1. What do you understand By Database ?

Ans. Database where the data was stored in a table form.

There is many key used in database all the detail full fill in the table form.

There is used query to create the table all the things create from the SQL

(structured query language).

Key used in sql :

1.Primary Key

2.Unique Key

3.Foreign Key

There are total 5 types of database

1.Relational Database

2.No SQL Database

3.Centralized Database

4.Distributed Database

5.Object-Oriented Database

1. What is Normalization?

Ans. DBMS normalization is referred to as a process to streamline database data correctly. This is because the redundancy, malfunctions, and integrity of the data are exceeded. In other words, normalization rearranges the database by splitting the tables to actually find the practical effects of the data management mixing up tables, any data will be lost.

The objective is to extensively reduce data redundancy and dependency. In essence, normalization was introduced and has continually been improved to rectify these specific aspects of data management. By organizing data in such a rigorous and stringent manner, normalization facilitates a significantly enhanced level of data integrity and enables more efficient data operations.

1. What is Difference between DBMS and RDBMS?

| **DBMS** | **RDBMS** |
| --- | --- |
| [DBMS](https://www.geeksforgeeks.org/introduction-of-dbms-database-management-system-set-1/) stores data as file. | [RDBMS](https://www.geeksforgeeks.org/rdbms-architecture/) stores data in tabular form. |
| Data elements need to access individually. | Multiple data elements can be accessed at the same time. |
| No relationship between data. | Data is stored in the form of tables which are related to each other. |
| Normalization is not present. | Normalization is present. |
| DBMS does not support distributed database. | RDBMS supports distributed database. |
| It stores data in either a navigational or hierarchical form. | It uses a tabular structure where the headers are the column names, and the rows contain corresponding values. |
| It deals with small quantity of data. | It deals with large amount of data. |
| Data redundancy is common in this model. | Keys and indexes do not allow Data redundancy. |
| It is used for small organization and deal with small data. | It is used to handle large amount of data. |
| Not all Codd rules are satisfied. | All 12 Codd rules are satisfied. |
| Security is less | More security measures provided. |
| It supports single user. | It supports multiple users. |
| Data fetching is slower for the large amount of data. | Data fetching is fast because of relational approach. |
| The data in a DBMS is subject to low security levels with regards to data manipulation. | There exists multiple levels of data security in a RDBMS. |
| Low software and hardware necessities. | Higher software and hardware necessities. |
| Examples:[XML](https://www.geeksforgeeks.org/xml-basics/), Window Registry, Forxpro, dbaseIIIplus etc. | Examples: [MySQL](https://www.geeksforgeeks.org/architecture-of-mysql/), [PostgreSQL](https://www.geeksforgeeks.org/what-is-postgresql-introduction/), [SQL](https://www.geeksforgeeks.org/what-is-sql/) Server, Oracle, Microsoft Access etc. |

4. What is MF Cod Rule of RDBMS Systems?

**Ans. Codd’s Rules in DBMS**

**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](https://www.geeksforgeeks.org/dml-full-form/)) 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.

1. What do you understand By Data Redundancy?

Ans. Data redundancy occurs when the same piece of data is stored in two or more separate places and**i**s a common occurrence in many businesses. As more companies are moving  away from siloed data to using a central repository to store information, they are finding that their database is filled with inconsistent duplicates of the same entry. Although it can be challenging to reconcile or even benefit from duplicate data entries, understanding how to reduce and track data redundancy efficiently can help mitigate long-term inconsistency issues for your business.

Sometimes data redundancy happens by accident while other times it is intentional. Accidental data redundancy can be the result of a complex process or inefficient coding while intentional data redundancy can be used to simply by leveraging the multiple occurrences of data for disaster recovery and quality checks.

If data redundancy is intentional, it’s important to have a central field or space for the data. This allows you to easily update all records of redundant data when necessary. When data redundancy isn’t purposeful, it can lead to a variety of issues which we’ll discuss below.

1. What is DDL Interpreter?

Ans. DDL is a standardized language with commands to define the storage groups (stogroups), different structures and objects in a database. DDL statements create, modify and remove database objects, such as tables, indexes and stogroups. DDL is also used in a generic sense to refer to any language that describes data.

