**MODULE: 5 (Database)**

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

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)](https://www.oracle.com/in/database/what-is-database/#WhatIsDBMS). Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened to just database.

Data within the most common types of databases in operation today is typically modeled in rows and columns in a series of tables to make processing and data querying efficient. The data can then be easily accessed, managed, modified, updated, controlled, and organized. Most databases use structured query language (SQL) for writing and querying data.

1. **What is Normalization?**

Normalization is the process to eliminate data redundancy and enhance data integrity in the table. Normalization also helps to organize the data in the database. It is a multi-step process that sets the data into tabular form and removes the duplicated data from the relational tables.

Normalization organizes the columns and tables of a database to ensure that database integrity constraints properly execute their dependencies. It is a systematic technique of decomposing tables to eliminate data redundancy (repetition) and undesirable characteristics like Insertion, Update, and Deletion anomalies.

**1st Normal Form (1NF)**

A table is referred to as being in its First Normal Form if atomicity of the table is 1.

Here, atomicity states that a single cell cannot hold multiple values. It must hold only a single-valued attribute.

The First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

**Second Normal Form (2NF)**

The first condition for the table to be in Second Normal Form is that the table has to be in First Normal Form. The table should not possess partial dependency. The partial dependency here means the proper subset of the candidate key should give a non-prime attribute.

**Third Normal Form (3NF)**

The first condition for the table to be in Third Normal Form is that the table should be in the Second Normal Form.

The second condition is that there should be no transitive dependency for non-prime attributes, which indicates that non-prime attributes (which are not a part of the candidate key) should not depend on other non-prime attributes in a table. Therefore, a transitive dependency is a functional dependency in which A → C (A determines C) indirectly, because of A → B and B → C (where it is not the case that B → A).

The third Normal Form ensures the reduction of data duplication. It is also used to achieve data integrity.

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

What is DBMS?

A database management system (DBMS) allows users to store, retrieve, and manipulate data in a database. A DBMS typically provides tools that enable users to create, update, and delete data in the database. In addition, a DBMS may provide tools for managing the database, such as creating and deleting tables and indexes and managing user access to the database. It also include, data integrity, and data recovery features.

Types of DBMS

There are four main types of DBMS: relational, object-oriented, graph-based, and NoSQL.

Relational DBMSs are the most common and use a tabular structure to store data.

Object-oriented DBMSs use an object-oriented model to store data, and Graph-based DBMSs use a graph structure to store data.

 DBMSs are a newer DBMS type that uses a non-tabular structure to store data.

There are some other types of DBMS also considerable one, and there are,

Hierarchical DBMS

Network DBMS

Cloud DBMS

In-Memory DBMS

Distributed DBMS

Multivalue DBMS

XML DBMS

Benefits of DBMS

A database management system (DBMS) is a software package designed to define, manipulate, and control a database. It is a system that enables the creation and maintenance of a central database. So, there are plenty of benefits of this DBMS, and the following are the considerable ones,

Data Integrity: A DBMS helps maintain data integrity due to its ability to enforce data integrity constraints. This integrity ensures that data is consistent, accurate, and reliable over time.

Data Security: A DBMS ensures data security through access control mechanisms like user profiles, passwords, and other authentication methods. This security ensures that only authorized users have access to the required data.

Data Redundancy: A DBMS eliminates data redundancy by storing data in a single centralized location. And this redundancy reduces the amount of data that needs to be stored and simplifies the data manipulation process.

Data Consistency: A DBMS ensures data consistency by enforcing data rules . More than this, this consistency ensures that all users have access to the same up-to-date information.

Cost Reduction: A DBMS system reduces the cost associated with data storage and data management.

What is RDBMS?

Generally, RDBMS stands for “relational database management system.” A relational database management system (RDBMS) is a database management system (DBMS) that uses relational techniques for storing and retrieving data. And also it is based on the relational model, which organizes data into rows and columns in tables.

RDBMSs use  (Structured Query Language) to manipulate data in the database. SQL is a standard language that most RDBMSs use and SQL can insert, update, delete, and query data in the database. RDBMSs have been the most popular type of DBMS since the 1980s. And nowadays, RDBMSs are the most widely used database systems because they provide a powerful and flexible way to store, retrieve and manage data.

