**SE - Database - Full stack and Back end**

**[DBMS]-**

DATABASE MANAGEMENT SYSTEM

**MODULE -:**5

**Submitted To:-**

**Jay Prajapati Sir**

**Submitted By:-**

**Ankita Sahu**

MODULE-: 5 (Database)

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

**Ans:-** A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a Database Management System (DBMS).

**2. What is Normalization?**

**Ans:-** Normalization is the process of minimizing redundancy (duplicity) from a relation or set of relations.

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

**Ans:-** To know what is the difference between RDBMS and DBMS, check out the table below :

|  |  |
| --- | --- |
| **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 normalization. |
| 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. |

**4. What is EF Codd Rule of RDBMS Systems?**

### Ans:- Dr Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database.

### Rules:-

## Rule 1: Information Rule

The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

## Rule 2: Guaranteed Access Rule

Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

## Rule 3: Systematic Treatment of NULL Values

The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following − data is missing, data is not known, or data is not applicable.

## Rule 4: Active Online Catalog

The structure description of the entire database must be stored in an online catalog, known as **data dictionary**, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

## Rule 5: Comprehensive Data Sub-Language Rule

A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

## Rule 6: View Updating Rule

All the views of a database, which can theoretically be updated, must also be updatable by the system.

## Rule 7: High-Level Insert, Update, and Delete Rule

A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

## Rule 8: Physical Data Independence

The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

## Rule 9: Logical Data Independence

The logical data in a database must be independent of its user’s view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.

## Rule 10: Integrity Independence

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

## Rule 11: Distribution Independence

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

## Rule 12: Non-Subversion Rule

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

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

**Ans:-** Data redundancy occurs when the same piece of data exists in multiple places, whereas data inconsistency is when the same data exists in different formats in multiple tables. Unfortunately, data redundancy can cause data inconsistency, which can provide a company with unreliable and/or meaningless information.

**6.** **What is DDL Interpreter?**

**Ans:-** Data Definition Language (DDL) is used to create and modify the structure of [objects](https://www.techtarget.com/searchapparchitecture/definition/object) in a [database](https://www.techtarget.com/searchdatamanagement/definition/database) using predefined [commands](https://www.techtarget.com/searchwindowsserver/definition/command) and a specific [syntax](https://www.techtarget.com/whatis/definition/syntax). These database objects include [tables](https://www.techtarget.com/whatis/definition/table), sequences, locations, [aliases](https://www.techtarget.com/whatis/definition/alias), [schemas](https://www.techtarget.com/searchdatamanagement/definition/schema) and indexes.  Interprets the DDL statements such as schema definition statements like CREATE, RENAME, ALTER.

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

**Ans:-** A DML (data manipulation language) refers to a computer programming language that allows you to insert (INSERT), delete (DELETE) and update (UPDATE) data in a database.

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

**Ans:-** Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

The following constraints are commonly used in SQL:

* [NOT NULL](https://www.w3schools.com/sql/sql_notnull.asp) - Ensures that a column cannot have a NULL value.
* [UNIQUE](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different.
* [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE Uniquely identifies each row in a table.
* [FOREIGN KEY](https://www.w3schools.com/sql/sql_foreignkey.asp) - Prevents actions that would destroy links between tables.
* [CHECK](https://www.w3schools.com/sql/sql_check.asp) - Ensures that the values in a column satisfies a specific condition.
* [DEFAULT](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column if no value is specified.

**Exapmle:**

Create Table table\_name (

Column 1 datatype constraint,

Column 1 datatype constraint,

Column 1 datatype constraint,

**……………**

);

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

**Ans:-** A **SAVEPOINT** is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.

The syntax for a SAVEPOINT command is as shown below:

SAVEPOINT SAVEPOINT\_NAME;

This command serves only in the creation of a SAVEPOINT among all the transactional statements. The ROLLBACK command is used to undo a group of transactions.

