Basics of Database

Q. 1. What do you understand By Database.

Database is a collection of raw data which stores the individual data in meaningful fact. It is basically a shared collection of interrelated data tables which are designed from varied information from organisation. On the other hand, DBMS is a collection of programmes that enables users for creating, manipulating, updating, deleting various entries in database.

Q. 2. What is Normalization?

Normalization is the process of minimizing redundancy (duplicity) from a relation or set of relations. Redundancy in relation may cause insertion, deletion and updating anomalies. So, it helps to minimize the redundancy in relations.

Most used normal forms:

First normal form(1NF)

Second normal form(2NF)

Third normal form(3NF)

Boyce & Codd normal form (BCNF)

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

DBMS stands for Database Management System, and RDBMS is the acronym for the Relational Database Management system. In DBMS, the data is stored as a file, whereas in RDBMS, data is stored in the form of tables. To know what is the difference between RDBMS and DBMS, check out the table below.

Difference between RDBMS and DBMS

RDBMS	DBMS
Data stored is in table format	Data stored is in the file format
Multiple data elements are accessible together	Individual access of data elements
Data in the form of a table are linked together	No connection between data
Normalisation is not achievable	There is normalisation
Support distributed database	No support for distributed database
Data is stored in a large amount	Data stored is a small quantity
Here, redundancy of data is reduced with the help of key and indexes in RDBMS	Data redundancy is common

RDBMS supports multiple users	DBMS supports a single user
It features multiple layers of security while handling data	There is only low security while handling data
The software and hardware requirements are higher	The software and hardware requirements are low
Oracle, SQL Server.	XML, Microsoft Access.

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

Rule 1: The Information Rule

All information, whether it is user information or metadata, that is stored in a database must be entered as a value in a cell of a table. It is said that everything within the database is organized in a table layout.

Rule 2: The Guaranteed Access Rule

Each data element is guaranteed to be accessible logically with a combination of the table name, primary key (row value), and attribute name (column value).

Rule 3: Systematic Treatment of NULL Values

Every Null value in a database must be given a systematic and uniform treatment.

Rule 4: Active Online Catalog Rule

The database catalog, which contains metadata about the database, must be stored and accessed using the same relational database management system.

Rule 5: The Comprehensive Data Sublanguage Rule

A crucial component of any efficient database system is its ability to offer an easily understandable data manipulation language (DML) that facilitates defining, querying, and modifying information within the database.

Rule 6: The View Updating Rule

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

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

A successful database system must possess the feature of facilitating high-level insertions, updates, and deletions that can grant users the ability to conduct these operations with ease through a single query.

Rule 8: Physical Data Independence

Application programs and activities should remain unaffected when changes are made to the physical storage structures or methods.

Rule 9: Logical Data Independence

Application programs and activities should remain unaffected when changes are made to the logical structure of the data, such as adding or modifying tables.

Rule 10: Integrity Independence

Integrity constraints should be specified separately from application programs and stored in the catalog. They should be automatically enforced by the database system.

Rule 11: Distribution Independence

The distribution of data across multiple locations should be invisible to users, and the database system should handle the distribution transparently.

Rule 12: Non-Subversion Rule

If the interface of the system is providing access to low-level records, then the interface must not be able to damage the system and bypass security and integrity constraints.

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

Data redundancy is a condition created within a database or data storage technology in which the same piece of data is held in two separate places.

This can mean two different fields within a single database, or two different spots in multiple software environments or platforms. Whenever data is repeated, it basically constitutes data redundancy.

Data redundancy can occur by accident but is also done deliberately for backup and recovery purposes.

Q. 6. What is DDL Interpreter?

The DDL Interpreter is responsible for processing DDL statements.

Its primary tasks include:

Interpreting DDL Statements: When DDL statements (e.g., CREATE TABLE, ALTER TABLE, DROP INDEX) are executed, the DDL Interpreter interprets them.

Creating Metadata Tables: It generates a set of tables containing metadata (data about data).

Data Dictionary: These metadata tables serve as a data dictionary, maintaining information about the database objects, their attributes, constraints, and relationships.

Storing Schema Definitions: The DDL Interpreter records the schema definitions (e.g., table names, column names, data types) in the data dictionary.

Maintaining Consistency: It ensures that the database remains consistent by enforcing rules and constraints defined in the DDL statements.

Q. 7. What is DML Compiler in SQL?

The DML Compiler is responsible for processing DML statements.

Its primary tasks include:

Interpreting DML Statements: When DML statements (e.g., SELECT, INSERT, UPDATE) are executed, the DML Compiler interprets them.

Query Execution: It compiles and executes queries against the database.

Data Retrieval: The DML Compiler retrieves data from the database based on the specified conditions.

Data Modification: It handles data insertion, modification, and deletion.

Transaction Handling: The DML Compiler ensures that changes made by DML statements are consistent and durable.

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

NOT NULL Constraint:

The NOT NULL constraint ensures that a column cannot store NULL values.

Example:

```
CREATE TABLE Students (
   student_id INT NOT NULL,
   first_name VARCHAR(50) NOT NULL,
   last_name VARCHAR(50) NOT NULL
);
```

Here, both student_id and first_name columns won't allow NULL values.

