DATABASE ASSIGNMENT

1. What do you understand By Database

A database is a structured collection of data organized in a way that a computer program can quickly search and retrieve specific pieces of data. It is designed to efficiently manage, store, and retrieve information. Databases are used in various applications and scenarios where large amounts of structured data need to be organized, accessed, and manipulated.

Key characteristics of databases include:

1. **Structured Data:** Data in a database is typically organized in tables with predefined columns and data types. This structured format allows for efficient storage and retrieval of information.
2. **Query Language:** Databases are accessed and manipulated using a query language, commonly SQL (Structured Query Language). SQL allows users to interact with the database by performing operations such as inserting, updating, deleting, and retrieving data.
3. **ACID Properties:** Transactions in databases adhere to ACID properties—Atomicity, Consistency, Isolation, and Durability. These properties ensure the reliability and consistency of the data, even in the event of system failures.
4. **Concurrent Access:** Databases allow multiple users or applications to access and modify data concurrently. Proper mechanisms are in place to manage concurrent access and maintain data integrity.
5. **Scalability:** Databases can scale to handle large amounts of data and increasing numbers of users. This scalability is crucial for applications that need to accommodate growth over time.

There are different types of databases, including:

- **Relational Databases:** Use tables to store data and establish relationships between tables. Examples include MySQL, PostgreSQL, and Oracle Database.
- **NoSQL Databases:** Designed to handle unstructured or semi-structured data. Examples include MongoDB (document-oriented), Cassandra (wide-column store), and Redis (key-value store).
- **In-memory Databases:** Store data primarily in the system's main memory (RAM) for faster access. Examples include Redis and Memcached.

Databases are fundamental to many applications, ranging from simple data storage to complex systems powering enterprise-level applications, e-commerce platforms, content management systems, and more.

DATABASE ASSIGNMENT

2. What is Normalization?

Normalization is a process used in database design to organize a relational database structure efficiently and reduce data redundancy. The goal of normalization is to eliminate or minimize data anomalies, such as insertion, update, and deletion anomalies, by breaking down large tables into smaller, well-structured tables. This helps ensure data integrity and maintainability in a relational database.

The normalization process involves applying a set of rules, usually up to a certain normal form, to a database schema. The most commonly used normal forms are the first normal form (1NF), second normal form (2NF), and third normal form (3NF). Higher normal forms, such as Boyce-Codd normal form (BCNF) and fourth normal form (4NF), exist but are less commonly used in practice.

Here's a brief overview of the first three normal forms:

1. **First Normal Form (1NF):** A table is in 1NF if it contains only atomic (indivisible) values, and there are no repeating groups or arrays of values in any column. Each attribute must contain only a single value.
2. **Second Normal Form (2NF):** A table is in 2NF if it is in 1NF and all non-key attributes are fully functionally dependent on the primary key. This means that there should be no partial dependencies, where only part of a composite key determines an attribute.
3. **Third Normal Form (3NF):** A table is in 3NF if it is in 2NF and there are no transitive dependencies. Transitive dependencies occur when a non-key attribute is dependent on another non-key attribute.

By applying normalization techniques, databases become more flexible, scalable, and less prone to data anomalies. However, it's important to note that normalization also introduces additional tables and relationships, which may slightly impact query performance. Database designers must strike a balance between normalization and denormalization based on the specific requirements and performance considerations of the application.

DATABASE ASSIGNMENT

3. What is Difference between DBMS and RDBMS?

DBMS (Database Management System) and RDBMS (Relational Database Management System) are terms often used in the context of databases, but they have distinct differences:

DBMS (Database Management System):

1. **Definition:**
   - DBMS is a software that manages databases. It provides an interface for interacting with the database, and it offers tools for creating, retrieving, updating, and managing data in the database.
2. **Data Model:**
   - DBMS may or may not follow a specific data model. It can be hierarchical, network, or relational, among others.
3. **Schema:**
   - In a DBMS, the schema defines the structure of the database, but it may not enforce relationships or constraints between tables.
4. **Data Integrity:**
   - DBMS may or may not support referential integrity and other advanced constraints.
5. **Example:**
   - Examples of DBMS include Microsoft Access, SQLite, and FileMaker.

