MODULE: 5 (Database)

• What do you understand By Database

A database is a sophisticated and organized system for storing, managing, and retrieving data efficiently. In today's digital age, databases are foundational to nearly every aspect of our lives, from business operations and healthcare to entertainment and scientific research. This comprehensive overview will delve into the fundamental principles, components, and functions of databases, exploring how they work, the various types of databases, and their essential roles in modern information systems.

Understanding Databases

At its core, a database is a structured collection of data. This data can encompass a wide range of information, including text, numbers, dates, multimedia, and more. Databases are designed to organize this data systematically, ensuring that it can be easily stored, accessed, and managed. They serve as repositories for structured information, providing a coherent and efficient method of dealing with data.

Databases are structured using tables, which consist of rows and columns. These tables are a fundamental component of relational databases, which are among the most commonly used database systems. Rows represent individual records, while columns define the attributes or properties of the data. For instance, in a database of customer information, each row might represent a single customer, and the columns could include attributes such as name, address, phone number, and email.

• What is Normalization?

Normalization is a database design process that organizes and structures data in a relational database to reduce data redundancy and improve data integrity. It involves breaking down large tables into smaller, related tables and defining relationships between them. The main goal of normalization is to minimize data duplication, ensure data consistency, and reduce the likelihood of data anomalies, such as update and delete anomalies, which can occur when data is not properly organized.

Normalization is typically applied to relational databases, which consist of tables with rows and columns. Each table in a database represents a specific entity or concept, and the columns in the table represent attributes or properties of that entity. When data is normalized, it is organized into multiple related tables, with each table serving a specific purpose. The normalization process is divided into a series of normal forms, which are rules or guidelines for structuring the data. The most commonly used normal forms are:

First Normal Form (1NF): This is the starting point of normalization. In 1NF, a table is free of repeating groups and has a primary key that uniquely identifies each row. Each column in the table contains atomic (indivisible) values. For example, if you have a table of orders, it should not contain lists of items in a single column; instead, each item should have its own row.

Second Normal Form (2NF): To achieve 2NF, a table must first be in 1NF. Additionally, all non-key attributes (columns) must be fully functionally dependent on the entire primary key. This means that if a table has a composite primary key (i.e., a primary key with multiple columns), each non-key column should be dependent on the entire key, not just a part of it. If not, these columns should be moved to a separate table.
Third Normal Form (3NF): In 3NF, a table must first be in 2NF. Furthermore, it should eliminate transitive dependencies, meaning that non-key columns should not depend on other non-key columns. If such dependencies exist, the data should be further split into separate tables to eliminate them.
Boyce-Codd Normal Form (BCNF): BCNF is a more advanced form of normalization. It extends the concept of 3NF by addressing situations where there might be multiple candidate keys (sets of columns that could serve as the primary key). To achieve BCNF, a table should ensure that for any non-trivial functional dependency, the left-hand side (determinant) is a superkey. This means that a table must satisfy a stricter condition than 3NF.
Fourth Normal Form (4NF): 4NF focuses on multi-valued dependencies. If a table contains multi-valued attributes (attributes that have sets of values), it should be split into separate tables to avoid data duplication and ensure data integrity.
Fifth Normal Form (5NF): 5NF deals with cases where a table has overlapping multi-valued dependencies. It further refines the organization of data to eliminate any redundancy related to complex multi-valued attributes.

• What is Difference between DBMS and RDBMS?

DBMS (Database Management System) and RDBMS (Relational Database Management System) are both software systems that facilitate the management, storage, retrieval, and manipulation of data in a structured way. However, they have distinct differences in terms of their capabilities and how they handle data. Here are the key differences between DBMS and RDBMS:

Data Structure:

DBMS: Supports various data structures, including hierarchical, network, or flat-file structures.
RDBMS: Enforces a tabular structure with tables, rows, and columns, following the relational model.

Data Integrity:

DBMS: Provides basic data integrity features but does not enforce all relational database constraints.
RDBMS: Enforces rules and constraints like primary keys, foreign keys, and data type constraints to ensure data integrity.

