needed, you can roll back the transaction to a specific savepoint using the ROLLBACK TO SAVEPOINT statement. This undoes all changes made after the savepoint was set.

*3. Releasing a Savepoint*

You can optionally release a savepoint using the RELEASE SAVEPOINT statement, which makes it no longer available for rolling back. Note that releasing a savepoint does not affect the transaction.

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

A trigger in SQL is a special kind of stored procedure that automatically executes in response to certain events on a particular table or view. These events can be actions such as INSERT, UPDATE, or DELETE. Triggers are used to enforce business rules, maintain audit trails, synchronize tables, and perform other automated tasks.

Types of Triggers

Before Triggers

Execute before the triggering event (INSERT, UPDATE, DELETE) occurs.

Example: Before an INSERT trigger checks if a value meets a certain condition before allowing the insert.

After Triggers

Execute after the triggering event has occurred.

Example: After an UPDATE trigger logs the update to an audit table.

Instead Of Triggers

Execute in place of the triggering event. Commonly used on views.

Example: Instead of an INSERT trigger on a view that performs custom logic before inserting data into the underlying tables.