Query:

**CREATE** **TABLE** student (ID INT, Name VARCHAR(20), Percentage INT, Location VARCHAR(20), DateOfBirth DATE);

**INSERT** **INTO** student(ID, Name, Percentage, Location, DateOfBirth) **VALUES**(1, "Manthan Koli", 79, "Delhi", "2003-08-20"), (2, "Dev Dixit", 75, "Pune", "1999-06-17"),  (3, "Aakash Deshmukh", 87, "Mumbai", "1997-09-12"),  (4, "Aaryan Jaiswal", 90, "Chennai", "2005-10-02");

START **TRANSACTION**;

SAVEPOINT ini;

**INSERT** **INTO** student **VALUES** (10, "Saurabh Singh", 54, "Kashmir", "1989-01-06");

SAVEPOINT ins;

**UPDATE** student **SET** Name = "Mahesh Kuwar" **WHERE** ID =1;

 SAVEPOINT upd;

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

**Ans:-**

A **trigger** is a stored procedure in database which automatically invokes whenever a special event in the database occurs.

For example, a trigger can be invoked when a row is inserted into a specified table.

How to create a Trigger in SQL:-

**Create Trigger** **[**trigger-name**]**

**[**Before**/**After**]**

**{**INSERT **|** UPDATE **|** DELETE**}**

ON **[**table\_name**] [**For each row**]**

**[**trigger\_body**];**

Task-

**1. Create Table Name : Student and Exam**

**Ans:-**

**Student:-**

CREATE TABLE student(Rollno int(11) PRIMARY KEY,Name varchar(50),Branch varchar(50));

INSERT INTO student (Rollno,Name,Branch) VALUES

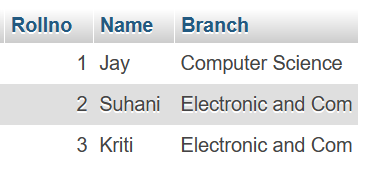
(1. 'Jay', 'Computer Science'),

(2. 'Suhani', 'Electronic and Com'),

(3. 'Kriti', 'Electronic and Com');

SELECT \* FROM 'student';

**Primary Key**

****

**Exam:-**

CREATE TABLE exam (Rollno int(11), S\_code varchar(255), Marks int(11), P\_code varchar(50),

FOREIGN KEY (Rollno) REFERENCES student (Rollno) ON DELETE CASCADE ON UPDATE CASCADE);

INSERT INTO exam (Roll no,S\_code,Marks,P\_code) VALUES

(1, 'CS11', 50, 'CS'),

(1, 'CS12', 60, 'CS'),

(2, 'EC101', 66, 'EC'),

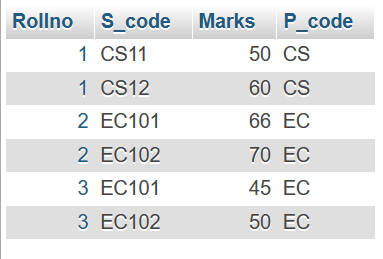
(2, 'EC102', 70, 'EC'),

(3, 'EC101', 45, 'EC'),

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

SELECT \* FROM ' exam';

Foreign Key



**2. Create table given below**

CREATE TABLE info (

Firstname varchar(50),

Lastname varchar(50),

Address varchar(50),

City varchar(50),

Age int(11));

INSERT Into info (FirstName,Lastname,Address,City,Age) VALUES

('Mickey','Mouse','123 Fantasy Way','Anaheim',73),

('Bat','Man','321 Cavern Ave','Gotham',54),

('Wonder','Women','987 Truth Way','Paradise',39),

('Donald','Duck','555 Quack Street','Mallard',65),

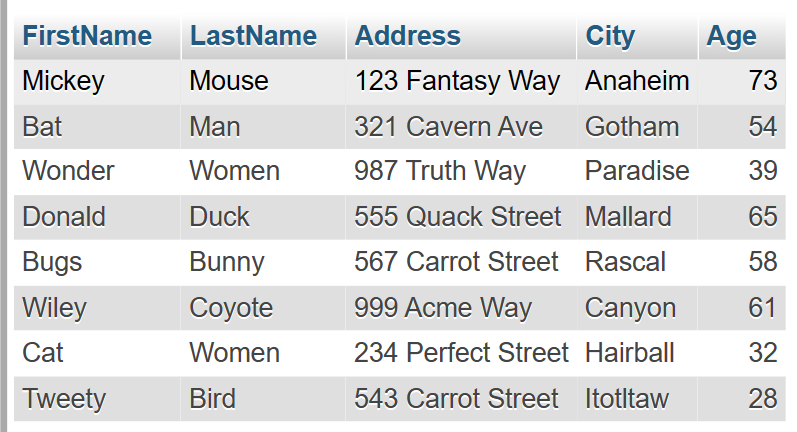
('Bugs','Bunny','567 Carrot Street','Rascal',58),