RDBMS (Relational Database Management System):

1. **Definition:**
   - RDBMS is a type of DBMS that specifically follows the relational model of data. It organizes data into tables with rows and columns, and it establishes relationships between tables.
2. **Data Model:**
   - RDBMS strictly adheres to the relational data model. It uses tables to store data and supports the principles of normalization.
3. **Schema:**
   - RDBMS enforces relationships and constraints between tables, maintaining data integrity.
4. **Data Integrity:**
   - RDBMS typically supports referential integrity, ensuring that relationships between tables are maintained.
5. **Example:**

- Examples of RDBMS include MySQL, PostgreSQL, Oracle Database, and Microsoft SQL Server.

In summary, while all RDBMS systems are DBMS, not all DBMS systems are RDBMS. RDBMS is a specific type of DBMS that follows the principles of the relational model, enforces relationships and constraints, and supports SQL (Structured Query Language) for managing and querying data.

4. What is MF Cod Rule of RDBMS Systems?

5. What do you understand By Data Redundancy?

Data redundancy refers to the duplication of data in a database or information system. It occurs when the same piece of data is stored in multiple places within a database or across different databases. While some level of redundancy is often unavoidable, excessive redundancy can lead to various issues in a database system. Here are some key points to consider:

1. **Wasted Storage:** Storing the same information in multiple locations consumes additional storage space, which may be unnecessary and inefficient.
2. **Update Anomalies:** Redundant data can lead to inconsistencies when updates are made to one instance of the data but not to others. This can result in data inconsistencies and errors.
3. **Insertion Anomalies:** Inserting new data into the database may become problematic if all instances of the redundant data are not updated simultaneously. This can lead to incomplete or inaccurate information.
4. **Deletion Anomalies:** Removing data from one location may cause issues if the same data is needed elsewhere. This can result in unintentional loss of related information.
5. **Complexity:** Redundant data can make database systems more complex and harder to maintain, especially as the volume of data increases.

To mitigate data redundancy, database designers often use normalization techniques. Normalization involves organizing data in a way that

minimizes redundancy and dependency. This typically includes breaking down large tables into smaller, related tables and establishing relationships between them.

It's important to strike a balance between normalization and performance, as overly normalized databases may require complex joins and impact query performance. Database designers need to carefully consider the specific requirements and characteristics of the application when addressing data redundancy issues.

6. What is DDL Interpreter?

DDL (Data Definition Language) is a subset of SQL (Structured Query Language) used to define and manage the structure of a relational database. DDL statements are responsible for defining, altering, and dropping database objects such as tables, indexes, and views. The DDL interpreter, in the context of a database management system (DBMS), is the component or module that processes and executes these DDL statements.

The DDL interpreter performs the following key functions:

1. **Syntax Checking:** It checks the syntax of the DDL statements to ensure they adhere to the rules and structure of the SQL language. If there are syntax errors, it reports them to the user.
2. **Authorization and Security:** The DDL interpreter checks whether the user executing the DDL statement has the necessary privileges to perform the requested operation. It enforces security rules defined in the database.
3. **Metadata Management:** DDL statements define the structure of the database objects, and the DDL interpreter updates the metadata or data dictionary of the database. Metadata includes information about tables, columns, constraints, and other database objects.
4. **Object Creation and Modification:** The DDL interpreter is responsible for creating new database objects (e.g., tables) or modifying existing ones based on the DDL statements provided by the user.
5. **Transaction Handling:** DDL statements often implicitly commit the current transaction, as they involve changes to the database structure that should be permanent. The DDL interpreter manages these transactions accordingly.