DDL includes Structured Query Language ([SQL)](https://www.techtarget.com/searchdatamanagement/definition/SQL) statements to create and drop databases, aliases, locations, indexes, tables and sequences. It also includes statements to alter these objects and impose or drop certain constraints on tables, such as the following:

* UNIQUE
* PRIMARY
* FOREIGN KEY
* CHECK

1. What is DML Compiler in SQL?

Ans. **1. Data Definition Language (DDL) in SQL**

**DDL or Data Definition Language** actually consists of the [SQL](https://www.geeksforgeeks.org/sql-tutorial/)commands that can be used to **defining**, **altering**, and **deleting**database structures such as **tables**, **indexes**, and **schemas**. It simply deals with descriptions of the database schema and is used to **create**and **modify**the structure of database objects in the database

**Common DDL Commands**

| **Command** | **Description** | **Syntax** |
| --- | --- | --- |
| [CREATE](https://www.geeksforgeeks.org/sql-create) | Create database or its objects (table, index, function, views, store procedure, and triggers) | **CREATE** TABLE table\_name (column1 data\_type, column2 data\_type, ...); |
| [DROP](https://www.geeksforgeeks.org/sql-drop-truncate) | Delete objects from the database | **DROP** TABLE table\_name; |
| [ALTER](https://www.geeksforgeeks.org/sql-alter-add-drop-modify) | Alter the structure of the database | **ALTER** TABLE table\_name ADD COLUMN column\_name data\_type; |
| [TRUNCATE](https://www.geeksforgeeks.org/sql-drop-truncate) | Remove all records from a table, including all spaces allocated for the records are removed | **TRUNCATE** TABLE table\_name; |
| [COMMENT](https://www.geeksforgeeks.org/sql-comments) | Add comments to the data dictionary | **COMMENT** 'comment\_text' ON TABLE table\_name; |
| [RENAME](https://www.geeksforgeeks.org/sql-alter-rename) | Rename an object existing in the database | **RENAME** TABLE old\_table\_name TO new\_table\_name; |

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

**SQL constraints are rules applied to columns or tables in a relational database to limit the type of data that can be inserted, updated, or deleted. These rules ensure the data is valid, consistent, and adheres to the business logic or database requirements. Constraints can be enforced during table creation or later using the ALTER TABLE statement. They play a vital role in maintaining the quality and integrity of your database.**

**SQL provides several types of constraints to manage different aspects of data integrity. These constraints are essential for ensuring that data meets the requirements of accuracy, consistency, and validity. Let’s go through each of them with detailed explanations and examples.**

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

**Ans.** **A save point is a logical rollback point within a transaction. When you set a save point, whenever an error occurs past a save point, you can undo the events you have done up to the save point using the rollback.**

**MySQL InnoDB provides support for the statements SAVEPOINT, ROLLBACK TO SAVEPOINT, RELEASE SAVEPOINT.**

**Syntax :SAVEPOINT identifier**

**Example:**

**CREATE TABLE EMP( FIRST\_NAME CHAR(20) NOT NULL, LAST\_NAME CHAR(20), AGE INT, SEX CHAR(1), INCOME FLOAT);**

**INSERT INTO EMP VALUES ('Krishna', 'Sharma', 19, 'M', 2000), ('Raj', 'Kandukuri', 20, 'M', 7000), ('Ramya', 'Ramapriya', 25, 'F', 5000);**

**START TRANSACTION; SELECT \* FROM EMP; UPDATE EMP SET AGE = AGE + 1; SAVEPOINT samplesavepoint; INSERT INTO EMP ('Mac', 'Mohan', 26, 'M', 2000); ROLLBACK TO SAVEPOINT samplesavepoint; COMMIT;**

**SELECT \* FROM EMP;**

**Output**

**The above query produces the following output −**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FIRST\_NAME** | **LAST\_NAME** | **AGE** | **SEX** | **INCOME** |
| **Krishna** | **Sharma** | **20** | **M** | **2000** |
| **Raj** | **Kandukuri** | **21** | **M** | **7000** |
| **Ramya** | **Ramapriya** | **26** | **F** | **5000** |

