Ex-1	PROBLEM IDENTIFICATION

AIM:

To develop a Flood Risk Prediction and A Warning System using DBMS that integrates historical flood data, terrain information, and climate models to predict flood risks and henceforth Provide early warning alerts to residents and authorities, allowing for proactive measures to mitigate damage.

PROBLEM STATEMENT:

In light of the escalating frequency and heightened severity of flooding incidents, there arises an imperative for the development of an advanced Flood Risk Prediction and Warning System. The objective is to construct a sophisticated Database Management System (DBMS) that seamlessly amalgamates historical flood data, terrain particulars, and climate models with the utmost precision for the purpose of predicting flood risks. The resultant system is envisioned to furnish punctual and dependable early warning alerts to both residents and authorities, affording them the opportunity to proactively undertake measures and execute efficacious strategies in order to preemptively address and alleviate potential damages. The Key Objectives of this project include Data Integration, Database Design and Management, Flood Risk Prediction,Early Warning System, User-friendly Interface, Proactive Measures coupled with Mitigation providing Scalability and Reliability.

RESULT:

Our Flood Risk Prediction System works for all regions irrespective of many topological conditions with keen consciousness of the micro errors to provide an early warning which helps the government and the people.
The goal is to provide a Flood Risk Prediction coupled with a Warning system that contributes significantly to enhancing community resilience, reducing property damage, and saving lives in flood-prone regions.

Ex-2	

REQUIREMENT GATHERING

AIM:

Designing a database management system(DBMS) for a Flood Risk Prediction System coupled with a warning system which involves identifying entd ities, Relationships, and attributes relevant to the application. Below is a relational database schema for the same.

REQUIREMENT GATHERING:

Requirement gathering for Flood Risk Prediction and warning system involves understanding the needs of the problem statement and a suitable implementation of the idea for the optimal solution.

Entities:

1. **Location:**
   - LocationID (Primary Key)
   - Latitude
   - Longitude
   - Region
   - Elevation

2. **HistoricalFloodData:**
   - FloodDataID (Primary Key)
   - LocationID (Foreign Key referencing Location.LocationID)
   - Date
   - Magnitude
   - Duration

3. **EarlyWarningAlert:**
   - AlertID (Primary Key)
   - LocationID (Foreign Key referencing Location.LocationID)
   - DateIssued
   - AlertLevel
   - RecommendedActions

4. **PreventiveMeasure:**
   - MeasureID (Primary Key)
   - LocationID (Foreign Key referencing Location.LocationID)
   - MeasureType
   - Description
   - ImplementationDate

5. **Resident:**
   - ResidentID (Primary Key)
   - Name
   - Address
   - ContactNumber
   - LocationID (Foreign Key referencing Location.LocationID)

6. **Authority:**
   - AuthorityID (Primary Key)
   - Name
   - Organization
   - ContactNumber
   - LocationID (Foreign Key referencing Location.LocationID)

7. **EmergencyResponsePlan:**
   - PlanID (Primary Key)
   - LocationID (Foreign Key referencing Location.LocationID)
   - PlanDescription
   - ActivationDate

8. **EmergencyResponseLog:**
   - LogID (Primary Key)
   - LocationID (Foreign Key referencing Location.LocationID)
   - Date
   - Time
   - LogDetails

9. **EvacuationCenter:**
   - CenterID (Primary Key)
   - LocationID (Foreign Key referencing Location. LocationID)
   - CenterName
   - Capacity
   - Location Details

RELATIONSHIPS:

- One-to-Many with HistoricalFloodData (One Location can have Many

HistoricalFloodData entries).
One-to-Many with EarlyWarningAlert (One Location can have Many EarlyWarningAlert entries).

- Many-to-One with Location (Many HistoricalFloodData entries can be associated with One Location).
- Many-to-One with Location (Many EarlyWarningAlert entries can be associated with One Location).
- Many-to-One with Location (Many PreventiveMeasures can be associated with One Location).
- Many-to-One with Location (Many Residents can be associated with One Location).
- Many-to-One with Location (Many Authorities can be associated with One Location).
- Many-to-One with Location (Many EmergencyResponsePlans can be associated with One Location).
- Many-to-One with Location (Many EmergencyResponseLogs can be associated with One Location).
- Many-to-One with Location (Many EvacuationCenters can be associated with One Location).

RESULT:

Hence, basic necessary entities are created with their relationships and the necessary requirement analysis is done. Additionally, incorporating normalization techniques to eliminate data redundancy can be ideal for optimal solutions for the above problem statement.

Ex-3	**DESIGN USING ER DIAGRAM**

AIM:

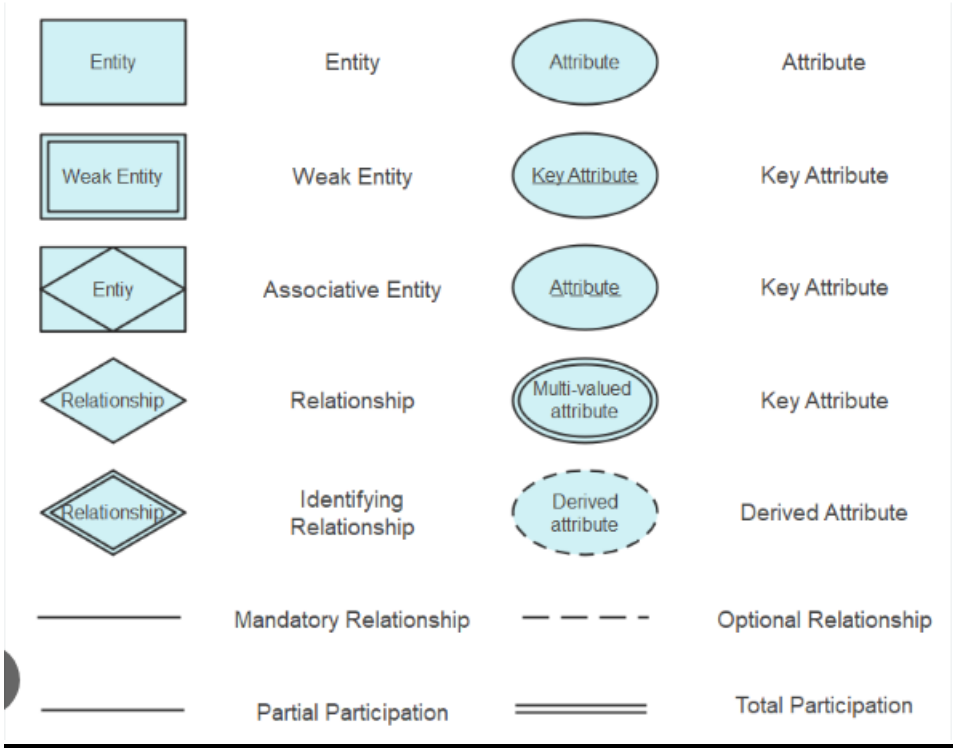
To brainstorm and create an ER Diagram for our project that is titled “Advanced Flood Risk Prediction and Warning System”.

ER DIAGRAM:

An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system. ER Diagrams are

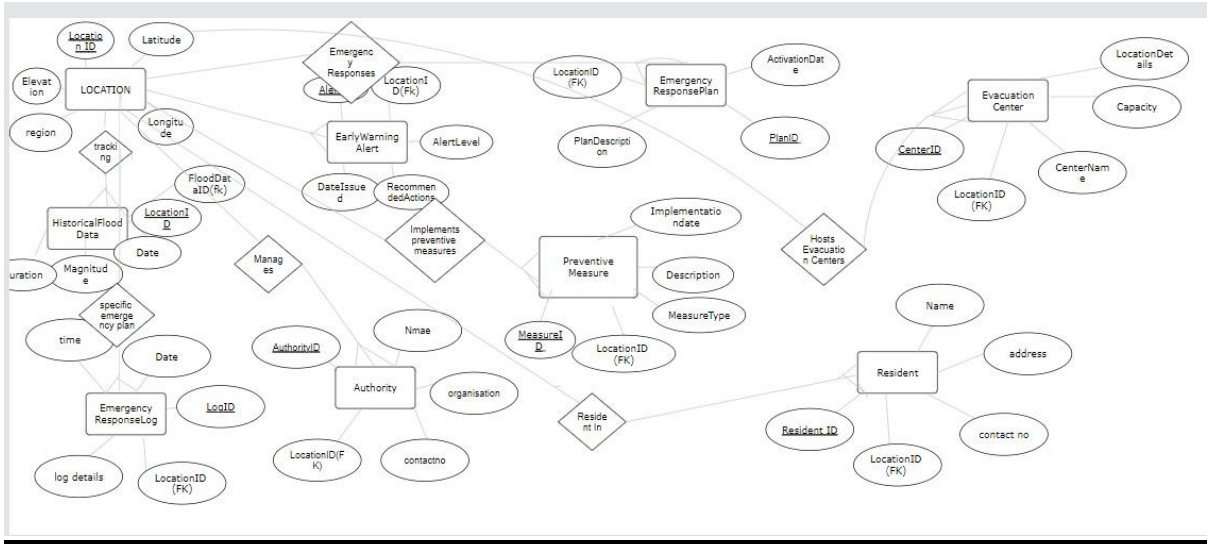
most often used to design or debug relational databases.

ER NOTATIONS:



Entities and their Attributes along with Primary Key:

ER DIAGRAM:



RESULT:

Therefore, an ER diagram for the topic Advanced Flood Risk Prevention and Early warning Database System has been designed successfully based on the given

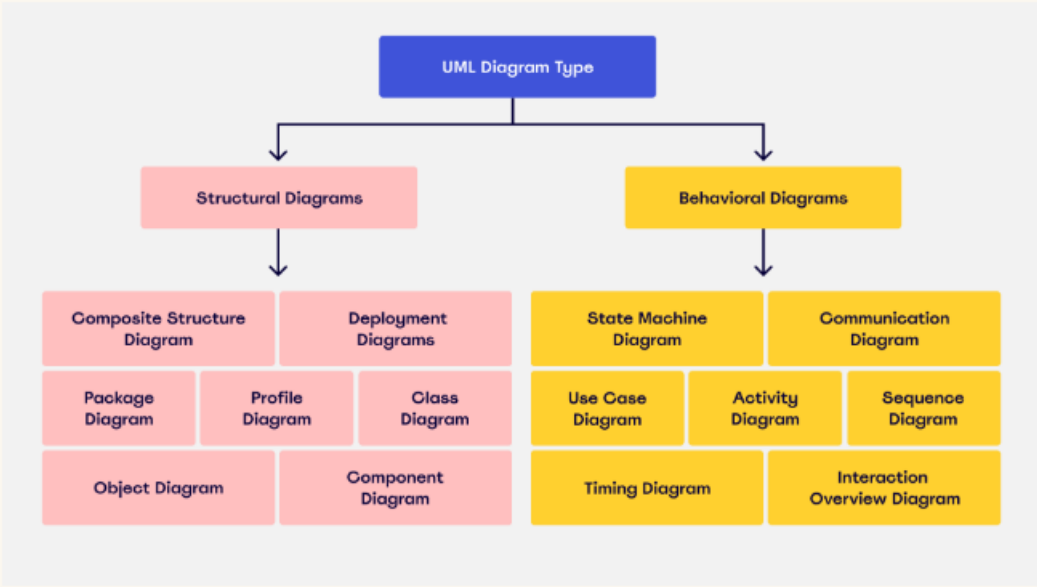
requirements.

