**Queries – INDIAN POLICE SYSTEM**

1. **Creating a Database**

mysql> CREATE DATABASE IndianPoliceService;

Query OK, 1 row affected (0.01 sec)

Explanation:

* This command creates a new database named IndianPoliceService.
* After executing this command, MySQL responds with Query OK, 1 row affected, indicating that the database was successfully created.

2. **Using the Database**

mysql> USE IndianPoliceService;

Database changed

Explanation:

* This command selects the IndianPoliceService database for use in subsequent operations.
* This SQL statement tells MySQL which database to work with for future queries.
* After executing this command, you see Database changed, which confirms that MySQL is now using the IndianPoliceService database, and any subsequent commands will be executed in the context of this database.

**DDL COMMANDS:**

**Creating tables:**

1. **Creating the Station Table**

mysql> CREATE TABLE Station (

-> StationID INT PRIMARY KEY AUTO\_INCREMENT,

-> StationName VARCHAR(100) NOT NULL,

-> Location VARCHAR(100) NOT NULL,

-> District VARCHAR(100) NOT NULL

-> );

Query OK, 0 rows affected (0.03 sec)

This command creates a table named Station to store information about police stations.

1. **Creating the Officer Table**

mysql> CREATE TABLE Officer (

-> OfficerID INT PRIMARY KEY AUTO\_INCREMENT,

-> Name VARCHAR(100) NOT NULL,

-> `Rank` VARCHAR(50) NOT NULL,

-> DateOfJoining DATE NOT NULL,

-> StationID INT,

-> FOREIGN KEY (StationID) REFERENCES Station(StationID)

-> );

Query OK, 0 rows affected (0.04 sec)

This command creates a table named Officer to store details about police officers.

3. **Creating the Case Table**

mysql> CREATE TABLE `Case` (

-> CaseID INT PRIMARY KEY AUTO\_INCREMENT,

-> CaseType VARCHAR(100) NOT NULL,

-> DateRegistered DATE NOT NULL,

-> Status VARCHAR(50) DEFAULT 'Open',

-> OfficerID INT,

-> FOREIGN KEY (OfficerID) REFERENCES Officer(OfficerID)

-> );

Query OK, 0 rows affected (0.05 sec)

This command creates a table named Case to store details about police cases.

4. **Creating the Suspect Table**

mysql> CREATE TABLE Suspect (

-> SuspectID INT PRIMARY KEY AUTO\_INCREMENT,

-> Name VARCHAR(100) NOT NULL,

-> Age INT CHECK (Age > 0),

-> Address VARCHAR(255),

-> CaseID INT,

-> FOREIGN KEY (CaseID) REFERENCES `Case`(CaseID)

-> );

Query OK, 0 rows affected (0.06 sec)

This command creates a table named Suspect to store details about suspects involved in cases.

5. **Creating the Victim Table**

mysql> CREATE TABLE Victim (

-> VictimID INT PRIMARY KEY AUTO\_INCREMENT,

-> Name VARCHAR(100) NOT NULL,

-> Age INT CHECK (Age > 0),

-> Address VARCHAR(255),

-> CaseID INT,

-> FOREIGN KEY (CaseID) REFERENCES `Case`(CaseID)

-> );

Query OK, 0 rows affected (0.04 sec)

This command creates a table named Victim to store details about victims involved in cases.

EXPLANATION:

**Table Creation**

Creating a table in SQL involves defining the structure of the table that represents an entity in your database. This includes specifying the table name, the columns, their data types, and any constraints that ensure data integrity. Each table typically has a primary key, which uniquely identifies each record, and may include foreign keys to establish relationships with other tables. For example, when creating a Station table, you would include columns for StationID, StationName, Location, and District, ensuring that StationID is set as the primary key and is auto-incremented for uniqueness. This structured approach allows for organized data storage and efficient data retrieval.

**Inserting values:**

1. **Inserting Data into the Station Table**

mysql> INSERT INTO Station (StationName, Location, District) VALUES

-> ('Central Police Station', 'MG Road', 'Bangalore'),

-> ('East Police Station', 'Indiranagar', 'Bangalore'),

-> ('West Police Station', 'Rajajinagar', 'Bangalore'),

-> ('North Police Station', 'Yeshwanthpur', 'Bangalore'),

-> ('South Police Station', 'Jayanagar', 'Bangalore');

Query OK, 5 rows affected (0.02 sec)

Records: 5 Duplicates: 0 Warnings: 0

2. **Inserting Data into the Officer Table**

mysql> INSERT INTO Officer (Name, `Rank`, DateOfJoining, StationID) VALUES

-> ('Amit Kumar', 'Inspector', '2010-05-01', 1),

-> ('Priya Sharma', 'Sub-Inspector', '2012-03-15', 2),

-> ('Ravi Desai', 'Inspector', '2009-11-23', 3),

-> ('Anjali Verma', 'Inspector', '2011-07-10', 4),

-> ('Rakesh Nair', 'Sub-Inspector', '2013-06-20', 5);

Query OK, 5 rows affected (0.01 sec)

Records: 5 Duplicates: 0 Warnings: 0

3. **Inserting Data into the Case Table**

mysql> INSERT INTO `Case` (CaseType, DateRegistered, Status, OfficerID) VALUES

-> ('Robbery', '2023-01-10', 'Open', 1),

-> ('Homicide', '2023-02-05', 'Closed', 2),

-> ('Kidnapping', '2023-03-12', 'Open', 3),

-> ('Assault', '2023-04-18', 'Under Investigation', 4),

-> ('Fraud', '2023-05-22', 'Open', 5);

Query OK, 5 rows affected (0.01 sec)

