1. What do you understand By Database.

* A database is simply a Structured and systematic way of storing information to be accessed, analyzed, transformed, updated and moved.

1. What is Normalization?

* Normalization is the process of organizing data within a relational database to eliminate data anomalies, such as redundancy.
* [It involves breaking down a large, complex table into smaller and simpler tables while maintaining data relationships](https://www.datacamp.com/tutorial/normalization-in-sql).
* Here are some key points about normalization:
* **Data Integrity**:
* Normalization ensures that data remains accurate and consistent. For example, if a customer changes their age, normalization allows us to update it in one place without affecting other records.
* **Efficient Querying**:
* By breaking down data into smaller tables, normalization simplifies querying and reduces the need for complex joins. This improves query performance.
* **Storage Optimization**:
* Redundant data occupies unnecessary storage space. Normalization eliminates redundancy by splitting data into separate tables

1. What is Difference between DBMS and RDBMS?

|  |  |  |
| --- | --- | --- |
| BASIS FOR COMPARISON | DBMS | RDBMS |
| Data Storage | Stores data as files. | Stores data in tabular form, with rows and columns. |
| Data Relationships | Does not support relationships between data. | Supports relationships between tables using foreign keys. |
| Normalization | Normalization is not typically supported. | Supports normalization to reduce data redundancy |
| Data Redundancy | Higher data redundancy. | Reduced data redundancy due to relational structure |
| Users | Generally supports single users. | Supports multiple users. |
| Security | Lower security measures. | Higher security measures, including access controls. |
| Example | XML, Windows Registry, and dBase. | MySQL, PostgreSQL, SQL Server, and Oracle. |

1. What is MF Cod Rule of RDBMS Systems?

* Codd’s 12 Rules are a set of thirteen rules (numbered from zero to twelve) proposed by Edgar F. Codd, a pioneer of the relational model for databases.
* [These rules define what is required for a database management system to be considered relational, i.e., a Relational Database Management System (RDBMS)](https://en.wikipedia.org/wiki/Codd%27s_12_rules)

* Here is a brief overview of Codd’s 12 Rules:

1. **Rule 0: The Foundation Rule**:

For any system to qualify as an RDBMS, it must be able to manage databases entirely through its relational capabilities.

1. **Rule 1: The Information Rule**:

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

1. **Rule 2: The Guaranteed Access Rule**:

Each and every datum (atomic value) in a relational database is guaranteed to be logically accessible by resorting to a combination of table name, primary key value, and column name.

1. **Rule 3: Systematic Treatment of Null Values**:

Null values must be uniformly treated as “missing information,” not as empty strings, blanks, or zeros.

1. **Rule 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 that authorized users can apply the same relational language to its interrogation as they apply to the regular data.

1. **Rule 5: The Comprehensive Data Sublanguage Rule**:

A relational system may support several languages and various modes of terminal use, but there must be at least one language that supports data definition, view definition, data manipulation, integrity constraints, authorization, and transaction boundaries.

1. **Rule 6: The View Updating Rule**:

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

1. **Rule 7: High-level Insert, Update, and Delete**:

The system must support set-level insert, update, and delete operations.

1. **Rule 8: Physical Data Independence**:

Changes to the physical storage of data should not affect the application programs.

10**.Rule 9: Logical Data Independence:**

Changes to the logical structure of the database should not affect the user’s ability to access data.

11.**Rule 10: Integrity Independence**:

Integrity constraints must be specified separately from application programs and stored in the catalog.

12.**Rule 11: Distribution Independence**:

The distribution of data across multiple locations should be invisible to users.

13. **[Rule 12: Non-Subversion Rule](https://en.wikipedia.org/wiki/Codd%27s_12_rules" \t "_blank)**[:](https://en.wikipedia.org/wiki/Codd%27s_12_rules" \t "_blank)

[If a system provides a low-level (record-at-a-time) interface, it must not be able to bypass the integrity rules and constraints expressed in the higher-level relational language](https://en.wikipedia.org/wiki/Codd%27s_12_rules" \t "_blank).

1. What do you understand By Data Redundancy?

* Data Redundancy refers to the situation where the same pieces of data are stored in multiple places within a database or data storage system.
* This can happen intentionally or accidentally.
* Redundancy can be useful for data recovery in case of corruption or loss.
* In computer memory and storage, data redundancy allows for error correction.