Ex-4	ARCHITECTURE OF DIAGRAMS

AIM:

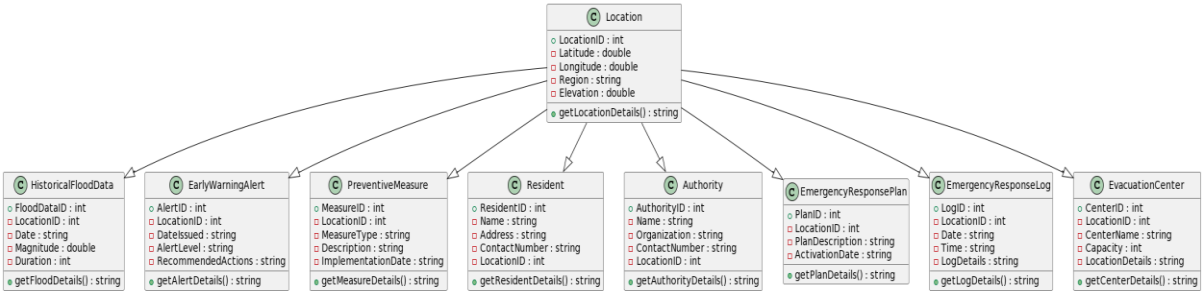
To understand the conceptually, the implantation part holds a huge role for it. A UML Diagram helps to visualize the system or progress of the project over time and its current situation. Errors can be reduced if detected.

Here's a breakdown of the different types of UML diagrams and whether they're classified as a structural diagram or behavioral diagram:



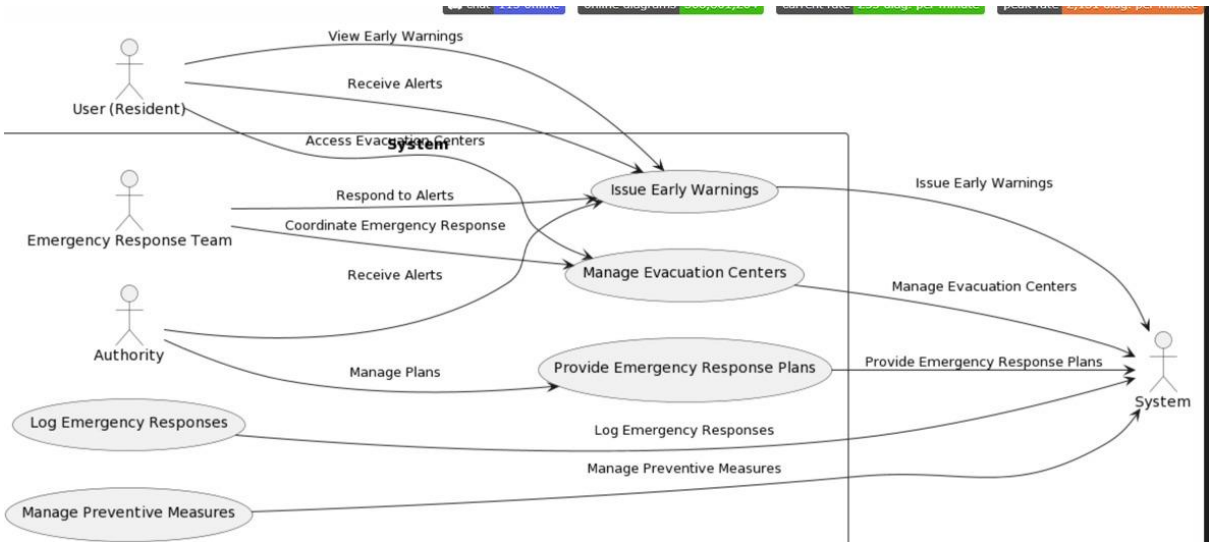
CLASS DIAGRAM:

A class diagram is a visual representation of the structure and relationships within a system, illustrating classes, their attributes, and methods. It provides a high-level view of the system's architecture, showcasing the static elements and their associations.



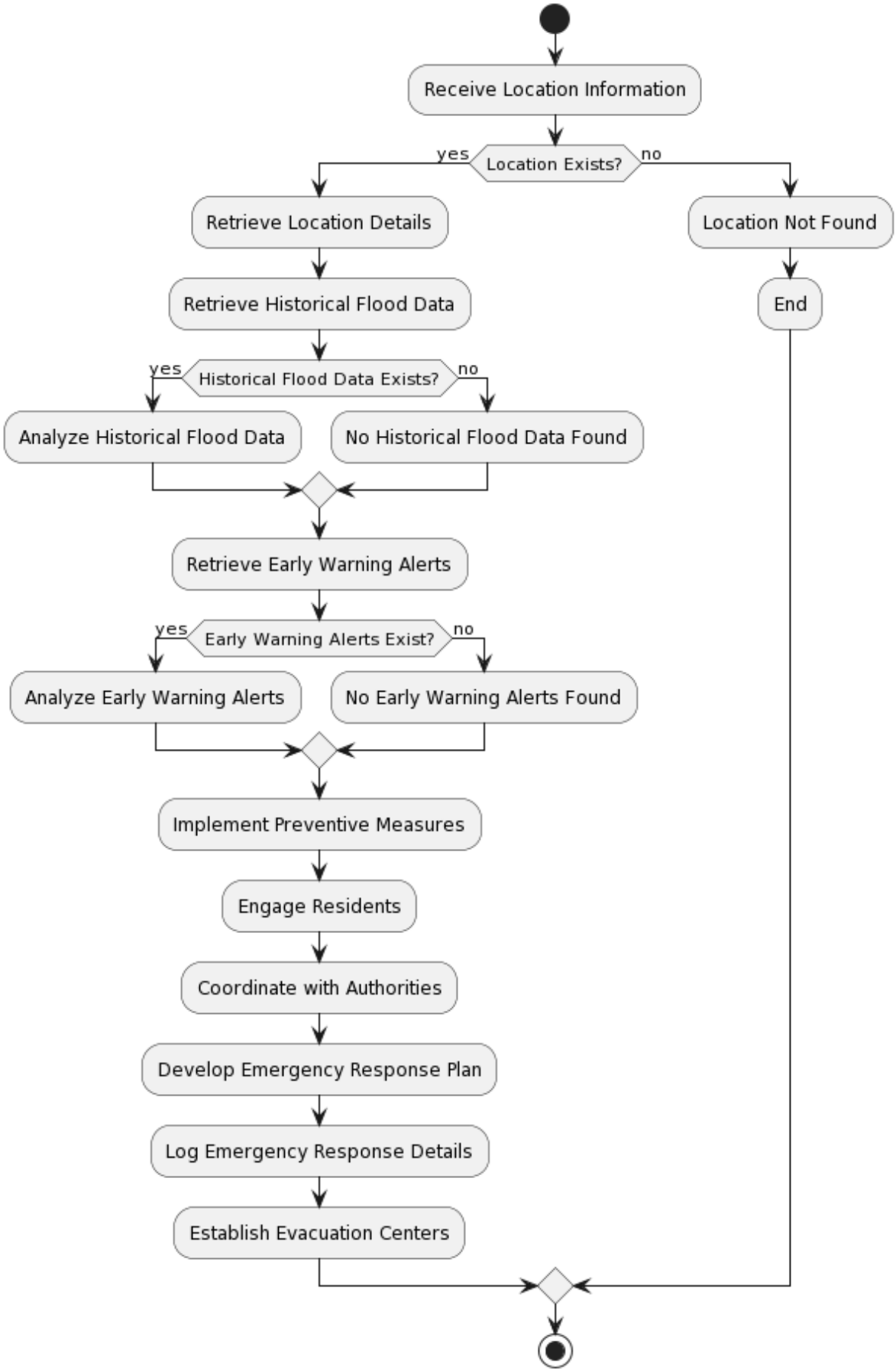
USE CASE DIAGRAM:

A use case diagram outlines the interactions between users and a system, illustrating various scenarios or use cases. It captures the functionalities a system offers from an external perspective, showcasing how actors (users) interact with the system to achieve specific goals.



ACTIVITY DIAGRAM:

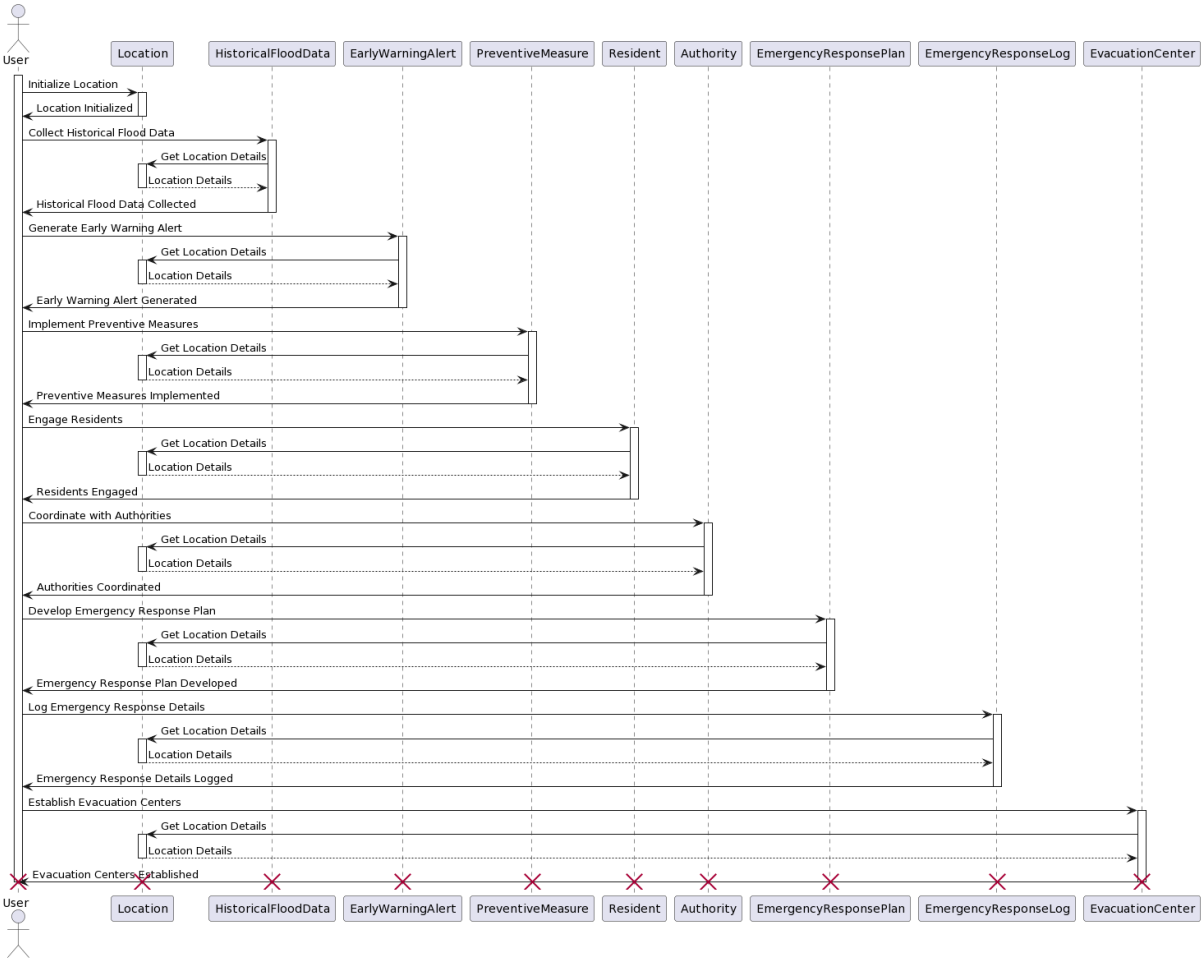
An activity diagram visually represents the flow of activities within a system, demonstrating the sequential steps and decision points in a process. It is particularly useful for modeling workflows and business processes, highlighting the dynamic aspects of system behavior.