Records: 5 Duplicates: 0 Warnings: 0

4. **Inserting Data into the Suspect Table**

mysql> INSERT INTO Suspect (Name, Age, Address, CaseID) VALUES

-> ('Rahul Singh', 32, 'MG Road, Bangalore', 1),

-> ('Arjun Patel', 28, 'Indiranagar, Bangalore', 2),

-> ('Vikas Khanna', 35, 'Rajajinagar, Bangalore', 3),

-> ('Suresh Rao', 29, 'Yeshwanthpur, Bangalore', 4),

-> ('Ravi Gupta', 40, 'Jayanagar, Bangalore', 5);

Query OK, 5 rows affected (0.01 sec)

Records: 5 Duplicates: 0 Warnings: 0

5. **Inserting Data into the Victim Table**

mysql> INSERT INTO Victim (Name, Age, Address, CaseID) VALUES

-> ('Manoj Kumar', 45, 'MG Road, Bangalore', 1),

-> ('Meera Joshi', 38, 'Indiranagar, Bangalore', 2),

-> ('Nisha Rao', 30, 'Rajajinagar, Bangalore', 3),

-> ('Aarti Desai', 27, 'Yeshwanthpur, Bangalore', 4),

-> ('Pradeep Verma', 50, 'Jayanagar, Bangalore', 5);

Query OK, 5 rows affected (0.01 sec)

Records: 5 Duplicates: 0 Warnings: 0

Explanation:

**Insertion of Values**

* Use the INSERT INTO statement to add new records to a table.
* Specify the table name followed by the column names in parentheses.
* Provide the values for each column in the same order, ensuring data types match the column definitions.
* For multiple records, use a comma-separated list of value tuples.
* It's important to ensure that any foreign key values correspond to valid primary key entries in the related table to maintain referential integrity.

**DDL COMMANDS**

mysql> -- DDL CREATING CHILD TABLE AND RENAMING

Query OK, 0 rows affected (0.00 sec)

* **CREATING CHILD TABLE**

mysql> CREATE TABLE child\_table1 (

-> ChildID INT PRIMARY KEY AUTO\_INCREMENT,

-> Name VARCHAR(100) NOT NULL,

-> OfficerID INT,

-> FOREIGN KEY (OfficerID) REFERENCES Officer(OfficerID)

-> );

Query OK, 0 rows affected (0.05 sec)

Explanation:

This command creates a new table named child\_table1 intended to store information about subordinate officers who report to the officers in the Officer table.

* **RENAME**

mysql> RENAME TABLE child\_table1 TO SubordinateOfficer;

Query OK, 0 rows affected (0.03 sec)

Explanation:

This command renames the child\_table1 to SubordinateOfficer. The new name provides clarity about the table’s purpose, which is to store details of subordinate officers.

mysql> show tables;

+-------------------------------+

| Tables\_in\_indianpoliceservice |

+-------------------------------+

| case |

| officer |

| station |

| subordinateofficer |

| suspect |

| victim |

+-------------------------------+

6 rows in set (0.03 sec)

This shows all the tables.

* **Inserting Data**

mysql> INSERT INTO SubordinateOfficer (Name, OfficerID) VALUES

-> ('Sanjay Menon', 1),

-> ('Vikram Rao', 2),

-> ('Neha Kapoor', 3),

-> ('Arun Kumar', 4),

-> ('Preeti Verma', 5);

Query OK, 5 rows affected (0.01 sec)

Records: 5 Duplicates: 0 Warnings: 0

Explanation:

Each record consists of the subordinate officer's name and the OfficerID of the officer they report to. The numbers (1, 2, 3, etc.) correspond to the IDs of officers previously added to the Officer table.

mysql> select \* from SubordinateOfficer;

+---------+--------------+-----------+

| ChildID | Name | OfficerID |

+---------+--------------+-----------+

| 1 | Sanjay Menon | 1 |

| 2 | Vikram Rao | 2 |

| 3 | Neha Kapoor | 3 |

| 4 | Arun Kumar | 4 |

| 5 | Preeti Verma | 5 |

+---------+--------------+-----------+

5 rows in set (0.01 sec)

This command retrieves all records from the SubordinateOfficer table, allowing you to view the data that has been inserted.

mysql> -- RENAME

Query OK, 0 rows affected (0.00 sec)

* **Renaming the SubordinateOfficer Table**

mysql> RENAME TABLE SubordinateOfficer TO OfficerAssistant;

Query OK, 0 rows affected (0.03 sec)

Explanation:

This command renames the SubordinateOfficer table to OfficerAssistant. This renaming may be for clarity, indicating that these officers assist in their respective duties.

* **Truncating the OfficerAssistant Table**

mysql> TRUNCATE TABLE OfficerAssistant;

Query OK, 0 rows affected (0.05 sec)

Explanation:

This command removes all records from the OfficerAssistant table without deleting the table structure itself. It is a faster way to clear out a table compared to using DELETE because it does not log individual row deletions.

* **Selecting Data from the OfficerAssistant Table**

mysql> select \* from OfficerAssistant;

Empty set (0.00 sec)

Explanation:

This command retrieves all records from the OfficerAssistant table after truncation to confirm that the table is indeed empty.

* **Dropping the OfficerAssistant Table**

mysql> drop table OfficerAssistant;

Query OK, 0 rows affected (0.03 sec)

Explanation:

This command permanently removes the OfficerAssistant table and its structure from the database. This is typically done when the table is no longer needed.

mysql> -- DML commands

Query OK, 0 rows affected (0.00 sec)

**DML COMMANDS**

mysql> CREATE TABLE SubordinateOfficer (

-> ChildID INT AUTO\_INCREMENT PRIMARY KEY,

-> Name VARCHAR(100) NOT NULL,

-> OfficerID INT,

-> FOREIGN KEY (OfficerID) REFERENCES Officer(OfficerID)

-> );

Query OK, 0 rows affected (0.04 sec)

mysql> INSERT INTO SubordinateOfficer (Name, OfficerID) VALUES

-> ('Rohit Sharma', 1),

-> ('Sneha Iyer', 2),

-> ('Rakesh Gupta', 3),

-> ('Neelam Raj', 4),

-> ('Vikas Shetty', 5);

Query OK, 5 rows affected (0.01 sec)

Records: 5 Duplicates: 0 Warnings: 0

mysql> -- update

Query OK, 0 rows affected (0.00 sec)

* **Update Statement (Initial)**

mysql> UPDATE SubordinateOfficer

-> SET ChildID = OfficerID + 10

-> WHERE Name = 'Rohit Sharma';

Query OK, 1 row affected (0.01 sec)

Rows matched: 1 Changed: 1 Warnings: 0

Explanation:

This appears to be a comment indicating that an update operation will follow. It has no effect on the database.