Data Relationships:

DBMS: Manages data with various types of relationships but doesn't enforce strict relationships.
RDBMS: Enforces well-defined and enforced relationships using foreign keys to maintain referential integrity.

Data Retrieval:
DBMS: Offers basic data retrieval capabilities through simple query languages, which may not be as powerful as SQL in RDBMS.
RDBMS: Provides robust querying capabilities using SQL, allowing complex and efficient data retrieval and manipulation.

Scalability:
DBMS: May not be as scalable, especially for large volumes of structured data.
RDBMS: Designed with scalability in mind, supporting features like sharding and replication for handling large datasets and high user loads.

Data Redundancy:
DBMS: Allows for data redundancy, which can lead to data anomalies and inconsistencies.
RDBMS: Minimizes data redundancy through normalization techniques to ensure data consistency.

Complexity:
DBMS: Generally simpler and more flexible, suitable for scenarios where data relationships are not a primary concern.
RDBMS: More structured and can be more complex due to adherence to the relational model, ideal for applications where data relationships are critical.

● What is MF Cod Rule of RDBMS Systems?

E.F. Codd's Rules are a set of 12 rules that were introduced by Edgar F. Codd, a computer scientist who is often credited with the invention of the relational database model. These rules are a set of criteria that a database management system (DBMS) must meet to be considered a true relational database management system (RDBMS). The rules are designed to ensure data integrity, consistency, and flexibility in relational databases. Here is a summary of Codd's 12 Rules for RDBMS:
Rule 0: The Information Rule
All information in the database is to be represented in one and only one way – as values in a table.
Rule 1: The Information Principle
Information is to be presented in table form.
Rule 2: Guaranteed Access Rule
Each data value in the database is guaranteed to be accessible by specifying a table name, primary key value, and column name.
Rule 3: Systematic Treatment of Null Values
DBMS must allow each field to remain null. This is a special marker for missing information and is distinct from an empty string or zero.
Rule 4: Dynamic Online Catalog Based on The Relational Model

The database schema, including metadata, must be stored and managed like regular data, and it should be accessible using SQL.

Rule 5: Comprehensive Data Sublanguage Rule

The DBMS must support a data sublanguage that is equivalent in expressive power to the relational algebra. In other words, it should be capable of performing all necessary operations on data.

Rule 6: View Updating Rule

All views that are theoretically updatable must be updatable by the system. Views provide a virtual representation of data.

Rule 7: High-level Insert, Update, and Delete

The DBMS must support high-level insert, update, and delete operations. These operations should not require the user to write low-level code to perform them.

Rule 8: Physical Data Independence

The physical storage structure of the data (e.g., file organization) should be separate from the logical structure (the tables, rows, and columns).

Rule 9: Logical Data Independence

Changes to the logical structure (tables, rows, columns) should not affect application programs. Users and programs should remain unaffected by logical changes.

Rule 10: Integrity Independence

Integrity constraints (e.g., foreign keys, unique constraints) must be definable in the relational data sublanguage and stored in the catalog, not in application code.

Rule 11: Distribution Independence

The distribution of data across a network should not be apparent to users. Users should be able to access distributed data as if it were all stored in one location.

Rule 12: The Nonsubversion Rule

If a relational system has low-level (record-at-a-time) access, it must also have high-level (set-at-a-time) access. This prevents users from subverting the system by using low-level access to bypass the high-level integrity rules.

Meeting all 12 of Codd's rules is a stringent standard for an RDBMS, and many widely used commercial RDBMS systems, such as Oracle, Microsoft SQL Server, and PostgreSQL, strive to adhere to these rules to ensure the reliability and integrity of their databases.

• What do you understand By Data Redundancy?

Data redundancy refers to the duplication of data within a database or information system. It occurs when the same data is stored in multiple places or multiple times within a database or across different databases. Data redundancy can be intentional or unintentional and is typically considered an undesirable characteristic in database design. Here are some key points to understand about data redundancy:

1.Causes of Data Redundancy:

Data redundancy can result from various factors, including poor database design, data denormalization, the use of flat-file systems, or the absence of proper data management practices.

Intentional data redundancy may occur for performance optimization, data caching, or to avoid complex and resource-intensive joins when querying data.