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

**Ans. Trigger is a statement that a system executes automatically when there is any modification to the database. In a trigger, we first specify when the trigger is to be executed and then the action to be performed when the trigger executes. Triggers are used to specify certain integrity constraints and referential constraints that cannot be specified using the constraint mechanism of SQL.**

**Example –   
Suppose, we are adding a tuple to the ‘Donors’ table that is some person has donated blood. So, we can design a trigger that will automatically add the value of donated blood to the ‘Blood\_record’ table.**

**Types of Triggers –   
We can define 6 types of triggers for each table:**

1. **AFTER INSERT activated after data is inserted into the table.**
2. **AFTER UPDATE: activated after data in the table is modified.**
3. **AFTER DELETE: activated after data is deleted/removed from the table.**
4. **BEFORE INSERT: activated before data is inserted into the table.**
5. **BEFORE UPDATE: activated before data in the table is modified.**
6. **BEFORE DELETE: activated before data is deleted/removed from the table.**

**Table Create :**

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

**Ans.**

**1st Table of Student = CREATE TABLE student(Rollno int PRIMARY KEY AUTO\_INCREMENT,**

**Name varchar(60) UNIQUE KEY,**

**Branch varchar(100) UNIQUE KEY**

**);**

**INSERT INTO student(Rollno,Name,Branch)**

**VALUES(1,"Jay","Computer Science");**

**INSERT INTO student(Rollno,Name,Branch)**

**VALUES(2,"Suhani","Electronic and Com");**

**INSERT INTO student(Rollno,Name,Branch)**

**VALUES(3,"Kriti","Electronic and Com");**

**2nd Table of Exam = CREATE TABLE exam(**

**Rollno int,**

**FOREIGN KEY(Rollno)**

**REFERENCES student(Rollno),**

**S\_code varchar(40),**

**Marks bigint UNIQUE KEY,**

**P\_code varchar(40)**

**);**

**INSERT INTO exam(Rollno,S\_code,Marks,P\_code)**

**VALUES(1,"CS11",50,"CS");**

**INSERT INTO exam(Rollno,S\_code,Marks,P\_code)**

**VALUES(1,"CS12",60,"CS");**

**INSERT INTO exam(Rollno,S\_code,Marks,P\_code)**

**VALUES(2,"EC101",66,"EC");**

**INSERT INTO exam(Rollno,S\_code,Marks,P\_code)**

**VALUES(3,"EC101",45,"EC");**

**INSERT INTO exam(Rollno,S\_code,Marks,P\_code)**

**VALUES(3,"EC102",55,"EC");**

1. Create Table Name = Employee and Incentive

Ans. 1st. Table of Employee = CREATE TABLE Employee(Employee\_id int PRIMARY KEY AUTO\_INCREMENT,

First\_name varchar(60) UNIQUE KEY,

Last\_name varchar(70) UNIQUE KEY,

Salary bigint,

Joining\_date varchar(100),

Department varchar(100)

);

INSERT INTO employee(Employee\_id,First\_name,Last\_name,Salary,Joining\_date,Department)

VALUES(21,"John","Abharam",800000,"01-JAN-13 12.00.00 AM","Banking");

INSERT INTO employee(Employee\_id,First\_name,Last\_name,Salary,Joining\_date,Department)

VALUES(2,"Micheal","Clarke",800000,"01-JAN-13 12.00.00 AM","Insurance");

INSERT INTO employee(Employee\_id,First\_name,Last\_name,Salary,Joining\_date,Department)

VALUES(3,"Roy","Thomas",700000,"01-Feb-13 12.00.00 AM","Banking");

INSERT INTO employee(Employee\_id,First\_name,Last\_name,Salary,Joining\_date,Department)

VALUES(4,"Tom","Jose",600000,"01-Feb-13 12.00.00 AM","Insurance");

INSERT INTO employee(Employee\_id,First\_name,Last\_name,Salary,Joining\_date,Department)