* **Updating ChildID Based on Name**

mysql> -- Update the Child Table by Changing the Entity Name Using the Primary Key

Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE SubordinateOfficer

-> SET Name = 'Rohit Kumar'

-> WHERE ChildID = 11;

Query OK, 1 row affected (0.01 sec)

Rows matched: 1 Changed: 1 Warnings: 0

Explanation:

This command updates the ChildID for the row where the Name is 'Rohit Kumar'. It sets ChildID to the value of OfficerID + 10 for that specific record.

* **Adding Referential Integrity (Foreign Key Constraint)**

mysql> -- Add Referential Integrity (Foreign Key Constraint)

Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE SubordinateOfficer

-> ADD CONSTRAINT FK\_Officer\_Subordinate

-> FOREIGN KEY (OfficerID) REFERENCES Officer(OfficerID);

Query OK, 5 rows affected (0.07 sec)

Records: 5 Duplicates: 0 Warnings: 0

Explanation:

This command adds a foreign key constraint to the SubordinateOfficer table, linking OfficerID in SubordinateOfficer to OfficerID in the Officer table. The constraint ensures that any OfficerID in SubordinateOfficer must correspond to an existing OfficerID in the Officer table, thus maintaining referential integrity.

* **Deleting Rows from SubordinateOfficer**

mysql> DELETE FROM SubordinateOfficer

-> WHERE OfficerID IN (4, 5);

Query OK, 2 rows affected (0.01 sec)

Explanation:

This command deletes records from the SubordinateOfficer table where the OfficerID is either 4 or 5.

* **Rename Child Table**

mysql> RENAME TABLE SubordinateOfficer TO OfficerAssistant;

Query OK, 0 rows affected (0.02 sec)

Explanation:

This command renames the SubordinateOfficer table to OfficerAssistant, potentially for clarity or to better reflect its purpose.

mysql> -- CONSTRAINTSC

Query OK, 0 rows affected (0.00 sec)

* **Modifying the Name Column in OfficerAssistant**

mysql> ALTER TABLE OfficerAssistant

-> MODIFY Name VARCHAR(100) NULL;

Query OK, 0 rows affected (0.12 sec)

Records: 0 Duplicates: 0 Warnings: 0

Explanation:

This command modifies the Name column in the OfficerAssistant table, allowing it to accept NULL values. Changing a column to allow NULL values can be useful when you want to indicate that a name might not be provided for some records.

mysql> desc officerassistant;

+-----------+--------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+-----------+--------------+------+-----+---------+----------------+

| ChildID | int | NO | PRI | NULL | auto\_increment |

| Name | varchar(100) | YES | | NULL | |

| OfficerID | int | YES | MUL | NULL | |

+-----------+--------------+------+-----+---------+----------------+

3 rows in set (0.02 sec)

Explanation:

The changes are seen in this.

**CLAUSE AND FUNCTIONS**

mysql> -- CLAUSE, FUNCTIONS

Query OK, 0 rows affected (0.00 sec)

* **WHERE CLAUSE**

mysql> SELECT \*

-> FROM Officer

-> WHERE DateOfJoining > '2020-01-01';

Empty set (0.01 sec)

mysql> SELECT \*

-> FROM Officer

-> WHERE DateOfJoining > '2021-08-20';

Empty set (0.00 sec)

mysql> SELECT \*

-> FROM Officer

-> WHERE DateOfJoining > '2013-06-20';

Empty set (0.00 sec)

mysql> SELECT \*

-> FROM Officer

-> WHERE DateOfJoining > '2009-11-23';

+-----------+--------------+---------------+---------------+-----------+

| OfficerID | Name | Rank | DateOfJoining | StationID |

+-----------+--------------+---------------+---------------+-----------+

| 1 | Amit Kumar | Inspector | 2010-05-01 | 1 |

| 2 | Priya Sharma | Sub-Inspector | 2012-03-15 | 2 |

| 4 | Anjali Verma | Inspector | 2011-07-10 | 4 |

| 5 | Rakesh Nair | Sub-Inspector | 2013-06-20 | 5 |

+-----------+--------------+---------------+---------------+-----------+

4 rows in set (0.00 sec)

Explanation:

Checked for officers who joined after specific dates. The query showed results for dates before 2009-11-23 and confirmed that officers joined on or after that date.

* **GROUP CLAUSE**

mysql> --GROUP BY Clause

SELECT Rank, COUNT(\*) AS NumberOfOfficers

' at line 1

mysql> -- group by clause

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT OfficerID, COUNT(\*) AS TotalCases, MAX(DateRegistered) AS LastCaseDate

-> FROM `Case`

-> GROUP BY OfficerID;

+-----------+------------+--------------+

| OfficerID | TotalCases | LastCaseDate |

+-----------+------------+--------------+

| 1 | 1 | 2023-01-10 |

| 2 | 1 | 2023-02-05 |

| 3 | 1 | 2023-03-12 |

| 4 | 1 | 2023-04-18 |

| 5 | 1 | 2023-05-22 |

+-----------+------------+--------------+

5 rows in set (0.01 sec)

mysql> SELECT CaseType, COUNT(\*) AS NumberOfCases

-> FROM `Case`

-> GROUP BY CaseType;

+------------+---------------+

| CaseType | NumberOfCases |

+------------+---------------+

| Robbery | 1 |

| Homicide | 1 |

| Kidnapping | 1 |

| Assault | 1 |

| Fraud | 1 |

+------------+---------------+

5 rows in set (0.00 sec)

Explanation:

 counted the total cases assigned to each officer and displayed the latest case date.

 counted the number of cases by type using the GROUP BY clause.