('Wiley','Coyote','999 Acme Way','Canyon',61),

('Cat','Women','234 Perfect Street','Hairball',32),

('Tweety','Bird','543 Carrot Street','Itotltaw',28);

SELECT \* FROM 'info';

****

**3. Create table given below: Employee and Incentive**

**Table Name : Employee:-**

CREATE TABLE employee(

Employee\_id int(11) PRIMARY KEY,

First\_name varchar(50),

Last\_name varchar(50),

Salary bigint(11),

Joining\_date varchar(50),

Department varchar(50));

INSERT INTO employee (Employee\_id, First\_name, Last\_name,

Salary, Joining\_date, Department) VALUES

(1,'John','Abraham',1000000,'01-Jan-13 12.00.00 AM','Banking'),

(2,'Michael','Clarke',800000,'01-Jan-13 12.00.00 AM','Insurance'),

(3,'Roy','Thomas',700000,'01-Feb-13 12.00.00 AM','Banking'),

(4,'Tom','Jose',600000,'01-Jan-13 12.00.00 AM','Insurance'),

(5,'Jerry','Pinto',650000,'01-Jan-13 12.00.00 AM','Insurance'),

(6,'Philip','Mathew',750000,'01-Jan-13 12.00.00 AM','Services'),

(7,'TestName1','123',650000,'01-Jan-13 12.00.00 AM','Services'),

(8,'TestName2','Lname%',600000,'01-Feb-13 12.00.00 AM',

'Insurance');

Select \* FROM 'employee';



**Table Name: Incentive:-**

CREATE TABLE incentive (Employee\_ref\_id int(11),Incentive\_date varchar(50),Incentive\_amount int(11),

FORIEGN KEY (Employee\_ref\_id) REFRENCES employee (Employee\_id) ON DELETE CASCADE ON UPDATE CASCADE);

INSERT INTO incentive (Employee\_ref\_id,Incentive\_date,Incentive\_amount) VALUES

(1, '01-Feb-13', 5000),

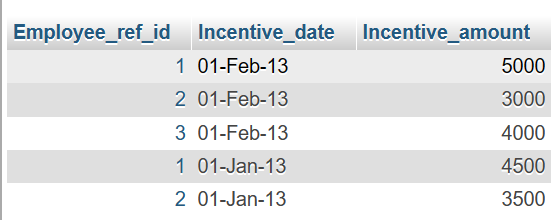
(2, '01-Feb-13', 3000),

(3, '01-Feb-13', 4000),

(1, '01-Jan-13', 4500),

(2, '01-Jan-13', 3500);

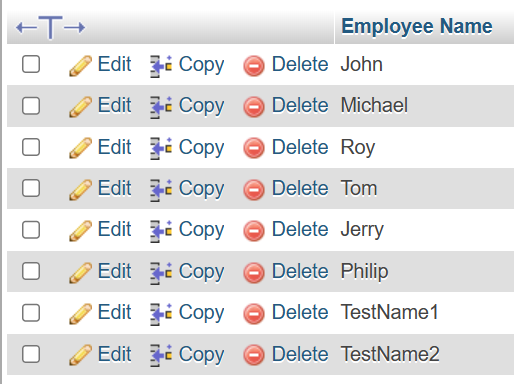
SELECT \* FROM incentive;



**a) Get First\_Name from employee table using Tom name “Employee Name”.**

**Query:-**

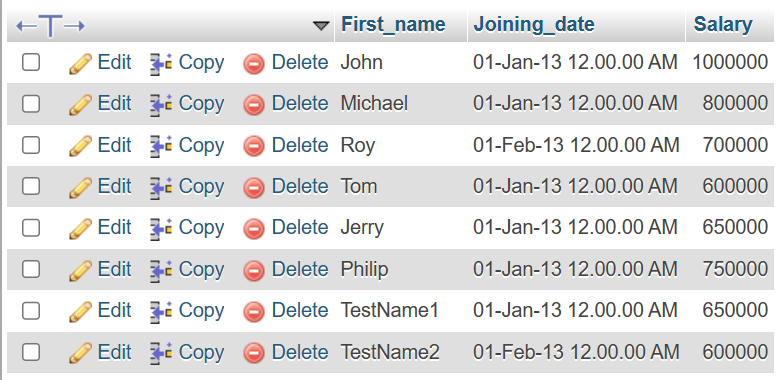
SELECT First\_name "Employee Name" FROM employee WHERE First\_name=Tom;