Example DDL statements include:

- **CREATE TABLE:** Defines a new table with its columns, data types, and constraints.

- **ALTER TABLE:** Modifies the structure of an existing table, such as adding or dropping columns.
- **DROP TABLE:** Removes an existing table from the database.

In summary, the DDL interpreter plays a crucial role in processing and executing DDL statements, ensuring the integrity and security of the database structure.

7. What is DML Compiler in SQL?

In the context of SQL (Structured Query Language) and database management systems (DBMS), the term "DML Compiler" is not a standard term used to describe a specific component or process. However, I can provide information about the components related to SQL and DML (Data Manipulation Language).

DML (Data Manipulation Language) in SQL consists of the statements used for querying and modifying data in a database. Common DML statements include SELECT (for querying data), INSERT (for adding new records), UPDATE (for modifying existing records), and DELETE (for removing records).

In a database system, the processing of SQL queries and DML statements generally involves several components:

1. **Parser:** The parser checks the syntax of the SQL statements to ensure they adhere to the language rules. It breaks down the statements into a parse tree, a hierarchical structure that represents the syntactic structure of the SQL statement.
2. **Optimizer:** The optimizer analyzes the parse tree and generates an optimized execution plan for the query or DML statement. The goal is to determine the most efficient way to access and manipulate the data, considering factors such as indexes and available resources.
3. **Executor:** The executor is responsible for executing the optimized plan generated by the optimizer. It interacts with the storage engine to retrieve or modify the data based on the SQL statements.
4. **Transaction Manager:** This component manages transactions, ensuring the consistency and integrity of the database. It handles aspects such as commit and rollback operations.

DATABASE ASSIGNMENT

While these components are involved in processing SQL queries and DML statements, the term "DML Compiler" is not a standard term used in the same way. If you have a specific context or a different term in mind, please provide additional details, and I'll do my best to assist you.

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

In SQL, key constraints are used to enforce the integrity and uniqueness of data in a relational database. There are different types of key constraints, including PRIMARY KEY, UNIQUE, and FOREIGN KEY.

PRIMARY KEY Constraint:

- The PRIMARY KEY constraint uniquely identifies each record in a table.
- It must contain unique values and cannot have NULL values.

UNIQUE Constraint:

- The UNIQUE constraint ensures that all values in a column are distinct, similar to the PRIMARY KEY constraint. However, unlike PRIMARY KEY, a table can have multiple UNIQUE constraints.

FOREIGN KEY Constraint:

- The FOREIGN KEY constraint establishes a link between two tables by referencing a column in one table to the primary key column in another table.
- It ensures referential integrity by preventing actions that would break the relationships between tables.

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

A savepoint in a relational database is a point within a transaction to which you can roll back the transaction without rolling back the entire transaction. Savepoints are useful in situations where you want to apply partial changes and have the option to roll back to a specific point if needed.

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

In SQL, a trigger is a set of instructions that are automatically executed ("triggered") in response to certain events on a particular table or view. These events can include data manipulation events (such as INSERT, UPDATE, DELETE), or specific database actions (such as database startup). Triggers are commonly used to enforce business rules, perform logging, or maintain data integrity.

DATABASE ASSIGNMENT

1. Create Table Name : Student and Exam

Primary Key

Student

Rollno	Name	Branch
1	Jay	Computer Science
2	Suhani	Electronic and Com
3	Kriti	Electronic and Com

Foreign Key

Exam

Rollno	S_code	Marks	P_code
1	CS11	50	CS
1	CS12	60	CS
2	EC101	66	EC
2	EC102	70	EC
3	EC101	45	EC
3	EC102	50	EC

```
create table student
(
        Roll_no int primary key,
        Name varchar(20),
       Address varchar(30),
       Branch varchar(30);

create table Exam
(
        Roll_no int,
        S_code varchar(20),
       Marks varchar(30),
P_code varchar(30)
Foreign key(Roll_no) reference student(Roll_no);
```