* **HAVING CLAUSE**

mysql> -- HAVING CLAUSE

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT OfficerID, COUNT(\*) AS NumberOfCases

-> FROM `Case`

-> GROUP BY OfficerID

-> HAVING COUNT(\*) > 2;

Empty set (0.00 sec)

mysql> SELECT \*

-> FROM Officer

-> ORDER BY Name ASC;

+-----------+--------------+---------------+---------------+-----------+

| OfficerID | Name | Rank | DateOfJoining | StationID |

+-----------+--------------+---------------+---------------+-----------+

| 1 | Amit Kumar | Inspector | 2010-05-01 | 1 |

| 4 | Anjali Verma | Inspector | 2011-07-10 | 4 |

| 2 | Priya Sharma | Sub-Inspector | 2012-03-15 | 2 |

| 5 | Rakesh Nair | Sub-Inspector | 2013-06-20 | 5 |

| 3 | Ravi Desai | Inspector | 2009-11-23 | 3 |

+-----------+--------------+---------------+---------------+-----------+

5 rows in set (0.00 sec)

* **ORDER BY CLAUSE**

mysql> -- order by clause

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT \*

-> FROM Officer

-> ORDER BY Name ASC;

+-----------+--------------+---------------+---------------+-----------+

| OfficerID | Name | Rank | DateOfJoining | StationID |

+-----------+--------------+---------------+---------------+-----------+

| 1 | Amit Kumar | Inspector | 2010-05-01 | 1 |

| 4 | Anjali Verma | Inspector | 2011-07-10 | 4 |

| 2 | Priya Sharma | Sub-Inspector | 2012-03-15 | 2 |

| 5 | Rakesh Nair | Sub-Inspector | 2013-06-20 | 5 |

| 3 | Ravi Desai | Inspector | 2009-11-23 | 3 |

+-----------+--------------+---------------+---------------+-----------+

5 rows in set (0.00 sec)

Explanation:

Retrieved and sorted the officers by name in ascending order.

* **Using DISTINCT and LIMIT**

mysql> -- distinct and limit keywords

Query OK, 0 rows affected (0.00 sec)

Explanation:

Fetched distinct case types from the Case table, limiting the results to three entries.

mysql> SELECT DISTINCT CaseType

-> FROM `Case`

-> LIMIT 3;

+------------+

| CaseType |

+------------+

| Robbery |

| Homicide |

| Kidnapping |

+------------+

3 rows in set (0.00 sec)

* **Aggregate Functions**

mysql> -- Aggregate Functions

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT MIN(DateOfJoining) AS EarliestJoiningDate

-> FROM Officer;

+---------------------+

| EarliestJoiningDate |

+---------------------+

| 2009-11-23 |

+---------------------+

1 row in set (0.00 sec)

Explanation:

calculated the earliest joining date of officers.

* **Pattern Matching with LIKE**

mysql> -- Pattern Matching Using LIKE

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT \*

-> FROM Officer

-> WHERE Name LIKE 'A%';

+-----------+--------------+-----------+---------------+-----------+

| OfficerID | Name | Rank | DateOfJoining | StationID |

+-----------+--------------+-----------+---------------+-----------+

| 1 | Amit Kumar | Inspector | 2010-05-01 | 1 |

| 4 | Anjali Verma | Inspector | 2011-07-10 | 4 |

+-----------+--------------+-----------+---------------+-----------+

2 rows in set (0.01 sec)

Explanation:

selected officers whose names start with 'A'

**JOIN**

**LEFT JOIN, INNER JOIN, and RIGHT JOIN operations to relate officers with their respective stations.**

mysql> -- JOIN

Query OK, 0 rows affected (0.00 sec)

mysql> -- LEFT JOIN Query

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT Officer.Name, Officer.Rank, Station.StationName

-> FROM Officer

-> LEFT JOIN Station ON Officer.StationID = Station.StationID;

+--------------+---------------+------------------------+

| Name | Rank | StationName |

+--------------+---------------+------------------------+

| Amit Kumar | Inspector | Central Police Station |

| Priya Sharma | Sub-Inspector | East Police Station |

| Ravi Desai | Inspector | West Police Station |

| Anjali Verma | Inspector | North Police Station |

| Rakesh Nair | Sub-Inspector | South Police Station |

+--------------+---------------+------------------------+

5 rows in set (0.00 sec)

mysql> -- INNER JOIN Query

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT Officer.Name, Officer.Rank, Station.StationName

-> FROM Officer

-> INNER JOIN Station ON Officer.StationID = Station.StationID;

+--------------+---------------+------------------------+

| Name | Rank | StationName |

+--------------+---------------+------------------------+

| Amit Kumar | Inspector | Central Police Station |

| Priya Sharma | Sub-Inspector | East Police Station |

| Ravi Desai | Inspector | West Police Station |

| Anjali Verma | Inspector | North Police Station |

| Rakesh Nair | Sub-Inspector | South Police Station |

+--------------+---------------+------------------------+

5 rows in set (0.00 sec)

mysql> -- RIGHT JOIN

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT Officer.Name, Station.StationName

-> FROM Officer

-> RIGHT JOIN Station ON Officer.StationID = Station.StationID;

+--------------+------------------------+

| Name | StationName |

+--------------+------------------------+

| Amit Kumar | Central Police Station |

| Priya Sharma | East Police Station |

| Ravi Desai | West Police Station |

| Anjali Verma | North Police Station |

| Rakesh Nair | South Police Station |

+--------------+------------------------+

5 rows in set (0.00 sec)

mysql> -- FULL OUTER JOIN

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT Officer.Name, Station.StationName

-> FROM Officer

-> LEFT JOIN Station ON Officer.StationID = Station.StationID

->

-> UNION ALL

->

-> SELECT Officer.Name, Station.StationName

-> FROM Officer

-> RIGHT JOIN Station ON Officer.StationID = Station.StationID

-> WHERE Officer.OfficerID IS NULL; -- To avoid duplicates

+--------------+------------------------+

| Name | StationName |

+--------------+------------------------+

| Amit Kumar | Central Police Station |

| Priya Sharma | East Police Station |

| Ravi Desai | West Police Station |

| Anjali Verma | North Police Station |

| Rakesh Nair | South Police Station |

+--------------+------------------------+

5 rows in set (0.00 sec)

Explanation:

Used LEFT JOIN and RIGHT JOIN combined with UNION ALL to emulate a full outer join.