2.Consequences of Data Redundancy:

Increased Storage: Storing the same data in multiple places consumes additional storage space, leading to increased storage costs.

Inconsistencies: Redundant data can lead to inconsistencies and data anomalies, as updates in one location may not be reflected in others.

Data Integrity Issues: Data anomalies, such as insertion, update, and deletion anomalies, can result from data redundancy. For example, updating a piece of data in one location may be forgotten in another, leading to inconsistent records.

Maintenance Challenges: Managing redundant data can be complex and error-prone, especially when changes need to be propagated to multiple locations.

Data Quality: Redundancy can lead to data quality issues, as it becomes challenging to ensure data accuracy and consistency.

3.Normalization to Reduce Redundancy:

Database normalization is a process used to reduce data redundancy by organizing data into related tables and eliminating duplicate information.

Normalization techniques involve breaking down large tables into smaller ones, defining relationships between tables, and ensuring that data is stored only once in a single place. This reduces redundancy and ensures data consistency.

4.Use of Unique Identifiers:

To avoid redundancy while still maintaining relationships between tables, unique identifiers (e.g., primary keys) are used. These identifiers link data in one table to related data in another without the need for duplicating information.

For example, in a relational database, each record in a child table can reference a unique identifier (primary key) from a parent table, allowing the child table to establish a relationship with the parent table.

5.Denormalization for Performance:

In some cases, denormalization, which intentionally introduces redundancy, is used to improve query performance. This is done by storing frequently used or computed data in a more accessible form.

Denormalization is often employed when read-heavy operations require faster data retrieval, even though it may lead to some trade-offs in terms of data integrity and storage.

6.Data Warehouse and Data Marts:

Data warehousing and data marts are specialized database systems that often incorporate redundant data for analytical purposes. Redundancy in these systems can enhance query performance and facilitate complex data analysis.

7.Balance Between Redundancy and Normalization:
Database designers must strike a balance between minimizing redundancy to ensure data integrity and optimizing the database for performance.
The appropriate level of redundancy in a database depends on the specific use case, the nature of the data, and the trade-offs between data integrity and query performance.

● What is DDL Interpreter?

A DDL (Data Definition Language) Interpreter is a component of a database management system (DBMS) responsible for processing and executing Data Definition Language statements. DDL is a subset of SQL (Structured Query Language) that is used for defining, managing, and modifying the structure and schema of a database, including tables, indexes, constraints, and other database objects. DDL statements are used to create, modify, or delete these database objects.
The DDL Interpreter performs the following key functions:
1.Parsing: It parses DDL statements to understand their syntax and structure. This involves breaking down the DDL statements into their constituent elements and ensuring they conform to the SQL language rules.
2.Validation: The interpreter checks the DDL statements for validity. It verifies that the requested operations can be performed without violating integrity constraints, data consistency, or database rules. For example, it checks if a proposed table structure conforms to the defined data types and constraints.
3.Execution: Once the DDL statements are parsed and validated, the DDL Interpreter carries out the requested database operations. This can include creating or altering database objects like tables, views, indexes, and constraints, or dropping (deleting) them.
4.Metadata Management: It updates the metadata of the database to reflect the changes made by the DDL statements. Metadata is data about the database itself, such as information about the tables, columns, indexes, and relationships.
5.Security and Permissions: The DDL Interpreter enforces security and permissions related to DDL operations. It ensures that only authorized users can execute DDL statements that modify the database's structure.

Examples of common DDL statements include:
- CREATE TABLE: Used to create a new table with specified columns and data types.
- ALTER TABLE: Used to modify the structure of an existing table, e.g., add or remove columns.
- DROP TABLE: Used to delete an existing table and its data.
- CREATE INDEX: Used to create an index on one or more columns for faster data retrieval.
- CREATE VIEW: Used to create a virtual table that presents data from one or more tables in a specific way.

The DDL Interpreter is an essential part of a DBMS, as it allows database administrators and users to define and manage the database's schema, ensuring that the data is stored and organized according to their requirements. It works in conjunction with the DML (Data Manipulation Language) interpreter, which is responsible for querying, inserting, updating, and deleting data in the database.

• What is DML Compiler in SQL?