**b) Get FIRST\_NAME, Joining Date, and Salary from employee table.**

**Query:-**

SELECT First\_name,Joining\_date,Salary FROM employee;



**c) Get all employee details from the employee table order by First\_Name Ascending and Salary descending?**

**Query:-**

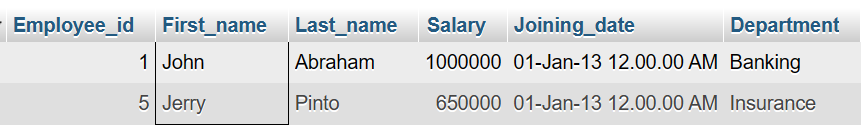
SELECT \* FROM employee ORDER BY First\_name ASC, Salary DESC;

****

**d) Get employee details from employee table whose first name contains ‘J’.**

**Query:-**

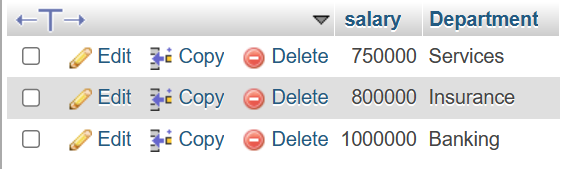
SELECT \* FROM employee WHERE First\_name LIKE '%j%';



**e) Get department wise maximum salary from employee table order by salary ascending?**

**Query:-**

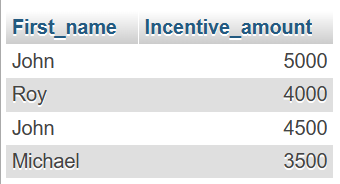
select max(Salary) as salary, Department from employee GROUP by Department ORDER BY MAX(Salary);

****

**f) Select first\_name, incentive amount from employee and incentives table for those employees who have incentives and incentive amount greater than 3000.**

**Query:-**

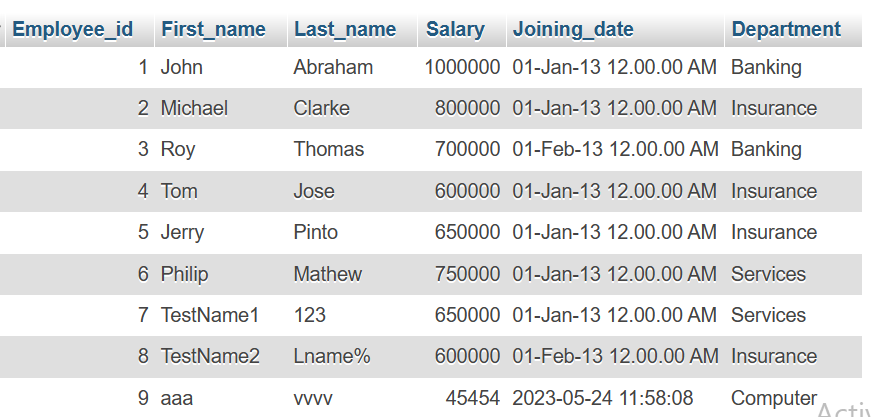
SELECT First\_name ,Incentive\_amount FROM employee A INNER JOIN incentive B ON A.Employee\_id=B.Employee\_ref\_id AND Incentive\_amount >3000;



**g) Create After Insert trigger on Employee table which insert records in view table.**

**Query:-**

CREATE TRIGGER 'after insert' AFTER INSERT ON 'employee' FOR EACH ROW INSERT INTO Views (id,first\_name,last\_name,salary,joining\_date,department) values (new.Employee\_id,new.First\_name,new.Last\_name,new.Salary,new.Joining\_date,new.Department);