**STRING FUNCTIONS**

mysql> -- String Functions

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT Name,

-> UPPER(Name) AS UppercaseName,

-> LOWER(Name) AS LowercaseName,

-> LENGTH(Name) AS NameLength

-> FROM Officer;

+--------------+---------------+---------------+------------+

| Name | UppercaseName | LowercaseName | NameLength |

+--------------+---------------+---------------+------------+

| Amit Kumar | AMIT KUMAR | amit kumar | 10 |

| Priya Sharma | PRIYA SHARMA | priya sharma | 12 |

| Ravi Desai | RAVI DESAI | ravi desai | 10 |

| Anjali Verma | ANJALI VERMA | anjali verma | 12 |

| Rakesh Nair | RAKESH NAIR | rakesh nair | 11 |

+--------------+---------------+---------------+------------+

5 rows in set (0.00 sec)

Explanation:

transformed officer names to uppercase, lowercase, and calculated their lengths.

**NUMERIC FUNCTIONS**

mysql> -- Numeric Functions

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT

-> COUNT(OfficerID) AS TotalOfficers,

-> MAX(DATEDIFF(CURDATE(), DateOfJoining)) AS MaxDaysSinceJoined,

-> MIN(DATEDIFF(CURDATE(), DateOfJoining)) AS MinDaysSinceJoined,

-> AVG(DATEDIFF(CURDATE(), DateOfJoining)) AS AvgDaysSinceJoined

-> FROM Officer;

+---------------+--------------------+--------------------+--------------------+

| TotalOfficers | MaxDaysSinceJoined | MinDaysSinceJoined | AvgDaysSinceJoined |

+---------------+--------------------+--------------------+--------------------+

| 5 | 5442 | 4137 | 4861.8000 |

+---------------+--------------------+--------------------+--------------------+

1 row in set (0.00 sec)

Explanation:

Summarized officer data by counting total officers and calculating the days since their joining.

**DATE – TIME FUNCTIONS**

mysql> -- Date-Time Functions

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT Name,

-> DateOfJoining,

-> YEAR(DateOfJoining) AS JoiningYear,

-> DATE\_FORMAT(DateOfJoining, '%W, %M %d, %Y') AS FormattedJoiningDate

-> FROM Officer;

+--------------+---------------+-------------+---------------------------+

| Name | DateOfJoining | JoiningYear | FormattedJoiningDate |

+--------------+---------------+-------------+---------------------------+

| Amit Kumar | 2010-05-01 | 2010 | Saturday, May 01, 2010 |

| Priya Sharma | 2012-03-15 | 2012 | Thursday, March 15, 2012 |

| Ravi Desai | 2009-11-23 | 2009 | Monday, November 23, 2009 |

| Anjali Verma | 2011-07-10 | 2011 | Sunday, July 10, 2011 |

| Rakesh Nair | 2013-06-20 | 2013 | Thursday, June 20, 2013 |

+--------------+---------------+-------------+---------------------------+

5 rows in set (0.00 sec)

Explanation:

Extracted the year from the joining date and formatted it for better readability.

**QUERY**

mysql> -- QUERY

Query OK, 0 rows affected (0.00 sec)

* **Self-Query**

mysql> -- Self-Query

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT o1.Name AS OfficerName, o1.DateOfJoining

-> FROM Officer o1

-> JOIN Officer o2 ON o1.DateOfJoining > o2.DateOfJoining

-> WHERE o2.Name = 'Priya Kumar';

Empty set (0.00 sec)

mysql> SELECT o1.Name AS OfficerName, o1.DateOfJoining

-> FROM Officer o1

-> JOIN Officer o2 ON o1.DateOfJoining > o2.DateOfJoining

-> WHERE o2.Name = 'Priya Sharma';

+-------------+---------------+

| OfficerName | DateOfJoining |

+-------------+---------------+

| Rakesh Nair | 2013-06-20 |

+-------------+---------------+

1 row in set (0.00 sec)

* **Sub Query**

mysql> -- Sub-query Involving More than 2 Tables

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT o.Name

-> FROM Officer o

-> WHERE o.OfficerID IN (

-> SELECT c.OfficerID

-> FROM `Case` c

-> INNER JOIN Suspect s ON c.CaseID = s.CaseID

-> WHERE s.Name = 'Ravi'

-> );

Empty set (0.01 sec)

mysql> SELECT o.Name

-> FROM Officer o

-> WHERE o.OfficerID IN (

-> SELECT c.OfficerID

-> FROM `Case` c

-> INNER JOIN Suspect s ON c.CaseID = s.CaseID

-> WHERE s.Name = 'Ravi Desai'

-> );

Empty set (0.00 sec)

Explanation;

attempted to find officers associated with a suspect named "Ravi" and checked for results.

* **Subquery with Two Different Tables**

mysql> -- Subquery with Two Different Tables using Relational Operators andGroup Functions

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT o.Name

-> FROM Officer o

-> WHERE o.OfficerID IN (

-> SELECT c.OfficerID

-> FROM `Case` c

-> GROUP BY c.OfficerID

-> HAVING COUNT(c.CaseID) > 1

-> );

Empty set (0.00 sec)

Explanation:

Attempted to retrieve officer names with more than one case assigned but returned an empty set, indicating no officers meet that criteria.