SEQUENCE DIAGRAM:

A sequence diagram depicts the interactions between objects in a system over time, illustrating the order of messages exchanged among them. It provides a dynamic view of system behavior, showing how objects collaborate to accomplish specific

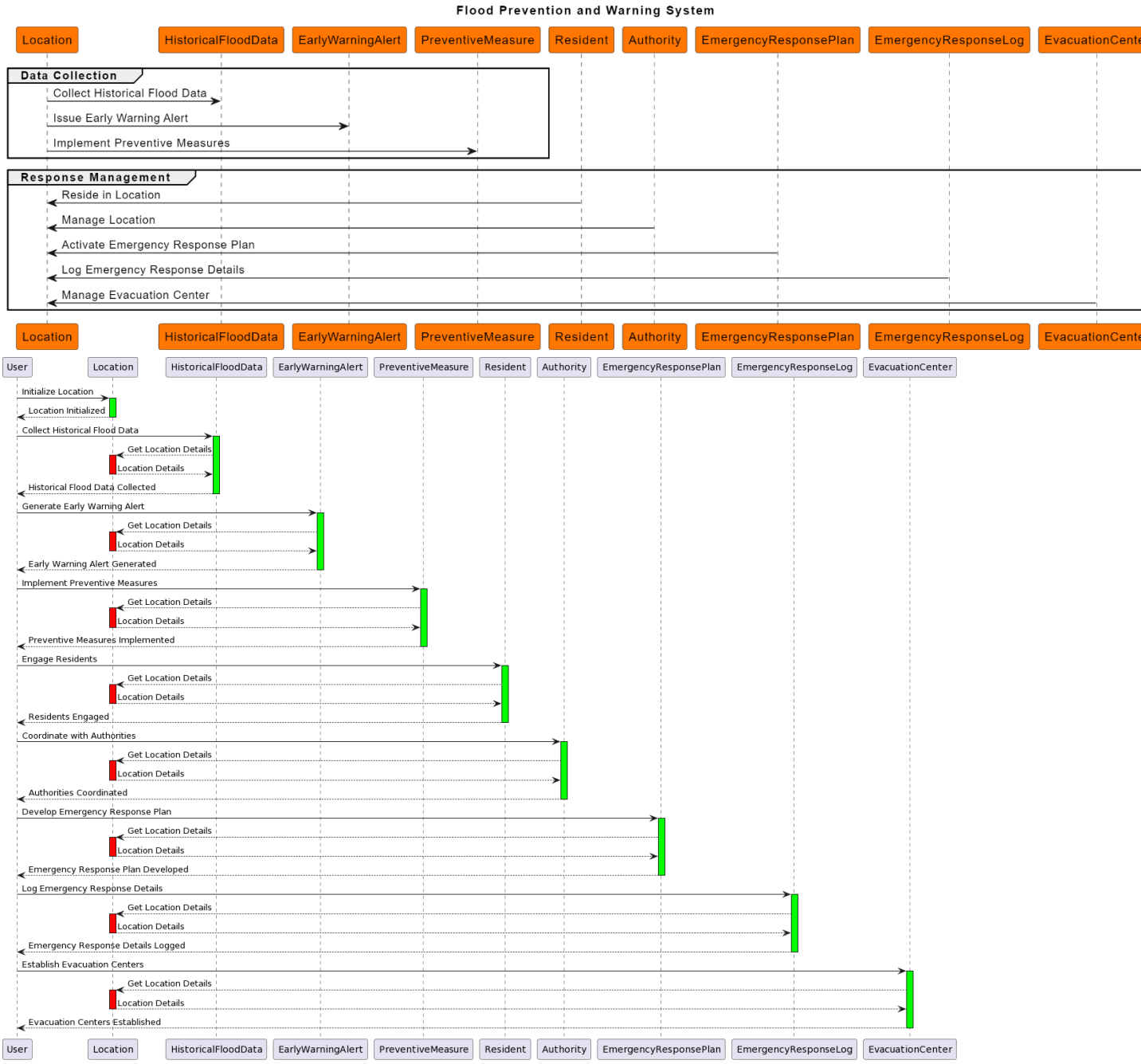
tasks or scenarios.



TIMING DIAGRAM:

A timing diagram portrays the timing aspects of interactions between objects in a system, emphasizing the durations of messages and the lifetimes of objects. It offers insights into the temporal aspects of a system's behavior, helping to understand the synchronization and timing constraints in a scenario.

=



RESULT:

To work on project designing using UML diagrams is successfully completed.

CONVERTING ER DIAGRAM TO TABLE

AIM:

To understand the conceptually, the implantation part holds a huge role for it. To

convert the Entity relationship diagram to tables and create data inside to feature and practice implementation through the venture of structured query language. SQL is a programming language which offers insights about the data inside the tables.

LIST OF TABLES:

1) Location Table
2) Historical Flood Data Table
3) Early Warning Alert Table
4) Preventive Measure Table
5) Resident Table
6) Authority Table
7) Emergency Response Plan Table
8) Emergency Response Log Table
9) Evacuation Center Table

1)LOCATION TABLE:

LocationID	Latitude	Longitude	Region	Elevation
1	40.7128	-74.0060	New York	10
2	34.0522	-118.2437	Los Angeles	15
3	51.5074	-0.1278	London	20
4	48.8566	2.3522	Paris	30
5	35.6895	139.6917	Tokyo	25
6	37.7749	-122.4194	San Francisco	12
7	40.4168	-3.7038	Madrid	22
8	37.9838	23.7275	Athens	18
9	41.9028	12.4964	Rome	28
10	55.7558	37.6176	Moscow	35

2)HISTORICAL FLOOD DATA TABLE:

FloodDataID	LocationID	Date	Magnitude	Duration
1	1	2022-01-05	4	48
2	2	2022-02-12	3	36
3	3	2022-03-20	5	72

4	4	2022-04-18	3.5	24
5	5	2022-05-25	4.5	60
6	6	2022-06-10	3	48
7	7	2022-07-15	4	36
8	8	2022-08-22	3.5	24
9	9	2022-09-30	5	72
10	10	2022-10-18	4.5	60

3)EARLY WARNING ALERT TABLE:

AlertID	LocationID	DateIssued	AlertLevel	RecommendedActions
1	1	2022-01-03	High	Evacuate
2	2	2022-02-10	Moderate	Stay Alert
3	3	2022-03-18	High	Prepare
4	4	2022-04-15	Low	Monitor
5	5	2022-05-22	Moderate	Prepare
6	6	2022-06-08	High	Evacuate
7	7	2022-07-13	Low	Monitor
8	8	2022-08-20	Moderate	Prepare
9	9	2022-09-28	High	Evacuate
10	10	2022-10-15	Low	Monitor

4)PREVENTIVE MEASURE TABLE:

MeasureID	LocationID	MeasureType	Description	ImplementationDate
1	1	Structural	Reinforcement of riverbanks	2022-01-15
2	2	Early Warning	Installation of flood sensors	2022-02-20
3	3	Community	Conducting awareness campaigns	2022-03-25
4	4	Infrastructure	Upgrading drainage systems	2022-04-30
5	5	Evacuation	Establishment of evacuation routes	2022-05-10
6	6	Monitoring	Implementing real-time monitoring	2022-06-15

7	7	Preparedness	Training emergency response teams	2022-07-20
8	8	Communication	Enhancing public communication channels	2022-08-25
9	9	Equipment	Procurement of emergency supplies	2022-09-05
10	10	Technology	Implementing GIS-based flood modeling	2022-10-10

5)RESIDENT TABLE:

ResidentID	Name	Address	ContactNumber	LocationID
1	John Doe	123 Main St, City	123-456-7890	1
2	Jane Smith	456 Oak Ave, Town	456-789-0123	2
3	David Johnson	789 Elm Rd, Village	789-012-3456	3
4	Sarah Brown	321 Pine Blvd, Suburb	321-654-9870	4
5	Emily Wilson	567 Maple Dr, Hamlet	567-890-1234	5
6	Michael Lee	901 Cedar Ln, County	901-234-5678	6
7	Laura Miller	234 Birch St, Township	234-567-8901	7
8	Kevin Clark	678 Walnut Ave, Parish	678-901-2345	8
9	Amanda White	890 Spruce Rd, Borough	890-123-4567	9
10	Eric Taylor	345 Oakwood Dr, Manor	345-678-9012	10

6)AUTHORITY TABLE:

AuthorityID	Name	Organization	ContactNumber	LocationID
1	Mark Anderson	City Emergency Dept	111-222-3333	1
2	Lisa Williams	County Disaster Team	444-555-6666	2
3	James Wilson	Township Fire Dept	777-888-9999	3
4	Sarah Brown	Parish Police Dept	000-111-2222	4
5	Michael Clark	Village Emergency	333-444-5555	5
6	Emma Johnson	Borough Rescue Squad	666-777-8888	6
7	Ryan Martinez	Hamlet Emergency	999-000-1111	7
8	Rachel Lee	Town Disaster Relief	222-333-4444	8
9	David White	County Emergency	555-666-7777	9
10	Jessica Taylor	Township Rescue	888-999-0000	10

7)EMERGENCY RESPONSE TABLE:

PlanID	LocationID	PlanDescription	ActivationDate
1	1	Emergency Evacuation Plan for City	2022-01-01
2	2	Disaster Preparedness Plan for County	2022-02-05
3	3	Town Emergency Response Plan	2022-03-10
4	4	Parish Crisis Management Plan	2022-04-15
5	5	Hamlet Disaster Recovery Plan	2022-05-20
6	6	County Emergency Action Plan	2022-06-25
7	7	Township Disaster Management Plan	2022-07-30
8	8	Village Emergency Preparedness Strategy	2022-08-05
9	9	Borough Contingency Plan	2022-09-10
10	10	Manor Disaster Response Framework	2022-10-15

8)EMERGENCY RESPONSE LOG TABLE:

LogID	LocationID	Date	Time	LogDetails
1	1	2022-01-02	09:00	Flood monitoring initiated
2	2	2022-02-08	12:30	Warning alert issued for flooding
3	3	2022-03-12	15:45	Activation of emergency response teams
4	4	2022-04-20	10:15	Drainage system inspection completed
5	5	2022-05-18	11:20	Evacuation routes marked and verified
6	6	2022-06-24	08:45	Real-time monitoring system activated
7	7	2022-07-28	14:00	Training session conducted for responders
8	8	2022-08-30	16:30	Public communication channels tested
9	9	2022-09-05	13:45	Emergency supplies inventory updated
10	10	2022-10-12	07:55	GIS-based flood modeling in progress

9)EVACUATION CENTER TABLE:

CenterID	LocationID	CenterName	Capacity	LocationDetails
1	1	City Evacuation Center	500	City Hall
2	2	County Shelter	300	Community Center
3	3	Town Emergency Center	200	Town Hall
4	4	Parish Safe Haven	400	Parish School
5	5	Hamlet Evacuation Site	250	Hamlet Park
6	6	County Relief Center	350	County Fairgrounds
7	7	Township Refuge	150	Township Library
8	8	Village Shelter	280	Village Hall
9	9	Borough Emergency Hub	180	Borough Community Center
10	10	Manor Safe Zone	400	Manor Arena

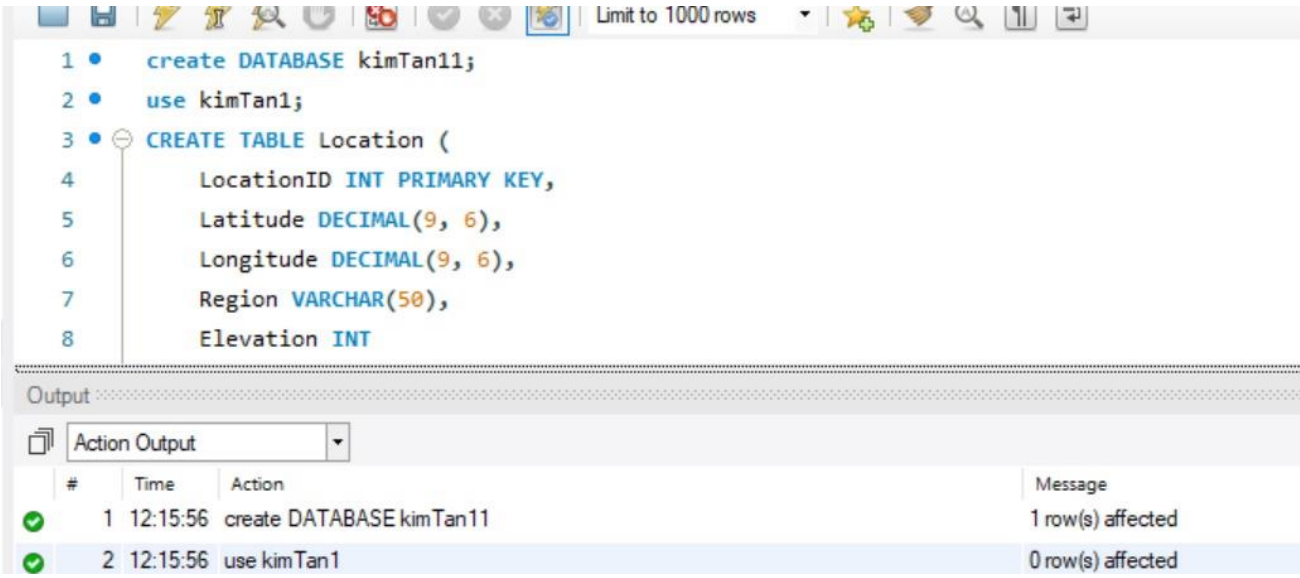
Ex-5	WORKING WITH DDL AND DML COMMANDS

AIM:

To work with DDL & DML commands.

1. DDL COMMANDS:-

a. **CREATE**: This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).



```sql
);

CREATE TABLE PreventiveMeasure (
    MeasureID INT PRIMARY KEY,
    LocationID INT,
    MeasureType VARCHAR(20),
    Description VARCHAR(100),
    ImplementationDate DATE,
    FOREIGN KEY (LocationID) REFERENCES Location(LocationID)
);

CREATE TABLE Resident (
    ResidentID INT PRIMARY KEY,
    Name VARCHAR(50),
    Address VARCHAR(100),
    ContactNumber VARCHAR(15),
    LocationID INT,
    FOREIGN KEY (LocationID) REFERENCES Location(LocationID)
);

CREATE TABLE Authority (
    AuthorityID INT PRIMARY KEY,
    Name VARCHAR(50),
    Organization VARCHAR(50),
    ContactNumber VARCHAR(15),
    LocationID INT,
```

```sql
CREATE TABLE EmergencyResponse (
    PlanID INT PRIMARY KEY,
    LocationID INT,
    PlanDescription VARCHAR(100),
    ActivationDate DATE,
    FOREIGN KEY (LocationID) REFERENCES Location(LocationID)
);

CREATE TABLE EmergencyResponseLog (
    LogID INT PRIMARY KEY,
    LocationID INT,
    Date DATE,
    Time TIME,
    LogDetails VARCHAR(100),
    FOREIGN KEY (LocationID) REFERENCES Location(LocationID)
);

CREATE TABLE EvacuationCenter (
    CenterID INT PRIMARY KEY,
    LocationID INT,
    CenterName VARCHAR(50),
    Capacity INT,
    LocationDetails VARCHAR(100),
    FOREIGN KEY (LocationID) REFERENCES Location(LocationID)
);
```

b. **DROP:** This command is used to delete objects from the database.

```
30  21:06:37  DROP TABLE IF EXISTS FloodData        0 row(s) affected        0.031 se
```

```sql
-- Drop FloodData table
DROP TABLE IF EXISTS FloodData;
```

c. **ALTER:** This is used to alter the structure of the database.

```sql
Alter table account1 ADD email varchar(20);
```

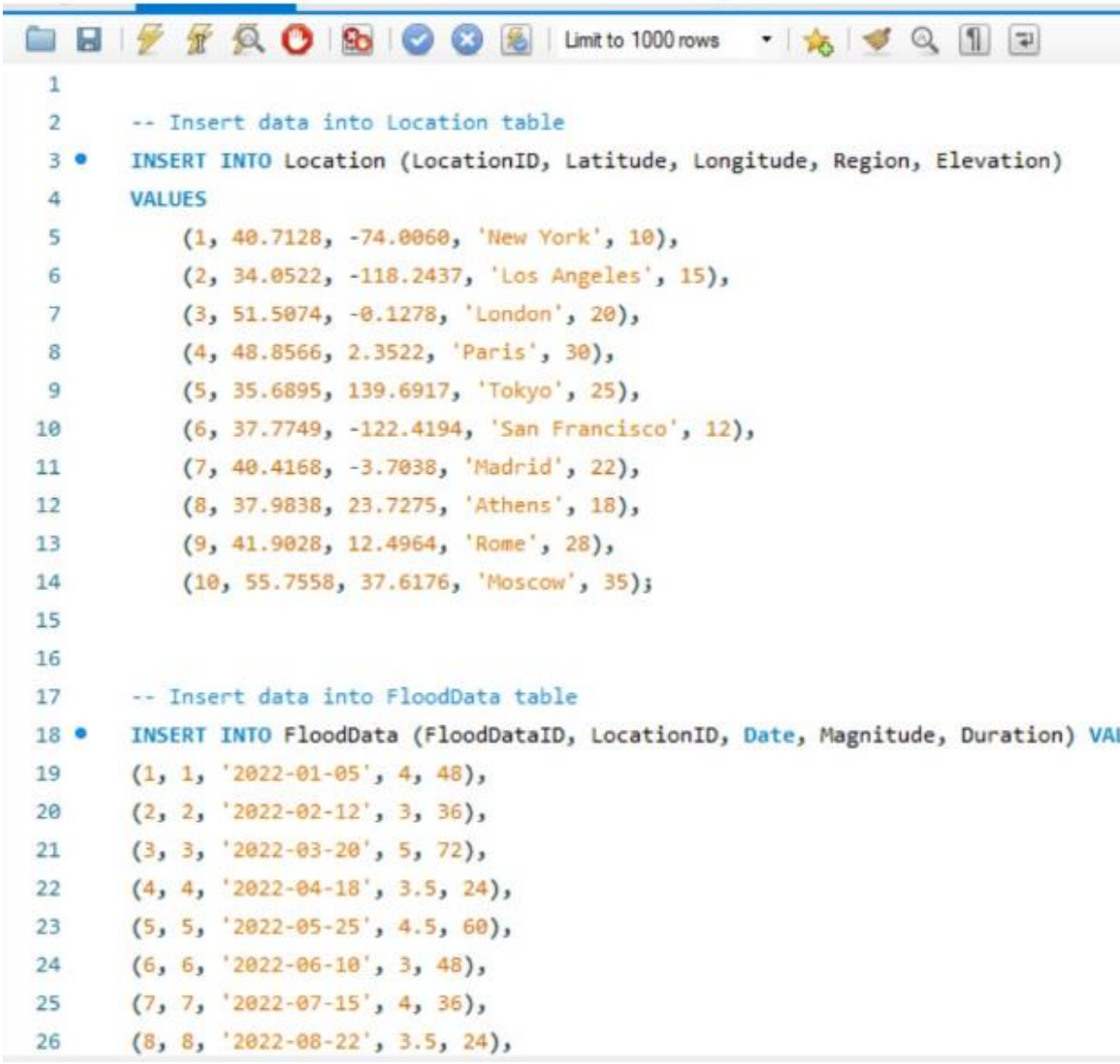
d. **TRUNCATE:** This is used to remove all records from a table, including all spaces allocated for the records are removed.

```
3
4     -- Truncate the tables with foreign key constraints
5     TRUNCATE TABLE EarlyWarningAlert;
6     TRUNCATE TABLE PreventiveMeasure;
7     TRUNCATE TABLE Resident;
8     TRUNCATE TABLE Authority;
9     TRUNCATE TABLE EmergencyResponse;
10    TRUNCATE TABLE EmergencyResponseLog;
11    TRUNCATE TABLE EvacuationCenter;
12
13    -- Now you can truncate the Location table
14    TRUNCATE TABLE Location;
15
16    -- Enable foreign key checks back
17    SET FOREIGN_KEY_CHECKS = 1;
18
```

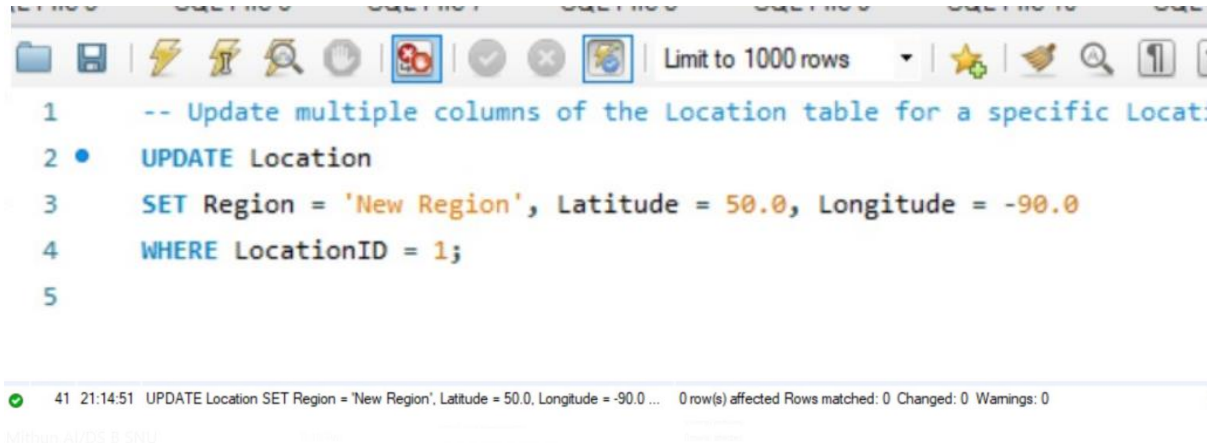
31	21:12:03	SET FOREIGN_KEY_CHECKS = 0	0 row(s) affected	0.000 sec
32	21:12:03	TRUNCATE TABLE EarlyWarningAlert	0 row(s) affected	0.063 sec
33	21:12:03	TRUNCATE TABLE PreventiveMeasure	0 row(s) affected	0.031 sec
34	21:12:03	TRUNCATE TABLE Resident	0 row(s) affected	0.031 sec
35	21:12:03	TRUNCATE TABLE Authority	0 row(s) affected	0.016 sec
36	21:12:03	TRUNCATE TABLE EmergencyResponse	0 row(s) affected	0.031 sec
37	21:12:03	TRUNCATE TABLE EmergencyResponseLog	0 row(s) affected	0.031 sec
38	21:12:03	TRUNCATE TABLE EvacuationCenter	0 row(s) affected	0.032 sec
39	21:12:03	TRUNCATE TABLE Location	0 row(s) affected	0.031 sec
40	21:12:03	SET FOREIGN_KEY_CHECKS = 1	0 row(s) affected	0.000 sec

