**DATABASE THEORY&Assigment**

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

**Ans.** A database is an electronically stored, systematic collection of data.

**2. What is Normalization?**

**Ans.** **1. First Normal Form (1NF):**

* Ensures that each table column contains atomic (indivisible) values and that each column contains values of a single type.
* Ensures that each row in the table is unique.

**2. Second Normal Form (2NF):**

* Builds on 1NF by ensuring that all non-key attributes are fully functionally dependent on the primary key.
* This means that there are no partial dependencies; that is, no non-key attribute is dependent on a part of a composite primary key.

**3. Third Normal Form (3NF):**

* Builds on 2NF by removing transitive dependencies.
* A transitive dependency occurs when a non-key attribute depends on another non-key attribute rather than directly
* depending on the primary key.

**4. Boyce-Codd Normal Form (BCNF):**

* A stricter version of 3NF, ensuring that every determinant (a column or set of columns that uniquely determines another column) is a candidate key.

**5. Fourth Normal Form (4NF):**

* Addresses multi-valued dependencies. It ensures that a record does not contain two or more independent multi-valued facts about an entity.

**6. Fifth Normal Form (5NF):**

* Deals with cases where information can be reconstructed from smaller pieces of information that are independent in structure.

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

**Ans. 1. Data Structure:**

* DBMS: A DBMS can manage data in various forms, such as hierarchical, network, object-oriented, and more. It doesn't necessarily have to use a tabular format. The data can be stored in files or other non-tabular formats.
* RDBMS: An RDBMS specifically manages data using a relational model, where data is organized into tables (relations) with rows and columns. Each table has a unique key, and relationships between tables are established through foreign keys.

**2. Data Integrity and ACID Properties:**

* DBMS: While a DBMS may offer some level of data integrity and consistency, it may not fully support ACID (Atomicity, Consistency, Isolation, Durability) properties, which are crucial for ensuring reliable transaction processing.
* RDBMS: RDBMSs are designed to fully support ACID properties, ensuring that all transactions are processed reliably and that data remains consistent even in the event of a failure.

**3. Data Relationships:**

* DBMS: Data relationships can be defined, but they are not as rigorously enforced as in an RDBMS. Relationships can be established manually by the application logic.
* RDBMS: An RDBMS inherently supports and enforces data relationships through primary keys, foreign keys, and referential integrity constraints.

**4. Data Manipulation Language:**

* DBMS: DBMSs may use a variety of query languages or custom APIs to interact with the data, and these may not be standardized.
* RDBMS: RDBMSs use Structured Query Language (SQL) as a standard language for defining, querying, and manipulating the data.

**5. Normalization:**

* DBMS: Normalization is not a mandatory aspect of data design in DBMSs, which can lead to data redundancy.
* RDBMS: Normalization is a core principle in RDBMS design, aimed at reducing data redundancy and improving data integrity.

**4. What is MF code rule of RDBMS Systems?**

**Ans.**

1. **Guaranteed Access Rule:** Every data item should be accessible via a combination of a table name, primary key, and column name.
2. **Systematic Treatment of Null Values:** The system must support null values to represent missing or inapplicable information.
3. **Dynamic Online Catalog Based on the Relational Model:** Metadata should be stored in the same way as regular data, allowing users to query it using SQL.
4. **Comprehensive Data Sublanguage Rule:** The database must support a comprehensive language that handles data definition, manipulation, and transactions.
5. **View Updating Rule:** Views must be updatable if theoretically possible.
6. **High-Level Insert, Update, and Delete:** The system should support set-based operations for insert, update, and delete.
7. **Physical Data Independence:** Changes to the physical storage of data should not require changes to the application.
8. **Logical Data Independence:** Changes to the logical schema (e.g., tables, columns) should not affect applications.
9. **Integrity Independence:** Integrity constraints must be stored in the catalog and not in the application programs.
10. **Distribution Independence:** The end-user should not be aware of whether the data is distributed across multiple locations.
11. **Non-Subversion Rule:** If there is a way to access the data without using the RDBMS's query language, it should not be able to bypass the integrity rules.