* **Conditional and Comparison Operators**

mysql> -- Conditional Operators and Comparison Operators in Subquery

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT o.Name

-> FROM Officer o

-> WHERE DATEDIFF(CURDATE(), o.DateOfJoining) > (

-> SELECT AVG(DATEDIFF(CURDATE(), DateOfJoining))

-> FROM Officer

-> );

+------------+

| Name |

+------------+

| Amit Kumar |

| Ravi Desai |

+------------+

2 rows in set (0.00 sec)

Explanation:

Selected officers who joined more than the average days ago; results included two officers: Amit Kumar and Ravi Desai.

* **Membership Operators**

mysql> -- Membership Operators with any

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT o.Name

-> FROM Officer o

-> WHERE DATEDIFF(CURDATE(), o.DateOfJoining) < ANY (

-> SELECT DATEDIFF(CURDATE(), DateOfJoining)

-> FROM Officer

-> );

+--------------+

| Name |

+--------------+

| Amit Kumar |

| Priya Sharma |

| Anjali Verma |

| Rakesh Nair |

+--------------+

4 rows in set (0.00 sec)

Explanation:

Retrieved names of officers who joined fewer days than any officer, resulting in four names.

**SET OPERATIONS**

mysql> -- Set Operations

Query OK, 0 rows affected (0.00 sec)

mysql> -- Union Example

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT o.Name

-> FROM Officer o

-> UNION

-> SELECT s.Name

-> FROM Suspect s;

+--------------+

| Name |

+--------------+

| Amit Kumar |

| Priya Sharma |

| Ravi Desai |

| Anjali Verma |

| Rakesh Nair |

| Rahul Singh |

| Arjun Patel |

| Vikas Khanna |

| Suresh Rao |

| Ravi Gupta |

+--------------+

10 rows in set (0.00 sec)

Explanation:

Combined officer names and suspect names, yielding ten unique entries.

* Set Difference without MINUS

mysql> -- Set Difference without Using MINUS Operator

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT o.Name

-> FROM Officer o

-> LEFT JOIN Suspect s ON o.Name = s.Name

-> WHERE s.Name IS NULL;

+--------------+

| Name |

+--------------+

| Amit Kumar |

| Priya Sharma |

| Ravi Desai |

| Anjali Verma |

| Rakesh Nair |

+--------------+

5 rows in set (0.00 sec)

Explanation:

Selected officers not listed as suspects, returning five names.

**CARTESIAN PRODUCT**

mysql> -- Cartesian Product

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT \*

-> FROM Officer AS o

-> CROSS JOIN Station AS s;

+-----------+--------------+---------------+---------------+-----------+-----------+------------------------+--------------+-----------+

| OfficerID | Name | Rank | DateOfJoining | StationID | StationID | StationName | Location | District |

+-----------+--------------+---------------+---------------+-----------+-----------+------------------------+--------------+-----------+

| 5 | Rakesh Nair | Sub-Inspector | 2013-06-20 | 5 | 1 | Central Police Station | MG Road | Bangalore |

| 4 | Anjali Verma | Inspector | 2011-07-10 | 4 | 1 | Central Police Station | MG Road | Bangalore |

| 3 | Ravi Desai | Inspector | 2009-11-23 | 3 | 1 | Central Police Station | MG Road | Bangalore |

| 2 | Priya Sharma | Sub-Inspector | 2012-03-15 | 2 | 1 | Central Police Station | MG Road | Bangalore |

| 1 | Amit Kumar | Inspector | 2010-05-01 | 1 | 1 | Central Police Station | MG Road | Bangalore |

| 5 | Rakesh Nair | Sub-Inspector | 2013-06-20 | 5 | 2 | East Police Station | Indiranagar | Bangalore |

| 4 | Anjali Verma | Inspector | 2011-07-10 | 4 | 2 | East Police Station | Indiranagar | Bangalore |

| 3 | Ravi Desai | Inspector | 2009-11-23 | 3 | 2 | East Police Station | Indiranagar | Bangalore |

| 2 | Priya Sharma | Sub-Inspector | 2012-03-15 | 2 | 2 | East Police Station | Indiranagar | Bangalore |

| 1 | Amit Kumar | Inspector | 2010-05-01 | 1 | 2 | East Police Station | Indiranagar | Bangalore |

| 5 | Rakesh Nair | Sub-Inspector | 2013-06-20 | 5 | 3 | West Police Station | Rajajinagar | Bangalore |

| 4 | Anjali Verma | Inspector | 2011-07-10 | 4 | 3 | West Police Station | Rajajinagar | Bangalore |

| 3 | Ravi Desai | Inspector | 2009-11-23 | 3 | 3 | West Police Station | Rajajinagar | Bangalore |

| 2 | Priya Sharma | Sub-Inspector | 2012-03-15 | 2 | 3 | West Police Station | Rajajinagar | Bangalore |

| 1 | Amit Kumar | Inspector | 2010-05-01 | 1 | 3 | West Police Station | Rajajinagar | Bangalore |

| 5 | Rakesh Nair | Sub-Inspector | 2013-06-20 | 5 | 4 | North Police Station | Yeshwanthpur | Bangalore |

| 4 | Anjali Verma | Inspector | 2011-07-10 | 4 | 4 | North Police Station | Yeshwanthpur | Bangalore |

| 3 | Ravi Desai | Inspector | 2009-11-23 | 3 | 4 | North Police Station | Yeshwanthpur | Bangalore |

| 2 | Priya Sharma | Sub-Inspector | 2012-03-15 | 2 | 4 | North Police Station | Yeshwanthpur | Bangalore |

| 1 | Amit Kumar | Inspector | 2010-05-01 | 1 | 4 | North Police Station | Yeshwanthpur | Bangalore |

| 5 | Rakesh Nair | Sub-Inspector | 2013-06-20 | 5 | 5 | South Police Station | Jayanagar | Bangalore |

| 4 | Anjali Verma | Inspector | 2011-07-10 | 4 | 5 | South Police Station | Jayanagar | Bangalore |

| 3 | Ravi Desai | Inspector | 2009-11-23 | 3 | 5 | South Police Station | Jayanagar | Bangalore |

| 2 | Priya Sharma | Sub-Inspector | 2012-03-15 | 2 | 5 | South Police Station | Jayanagar | Bangalore |

| 1 | Amit Kumar | Inspector | 2010-05-01 | 1 | 5 | South Police Station | Jayanagar | Bangalore |