Benefits of RDBMS

A relational database management system (RDBMS) is a powerful tool for storing and retrieving data. RDBMSs are used in various web-based applications, financial (CRM) systems. RDBMSs are also well-suited for managing large volumes of data.RDBMSs offer several benefits over other database management systems.

Structured Query Language (SQL): The most powerful benefit of RDBMS is the use of SQL to store, retrieve, manipulate, and manage data. SQL is a powerful language that enables users to interact with the database efficiently and effectively.

Atomicity: This feature of RDBMS ensures that all the transactions in the system are completed in an atomic manner. This means that all the transaction operations are done successfully, or none are done, ensuring data integrity and consistency.

Reliability: RDBMS provides a reliable system for storing, updating, and retrieving data. It also guarantees data security since it maintains an audit trail of every transaction, which helps detect any anomalies in the system.

Scalability: RDBMS is highly scalable and can easily be expanded to accommodate larger datasets. And this scalability leads to support for more users and data.

Flexibility: RDBMSs offer a high degree of flexibility, allowing users to add, delete, and update data easily.

1. **What is MF Cod Rule of RDBMS Systems?**

It seems like there might be a typo or misunderstanding in your question. As of my last update in January 2022, there's no widely recognized term or concept called "MF Cod Rule" in the context of RDBMS (Relational Database Management Systems).

However, there are various principles and rules associated with relational databases, such as those defined by Codd's 12 rules or the principles of normalization.

Codd's 12 rules, proposed by Edgar F. Codd, are a set of criteria used to determine whether a database management system qualifies as a relational database management system (RDBMS). These rules are designed to ensure the integrity and flexibility of relational databases. They cover aspects such as data manipulation, data definition, and data integrity.

If you meant something else by "MF Cod Rule" or if you have any other questions related to relational databases or RDBMS principles, please feel free to clarify, and I'd be happy to assist you further.

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

Data redundancy refers to the practice of keeping data in two or more places within a database or data storage system. Data redundancy ensures an organization can provide continued operations or services in the event something happens to its data -- for example, in the case of data corruption or . The concept applies to areas such as databases, computer memory and file storage systems.

Data redundancy can occur within an organization intentionally or accidentally. If done intentionally, the same data is kept in different locations with the organization making a conscious effort to protect it and ensure its consistency. This data is often used for backups or disaster recovery.

If carried out by accident, duplicate data may cause data inconsistencies. Even though data redundancy can help minimize the chance of data loss, redundancy issues can affect larger data sets. For example, data that is stored in several places takes up valuable storage space and makes it difficult for the organization to identify which data they should access or update.

1. **What is DDL Interpreter?**

In the context of databases, DDL stands for Data Definition Language. It's a subset of SQL (Structured Query Language) used for defining and modifying the structure of a database schema, such as creating, altering, or dropping tables, indexes, and other database objects.

A DDL interpreter is a component of a database management system (DBMS) that processes DDL statements. It's responsible for executing commands that define or modify the structure of the database schema. This includes tasks such as creating tables, defining constraints, altering column definitions, and dropping objects from the database.

The DDL interpreter translates the DDL statements issued by users or applications into actions that manipulate the underlying database schema. It ensures that the requested changes adhere to the syntax and semantics of the database system, and it performs the necessary operations to implement those changes in the database.

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

In SQL (Structured Query Language), DML stands for Data Manipulation Language. It's a subset of SQL used for querying and modifying data in a database. Common DML statements include SELECT (for querying data), INSERT (for adding new rows), UPDATE (for modifying existing rows), and DELETE (for removing rows).

A DML compiler, sometimes referred to as the query compiler, is a component of a database management system (DBMS) responsible for translating DML statements written in SQL into low-level instructions or an execution plan that the database engine can understand and execute efficiently.

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

In SQL, key constraints are rules applied to columns in a database table that enforce uniqueness and integrity.

There are primarily three types of key constraints:

Primary Key Constraint: A primary key constraint uniquely identifies each record in a table and ensures that the column(s) it's applied to contain unique and non-null values.

Unique Constraint: A unique constraint ensures that all values in a column or a group of columns are unique, but unlike the primary key constraint, it allows null values.

Foreign Key Constraint: A foreign key constraint establishes a relationship between two tables by enforcing referential integrity. It ensures that values in a column (or a set of columns) in one table match values in another table's column (usually the primary key column).