2. Create table given below: Employee and IncentiveTable

```
create table sco
(
        firs_tname varchar(20),
        Last_name varchar(20),
       Address varchar(30),
       City varchar(10),
       age int
       );


insert into sco
VALUES
    ('micky','mouse','123fantasy way','anahelm',73 ),
    ('bat','man','321fantasy way','gotham',54),
    ('wonder','woman',' 987truth way','paradise',39 ),
    ('donald','duck','555quack stree','mallard',65 ),
    ('bugs','bunny','567carrot street','rascal',58 ),
    ('cat','woman','234purrfect street','hairball',32 ),
    ('micky','mouse','999fantasy way','anahelm',73 ),
    ('tweety','bird','543fantasy way','ititlaw',28 );
```

DATABASE ASSIGNMENT

first_name	last_name	address	city	age
micky	mouse	123fantasy way	anahelm	73
bat	man	321fantasy way	gotham	54
wonder	woman	987truth way	paradise	39
donald	duck	555quack street	mallard	65
bugs	bunny	567carrot street	rascal	58
cat	woman	234purrfect street	hairball	32
micky	mouse	999fantasy way	anahelm	73
tweety	bird	543fantasy way	ititlaw	28

```
CREATE TABLE incentives(employe_ref_id int,incentives_date date,inc
_amount INT, FOREIGN KEY(employe_ref_id) REFERENCES employee(employ
e_id));

INSERT INTO incentives(employe_ref_id,incentives_date,inc_amount) V
ALUES (1,2013-01-02,5000), (2,2013-01-02,3000), (3,2013-01-
02,4000), (1,2013-01-01,4500), (1,2013-01-01,3500);
```

employe_ref_id	incentives_date	inc_amount
1	0000-00-00	5000
2	0000-00-00	3000
3	0000-00-00	4000
1	0000-00-00	4500
1	0000-00-00	3500

3. Get First Name from employee table using Tom name “Employee Name”.

```
SELECT * FROM employee WHERE first_name='tom;';
```

employe_id	first_name	Last_name	salary	joining_date	Department
4	Tom	jose	600000	2024-01-02 18:12:37	insurance

4. Get FIRST_NAME, Joining Date, and Salary from employee table.

```
SELECT DISTINCT first_name,joining_date,salary FROM employee;
```

first_name	joining_date	salary
john	2024-01-02 18:12:37	1000000
michel	2024-01-02 18:12:37	8000000
Roy	2024-01-02 18:12:37	700000
Tom	2024-01-02 18:12:37	600000
jerry	2024-01-02 18:12:37	750000
phillip	2024-01-02 18:12:37	650000
testname_1	2024-01-02 18:12:37	60000

5. Get all employee details from the employee table order by First Name Ascending and Salary descending?

DATABASE ASSIGNMENT

(a) First name ascending

```
SELECT * FROM employee ORDER by first_name;
```

employe_id	first_name ▲ 1	Last_name	salary	joining_date	Department
5	jerry	pinto	750000	2024-01-02 18:12:37	insurance
1	john	abraham	1000000	2024-01-02 18:12:37	Banking
2	michel	cleark	8000000	2024-01-02 18:12:37	insurance
6	phillip	mathew	650000	2024-01-02 18:12:37	service
3	Roy	Thomus	700000	2024-01-02 18:12:37	Banking
7	testname_1	123	60000	2024-01-02 18:12:37	service
4	Tom	jose	600000	2024-01-02 18:12:37	insurance

(b) Salary descending

```
SELECT * FROM employee ORDER by salary DESC;
```

employe_id	first_name	Last_name	salary ▼ 1	joining_date	Department
2	michel	cleark	8000000	2024-01-02 18:12:37	insurance
1	john	abraham	1000000	2024-01-02 18:12:37	Banking
5	jerry	pinto	750000	2024-01-02 18:12:37	insurance
3	Roy	Thomus	700000	2024-01-02 18:12:37	Banking
6	phillip	mathew	650000	2024-01-02 18:12:37	service
4	Tom	jose	600000	2024-01-02 18:12:37	insurance
7	testname_1	123	60000	2024-01-02 18:12:37	service