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

**Ans.** **Increased Storage Costs:** Storing the same data multiple times can lead to wasted storage space, which can be costly, especially with large datasets.

**Data Inconsistency:** When data is redundant, there is a risk of inconsistencies arising if one copy of the data is updated while another is not. This can lead to incorrect or outdated information being presented to users.

**Maintenance Challenges:** Redundant data requires additional effort to maintain, as updates need to be propagated to all copies of the data. This can make database management more complex and error-prone.

**Potential for Anomalies:** Data redundancy can lead to anomalies, such as update anomalies (where changes in data require updates in multiple places), insert anomalies (difficulty in adding new data due to missing information), and delete anomalies (removal of data that might accidentally result in losing other important information).

**6. What is DDL Interpreter?**

**Ans. Parsing and Validation:** The DDL Interpreter first parses the DDL commands to ensure they are syntactically correct. It also validates the commands against the database schema and other constraints to ensure they are semantically correct.

**Execution:** Once the DDL commands are parsed and validated, the DDL Interpreter executes them by modifying the database's metadata. This involves creating, altering, or deleting tables, columns, indexes, and other database objects as specified**.**

**Metadata Management:** The DDL Interpreter updates the system catalog or data dictionary, which is a special set of tables that store metadata about the database objects. This includes information about the structure and constraints of the database.

**Transaction Management:** Some database systems support transactional DDL, where changes made by DDL commands can be committed or rolled back. The DDL Interpreter handles these transactions to ensure atomicity and consistency.

**7. What is DML Compiler in SQL?**

**Ans. Parsing:** The DML Compiler first parses the DML statement, breaking it down into its constituent parts such as keywords, table names, column names, and values. This step involves checking the syntax of the SQL statement to ensure it conforms to the language's grammar rules.

1. **Semantic Analysis:** After parsing, the DML Compiler performs semantic analysis to verify that the SQL statement is meaningful in the context of the database schema. This includes:
   * Checking that the specified tables and columns exist.
   * Ensuring data types are compatible.
   * Verifying that the statement respects integrity constraints (e.g., foreign key constraints).
   * Ensuring that the user has the necessary permissions to execute the command.
2. **Query Optimization:** One of the key functions of the DML Compiler is to optimize the query for efficient execution. This involves:
   * Analyzing different ways to execute the query.
   * Selecting the most efficient execution plan, which may involve choosing the best indexes, join strategies, and access paths.
   * Reordering operations to minimize the computational cost, such as reducing the number of I/O operations.
3. **Code Generation:** The DML Compiler translates the optimized query plan into a set of low-level instructions that the database engine can execute. This executable code is tailored to the database's internal architecture and is designed to perform the data manipulation operations as efficiently as possible.
4. **Execution:** While the DML Compiler itself does not execute the code, it prepares the execution plan. The compiled code is then handed over to the database engine's execution component, which carries out the operations on the database.

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

**Ans.** **Primary Key Constraint:** Ensures that each row in a table is uniquely identifiable. A primary key cannot contain NULL values and must contain unique values.

**Foreign Key Constraint:** Ensures that the value in a column or a group of columns matches values in a column of another table, enforcing referential integrity between tables.

**Unique Constraint:** Ensures that all values in a column or a group of columns are unique across all rows in the table.

**Check Constraint:** Ensures that all values in a column satisfy a specific condition.

**Not Null Constraint:** Ensures that a column cannot have NULL values.

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

**Ans. Transaction Management:** Savepoints are used within transactions, which are sequences of SQL statements that are executed as a single unit of work. Transactions ensure that operations are atomic, consistent, isolated, and durable (ACID properties).

**Partial Rollback:** By creating savepoints, you can roll back to a specific point within a transaction, undoing only certain parts of the transaction instead of rolling back the entire transaction.

**Nested Transactions:** Savepoints can be considered a form of nested transactions, where you can revert to an earlier state within the main transaction.

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

**Ans. Automatic Execution:** Triggers are executed automatically when a specified event occurs on the associated table or view.