+-----------+--------------+---------------+---------------+-----------+-----------+------------------------+--------------+-----------+

25 rows in set (0.00 sec)

Explanation:

Cross-joined Officer and Station tables, resulting in a full combination of all officers with all stations (25 rows).

**DIVISION**

mysql> -- DIVISION

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT OfficerID

-> FROM Officer

-> WHERE OfficerID IN (

-> SELECT o.OfficerID

-> FROM Officer o

-> JOIN Station s ON o.StationID = s.StationID

-> GROUP BY o.OfficerID

-> HAVING COUNT(DISTINCT s.StationID) = (SELECT COUNT(\*) FROM Station)

-> );

Empty set (0.01 sec)

Explanation:

Attempted to find officers assigned to all stations, but returned an empty set, indicating no officers met that requirement.

**Rename and Select**

mysql> -- Rename

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT o.OfficerID AS OfficerID,

-> o.Name AS OfficerName,

-> o.Rank AS OfficerRank

-> FROM Officer AS o;

+-----------+--------------+---------------+

| OfficerID | OfficerName | OfficerRank |

+-----------+--------------+---------------+

| 1 | Amit Kumar | Inspector |

| 2 | Priya Sharma | Sub-Inspector |

| 3 | Ravi Desai | Inspector |

| 4 | Anjali Verma | Inspector |

| 5 | Rakesh Nair | Sub-Inspector |

+-----------+--------------+---------------+

5 rows in set (0.00 sec)

Explanation:

Renamed columns in the SELECT query for clarity and displayed five officers with their respective ranks.

**TTL COMMANDS**

mysql> -- TCL COMMANDS

Query OK, 0 rows affected (0.00 sec)

* **Start transaction**

mysql> START TRANSACTION;

Query OK, 0 rows affected (0.00 sec)

mysql> -- Auto Commit (on/off)

Query OK, 0 rows affected (0.00 sec)

* **Auto commit off**

mysql> SET AUTOCOMMIT = 0;

Query OK, 0 rows affected (0.00 sec)

mysql> -- Insert a New Officer and a Case

Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO Officer (Name, `Rank`, DateOfJoining, StationID) VALUES

-> ('Deepak Mehta', 'Inspector', '2024-10-17', 1);

Query OK, 1 row affected (0.00 sec)

mysql>

mysql> INSERT INTO `Case` (CaseType, DateRegistered, Status, OfficerID) VALUES

-> ('Theft', '2024-10-17', 'Open', LAST\_INSERT\_ID());

Query OK, 1 row affected (0.00 sec)

mysql> -- SAVEPOINT

Query OK, 0 rows affected (0.00 sec)

mysql> SAVEPOINT BeforeSuspectInsert;

Query OK, 0 rows affected (0.00 sec)

mysql> -- Insert a Suspect

Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO Suspect (Name, Age, Address, CaseID) VALUES

-> ('Ravi Kumar', 30, 'MG Road, Bangalore', LAST\_INSERT\_ID());

Query OK, 1 row affected (0.00 sec)

mysql> -- ROLLBACK

Query OK, 0 rows affected (0.00 sec)

mysql> ROLLBACK TO SAVEPOINT BeforeSuspectInsert; -- Optionally roll back if the suspect insertion fails

Query OK, 0 rows affected (0.00 sec)

mysql> -- Commit the Transaction

Query OK, 0 rows affected (0.00 sec)

mysql> COMMIT;

Query OK, 0 rows affected (0.01 sec)

mysql> -- AUTOCOMMIT ON

Query OK, 0 rows affected (0.00 sec)

mysql> SET AUTOCOMMIT = 1;

Query OK, 0 rows affected (0.00 sec)

Explanation:

• Started a transaction and set auto-commit to off, inserted a new officer, and attempted to insert a suspect.

• Rolled back to the previous savepoint to avoid the suspect insertion.

• Committed the transaction successfully.

**VDL COMMANDS**

mysql> -- VDL COMMANDS

Query OK, 0 rows affected (0.00 sec)

mysql> -- Create a View Using an Original Table with All Fields

Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW View\_Officer AS

-> SELECT \* FROM Officer;

Query OK, 0 rows affected (0.01 sec)

mysql> -- Create a View from the Master Table with Selected Fields Only Satisfying Certain Conditions

Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW View\_Inspectors AS

-> SELECT OfficerID, Name, `Rank`, StationID

-> FROM Officer

-> WHERE `Rank` = 'Inspector';

Query OK, 0 rows affected (0.01 sec)

mysql> -- EQUI JOIN

Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW View\_Officer\_Station AS

-> SELECT o.Name AS OfficerName, s.StationName

-> FROM Officer o

-> JOIN Station s ON o.StationID = s.StationID;

Query OK, 0 rows affected (0.01 sec)

mysql> -- Update a View

Query OK, 0 rows affected (0.00 sec)

mysql> -- First, make sure the view is defined

Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW View\_Officer AS

-> SELECT \* FROM Officer;

ERROR 1050 (42S01): Table 'View\_Officer' already exists

mysql>

mysql> -- Update the Officer directly

Query OK, 0 rows affected (0.00 sec)

mysql>

mysql> -- Verification

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT \* FROM Officer WHERE OfficerID = 1; -- Check the Officer table