2)DML COMMANDS:-

a. **INSERT**: It is used to insert data into a table.



b. **UPDATE:** It is used to update existing data within a table.



c. **DELETE:** It is used to delete records from a database table.

```
3 •  delete from account1 where id=1;
4
```

Action Output

#	Time	Action	Message
3	09:07:37	Alter table account1 ADD email varchar(20)	Error Code: 1146. Table 'dbms_prj.account1' doesn't exist
4	09:08:10	CREATE TABLE account1 (id INT AUTO_INCREME...	0 row(s) affected
5	09:08:13	Alter table account1 ADD email varchar(20)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
6	09:09:32	Truncate table account1	0 row(s) affected
7	09:10:53	delete from account1 where id=1	0 row(s) affected

RESULT:

Thus, working with DDL & DML commands on MySQL is successfully verified.

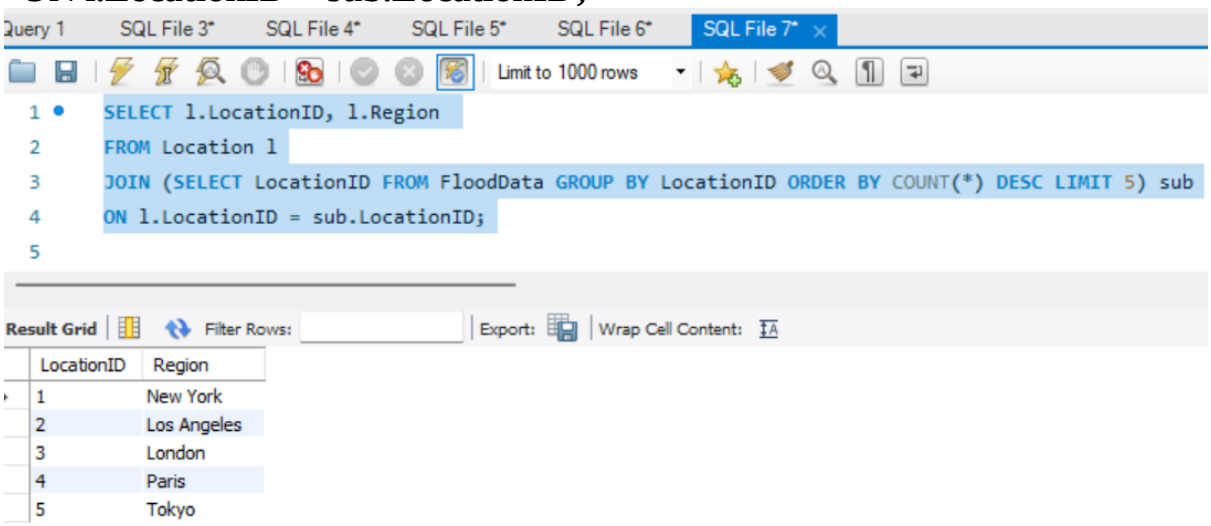
Ex-6	WORKING WITH JOIN,SET OPERATIONS AND AGGREGATE FUNCTIONS

AIM:

To perform join, set and aggregate functions on MySQL.

Query to Identify Regions Most Affected by Floods:

```
SELECT l.LocationID, l.Region
FROM Location l
JOIN (SELECT LocationID FROM FloodData GROUP BY LocationID ORDER
    BY COUNT(*) DESC LIMIT 5) sub
ON l.LocationID = sub.LocationID;
```



Count the number of preventive measures of each type:

```
SELECT MeasureType, COUNT(*) AS MeasureCount
FROM PreventiveMeasure
GROUP BY MeasureType;
```

```
SELECT MeasureType, COUNT(*) AS MeasureCount
FROM PreventiveMeasure
GROUP BY MeasureType;
```

Result Grid | Filter Rows: | Export: | Wrap Cell C

MeasureType	MeasureCount
Structural	1
Early Warning	1
Community	1
Infrastructure	1
Evacuation	1
Monitoring	1
Preparedness	1
Communication	1
Equipment	1
Technology	1

Retrieve residents' contact information for regions with flood occurrences:

```
SELECT DISTINCT r.Name, r.Address, r.ContactNumber
FROM Resident r
JOIN Location l ON r.LocationID = l.LocationID
JOIN FloodData f ON l.LocationID = f.LocationID;
```

Limit to 1000 rows

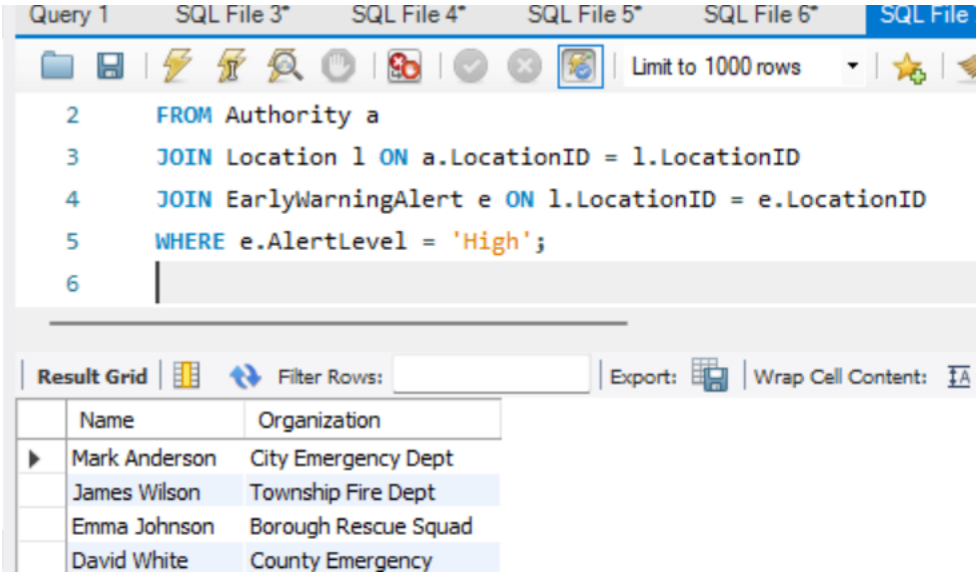
```
SELECT DISTINCT r.Name, r.Address, r.ContactNumber
FROM Resident r
JOIN Location l ON r.LocationID = l.LocationID
JOIN FloodData f ON l.LocationID = f.LocationID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell C

Name	Address	ContactNumber
John Doe	123 Main St, City	123-456-7890
Jane Smith	456 Oak Ave, Town	456-789-0123
David Johnson	789 Elm Rd, Village	789-012-3456
Sarah Brown	321 Pine Blvd, Suburb	321-654-9870
Emily Wilson	567 Maple Dr, Hamlet	567-890-1234
Michael Lee	901 Cedar Ln, County	901-234-5678
Laura Miller	234 Birch St, Township	234-567-8901
Kevin Clark	678 Walnut Ave, Parish	678-901-2345
Amanda White	890 Spruce Rd, Borough	890-123-4567
Eric Taylor	345 Oakwood Dr, Manor	345-678-9012

Retrieve authorities for regions with flood occurrences and high alert levels:

```
SELECT DISTINCT a.Name, a.Organization
FROM Authority a
JOIN Location l ON a.LocationID = l.LocationID
JOIN EarlyWarningAlert e ON l.LocationID = e.LocationID
WHERE e.AlertLevel = 'High';
```



Retrieve flood data for regions with elevation less than 20 and magnitude greater than 2:

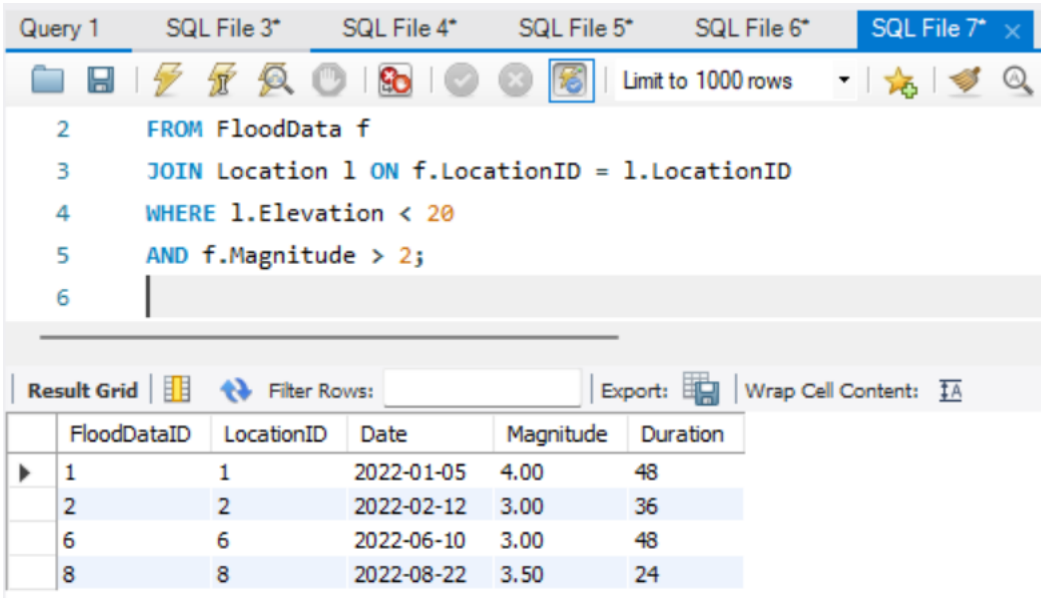
SELECT f.*

FROM FloodData f

JOIN Location l ON f.LocationID = l.LocationID

WHERE l.Elevation < 20

AND f.Magnitude > 2;



RESULT: Thus, working with Join, Set operations and Aggregate functions on MySQL is successfully verified.

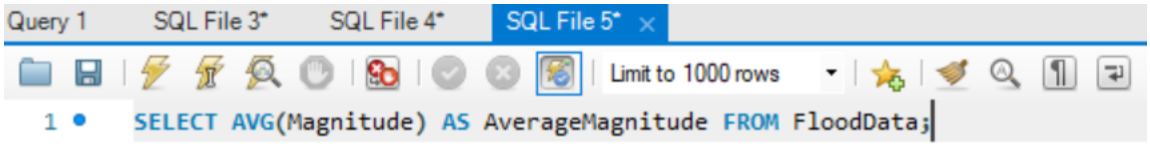
Ex-7	WORKING WITH QUERIES RELATED TO PROJECT

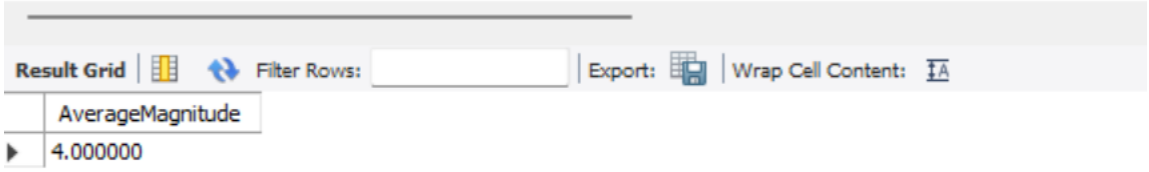
AIM:

To work with queries related to project on MySQL.

1) Query to Calculate Average Magnitude of Floods:

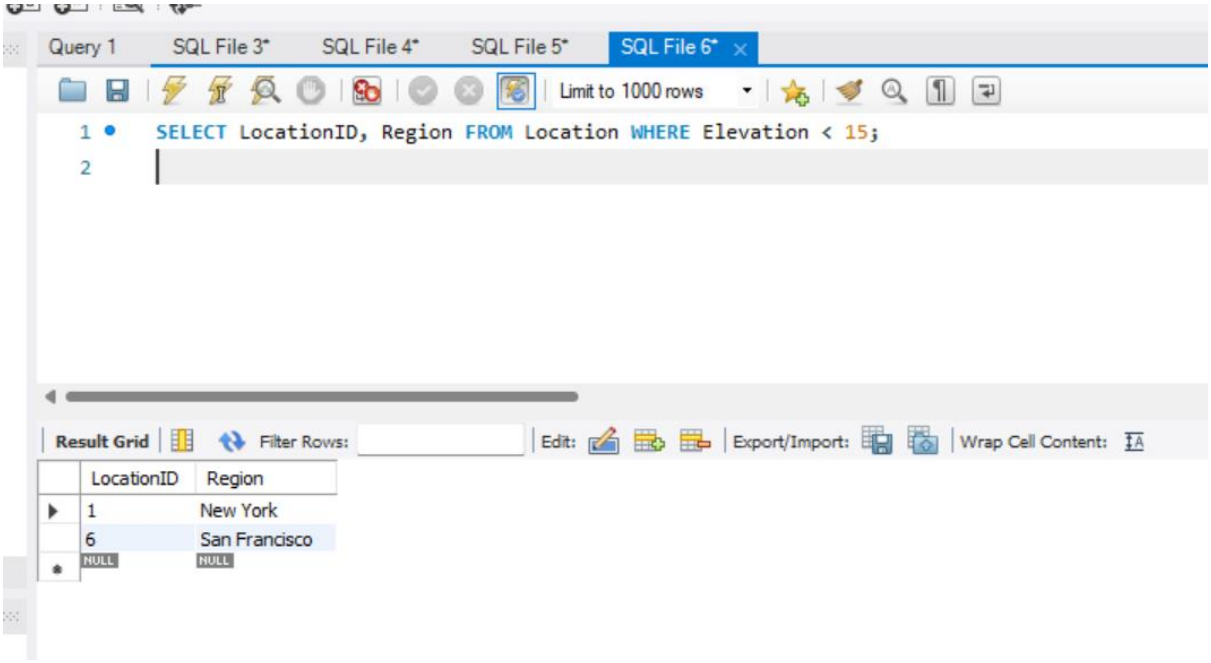
SELECT AVG(Magnitude) AS AverageMagnitude FROM FloodData;





2) Query to Identify Regions with Low Elevation:

SELECT LocationID, Region FROM Location WHERE Elevation < 15;

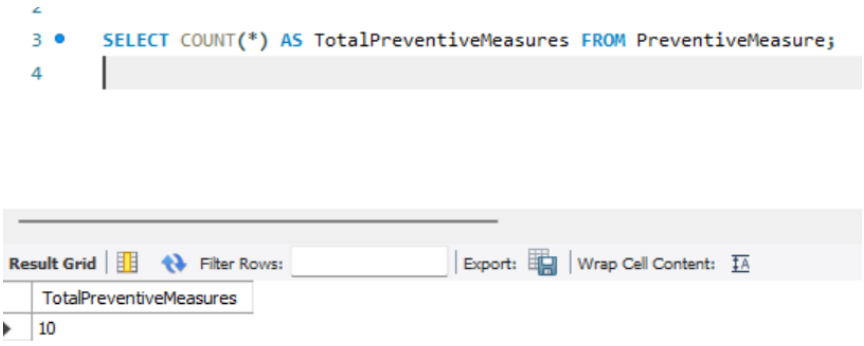


3) Query to Retrieve Early Warning Alerts:

SELECT * FROM EarlyWarningAlert;

4) Query to Count Total Preventive Measures Implemented:

SELECT COUNT(*) AS TotalPreventiveMeasures FROM PreventiveMeasure;

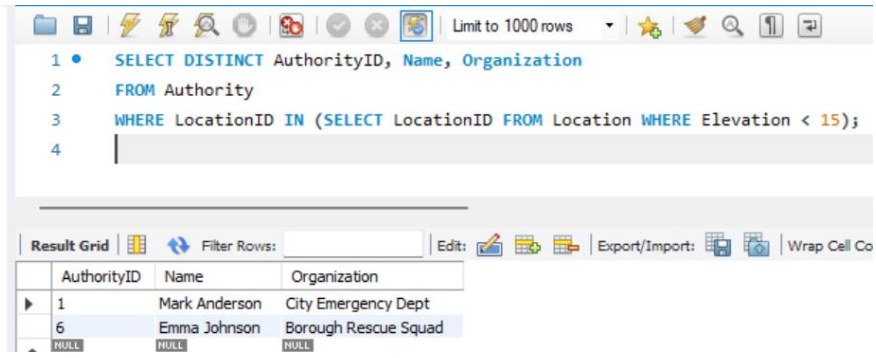


```
SELECT * FROM EarlyWarningAlert;
```

AlertID	LocationID	DateIssued	AlertLevel	RecommendedActions
1	1	2022-01-03	High	Evacuate
2	2	2022-02-10	Moderate	Stay Alert
3	3	2022-03-18	High	Prepare
4	4	2022-04-15	Low	Monitor
5	5	2022-05-22	Moderate	Prepare
6	6	2022-06-08	High	Evacuate
7	7	2022-07-13	Low	Monitor
8	8	2022-08-20	Moderate	Prepare
9	9	2022-09-28	High	Evacuate
10	10	2022-10-15	Low	Monitor

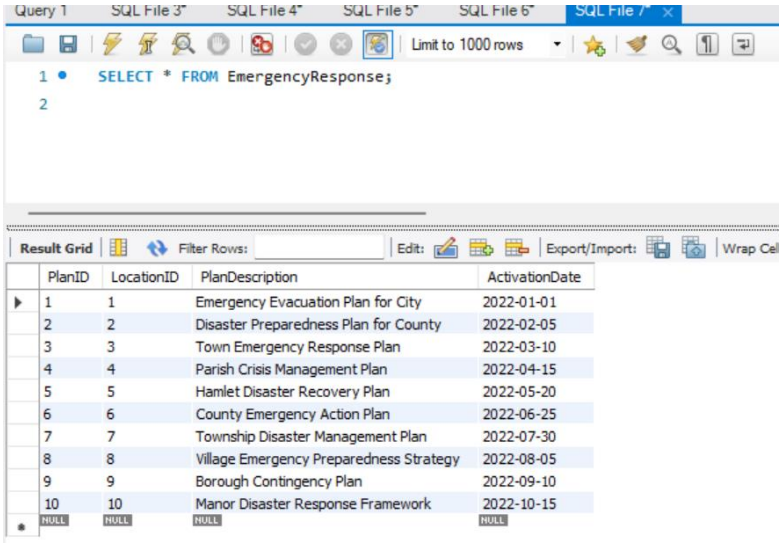
5) Query to Identify Authorities Responsible for Flood-Prone Areas:

```
SELECT DISTINCT AuthorityID, Name, Organization
FROM Authority
WHERE LocationID IN (SELECT LocationID FROM Location WHERE Elevation < 15);
```



6) Query to Retrieve Activated Emergency Response Plans:

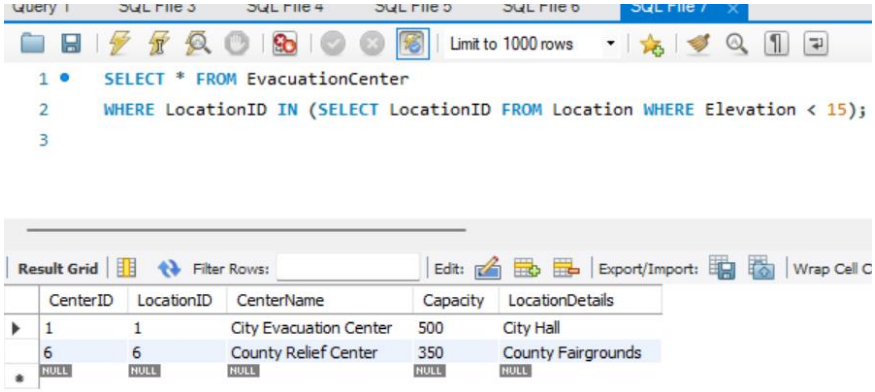
```
SELECT * FROM EmergencyResponse;
```



7) Query to List Evacuation Centers in Flood-Prone Areas:

SELECT * FROM EvacuationCenter

WHERE LocationID IN (SELECT LocationID FROM Location WHERE Elevation < 15);

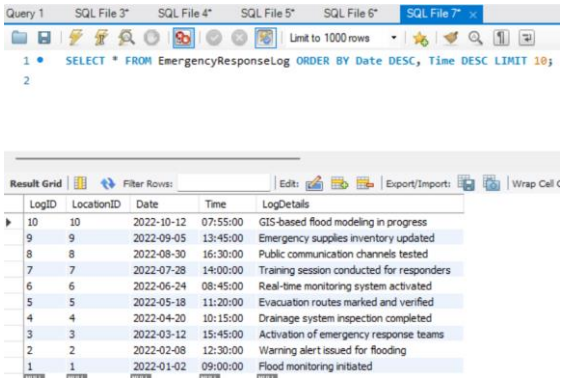


```
SELECT * FROM EvacuationCenter
WHERE LocationID IN (SELECT LocationID FROM Location WHERE Elevation < 15);
```

CenterID	LocationID	CenterName	Capacity	LocationDetails
1	1	City Evacuation Center	500	City Hall
6	6	County Relief Center	350	County Fairgrounds
NULL	NULL	NULL	NULL	NULL

8) Query to Retrieve Recent Emergency Response Logs:

SELECT * FROM EmergencyResponseLog ORDER BY Date DESC, Time DESC LIMIT 10;

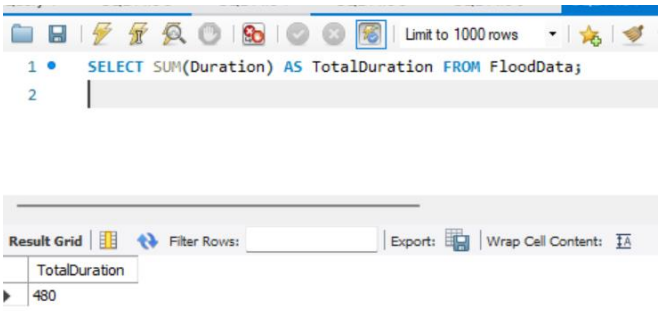