In SQL (Structured Query Language), there isn't a specific component or concept known as a "DML Compiler." However, SQL includes DML (Data Manipulation Language) statements that are used to manipulate data within a database. These DML statements are typically executed by the SQL database management system (DBMS) without the need for a separate "DML Compiler."

DML statements in SQL are used for tasks such as:
1. SELECT: To retrieve data from one or more database tables.
2. INSERT: To add new rows of data into a table.
3. UPDATE: To modify existing data in a table.
4. DELETE: To remove data from a table.

These statements are executed directly within an SQL environment (e.g., through SQL client software, database management tools, or programming languages that support SQL) without the need for a separate compilation process.
It's important to understand that SQL is a declarative language, meaning that you specify what you want to do with the data, and the database system's query optimizer determines the most efficient way to perform the requested operation. In this sense, you don't typically deal with a "compiler" in the same way you would with a programming language like C or Java.
So, while SQL doesn't have a specific "DML Compiler," it does include DML statements for manipulating data, and the DBMS itself handles the execution of these statements.

• What is SQL Key Constraints writing an Example of SQL Key Constraints

In SQL, key constraints are used to enforce the uniqueness and integrity of data within a database table. There are three main types of key constraints: primary key, unique key, and foreign key. Here's an explanation of each, along with examples:

1.Primary Key Constraint:

A primary key constraint is used to ensure that a column (or a set of columns) in a table contains unique values and cannot contain NULL values. It is a unique identifier for each row in the table.
A table can have only one primary key.
Primary keys are often used to establish relationships between tables using foreign keys.
Example of defining a primary key in SQL:

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    DepartmentID INT
);
```

In this example, the "EmployeeID" column is defined as the primary key. It ensures that each "EmployeeID" value is unique in the "Employees" table.

2.Unique Key Constraint:

A unique key constraint ensures that the values in a column (or a set of columns) are unique, just like a primary key. However, it allows NULL values in the column.

A table can have multiple unique keys.

Example of defining a unique key in SQL:

```
CREATE TABLE Products (
   ProductID INT UNIQUE,
   ProductName VARCHAR(100) UNIQUE,
   Price DECIMAL(10, 2)
);
```

In this example, the "ProductID" and "ProductName" columns both have unique key constraints. They ensure that values in these columns are unique, but NULL values are allowed.

3.Foreign Key Constraint:

A foreign key constraint is used to establish a relationship between two tables. It enforces referential integrity by ensuring that values in a column in one table correspond to values in another table's primary key.

It helps maintain consistency and data integrity in the database.

Example of defining a foreign key in SQL:

```
CREATE TABLE Orders (
   OrderID INT PRIMARY KEY,
   CustomerID INT,
   OrderDate DATE,
   FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

In this example, the "CustomerID" column in the "Orders" table is a foreign key that references the "CustomerID" primary key in the "Customers" table. It ensures that a customer must exist in the "Customers" table before an order can be associated with them.

These key constraints are fundamental to maintaining data accuracy and relationships in a relational database management system (RDBMS) using SQL. They help ensure data integrity and enforce rules that define how data can be stored and related in the database.

● What is save Point? How to create a save Point write a Query?

A savepoint in a database management system is a point within a transaction where you can set a marker, allowing you to roll back to that specific point if needed. Savepoints are useful for breaking a complex transaction into smaller, more manageable parts and rolling back to a specific savepoint in case of errors or issues, without having to undo the entire transaction. Savepoints are typically used within the context of transactions, especially in systems that support the SQL standard.

Here's how you can create a savepoint and use it within a transaction using SQL:

```
-- Start a transaction
BEGIN;

-- Create a savepoint within the transaction
SAVEPOINT my_savepoint;

-- Execute SQL statements within the transaction
INSERT INTO Employees (EmployeeID, FirstName, LastName) VALUES (101, 'John', 'Doe');
UPDATE Employees SET DepartmentID = 2 WHERE EmployeeID = 101;

-- Suppose an error occurs, and you want to roll back to the savepoint
-- You can issue a ROLLBACK TO statement
ROLLBACK TO my_savepoint;

-- You can continue with the transaction, or if needed, you can choose to commit or rollback the entire transaction
-- For example, if everything is fine, you can commit the transaction:
-- COMMIT;

-- If there is an issue and you want to undo all changes made within the transaction, you can rollback the entire transaction:
-- ROLLBACK;

-- If you commit, the changes made before the savepoint (in this case, the INSERT statement) will be permanent, and the changes made after the savepoint (in this case, the UPDATE statement) will be undone.
-- If you rollback the entire transaction, all changes made within the transaction will be undone.

-- End the transaction
COMMIT;
```

In the example above:

1.We begin a transaction using BEGIN;.