**Event-Driven:** They respond to events such as data modifications (INSERT, UPDATE, DELETE).

**Timing:** Triggers can be set to execute **BEFORE or AFTER** the event.

* **BEFORE Trigger:** Executes before the event occurs, allowing modifications or validations before the actual data change.
* **AFTER Trigger:** Executes after the event occurs, allowing actions to be taken once the data has been changed.

**Row-Level vs. Statement-Level:**

* **Row-Level Trigger:** Executes once for each row affected by the event.
* **Statement-Level Trigger:** Executes once per SQL statement, regardless of the number of rows affected.

1) create database st77;

CREATE TABLE H45(Rollno int PRIMARY key AUTO\_INCREMENT not null

name varchar(20),branch varchar(20) );

insert into H45(name , branch) values

(“jay”, computer sci )

(“suhani”, Elctronic and com)

(“kirit”, Elctronic and com)

CREATE TABLE E21(ERollno int, S\_code varchar(20), Marks int, P\_code varchar(20),

FOREIGN KEY (ERollno) REFERENCES h45(Rollno) );

INSERT INTO E21(Erollno, S\_code ,Marks, P\_code)VALUES

(1,"CS11",50,"CS"),

(1,"CS12",60,"CS"),

(2,"EC101",66,"EC"),

(2,"EC102",70,"EC"),

(3,"EC101",45,"EC"),

(3,"EC102",50,"EC");

2) create database employee;

Create table emp(Employee id int primary key auto increment not null,

First\_name varchar(20),

last\_name varchar(20),

salary bigint , joining\_date varchar ,department varchar(20) );

insert into emp(First\_name, last\_name , salary,joining\_date, department)VALUES

("johan", "Abrahan" , 10000000 ,"01-JAN-13 12.00.00 AM","banking" ),

("Mitchal", "clarke", 8000000 ,"01-JAN-13 12.00.00 AM","insurance" ),

("Roy", "Thomes", 7000000 ,"01-FEB-13 12.00.00 AM","banking" ),

("Tom" , "jose" ,6000000,"01-FEB-13 12.00.00 AM","insurance"),

("jerry" , "pinto" , 650000 ,"01-FEB-13 12.00.00 AM", "insurance"),

("Philip" , "Mathew" , 750000 ,"01-JAN-13 12.00.00 AM", "service"),

("test name 1" ," 123" , 650000 ,"01-JAN-13 12.00.00 AM", "service"),

("teat name 2", "lname % ", 600000 ,"01-FEB-13 12.00.00 AM", "insurance");

CREATE TABLE Incentive(Employee\_refer\_id int, Incentive\_date int ,Incentive\_amount bigint);

Insert into Incentive(Employee\_refer\_id , Incentive\_date ,Incentive\_amount );

(1,01-feb-13,5000),

(2,01-feb-13,3000),

(3,01-feb-13,4000),

(1,01-feb-13,4500),

(2,01-feb-13,3500);

4) SELECT FIRST\_NAME, Joining\_Date, Salary FROM employee;

5) SELECT \* FROM emp ORDER BY First\_Name ASC, Salary DESC;

6) Select\*From emp Where First\_name like 'j%';

7) SELECT emp MAX(salary) AS max\_salary FROM employee;

8) SELECT\*From emp order by Asc;

9) SELECT e.first\_name, i.incentive\_amount FROM emp JOIN incentives ON e.employee\_id = i.employee\_id WHERE i.incentive\_amount > 3000;

10)

DELIMETER$$

BEGIN

CREATE TABLE emp\_trigger (tid int PRIMARY KEY AUTO\_INCRIMENT,

TFirst\_name varchar(20), Tlast\_name varchar(20), Tsalary varchar(50), Tjoinigdate varchar(20),

Tdeparment varchar(20), Action\_perform varchar(60) )

End;

CREATE TRIGGER trigger01 BEFORE DELETE ON emp for EACH ROW

INSERT INTO emp\_trigger(tid, tFirst\_name, t\_lastname, tsalry , tprice, t\_department ,Action\_perform) VALUES

(old.id ,old.firstname, old.lastname , old.salary ,old.price ,old.price , old department)

3) CREATE DATABASE h777;

CREATE TABLE Salesperson(pk\_snno int, Sname varchar(20), city varchar(20),

Comm int);