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

```
SELECT * FROM employee WHERE first_name LIKE 'j%';
```

employe_id	first_name	Last_name	salary	joining_date	Department
1	john	abraham	1000000	2024-01-02 18:12:37	Banking
5	jerry	pinto	750000	2024-01-02 18:12:37	insurance

7. Get department wise maximum salary from employee table order by

```
SELECT max(salary) as maximum_salary FROM employee;
```

SELECT max(salary) as maximum_salary FROM employee;

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP co

Show all | Number of rows: 25 Filter rows:

Extra options

maximum_salary
8000000

8. salary ascending?

DATABASE ASSIGNMENT

```
SELECT * from employee ORDER by salary;
```

```
SELECT * from employee ORDER by salary;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

employe_id	first_name	Last_name	salary ▲ 1	joining_date	Department
7	testname_1	123	60000	2024-01-02 18:12:37	service
4	Tom	jose	600000	2024-01-02 18:12:37	insurance
6	phillip	mathew	650000	2024-01-02 18:12:37	service
3	Roy	Thomus	700000	2024-01-02 18:12:37	Banking
5	jerry	pinto	750000	2024-01-02 18:12:37	insurance
1	john	abraham	1000000	2024-01-02 18:12:37	Banking
2	michel	cleark	8000000	2024-01-02 18:12:37	insurance

9. Select first name, incentive amount from employee and incentives table for those employees who have incentives and incentive amount greater than 3000

first give primary id to employe id in employe table
ALTER TABLE employee ADD PRIMARY KEY employee(employe_id);

Than give foreign key id to employe_ref_id in incentives table.

than

```
SELECT e.first_name, i.inc_amount FROM employee e INNER JOIN incentives i
 ON e.employe_id = i.employe_ref_id WHERE i.inc_amount > 3000;
```

first_name	inc_amount
john	5000
Roy	4000
john	4500
john	3500

10. Create After Insert trigger on Employee table which insert records in view table

```
CREATE TABLE view_table
(
id int,
name varchar(30),
date_time timestamp,
action_ text);
```

DATABASE ASSIGNMENT

```
CREATE TRIGGER tri_insert AFTER INSERT on employee
 for EACH ROW
BEGIN
 INSERT INTO view_table(id,name,action_) VALUES (new.first_name,'Record
inserted');
END;
```

11.Create table given below: Salesperson and Customer

TABLE NAME- SALSEPERSON

(PK)SNo	SNAME	CITY	COMM
1001	Peel	London	.12
1002	Serres	San Jose	.13
1004	Motika	London	.11
1007	Rafkin	Barcelona	.15
1003	Axelrod	New York	.1

TABLE-2

TABLE NAME- CUSTOMER

(PK)CNM.	CNAME	CITY	RATING	(FK)SNo
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

```
CREATE TABLE salesperson
(Sno int PRIMARY KEY,
Sname varchar(20),
city varchar(20),
COMM int);
```

```
INSERT into salesperson
(Sno,Sname,city,COMM)VALUES
(1001,'Peel','London',12),
(1002,'Serres','San jose',13),
(1004,'Mortika','London',11),
(1007,'Rafkin','Barcelona',15),
(1003,'Axelord','New york',1);
```

	Sno	Sname	city	COMM
Edit Copy Delete	1001	Peel	London	12
Edit Copy Delete	1002	Serres	San jose	13
Edit Copy Delete	1003	Axelord	New york	1
Edit Copy Delete	1004	Mortika	London	11
Edit Copy Delete	1007	Rafkin	Barcelona	15

For coustomer

```
CREATE table Customer
(CNM int PRIMARY KEY,
CNAME varchar(20),
```

DATABASE ASSIGNMENT