2.We create a savepoint named my_savepoint using the SAVEPOINT statement.

3.We execute some SQL statements within the transaction, including an INSERT and an UPDATE.

4.If an error or issue occurs after the savepoint, we can roll back to the my_savepoint using ROLLBACK TO my_savepoint. This will undo changes made after the savepoint while keeping changes made before the savepoint intact.

5.Depending on the situation, you can choose to commit the entire transaction using COMMIT; or roll it back using ROLLBACK;.

Savepoints provide a way to maintain data integrity and consistency within a transaction while allowing for selective rollbacks when errors occur.

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

In SQL, a trigger is a database object that automatically executes a specified set of actions when a specific event occurs within a database table. Triggers are typically used to enforce data integrity rules, audit changes to data, or automate certain tasks based on events like INSERT, UPDATE, or DELETE operations on a table. There are two main types of triggers: "AFTER" triggers and "INSTEAD OF" triggers.

Here's how you can create a trigger in SQL:

1. AFTER Trigger:

An AFTER trigger is executed after an event (e.g., INSERT, UPDATE, DELETE) has occurred.

Example of creating an AFTER trigger:

```
CREATE TRIGGER UpdateLastModified
AFTER UPDATE ON Employees
FOR EACH ROW
BEGIN
   UPDATE Employees
   SET LastModified = NOW()
   WHERE EmployeeID = NEW.EmployeeID;
END;
```

In this example, the trigger UpdateLastModified is created to execute after an UPDATE operation on the Employees table. It updates the LastModified column with the current timestamp when a row is updated.

2. INSTEAD OF Trigger:

An INSTEAD OF trigger is executed instead of the triggering event. These are commonly used with views, allowing you to customize the action to take when an event occurs.

Example of creating an INSTEAD OF trigger:

```
CREATE TRIGGER InsertInsteadOf
INSTEAD OF INSERT ON MyView
FOR EACH ROW
BEGIN
   INSERT INTO MyTable (Column1, Column2)
   VALUES (NEW.Column1, NEW.Column2);
END;
```

In this example, the trigger InsertInsteadOf is created for an INSTEAD OF INSERT operation on a view named MyView. Instead of inserting data into the view, it inserts data into the table MyTable.

Triggers are powerful, but they should be used with caution. They can have a significant impact on database performance and complexity, and they should be well-documented to ensure that their behavior is understood and maintained. It's important to create triggers that serve a specific purpose, such as enforcing business rules, and to thoroughly test them to ensure they work as intended.

Task :

1. Create Table Name : Student and Exam

Student :

CREATE TABLE Student (RollNo int(5)PRIMARY KEY, Name VARCHAR(20),Branch VARCHAR(20));

		Roll_No	Name	Branch
☐	Edit Copy Delete	1	Jay	Computer Science
☐	Edit Copy Delete	2	Suhani	Electronic and Com
☐	Edit Copy Delete	3	Kriti	Electronic and Com

Exam :

CREATE TABLE exam (Roll No int(5), S_code VARCHAR(20),Marks VARCHAR(20), P_code VARCHAR(20));

Roll No	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

---

2. Create table given below

CREATE TABLE table(First Name VARCHAR(20), Last Name VARCHAR(20),Address VARCHAR(20), City VARCHAR(20), Age int(10));

First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	73
Bat	Man	321 Cavern Ave	Gotham	54
Wonder	Woman	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	Woman	234 Purrfect Street	Hairball	32
Tweety	Bird	543	Itotltaw	28