VALUES(5,"Jerry","Pinto",650000,"01-Feb-13 12.00.00 AM","Insurance");

INSERT INTO employee(Employee\_id,First\_name,Last\_name,Salary,Joining\_date,Department)

VALUES(6,"Philip","Mathew",750000,"01-Jan-13 12.00.00 AM","Services");

INSERT INTO employee(Employee\_id,First\_name,Last\_name,Salary,Joining\_date,Department)

VALUES(7,"TestName1","123",650000,"01-Jan-13 12.00.00 AM","Services");

INSERT INTO employee(Employee\_id,First\_name,Last\_name,Salary,Joining\_date,Department)

VALUES(8,"TestName2","Lname%",600000,"01-Feb-13 12.00.00 AM","Insurance");

2nd.Table for Increntive = CREATE TABLE incetive(Employee\_ref\_id int,

Incetive\_date varchar(40),

Incentive\_amount bigint);

INSERT INTO incetive(Employee\_ref\_id,Incetive\_date,Incentive\_amount)

VALUES(1,"01-FEB-13", 5000);

INSERT INTO incetive(Employee\_ref\_id,Incetive\_date,Incentive\_amount)

VALUES(2,"01-FEB-13", 3000);

INSERT INTO incetive(Employee\_ref\_id,Incetive\_date,Incentive\_amount)

VALUES(3,"01-FEB-13", 4000);

INSERT INTO incetive(Employee\_ref\_id,Incetive\_date,Incentive\_amount)

VALUES(1,"01-JAN-13", 4500);

INSERT INTO incetive(Employee\_ref\_id,Incetive\_date,Incentive\_amount)

VALUES(2,"01-JAN-13", 3500);

Que3.

Ans. SELECT First\_name FROM Employee WHERE First\_name = 'Tom';

Que4.

Ans. SELECT First\_name, Joining\_date, Salary FROM Employee;

Que5.

Ans. SELECT \* FROM Employee ORDER BY First\_name;

Que6.

Ans. SELECT \* FROM Employee WHERE First\_name LIKE '%J%';

Que7.

Ans. SELECT Department, MAX(Salary) AS Max\_Salary

FROM Employee

GROUP BY Department

ORDER BY Max\_Salary ASC;

Que8.

Ans. SELECT \* FROM Employee ORDER BY Salary ASC;

Que9.

Ans. SELECT First\_name, Incentive\_amount

FROM employee

JOIN incetive ON Employee\_id = Employee\_ref\_id

WHERE Incentive\_amount > 3000;

Que10.

Ans. CREATE TRIGGER AfterInsertEmployee

AFTER INSERT ON employee

FOR EACH ROW

INSERT INTO EmployeeViewTable (Employee\_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);

1. Create Table Name = Salesmperson And Customer

Ans. 1st.Table of Salesperson = CREATE TABLE salesperson(

SNO int PRIMARY KEY AUTO\_INCREMENT,

SNAME varchar(60) UNIQUE KEY,

CITY varchar(60) UNIQUE KEY,

COMM bigint

);

INSERT INTO salesperson(SNO,SNAME,CITY,COMM)

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

INSERT INTO salesperson(SNO,SNAME,CITY,COMM)

VALUES(1002,"Serres","San Jose",.13);

INSERT INTO salesperson(SNO,SNAME,CITY,COMM)

VALUES(1003,"Motika","London",.11);

INSERT INTO salesperson(SNO,SNAME,CITY,COMM)

VALUES(1004,"Rafkin","Barcelona",.15);

INSERT INTO salesperson(SNO,SNAME,CITY,COMM)

VALUES(1005,"Axelrod","New York",.1);

2nd.Table of Customer = CREATE TABLE customer(CNM int PRIMARY KEY AUTO\_INCREMENT,

CNAME varchar(40) UNIQUE KEY,

CITY varchar(60),

RATING bigint ,

SNO int);

INSERT INTO customer(CNM,CNAME,CITY,RATING,SNO)

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

INSERT INTO customer(CNM,CNAME,CITY,RATING,SNO)

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

INSERT INTO customer(CNM,CNAME,CITY,RATING,SNO)

VALUES(203,"Liu","San Jose",300,1002);

INSERT INTO customer(CNM,CNAME,CITY,RATING,SNO)

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

INSERT INTO customer(CNM,CNAME,CITY,RATING,SNO)

VALUES(206,"Clemens","London",300,1007);

INSERT INTO customer(CNM,CNAME,CITY,RATING,SNO)

VALUES(207,"Perecia","Roe",100,1004);

Que13.