1. What is DDL Interpreter?

* A **DDL Interpreter** is a component within a Database Management System (DBMS) that processes Data Definition Language (DDL) statements.
* These statements (like **CREATE**, **ALTER** and **DELETE**) are used to define and modify the structure of database objects like tables, indexes, and schemas.
* [The DDL Interpreter translates these statements into a set of tables containing metadata, which is stored in the data dictionary](https://www.geeksforgeeks.org/structure-of-database-management-system/).

1. What is DML Compiler in SQL?

* [The **DML Compiler** processes Data Manipulation Language (DML) statements into low-level instructions (machine language) that can be executed by the database system](https://www.geeksforgeeks.org/structure-of-database-management-system/).
* In SQL, DML includes commands like **SELECT**, **INSERT**, **UPDATE**, and **DELETE**, which allow you to manipulate data within tables.

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

* **NOT NULL Constraint** :
* Ensures that a column cannot store NULL values.
* Example :

**CREATE TABLE** Colleges (

college\_id INT ***NOT NULL***,

college\_code VARCHAR(20) ***NOT NULL***,

college\_name VARCHAR(50)

);

* **UNIQUE Constraint** :
* Requires that the values in a column must be unique.
* That is, the values in any row of a column must not be repeated.
* Example :

**CREATE TABLE** Colleges (

college\_id INT NOT NULL ***UNIQUE***,

college\_code VARCHAR(20) ***UNIQUE***,

college\_name VARCHAR(50)

);

* **PRIMARY KEY Constraint** :
* Combines NOT NULL and UNIQUE constraints to uniquely identify rows.
* A primary key is a field which can uniquely identify each row in a table And this constraint is used to specify a field in a table as primary key.
* Example :

**CREATE TABLE** Colleges (

college\_id INT ***PRIMARY KEY***,

college\_code VARCHAR(20) NOT NULL,

college\_name VARCHAR(50)

);

* **FOREIGN KEY Constraint** :
* References a record in another table.
* A Foreign key is a field which can uniquely identify each row in a another table. And this constraint is used to specify a field as Foreign key.
* Example :

**CREATE TABLE** Orders (

order\_id INT PRIMARY KEY,

customer\_id INT,

***FOREIGN KEY customer\_id REFERENCES Customers(id)***

);

* **CHECK Constraint** :
* Validates conditions before allowing values in a table.
* This constraint helps to validate the values of a column to meet a particular condition. That is, it helps to ensure that the value stored in a column meets a specific condition.
* Example :

**CREATE TABLE** Orders (

order\_id INT PRIMARY KEY,

amount INT ***CHECK (amount >= 100)***

);

* **DEFAULT Constraint** :
* Sets a default value if NULL is stored in a column.
* This constraint specifies a default value for the column when no value is specified by the user.
* Example :

**CREATE TABLE** College (

college\_id INT PRIMARY KEY,

college\_code VARCHAR(20),

***college\_country VARCHAR(20) DEFAULT 'US'***

);

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

* SAVEPOINT is a mechanism that allows you to create a point within a transaction to which you can later roll back.
* [This feature is particularly useful when you want to implement a partial rollback in case of errors or other exceptional conditions within a transaction](https://www.sqltutorial.net/savepoint.html).
* Creation and of savepoint and use it:
* Creating a Savepoint :

To create a savepoint, use the SAVEPOINT statement followed by a name for the savepoint. You can choose any valid identifier as the savepoint name.

For example:

**SAVEPOINT** my\_savepoint;

* **Performing SQL Operations :**

After creating the savepoint, you can perform various SQL operations within the transaction.

* **Rolling Back to the Savepoint :**

If an error occurs or you need to revert to a known good state, you can use the ROLLBACK TO statement to undo changes made after the savepoint. For example:

**ROLLBACK** **TO** my\_savepoint;

* **Continuing or Committing the Transaction :**

After rolling back to the savepoint, you can continue with the transaction or commit it using COMMIT.

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

* A trigger in SQL is a piece of code that automatically executes in response to specific events occurring on a particular table within the database.
* Here are the key points about triggers:

**Event Types for Triggers:**

* + Triggers can be invoked either before or after certain events:
    - INSERT: When a new row is inserted into the table.
    - UPDATE: When an existing row is modified.
    - DELETE: When a row is deleted.
  + Some databases also invoke triggers for other statements, like LOAD DATA INFILE in MySQL.

**Trigger Creation Syntax:**

* + To create a trigger, use the following statement:

***CREATE TRIGGER*** trigger\_name ***[BEFORE | AFTER]*** event **ON** table\_name

**FOR EACH ROW**

BEGIN

-- trigger\_logic

END;