3. Create table given below: Employee and Incentive

Table Name: Employee

	Employee_id	First_Name	Last_Name	Salary	Joining_Date	Department
Edit Copy Delete	1	John	Abraham	1000000	01-JAN-13 12.00.00 A	Banking
Edit Copy Delete	2	Michael	Clarke	800000	01-JAN-13 12.00.00 A	Insurance
Edit Copy Delete	3	Roy	Thomas	700000	01-FEB-13 12.00.00 A	Banking
Edit Copy Delete	4	Tom	Jose	600000	01-FEB-13 12.00.00 A	Insurance
Edit Copy Delete	5	Jerry	Pinto	650000	01-FEB-13 12.00.00 A	Insurance
Edit Copy Delete	6	Philip	Mathew	750000	01-JAN-13 12.00.00 A	Services
Edit Copy Delete	7	TestName1	123	650000	01-JAN-13 12.00.00 A	Services
Edit Copy Delete	8	TestName2	Lname%	600000	01-FEB-13 12.00.00 A	Insurance

Table Name: Incentive

Employee_ref_id	Incentive_date	Incentive_amount
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

a) Get First_Name from employee table using Tom name “Employee Name”

SELECT * FROM `employee` WHERE First_Name='Tom';

	Employee_id	First_Name	Last_Name	Salary	Joining_Date	Department
Edit Copy Delete	4	Tom	Jose	600000	01-FEB-13 12.00.00 A	Insurance

b) Get FIRST_NAME, Joining Date, and Salary from employee table.

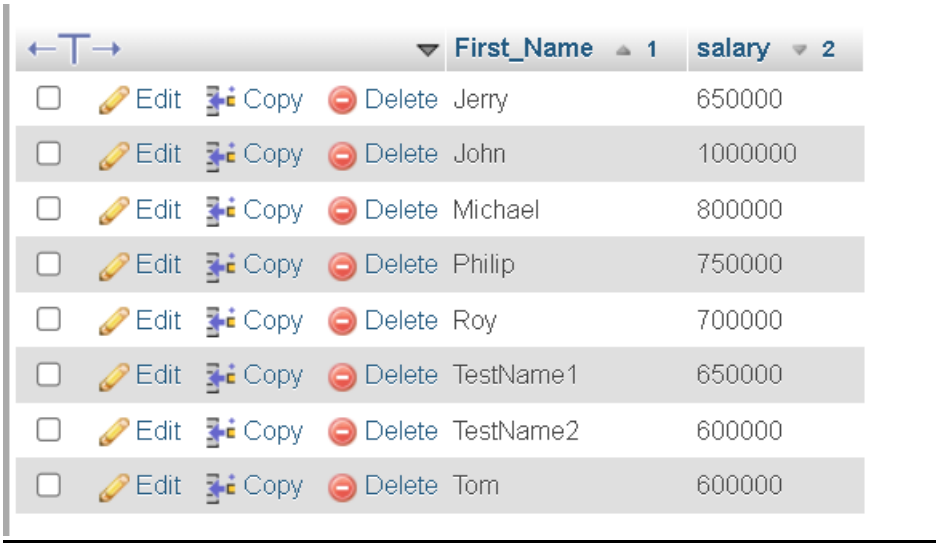
SELECT `First_Name`,`Salary`, `Joining_Date` FROM `employee`

	First_Name	Salary	Joining_Date
Edit Copy Delete	John	1000000	01-JAN-13 12.00.00 A
Edit Copy Delete	Michael	800000	01-JAN-13 12.00.00 A
Edit Copy Delete	Roy	700000	01-FEB-13 12.00.00 A
Edit Copy Delete	Tom	600000	01-FEB-13 12.00.00 A
Edit Copy Delete	Jerry	650000	01-FEB-13 12.00.00 A
Edit Copy Delete	Philip	750000	01-JAN-13 12.00.00 A
Edit Copy Delete	TestName1	650000	01-JAN-13 12.00.00 A
Edit Copy Delete	TestName2	600000	01-FEB-13 12.00.00 A

---

c) Get all employee details from the employee table order by First_Name Ascending and Salary descending?