Ans. SELECT \* FROM Orders WHERE OrderAmount > 1000;

Que14.

Ans. SELECT SNAME, CITY FROM Salesperson

WHERE CITY = 'London' AND COMM> 0.12;

Que15.

Ans. SELECT \* FROM Salesperson

WHERE CITY IN ('Barcelona', 'London');

Que16.

Ans. SELECT \* FROM Salesperson

WHERE COMM > 0.10 AND COMM < 0.12;

Que17.

Ans. SELECT \* FROM customer

WHERE RATING > 100 OR CITY = 'Rome';

Que 18. Write a SQL statement that displays all the information about all the salesperson.

Ans.

CREATE TABLE salespeople(employee\_id int PRIMARY KEY AUTO\_INCREMENT,

name varchar(40) UNIQUE KEY,

city varchar(60),

commision bigint);

INSERT INTO salespeople(employee\_id,name,city,commision)

VALUES(5001,"James Hoog","New York",0.15);

INSERT INTO salespeople(employee\_id,name,city,commision)

VALUES(5002,"Nail Knite","Paris",0.13);

INSERT INTO salespeople(employee\_id,name,city,commision)

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

INSERT INTO salespeople(employee\_id,name,city,commision)

VALUES(5006,"Mc Lyon","Paris",0.14);

INSERT INTO salespeople(employee\_id,name,city,commision)

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

INSERT INTO salespeople(employee\_id,name,city,commision)

VALUES(5003,"Lauson Hen","San Jose",0.12);

SELECT \* FROM salespeople;

QUE19. From the following table. write a SQL query to find ordeds that are delivered by a salesperson with id.5001.return ord\_no,ord\_date.

Purch\_amy.

Ans.

Que19.

CREATE TABLE salesmen(ord\_no bigint PRIMARY KEY AUTO\_INCREMENT,

purch\_amt bigint,

ord\_date bigint,

customer\_id int,

salesmen\_id int );

INSERT INTO salesmen(ord\_no,purch\_amt,ord\_date,customer\_id,salesmen\_id)

VALUES(70001,150.5,2012-10-05,3005,5002);

INSERT INTO salesmen(ord\_no,purch\_amt,ord\_date,customer\_id,salesmen\_id)

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

INSERT INTO salesmen(ord\_no,purch\_amt,ord\_date,customer\_id,salesmen\_id)

VALUES(70002,65.26,2012-10-05,3002,5001);

INSERT INTO salesmen(ord\_no,purch\_amt,ord\_date,customer\_id,salesmen\_id)

VALUES(70004,110.05,2012-08-17,3009,5003);

INSERT INTO salesmen(ord\_no,purch\_amt,ord\_date,customer\_id,salesmen\_id)

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

INSERT INTO salesmen(ord\_no,purch\_amt,ord\_date,customer\_id,salesmen\_id)

VALUES(70005,2400.6,2012-09-10,3007,5001);

INSERT INTO salesmen(ord\_no,purch\_amt,ord\_date,customer\_id,salesmen\_id)

VALUES(70008,5760,2012-10-10,3002,5001);

INSERT INTO salesmen(ord\_no,purch\_amt,ord\_date,customer\_id,salesmen\_id)

VALUES(70010,1983.43,2012-07-10,3004,5006);

INSERT INTO salesmen(ord\_no,purch\_amt,ord\_date,customer\_id,salesmen\_id)

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

INSERT INTO salesmen(ord\_no,purch\_amt,ord\_date,customer\_id,salesmen\_id)

VALUES(700012,250.45,2012-06-27,3008,5002);