Insert Into Salesperson(pk\_sunno,sname,city,Comm) VALUES

(1001,"Peel","London",.12),

(1002,"Serrees","San jose",.13),

(1004,"Motika","london",.11),

(1007,"Rafkin","barcelona",.15),

(1003,"Axelrod","new york",.1);

CREATE TABLE Customer(pk\_Cnm int,cname varchar(20),city varchar(20),Rating bigint,Fk\_snno int);

INSERT INTO customer(pk\_Cnm,cname,city,Rating,Fk\_snno) VALUES

(201,"Hoffman","London",100,1001),

(202,"Giovanne","Roe",200,1003),

(203,"Liu","San jose",300,1002),

(204,"Grass","Barcelona",100,1002),

(206,"Clemens","London",300,1003),

(207,"Pereira","Roe",100,1004);

13) [SELECT](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/select.html) cname FROM customer WHERE rating > 100;

14) SELECT name, city FROM salesperson WHERE city = 'London' AND commission > 0.12;

15) SELECT name, city FROM salesperson WHERE city = 'Barcelona' OR city = 'London';

16) SELECT name, city, commission FROM salespeople WHERE commission > 0.10 AND

commission < 0.12;

17) SELECT customer\_name, city, rating FROM customers WHERE rating > 100 OR (city = 'Rome' AND rating <= 100);

18) CREATE DATABASE Salespeople;

CREATE TABLE sales5(salesman\_id int,name varchar(20),city varchar(20),commission float);

INSERT INTO sales5(salesman\_id,name,city,commission) VALUES

(5001,"James Hoog","New york",0.15),

(5002,"Nail knite","Paries",0.13),

(5005,"Pit Alex","London",0.11),

(5006,"Mc Lyon","paris",0.14),

(5007,"Paul Adam","Rome",0.13),

(5007,"Lauson Hen","San Jose",0.12);

19) CREATE DATABASE Salesperson;

CREATE TABLE order8(ord\_no int, purch\_amt float, ord\_date bigint, customer\_id int, salesman\_id int);

INSERT INTO order8(ord\_no,purch\_amt,ord\_date,customer\_id,salesman\_id) VALUES

(70001,150.5,2012-10-5,3005,5002),

(70009,270.65, 2012-09-10,3001,5005),

(70002,65.26, 2012-10-5,3002,5001),

(70004,110.5, 2012-08-17,3009,5003),

(70007,948.5, 2012-09-10,3005,5002),

(70005,2400.6, 2012-07-27,3007,5001),

(70008,5760, 2012-09-10,3002,5001),

(70010,1983.43, 2012-10-10,3004,5006),

(70003,2480.4, 2012-10-10,3009,5003),

(70012,250.45, 2012-06-27,3008,5002),

(70011,75.29, 2012-08-17,3003,5007),

(70013,3045.6, 2012-04-25,3002,5001);

SELECT \* FROM orders WHERE salesman\_id = 5001;

SELECT ord\_no, ord\_date, purch\_amt FROM orders WHERE salesman\_id = 5001;

21) CREATE DATABASE PROUDUCT;

CREATE TABLE P11(PRO\_ID int primary key auto\_incriment not null,

PRO\_NAME varchar(20),PRO\_PRICE int,PRO\_COM int);

INSERT INTO p11(pro\_id,pro\_name,pro\_price,pro\_com)VALUES

(101,"Mother board","3200",15),

(102,"key board","450",16),

(103,"Zip drive","250",14),

(104,"Speaker","550",16),

(105,"Moniter","5000",11),

(106,"DVD drive","900",12),

(107,"CD drive","800",12),

(108,"Printer","2600",13),

(109,"Refeer catridge","350",13),

(110,"Mouse","250",12);

SELECT \* FROM p11 WHERE pro\_price <= 250;

SELECT \* FROM p11 ORDER BY pro\_name ASC;

SELECT \* FROM p11 ORDER BY pro\_price DESC;

SELECT AVG(pro\_price) FROM P11;

SELECT SUM(pro\_price) FROM p11;

SELECT AVG(8 \* pro\_price) AS average\_total FROM p11;