Here's an example of each type of key constraint:

**Primary Key Constraint Example:**

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DepartmentID INT

);

**Unique Constraint Example:**

CREATE TABLE Products (

ProductID INT UNIQUE,

ProductName VARCHAR(100),

Price DECIMAL(10, 2)

);

**Foreign Key Constraint Example:**

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

ProductID INT,

Quantity INT,

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

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

A savepoint in SQL is a point in a transaction where you can save the current state of the transaction so that you can later roll back to that point if needed. This is useful in situations where you want to make changes within a transaction but might want to undo those changes without rolling back the entire transaction.

To create a savepoint, you can use the SAVEPOINT statement in SQL. Here's the syntax:

SAVEPOINT savepoint\_name;

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

In SQL, a trigger is a database object that is automatically executed or fired in response to certain events or actions occurring in the database. These events could include INSERT, UPDATE, DELETE operations on a table, or even database schema changes. Triggers are often used to enforce data integrity rules, perform logging, or automate complex business logic within the database.

To create a trigger in SQL, you typically define the trigger's behavior, including the event that will activate it, the table it's associated with, and the actions it will perform when triggered.

Syntax : CREATE [OR REPLACE] TRIGGER trigger\_name

{BEFORE | AFTER | INSTEAD OF} {INSERT | UPDATE | DELETE | TRUNCATE}

ON table\_name

[REFERENCING {OLD AS old | NEW AS new}]

[FOR EACH ROW]

WHEN (condition)

BEGIN

-- Trigger action statements

END;

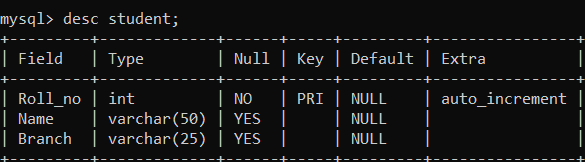
**SQL Queries**

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

Student table :

create table student(Roll\_no int auto\_increment primary key,Name varchar(50),Branch varchar(25));

desc student;

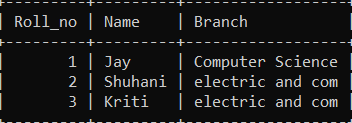


insert into student values(1,'Jay','Computer Science');

insert into student values(2,'Shuhani','electric and com');

insert into student values(3,'Kriti','electric and com');

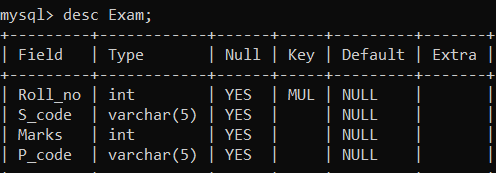
select \* from student;



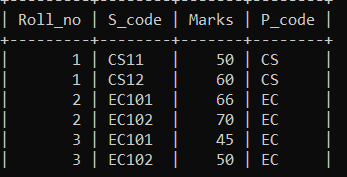
Exam table :

create table Exam(Roll\_no int,S\_code varchar(5),Marks int,P\_code varchar(5));

alter table Exam add foreign key(Roll\_no) REFERENCES student(Roll\_no);



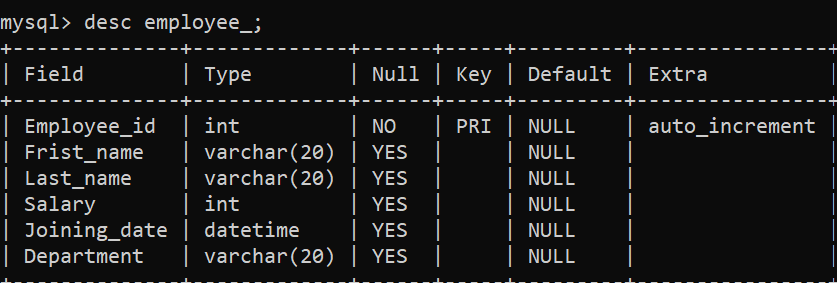
select \* from Exam;



**2. Create table given below: Employee and IncentiveTable**

**Employee :**

**create table Employee\_(Employee\_id int auto\_increment primary key,Frist\_name varchar(20),Last\_name varchar(20),Salary int,Joining\_date datetime,Department varchar(20));**