INSERT INTO salesmen(ord\_no,purch\_amt,ord\_date,customer\_id,salesmen\_id)

VALUES(700011,75.29,2012-08-17,3003,5007);

INSERT INTO salesmen(ord\_no,purch\_amt,ord\_date,customer\_id,salesmen\_id)

VALUES(700013,3045.6,2012-04-25,3001,5001);

SELECT ord\_no, ord\_date, purch\_amt

FROM salesmen

WHERE salesmen\_id = 5001;

Que20. From the following table. Write a SQL query to select a range pf products whose price is in the range RS.200 To 600 . Begin and End values are included. Return pro\_id,pro\_name,pro\_price,and pro\_comm.

Ans.

Que20.

CREATE TABLE item(PRO\_ID int PRIMARY KEY AUTO\_INCREMENT,

PRO\_NAME varchar(60) UNIQUE KEY,

PRO\_PRICE bigint UNIQUE KEY,

PRO\_COM int);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(101,"Mother Board",3200,15);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(102,"KEy Board",450,16);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(103,"ZIP drive",250,14);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(105,"Monitor",5000,11);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(109,"Refill catridge",350,13);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(110,"Mouse",251,12);

SELECT PRO\_ID,PRO\_PRICE,PRO\_COM,PRO\_NAME

FROM item

WHERE PRO\_PRICE BETWEEN 200 AND 600;

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

Ans.

Que20.

CREATE TABLE item(PRO\_ID int PRIMARY KEY AUTO\_INCREMENT,

PRO\_NAME varchar(60) UNIQUE KEY,

PRO\_PRICE bigint UNIQUE KEY,

PRO\_COM int);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(101,"Mother Board",3200,15);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(102,"KEy Board",450,16);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(103,"ZIP drive",250,14);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(105,"Monitor",5000,11);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(109,"Refill catridge",350,13);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(110,"Mouse",251,12);

SELECT AVG(PRO\_PRICE) AS avg\_price

FROM item

WHERE PRO\_COM = 16;

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

Ans.

Que20.

CREATE TABLE item(PRO\_ID int PRIMARY KEY AUTO\_INCREMENT,

PRO\_NAME varchar(60) UNIQUE KEY,

PRO\_PRICE bigint UNIQUE KEY,

PRO\_COM int);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(101,"Mother Board",3200,15);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(102,"KEy Board",450,16);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(103,"ZIP drive",250,14);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(105,"Monitor",5000,11);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(109,"Refill catridge",350,13);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(110,"Mouse",251,12);

SELECT PRO\_NAME AS "Item Name", PRO\_PRICE AS "Price in Rs."

FROM item;

Que 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.

Ans.

Que20.

CREATE TABLE item(PRO\_ID int PRIMARY KEY AUTO\_INCREMENT,

PRO\_NAME varchar(60) UNIQUE KEY,

PRO\_PRICE bigint UNIQUE KEY,

PRO\_COM int);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(101,"Mother Board",3200,15);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(102,"KEy Board",450,16);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(103,"ZIP drive",250,14);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(105,"Monitor",5000,11);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(109,"Refill catridge",350,13);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(110,"Mouse",251,12);

SELECT PRO\_NAME, PRO\_PRICE

FROM item

WHERE PRO\_PRICE >= 250

ORDER BY PRO\_PRICE DESC, PRO\_NAME ASC;

Que 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.

Ans.

Que20.

CREATE TABLE item(PRO\_ID int PRIMARY KEY AUTO\_INCREMENT,

PRO\_NAME varchar(60) UNIQUE KEY,

PRO\_PRICE bigint UNIQUE KEY,

PRO\_COM int);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(101,"Mother Board",3200,15);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(102,"KEy Board",450,16);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(103,"ZIP drive",250,14);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(105,"Monitor",5000,11);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

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

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(109,"Refill catridge",350,13);

INSERT INTO item(PRO\_ID,PRO\_NAME,PRO\_PRICE,PRO\_COM)

VALUES(110,"Mouse",251,12);

SELECT PRO\_COM, AVG(PRO\_PRICE) AS average\_price

FROM item

GROUP BY PRO\_COM;