UNIQUE Constraint:

The UNIQUE constraint ensures that a column must have unique values.

Example:

```
CREATE TABLE Employees (
   employee_id INT NOT NULL UNIQUE,
   employee_code VARCHAR(20) UNIQUE,
   employee_name VARCHAR(50)
);
```

The employee_code column must have unique values.

PRIMARY KEY Constraint:

The PRIMARY KEY constraint uniquely identifies a row in a table.

Example:

```
CREATE TABLE Customers (
    customer_id INT PRIMARY KEY,
    customer_name VARCHAR(100),
    email VARCHAR(100) UNIQUE
);
```

The customer_id serves as a unique identifier for each row.

FOREIGN KEY Constraint:

The FOREIGN KEY constraint references a row in another table.

Example:

```
CREATE TABLE Orders (
    order_id INT PRIMARY KEY,
    customer_id INT REFERENCES Customers(customer_id),
    order_total DECIMAL(10, 2)
);
```

The customer_id in the Orders table references the customer_id in the Customers table.

CHECK Constraint:

The CHECK constraint validates a condition before allowing values in a table.

Example:

```
CREATE TABLE Products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    price DECIMAL(8, 2) CHECK (price >= 0)
);
```

The price must be greater than or equal to 0.

DEFAULT Constraint:

The DEFAULT constraint sets a default value if NULL is stored in a column.

Example:

```
CREATE TABLE Orders (
    order_id INT PRIMARY KEY,
    order_date DATE DEFAULT CURRENT_DATE,
    total_amount DECIMAL(10, 2)
);
```

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

A savepoint in SQL is a marker within a transaction that allows you to roll back only part of the transaction instead of the entire transaction. It's like creating a bookmark within a transaction, allowing you to return to that specific point if needed.

Here's how you can create a savepoint and use it in SQL (using MySQL as an example):

1. Creating a Savepoint:

To initiate a transaction and create a savepoint, use the START TRANSACTION command:

START TRANSACTION;

SAVEPOINT my_savepoint;

In this example, we've named our savepoint as my_savepoint.

2. Inserting Data:

Let's say we insert a new record into a table:

INSERT INTO my_table (column1, column2) VALUES (value1, value2);

3. Rolling Back to the Savepoint:

If something goes wrong and you want to undo changes made after the savepoint, use the ROLLBACK TO command:

ROLLBACK TO my_savepoint;

This will revert changes made after the my_savepoint.

4. Committing or Rolling Back the Entire Transaction:

To commit all changes made since the savepoint, use the COMMIT command:

COMMIT;

To discard all changes made since the savepoint and end the transaction, use the ROLLBACK command:

ROLLBACK;

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

A trigger in SQL is a piece of code that executes automatically in response to a specific event occurring on a table in the database. Triggers allow you to perform actions automatically when certain events (such as INSERT, UPDATE, or DELETE) occur. Here are the key points about triggers:

1. Types of Triggers:

- AFTER INSERT: Activated after data is inserted into the table.
- AFTER UPDATE: Activated after data in the table is modified.
- AFTER DELETE: Activated after data is deleted/removed from the table.
- BEFORE INSERT: Activated before data is inserted into the table.
- BEFORE UPDATE: Activated before data in the table is modified.
- BEFORE DELETE: Activated before data is deleted/removed from the table.

2. Virtual Tables for Triggers:

- SQL Server provides two virtual tables specifically for triggers:
  - INSERTED: Holds rows to be inserted.
  - DELETED: Holds rows to be deleted.

3. Example Scenarios:

- Let's look at some examples of creating triggers:

1. Age Validation Trigger:

- Suppose we want to ensure that no employee younger than 25 can be inserted into the database. We can create a trigger named Check_age:

```
CREATE TRIGGER Check_age

BEFORE INSERT ON employee

FOR EACH ROW

BEGIN

  IF NEW.age < 25 THEN

    SIGNAL SQLSTATE '45000'

    SET MESSAGE_TEXT = 'ERROR: AGE MUST BE AT LEAST 25 YEARS!';

  END IF;

END;
```

This trigger checks the age attribute before inserting a tuple into the employee table.

2. Backup Trigger:

- Let's create a backup table (employee_backup) to hold values of employees who are no longer part of the institution:

```
CREATE TABLE employee_backup (

  employee_no INT,

  employee_name VARCHAR(40),

  job VARCHAR(40),

  hiredate DATE,

  salary INT,
```

```
    PRIMARY KEY (employee_no)
);
CREATE TRIGGER Backup
BEFORE DELETE ON employee
FOR EACH ROW
BEGIN
    INSERT INTO employee_backup
    VALUES (OLD.employee_no, OLD.name, OLD.job, OLD.hiredate, OLD.salary);
END;
```

3. Counting Tuples Trigger:

- To count the number of new tuples inserted using each INSERT statement:

```
DECLARE count INT;
SET count = 0;
CREATE TRIGGER Count_tuples
AFTER INSERT ON employee
FOR EACH ROW
BEGIN
    SET count = count + 1;
END;
```

This trigger keeps track of the number of new tuples in the employee table.