****

**………………………………Insert data in employee table…………………………………**

**insert into Employee\_ values (1, 'John', 'Abraham', 1000000, '2013-01-01 12:00:00', 'Banking');**

**insert into Employee\_ values (2, 'Michael', 'Clarke', 800000, '2013-01-01 12:00:00', 'Insurance');**

**insert into Employee\_ values (3, 'Roy', 'Thomas', 700000, '2013-02-01 12:00:00', 'Banking');**

**insert into Employee\_ values (4, 'Tom', 'Jose',600000, '2013-02-01 12:00:00', 'Insurance');**

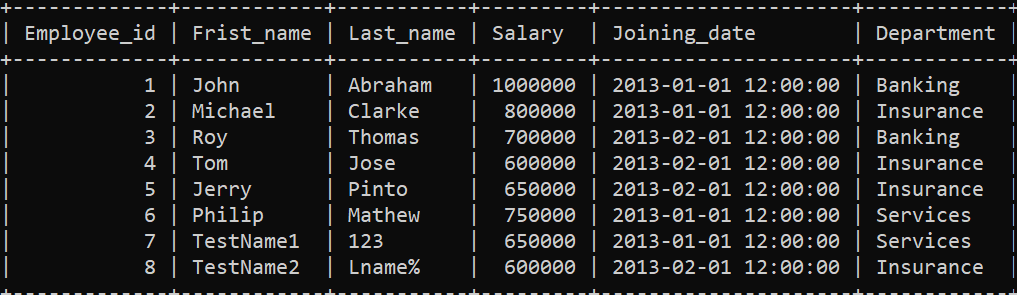
**insert into Employee\_ values (5, 'Jerry', 'Pinto',650000, '2013-02-01 12:00:00', 'Insurance');**

**insert into Employee\_ values (6, 'Philip', 'Mathew',750000, '2013-01-01 12:00:00', 'Services');**

**insert into Employee\_ values (7, 'TestName1', '123',650000, '2013-01-01 12:00:00', 'Services');**

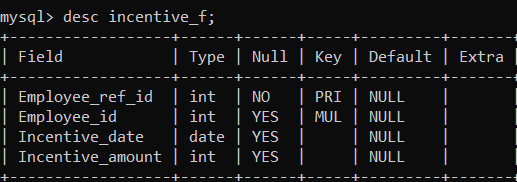
**insert into Employee\_ values (8, 'TestName2', 'Lname%',600000, '2013-02-01 12:00:00', 'Insurance');**

**select \* from Employee\_;**

****

**IncentiveTable**

CREATE TABLE incentive\_f (Employee\_ref\_id INT PRIMARY KEY,Employee\_id INT,Incentive\_date DATE,Incentive\_amount int, FOREIGN KEY (Employee\_id) REFERENCES Employee\_(Employee\_id));

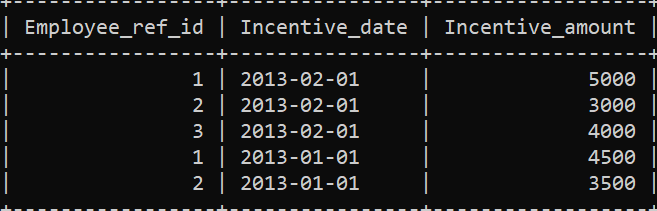
****

**………………………………Insert data in Incentive table………………………………….**

**insert into Incentive\_f values(1,'2013-02-01',5000),(2,'2013-02-01',3000),**

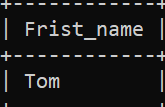
**(3,'2013-02-01',4000),(1,'2013-01-01',4500),(2,'2013-01-01',3500);**

**select \* from Incentive\_f;**

****

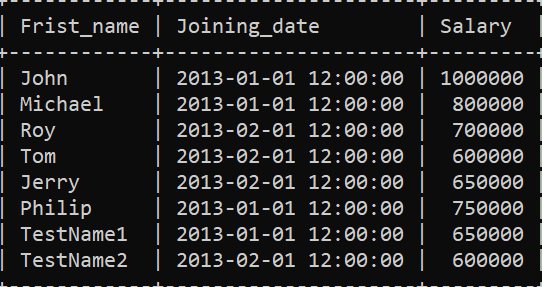
**Get First\_Name from employee table using Tom name “Employee Name”.**