SELECT First_Name, salary FROM employee ORDER BY First_Name, salary DESC;

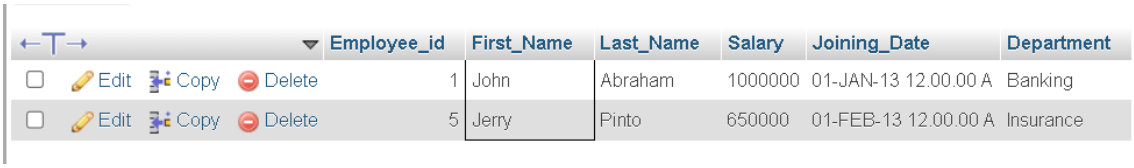


	First_Name ▲ 1	salary ▼ 2
Edit Copy Delete	Jerry	650000
Edit Copy Delete	John	1000000
Edit Copy Delete	Michael	800000
Edit Copy Delete	Philip	750000
Edit Copy Delete	Roy	700000
Edit Copy Delete	TestName1	650000
Edit Copy Delete	TestName2	600000
Edit Copy Delete	Tom	600000

---

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

SELECT * FROM employee WHERE First_Name LIKE 'J%'

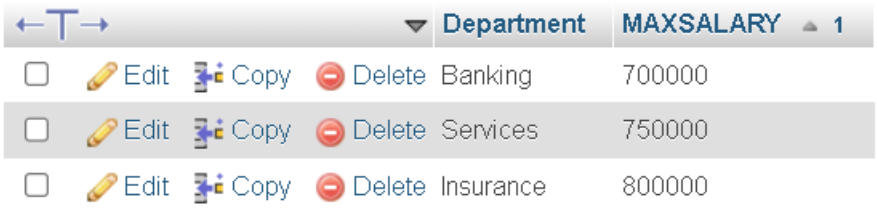


	Employee_id	First_Name	Last_Name	Salary	Joining_Date	Department
Edit Copy Delete	1	John	Abraham	1000000	01-JAN-13 12.00.00 A	Banking
Edit Copy Delete	5	Jerry	Pinto	650000	01-FEB-13 12.00.00 A	Insurance

---

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

SELECT Department,MAX(SALARY) MAXSALARY FROM employee GROUP BY Department ORDER BY MAXSALARY ASC;

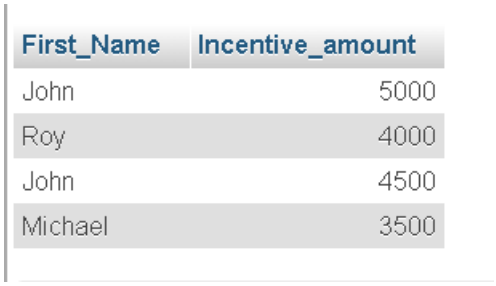


	Department	MAXSALARY
Edit Copy Delete	Banking	700000
Edit Copy Delete	Services	750000
Edit Copy Delete	Insurance	800000

---

f) Select first_name, incentive amount from employee and incentives table for those employees who have incentives and incentive amount greater than 3000

SELECT First_Name,Incentive_amount FROM employee A INNER JOIN incentive B ON A.Employee_id=B.Employee_ref_id AND Incentive_amount >3000;



First_Name	Incentive_amount
John	5000
Roy	4000
John	4500
Michael	3500

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

CREATE TRIGGER insertrecords ON employee AFTER INSERT AS BEGIN PRINT('Record(s) inserted successfully')END

4. Create table given below: Salesperson and Customer

Table-1 : Salesperson

(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 : 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

a) All orders for more than $1000.
Order details no available.

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

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

No Data Found…..

c) All salespeople either in Barcelona or in London

SELECT * FROM `salesperson` WHERE CITY='Barcelona' OR CITY='London'

(PK)SNo	SNAME	CITY	COMM
1001	Peel	London	.12
1004	Motika	London	.11
1007	Rafkin	Barcelona	.15

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

Select SNAME, COMM from salesperson where COMM > 0.10 and COMM < 0.12;

SNAME	COMM
Motika	.11

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

SELECT * FROM `customer` WHERE RATING<=100 AND CITY='Roe';

(PK)CNM.	CNAME	CITY	RATING	(FK)SNo
207	Pereira	Roe	100	1004