**Table Name : Views:-**

****

**4. Create table given below: Salesperson and Customer**

**Table 1:**

**Table Name : Salesperson:-**

CREATE TABLE salesperson (Sno int(11) PRIMARY KEY, Sname varchar(50), City varchar(50), Comm int(11));

INSERT INTO salesperson(Sno,Sname,City,Comm) VALUES

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

(1002,'Serrs','San Jose', .13),

(1004,'Motika','London', .11),

(1007,'Rafkin','Barcelone', .15),

(1003,'Axelrod','New York', .1);

SELECT \* FROM 'Salesperson';



**Table 2:**

**Table Name : Customer:-**

CREATE TABLE customer(Cno int(11) PRIMARY KEY,Cname varchar(50),City varchar(50),Rating int(11),Sno int(11),

FOREIGN KEY (Sno) REFERENCES salesperon (Sno) ON DELETE CASCADE ON UPDATE CASCADE);

INSERT INTO customer (Cno,Cname,City,Rating,Sno) VALUES

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

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

(203,'Liu','San Jose', 300, 1002),

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

(205,'Clemens','London', 300, 1007),

(206,'Pereira','Roe', 100, 1004);

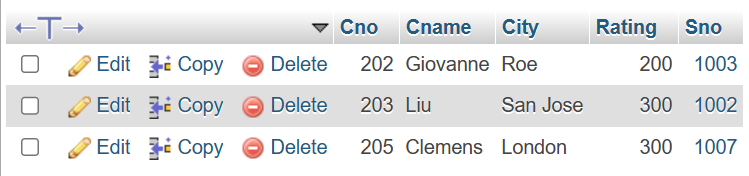
SELECT \* FROM 'customer';



**a) All orders for more than $1000.**

**Query:-**

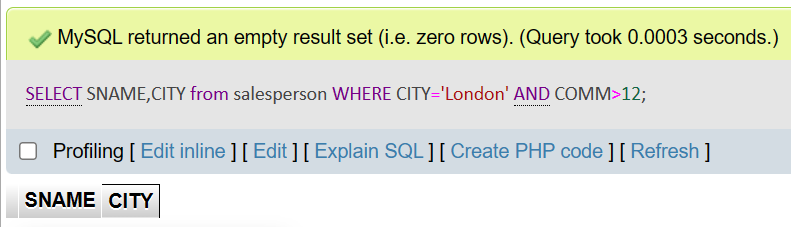
SELECT \* FROM customer where RATING>100;



**b) Names and cities of all salespeople in London with commission above 0.12.**

**Query:-**

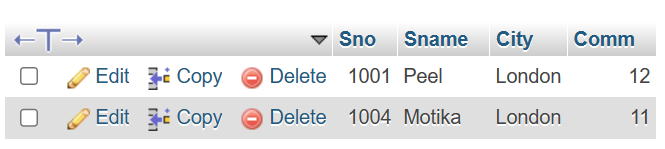
SELECT SNAME,CITY from salesperson WHERE CITY='London' AND COMM>12;



**c) All salespeople either in Barcelona or in London.**

**Query:-**

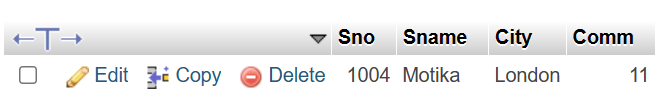
SELECT \* from salesperson where CITY='barcelona' OR CITY='london';



**d) All salespeople with commission between 0.10 and 0.12. (Boundary values should be excluded).**

**Query:-**

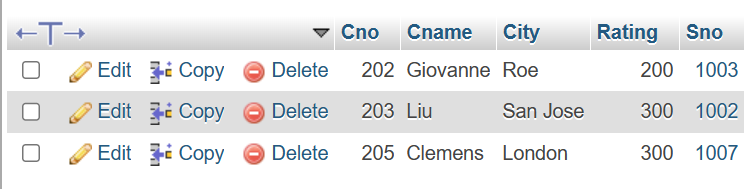
SELECT \* from salesperson where COMM>10 and COMM<12;

****

**e) All customers excluding those with rating <= 100 unless they are located in Rome.**

**Query:-**

SELECT \* FROM CUSTOMER WHERE RATING>100 AND CITY !='ROM';



**--XOXOXOXOXOXOXO--**