**select Frist\_name from Employee\_ where Frist\_name = 'Tom';**

****

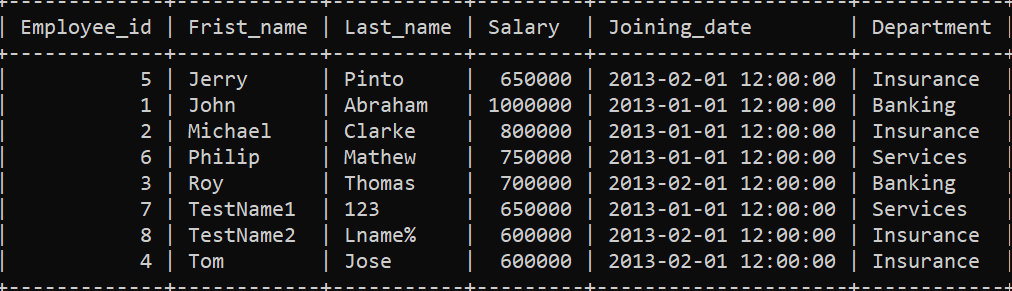
**Get FIRST\_NAME, Joining Date, and Salary from employee table.**

**select Frist\_name,Joining\_date,Salary from employee\_;**

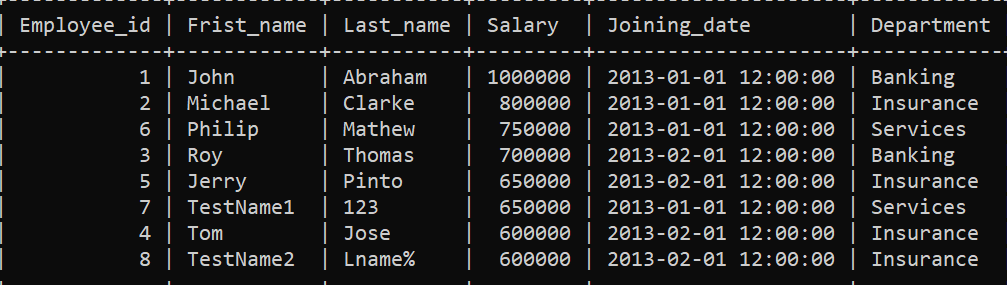
****

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

**select \* from Employee\_ order by Frist\_name ASC;**

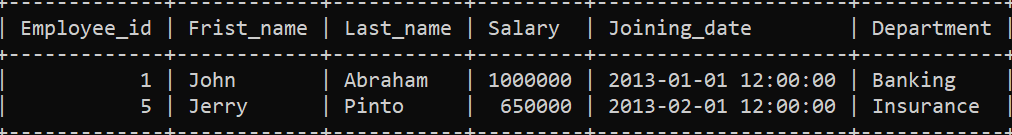
****

**select \* from employee\_ order by Salary desc;**

****

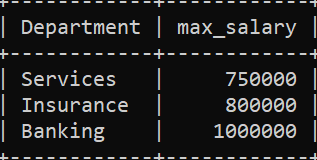
**Get employee details from employee table whose first name contains ‘J’.**

**select \* from employee\_ where Frist\_name LIKE '%J%';**



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

**select Department,max(Salary) as max\_salary from employee\_ group by Department order by max\_salary asc;**

****

**Create After Insert trigger on Employee table which insert records in viewtable**

**DELIMITER //**

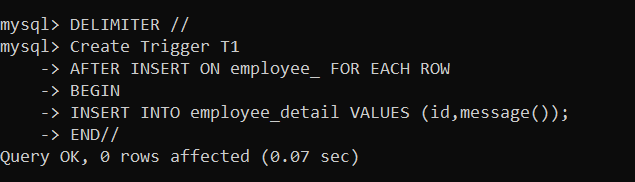
**mysql> Create Trigger T1**

**-> AFTER INSERT ON employee\_ FOR EACH ROW**

**-> BEGIN**

**-> INSERT INTO employee\_detail VALUES (id,message());**

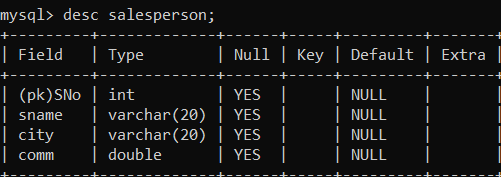
**-> END//**

****

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

**Salesperson :**

Create table SALESPERSON1(`(pk)SNo`INT,sname VARCHAR(20),city VARCHAR(20),comm DOUBLE );