```
CITY varchar(20),
RATING int,
Sno1 int,
FOREIGN key(Sno1) REFERENCES salesperson(Sno));

INSERT INTO customer(CNM,CNAME,CITY,RATING,Sno1)
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);
```

	CNM	CNAME	CITY	RATING	Sno1
Edit Copy Delete	201	Hoffman	London	100	1001
Edit Copy Delete	202	Giovanne	Roe	200	1003
Edit Copy Delete	203	Liu	San jose	300	1002
Edit Copy Delete	204	Grass	Barcelona	100	1002
Edit Copy Delete	206	Clemens	London	300	1007
Edit Copy Delete	207	Pereira	Roe	100	1004

12.Retrieve the below data from above table
13.All orders for more than $1000.

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

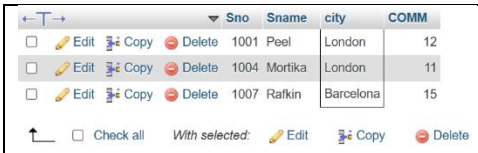
```
SELECT * FROM salesperson WHERE city='london' AND COMM>0.12;
```

	Sno	Sname	city	COMM
Edit Copy Delete	1001	Peel	London	12
Edit Copy Delete	1004	Mortika	London	11

15.All salespeople either in Barcelona or in London

```
SELECT * FROM `salesperson` WHERE city='london' or city='barcelona';
```

DATABASE ASSIGNMENT



	Sno	Sname	city	COMM
Edit Copy Delete	1001	Peel	London	12
Edit Copy Delete	1004	Mortika	London	11
Edit Copy Delete	1007	Rafkin	Barcelona	15

Check all With selected: Edit Copy Delete

16.All salespeople with commission between 0.10 and 0.12.(Boundary valuesshould be excluded).

```
SELECT * FROM salesperson WHERE COMM>0.10 AND COMM<0.12;
```

17.All customers excluding those with rating <= 100 unless they are locatedinRome

```
SELECT * FROM customer WHERE RATING<=100;
```

	CNM	CNAME	CITY	RATING	Sno1
Edit Copy Delete	201	Hoffman	London	100	1001
Edit Copy Delete	204	Grass	Barcelona	100	1002
Edit Copy Delete	207	Pereira	Roe	100	1004

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

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

```
salesman_id |    name    |   city    | commission
------------+------------+-----------+------------
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
```

```
CREATE TABLE salesperson_1
(salesman_id int,name varchar(20),city varchar(20),commission float );
```

```
INSERT INTO salesperson_1
(salesman_id,name,city,commission)
```

DATABASE ASSIGNMENT

```
VALUES
(5001,'James Hoog','New York',0.15),
(5002,'Nail Alex','Paris',0.13),
(5005,'Pit Alex','London',0.11),
(5006,'Mc Lyon','Paris',0.14),
(5003,'Lauson hen','San Jose',0.12);
```

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Alex	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
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.

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.

Sample table: orders

ord_no	purch_amt	ord_date	customer_id	salesman_id
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

```
CREATE TABLE orders
(ord_no int,punch_amt int,
ord_date date,
customer_id int,salesman_id_1 INT,
FOREIGN KEY(salesman_id_1)REFERENCES salesperson_1(salesman_id));

INSERT INTO orders(ord_no,ord_date,customer_id,salesman_id_1,purch_amt)
VALUES
(70001,2012-10-05,3005,5002,1505),
(70009,2012-09-10,3005,5002,270.65),
(70002,2012-10-05,3002,5001,65.26),
(70004,2012-08-17,3009,5003,110.5),
(70007,2012-09-17,3005,5002,948.5),
(70005,2012-07-27,3007,5001,2400.6),
 (70008,2012-09-10,3002,5001,5760),
(70010,2012-10-10,3004,5006,1983.43),
 (70003,2012-10-10,3009,5003,2480.4),
 (70012,2012-06-27,3008,5002,250.45),
```

DATABASE ASSIGNMENT