+-----------+------------+-----------+---------------+-----------+

| OfficerID | Name | Rank | DateOfJoining | StationID |

+-----------+------------+-----------+---------------+-----------+

| 1 | Amit Kumar | Inspector | 2010-05-01 | 1 |

+-----------+------------+-----------+---------------+-----------+

1 row in set (0.00 sec)

mysql> SELECT \* FROM View\_Officer WHERE OfficerID = 1; -- Check the view

+-----------+------------+-----------+---------------+-----------+

| OfficerID | Name | Rank | DateOfJoining | StationID |

+-----------+------------+-----------+---------------+-----------+

| 1 | Amit Kumar | Inspector | 2010-05-01 | 1 |

+-----------+------------+-----------+---------------+-----------+

1 row in set (0.00 sec)

mysql> -- Delete Multiple Records from the Master Table and Verify

Query OK, 0 rows affected (0.00 sec)

mysql> -- Delete multiple suspects

Query OK, 0 rows affected (0.00 sec)

mysql> DELETE FROM Suspect

-> WHERE SuspectID IN (1, 2); -- Deleting suspects with IDs 1 and 2

Query OK, 2 rows affected (0.00 sec)

mysql>

mysql> -- Verification

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT \* FROM Suspect; -- Check the Suspect table

+-----------+--------------+------+-------------------------+--------+

| SuspectID | Name | Age | Address | CaseID |

+-----------+--------------+------+-------------------------+--------+

| 3 | Vikas Khanna | 35 | Rajajinagar, Bangalore | 3 |

| 4 | Suresh Rao | 29 | Yeshwanthpur, Bangalore | 4 |

| 5 | Ravi Gupta | 40 | Jayanagar, Bangalore | 5 |

+-----------+--------------+------+-------------------------+--------+

3 rows in set (0.00 sec)

mysql> -- Insert Records into a View and Verify

Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW View\_Officer\_Selected AS

-> SELECT OfficerID, Name

-> FROM Officer;

Query OK, 0 rows affected (0.01 sec)

Explanation:

 **View Commands**:

* Created views for officers and inspectors, as well as a view joining officers with their stations.
* Attempted to create a view that already exists, resulting in an error.

 **Verification**:

* Checked individual officer and view records to confirm data consistency.

 **Delete Multiple Records**:

* Deleted specific suspects and confirmed their removal by querying the Suspect table.

 **Insert Records into a View**:

* Attempted to create a new view for selected officers but noted that views cannot be directly modified unless the underlying table allows it.

mysql> -- Update the View Using IF-ELSE-Like Condition

Query OK, 0 rows affected (0.00 sec)

mysql> -- Update case status based on condition

Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE `Case`

-> SET Status = CASE

-> WHEN CaseType = 'Robbery' THEN 'Under Investigation'

-> WHEN CaseType = 'Homicide' THEN 'Closed'

-> WHEN CaseType = 'Assault' THEN 'Open'

-> ELSE Status

-> END

-> WHERE OfficerID = 1; -- Update cases handled by a specific officer

Query OK, 1 row affected (0.01 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql>

mysql> -- Verification

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT \* FROM `Case` WHERE OfficerID = 1; -- Check the updated cases

+--------+----------+----------------+---------------------+-----------+

| CaseID | CaseType | DateRegistered | Status | OfficerID |

+--------+----------+----------------+---------------------+-----------+

| 1 | Robbery | 2023-01-10 | Under Investigation | 1 |

+--------+----------+----------------+---------------------+-----------+

1 row in set (0.00 sec)

**CONCEPTS EXPLAINED/ CONCEPT SUMMARY:**

### Summary of Database Creation and Operations

1. **Database Creation**:
   * A database named "PoliceDepartment" was established to manage information related to police officers, cases, suspects, and police stations.
2. **Table Structure**:
   * **Officer Table**: This table holds details about police officers, including their ID, name, rank, date of joining, and associated station ID.
   * **Case Table**: It records various cases handled by officers, containing case ID, type, registration date, status, and the officer assigned to the case.
   * **Suspect Table**: This table includes information about suspects related to specific cases, detailing their ID, name, age, address, and the associated case ID.
   * **Station Table**: It stores information about police stations, including their ID, name, location, and district.
3. **Data Manipulation**:
   * **Insertion**: Initial records are added to each table, establishing a baseline of data for officers, cases, suspects, and stations.
   * **Updating**: Existing records can be modified to reflect changes, such as an officer's rank or case status.
   * **Deletion**: Records can be removed from the tables as necessary, such as when a suspect is no longer relevant to ongoing investigations.
4. **SQL Queries and Operations**:
   * **Subqueries**: These allow for complex filtering of data by evaluating conditions in nested queries.
   * **Membership Operators**: Operators like ANY and ALL help assess conditions across sets of values.
   * **Set Operations**: Operations such as union and set difference are utilized to combine or differentiate datasets effectively.
5. **Joins and Cartesian Products**:
   * **Join Operations**: Different types of joins (e.g., equi joins) enable the combination of related data from multiple tables based on common attributes.
   * **Cartesian Product**: This operation generates a combination of all records from two tables, providing a complete dataset for analysis.
6. **Views**:
   * Views are created to simplify complex queries and present data in a structured format, allowing users to access specific subsets of information without altering the underlying tables.
7. **Transactions and Control**:
   * Transaction control mechanisms ensure data integrity during operations by allowing multiple actions to be treated as a single unit of work. This includes the ability to roll back changes if necessary.

### Overall Concept

The designed database facilitates efficient management of police department operations, enabling quick access to information about officers, cases, and suspects. The use of SQL queries, joins, and views enhances the capability to analyze and manipulate data, supporting effective decision-making and reporting within the department. Transaction control provides additional assurance that the data remains consistent and reliable throughout various operations.