****

**……………Insert value in this table……………..**

insert into salesperson1 values(1001,'Peel','London',.12),

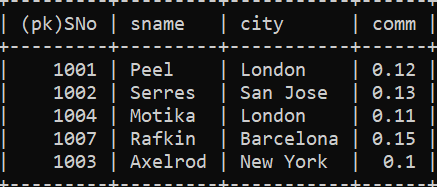
(1002,'Serres','SanJose',.13),

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

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

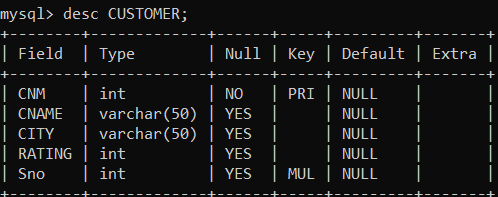
(1003,'Axelrod','New York',.1);

select \* from salesperson1;



**Customer :**

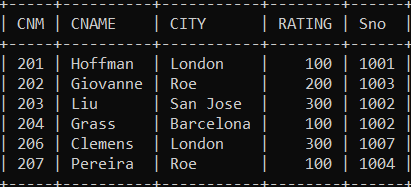
Create table CUSTOMER ( CNM int primary key, CNAME varchar(50), CITY varchar(50),RATING int,Sno int, foreign key (Sno) REFERENCES SALESPERSON1(Sno));



…………………Insert data in customer table……………

Insert into CUSTOMER (CNM, 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),(206, 'Clemens', 'London', 300, 1007),(207, 'Pereira', 'Roe', 100, 1004);



**13.All orders for more than $1000**.

select \* from CUSTOMER where RATING > 1000;



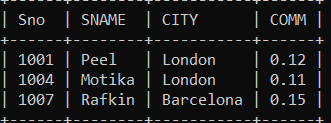
**14.Names and cities of all salespeople in London with commission above 0.12**

select SNAME, CITY from SALESPERSON1 where CITY = 'London' AND COMM > 0.12;



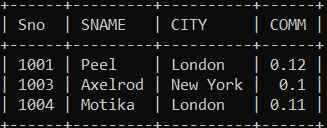
**15.All salespeople either in Barcelona or in London**

SELECT \*FROM SALESPERSON1 WHERE CITY IN ('Barcelona', 'London')



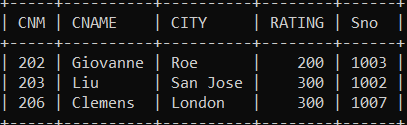
**16. All salespeople with commission between 0.10 and 0.12. (Boundary values should be excluded).**

SELECT \* FROM SALESPERSON1 WHERE COMM > 0.10 AND COMM < 0.12;

****

**17. All customers excluding those with rating <= 100 unless they are located inRome**.

**SELECT \*FROM CUSTOMER WHERE RATING > 100 OR (RATING <= 100 AND CITY = 'Rome');**

****

**18. Write a SQL statement that displays all the information about all salespeople.**

**Create table Salespeople (salesman\_id int primary key,name varchar(255),city varchar(255),commission float);**

Insert into Salespeople (salesman\_id, name, city, commission)values

(5001, 'James Hoog', 'New York', 0.15),

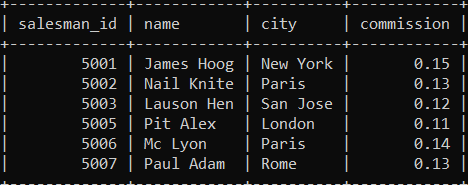
(5002, 'Nail Knite', 'Paris', 0.13),

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

(5006, 'Mc Lyon', 'Paris', 0.14),

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

(5003, 'Lauson Hen', 'San Jose', 0.12);

****

**19. From the following table, write a SQL query to find orders that are delivered by a salesperson with ID. 5001. Return ord\_no, ord\_date, purch\_amt.**

CREATE TABLE orders (ord\_no INT PRIMARY KEY, purch\_amt FLOAT,ord\_date DATE,customer\_id INT,salesman\_id INT);

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

-> (70001, 150.5, '2012-10-05', 3005, 5002),

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

-> (70002, 65.26, '2012-10-05', 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),