```
 (70011,2012-08-17,3003,null,75.29),
(70013,2012-04-25,3002,5001,3045.6);
```

ord_no	ord_date 1	customer_id	salesman_id_1	purch_amt
70001	0000-00-00	3005	5002	1505
70011	0000-00-00	3003	NULL	75.29
70012	0000-00-00	3008	5002	250.45
70003	0000-00-00	3009	5003	2480.4
70010	0000-00-00	3004	5006	1983.43
70008	0000-00-00	3002	5001	5760
70005	0000-00-00	3007	5001	2400.6
70007	0000-00-00	3005	5002	948.5
70004	0000-00-00	3009	5003	110.5
70002	0000-00-00	3002	5001	65.26
70009	0000-00-00	3005	5002	270.65
70013	0000-00-00	3002	5001	3045.6

```
SELECT * FROM orders WHERE salesman_id_1=5001;
```

ord_no	ord_date	customer_id	salesman_id_1	purch_amt
70002	0000-00-00	3002	5001	65.26
70005	0000-00-00	3007	5001	2400.6
70008	0000-00-00	3002	5001	5760
70013	0000-00-00	3002	5001	3045.6
70002	0000-00-00	3002	5001	65.26
70005	0000-00-00	3007	5001	2400.6
70008	0000-00-00	3002	5001	5760
70013	0000-00-00	3002	5001	3045.6

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. Sample table: item mast

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.

Sample table: item_mast

PRO_ID	PRO_NAME	PRO_PRICE	PRO_COM
101	Mother Board	3200.00	15
102	Key Board	450.00	16
103	ZIP drive	250.00	14
104	Speaker	550.00	16
105	Monitor	5000.00	11
106	DVD drive	900.00	12
107	CD drive	800.00	12
108	Printer	2600.00	13
109	Refill cartridge	350.00	13
110	Mouse	250.00	12

```
SELECT * FROM `item_mast` WHERE pro_price BETWEEN '200' AND '600';
```

DATABASE ASSIGNMENT

pro_id	pro_name	pro_price	pro_com
102	Key Board	450	16
103	Zip Driver	250	14
104	Speaker	550	16
109	Refill cartridge	350	13
110	Mouse	250	12

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

```
SELECT AVG (pro_price)avg_pro_price FROM item_mast;
```

Extra options

avg_pro_price
1435.0000

22.From the following table, write a SQL query to display the pro nameas 'Item Name' and pro priceas 'Price in Rs.' Sample table: item mast

```
ALTER TABLE item_mast CHANGE pro_name item_name varchar(30);

ALTER TABLE item_mast CHANGE pro_price price_in_rs int;
```

pro_id	item_name	price_in_rs	pro_com
101	Mother Board	3200	15
102	Key Board	450	16
103	Zip Driver	250	14
104	Speaker	550	16
105	Monitor	5000	11
106	DVD drive	900	12
107	CD drive	800	12
108	Printer	2600	13
109	Refill cartridge	350	13
110	Mouse	250	12

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. Sample table: item mast

```
SELECT * FROM item_mast WHERE price_in_rs>250;
```

DATABASE ASSIGNMENT

pro_id	item_name	price_in_rs	pro_com
101	Mother Board	3200	15
102	Key Board	450	16
103	Zip Driver	250	14
104	Speaker	550	16
105	Monitor	5000	11
106	DVD drive	900	12
107	CD drive	800	12
108	Printer	2600	13
109	Refill cartridge	350	13
110	Mouse	250	12

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.
Sample table: item_mast

```
SELECT PRO_COM AS company_code, AVG(price_in_rs) AS average_price FROM it
em_mast GROUP BY PRO_COM;
```

company_code	average_price
11	5000.0000
12	650.0000
13	1475.0000
14	250.0000
15	3200.0000
16	500.0000