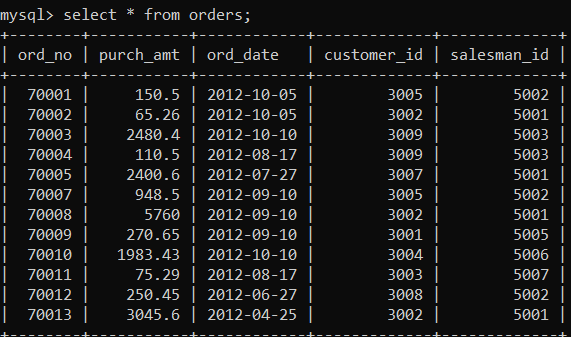
-> (70010, 1983.43, '2012-10-10', 3004, 5006),

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

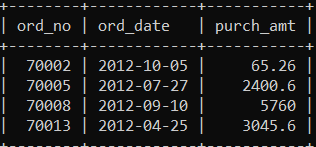
-> (70012, 250.45, '2012-06-27', 3008, 5002),

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

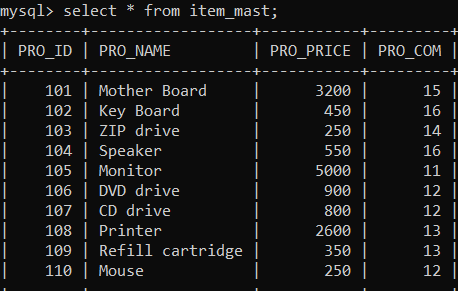
-> (70013, 3045.6, '2012-04-25', 3002, 5001);

****

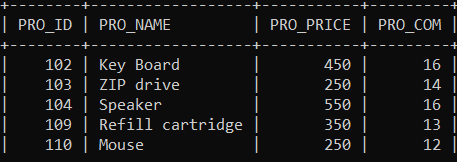
**SELECT ord\_no, ord\_date, purch\_amt FROM orders WHERE salesman\_id = 5001;**

****

**20. From the following table, write a SQL query to select a range of products whose price is in the range Rs.200 to Rs.600. Begin and end values are included. Return pro\_id, pro\_name, pro\_price, and pro\_com.**

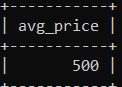
****

SELECT PRO\_ID, PRO\_NAME, PRO\_PRICE, PRO\_COMFROM item\_mast WHERE PRO\_PRICE BETWEEN 200.00 AND 600.00;

****

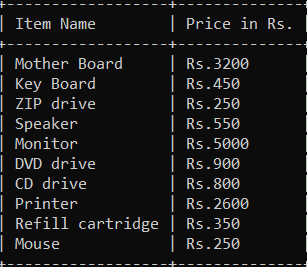
**21.From the following table, write a SQL query to calculate the average price for a manufacturer code of 16. Return avg.**

SELECT AVG(PRO\_PRICE) AS avg\_price FROM item\_mast WHERE PRO\_COM =16;

****

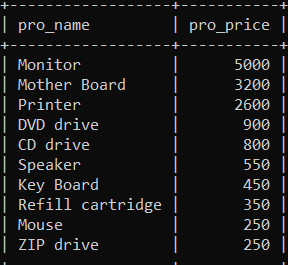
**22. From the following table, write a SQL query to display the pro\_name as 'Item Name' and pro\_priceas 'Price in Rs.'**

SELECT PRO\_NAME AS 'Item Name', CONCAT('Rs.', PRO\_PRICE) AS 'Price in Rs.'FROM item\_mast;

****

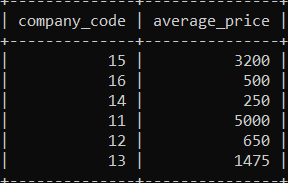
**23. From the following table, write a SQL query to find the items whose prices are higher than or equal to $250. Order the result by product price in descending, then product name in ascending. Return pro\_name and pro\_price.**

SELECT pro\_name, pro\_price FROM item\_mast WHERE pro\_price >= 250 ORDER BY pro\_price DESC, pro\_name ASC;

****

**24. From the following table, write a SQL query to calculate average price of the items for each company. Return average price and company code.**

SELECT PRO\_COM AS company\_code, AVG(PRO\_PRICE) AS average\_price FROM item\_mast GROUP BY PRO\_COM;

****

**CREATE TRIGGER employee1 AFTER INSERT ON Employee1 FOR EACH ROW**

**BEGIN**

**INSERT INTO ViewTable (column1, column2, column3, ...)**

**VALUES (NEW.column1, NEW.column2, NEW.column3, ...);**

**END;**