```
SELECT * FROM EmergencyResponseLog ORDER BY Date DESC, Time DESC LIMIT 10;
```

LogID	LocationID	Date	Time	LogDetails
10	10	2022-10-12	07:55:00	GIS-based flood modeling in progress
9	9	2022-09-05	13:45:00	Emergency supplies inventory updated
8	8	2022-08-30	16:30:00	Public communication channels tested
7	7	2022-07-28	14:00:00	Training session conducted for responders
6	6	2022-06-24	08:45:00	Real-time monitoring system activated
5	5	2022-05-18	11:20:00	Evacuation routes marked and verified
4	4	2022-04-20	10:15:00	Drainage system inspection completed
3	3	2022-03-12	15:45:00	Activation of emergency response teams
2	2	2022-02-08	12:30:00	Warning alert issued for flooding
1	1	2022-01-02	09:00:00	Flood monitoring initiated

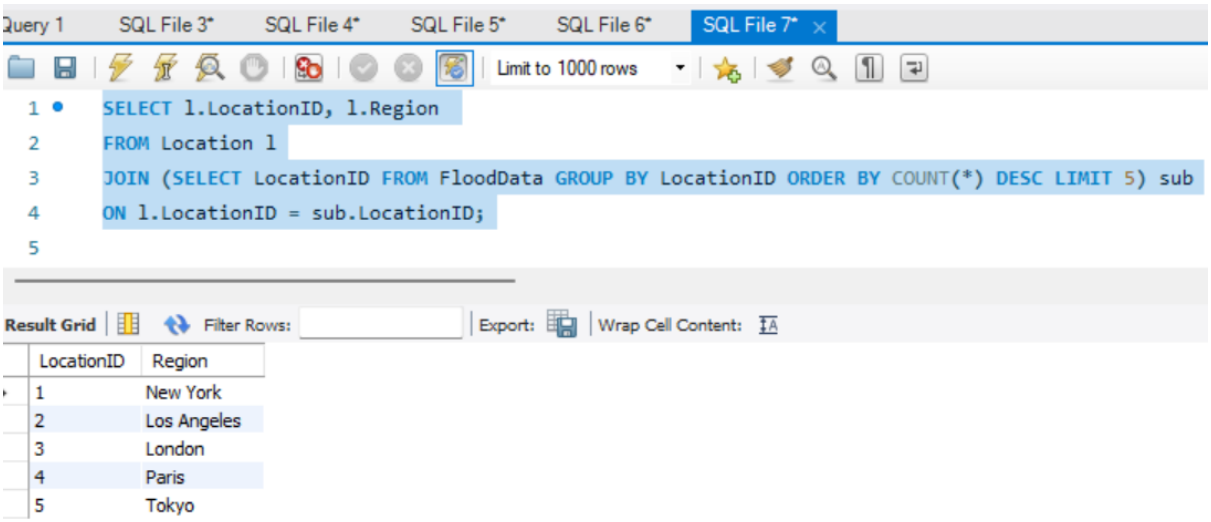
9) Query to Calculate Total Duration of Floods:

SELECT SUM(Duration) AS TotalDuration FROM FloodData;



10) Query to Identify Regions Most Affected by Floods:

```
SELECT l.LocationID, l.Region

FROM Location l

JOIN (SELECT LocationID FROM FloodData GROUP BY LocationID ORDER
    BY COUNT(*) DESC LIMIT 5) sub

ON l.LocationID = sub.LocationID;
```



11) Retrieve all floods with magnitude greater than 4:

```
SELECT * FROM FloodData WHERE Magnitude > 4;
```

```
SELECT * FROM FloodData WHERE Magnitude > 4;
```

Result Grid | Filter Rows: | Edit:

FloodDataID	LocationID	Date	Magnitude	Duration
3	3	2022-03-20	5.00	72
5	5	2022-05-25	4.50	60
9	9	2022-09-30	5.00	72
10	10	2022-10-18	4.50	60
NULL	NULL	NULL	NULL	NULL

12) Retrieve preventive measures for a specific location:

SELECT * FROM PreventiveMeasure

WHERE LocationID = 1;

Limit to 1000 rows

```
SELECT * FROM PreventiveMeasure
WHERE LocationID = 1;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

MeasureID	LocationID	MeasureType	Description	ImplementationDate
1	1	Structural	Reinforcement of riverbanks	2022-01-15
NULL	NULL	NULL	NULL	NULL

13) Retrieve residents in low elevation areas:

SELECT * FROM Resident

WHERE LocationID IN (SELECT LocationID FROM Location WHERE Elevation < 15);

Query 1 | SQL File 3* | SQL File 4* | SQL File 5* | SQL File 6* | SQL File 7*

Limit to 1000 rows

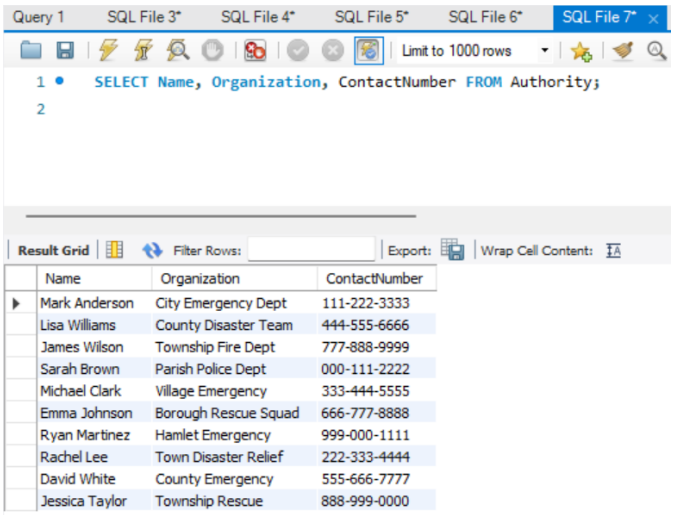
```
SELECT * FROM Resident
WHERE LocationID IN (SELECT LocationID FROM Location WHERE Eleva
```

Result Grid | Filter Rows: | Edit: | Export/Import:

ResidentID	Name	Address	ContactNumber	LocationID
1	John Doe	123 Main St, City	123-456-7890	1
6	Michael Lee	901 Cedar Ln, County	901-234-5678	6
NULL	NULL	NULL	NULL	NULL

14) Retrieve authorities' contact information:

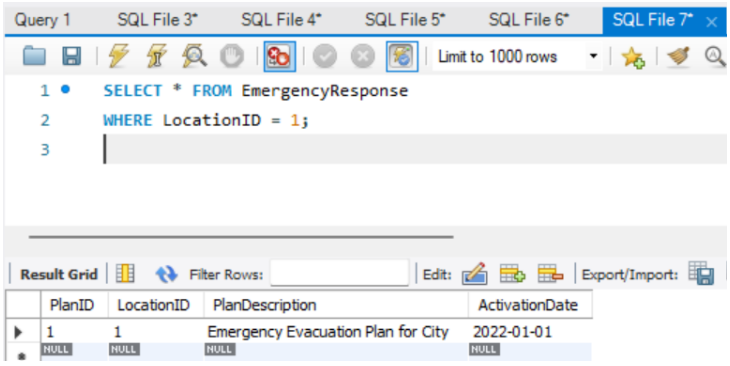
SELECT Name, Organization, ContactNumber FROM Authority;



15) Retrieve emergency response plans for a specific location:

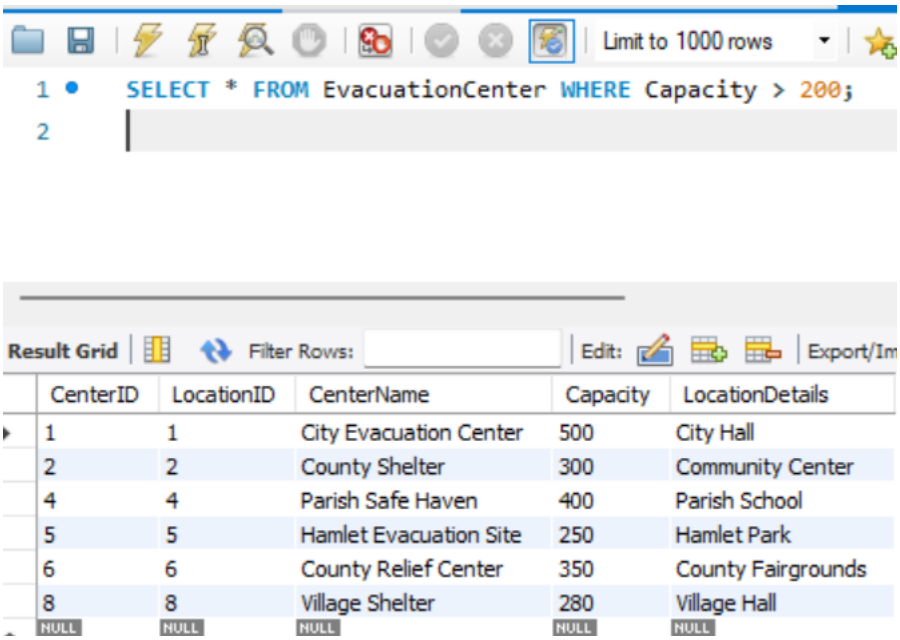
SELECT * FROM EmergencyResponse

WHERE LocationID = 1;



16) Retrieve evacuation centers with capacity greater than 200:

SELECT * FROM EvacuationCenter WHERE Capacity > 200;



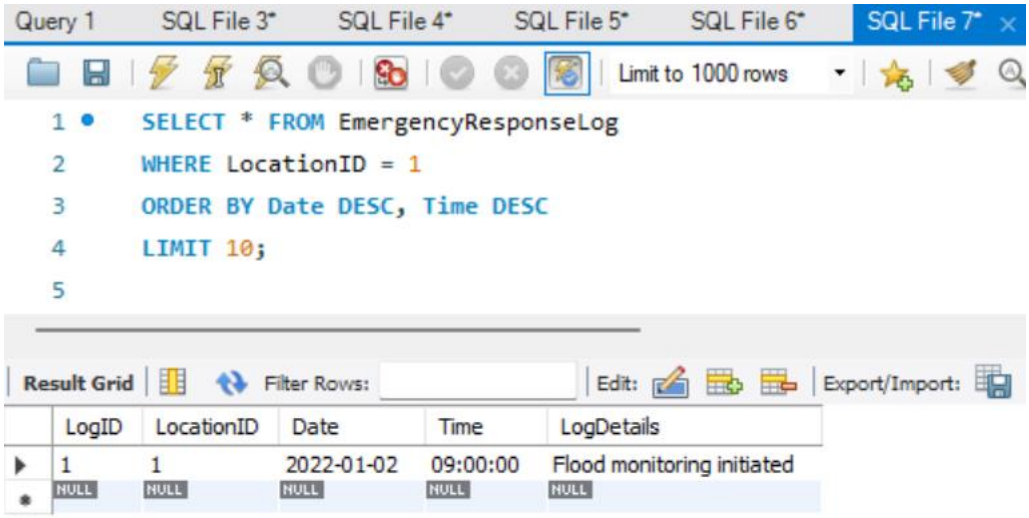
17) Retrieve recent emergency response logs for a specific location:

SELECT * FROM EmergencyResponseLog

WHERE LocationID = 1

ORDER BY Date DESC, Time DESC

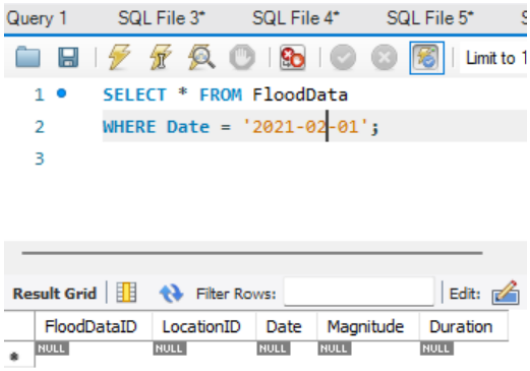
LIMIT 10;



18) Retrieve flood data for a specific date:

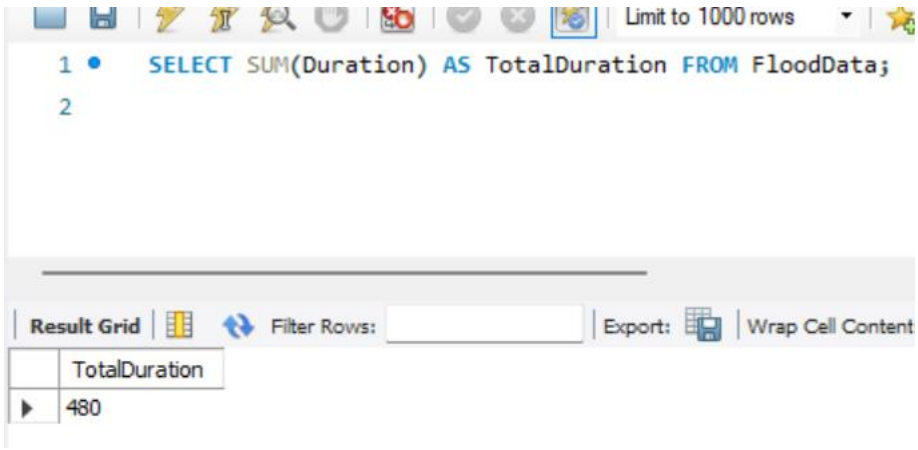
SELECT * FROM FloodData

WHERE Date = '2022-01-01';



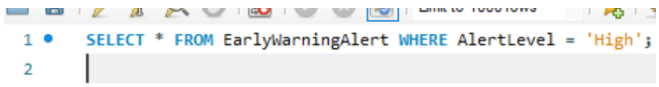
19) Calculate the total duration of floods:

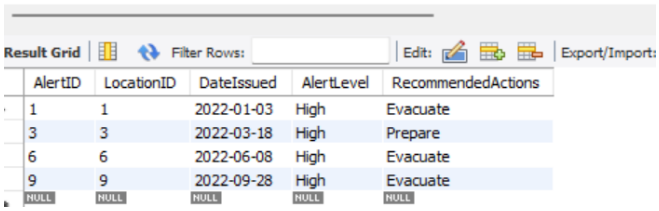
SELECT SUM(Duration) AS TotalDuration FROM FloodData;



20) Retrieve all early warning alerts with high alert levels:

```
SELECT * FROM EarlyWarningAlert WHERE AlertLevel = 'High';
```

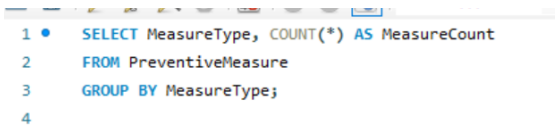




AlertID	LocationID	DateIssued	AlertLevel	RecommendedActions
1	1	2022-01-03	High	Evacuate
3	3	2022-03-18	High	Prepare
6	6	2022-06-08	High	Evacuate
9	9	2022-09-28	High	Evacuate
NULL	NULL	NULL	NULL	NULL

21) Count the number of preventive measures of each type:

```
SELECT MeasureType, COUNT(*) AS MeasureCount
FROM PreventiveMeasure
GROUP BY MeasureType;
```



MeasureType	MeasureCount
Structural	1
Early Warning	1
Community	1
Infrastructure	1
Evacuation	1
Monitoring	1
Preparedness	1
Communication	1
Equipment	1
Technology	1

22) Retrieve residents' contact information for a specific region:

```
SELECT Name, Address, ContactNumber
FROM Resident
WHERE LocationID IN (SELECT LocationID FROM Location WHERE Region
    = 'New York');
```

```
SELECT Name, Address, ContactNumber
FROM Resident
WHERE LocationID IN (SELECT LocationID FROM Location WHERE Region = 'New York');
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Name	Address	ContactNumber
John Doe	123 Main St, City	123-456-7890

23) Retrieve emergency response plans activated after a specific date:

```
SELECT * FROM EmergencyResponse
WHERE ActivationDate > '2022-01-01';
```

Query 1 | SQL File 3* | SQL File 4* | SQL File 5* | SQL File 6* | SQL Fi

Limit to 1000 rows

```
SELECT * FROM EmergencyResponse
WHERE ActivationDate > '2022-01-01';
```

Result Grid | Filter Rows: | Edit: | Export/Impo

PlanID	LocationID	PlanDescription	ActivationDate
2	2	Disaster Preparedness Plan for County	2022-02-05
3	3	Town Emergency Response Plan	2022-03-10
4	4	Parish Crisis Management Plan	2022-04-15
5	5	Hamlet Disaster Recovery Plan	2022-05-20
6	6	County Emergency Action Plan	2022-06-25
7	7	Township Disaster Management Plan	2022-07-30
8	8	Village Emergency Preparedness Strategy	2022-08-05
9	9	Borough Contingency Plan	2022-09-10
10	10	Manor Disaster Response Framework	2022-10-15
NULL	NULL	NULL	NULL

24) Retrieve evacuation centers with capacity greater than 300 and located in low elevation areas:

```
SELECT * FROM EvacuationCenter
WHERE Capacity > 300
AND LocationID IN (SELECT LocationID FROM Location WHERE Elevation < 15);
```

Limit to 1000 rows

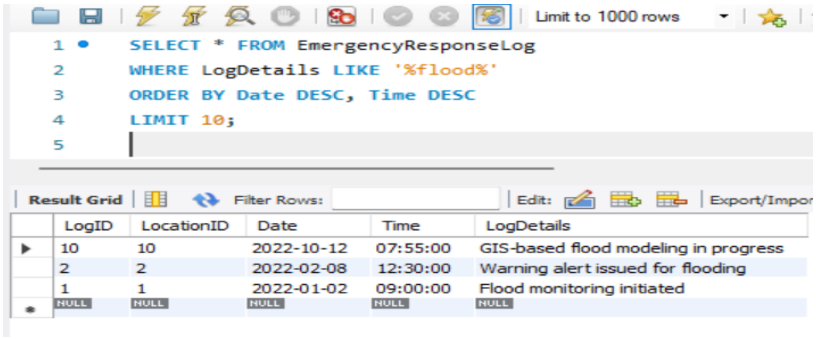
```
SELECT * FROM EvacuationCenter
WHERE Capacity > 300
AND LocationID IN (SELECT LocationID FROM Location WHERE Elevation < 15);
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Ce

CenterID	LocationID	CenterName	Capacity	LocationDetails
1	1	City Evacuation Center	500	City Hall
6	6	County Relief Center	350	County Fairgrounds
NULL	NULL	NULL	NULL	NULL

25) Retrieve recent emergency response logs with flood-related details:

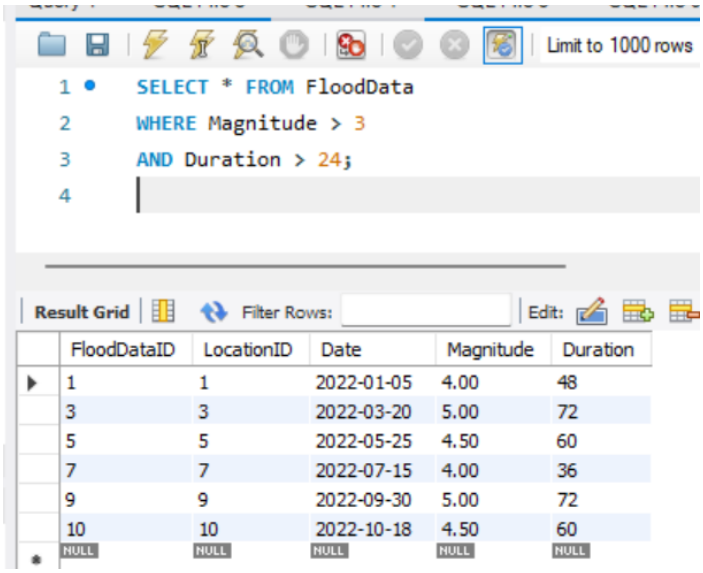
```
SELECT * FROM EmergencyResponseLog
WHERE LogDetails LIKE '%flood%'
ORDER BY Date DESC, Time DESC
LIMIT 10;
```



LogID	LocationID	Date	Time	LogDetails
10	10	2022-10-12	07:55:00	GIS-based flood modeling in progress
2	2	2022-02-08	12:30:00	Warning alert issued for flooding
1	1	2022-01-02	09:00:00	Flood monitoring initiated

26) Retrieve flood data with magnitude greater than 3 and duration more than 24 hours:

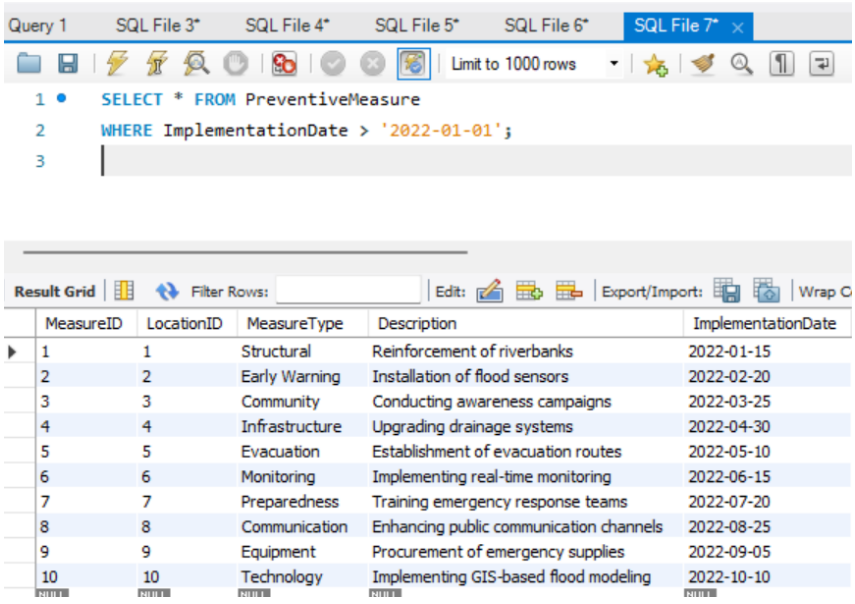
```
SELECT * FROM FloodData
WHERE Magnitude > 3
AND Duration > 24;
```



FloodDataID	LocationID	Date	Magnitude	Duration
1	1	2022-01-05	4.00	48
3	3	2022-03-20	5.00	72
5	5	2022-05-25	4.50	60
7	7	2022-07-15	4.00	36
9	9	2022-09-30	5.00	72
10	10	2022-10-18	4.50	60

27) Retrieve preventive measures implemented after a specific date:

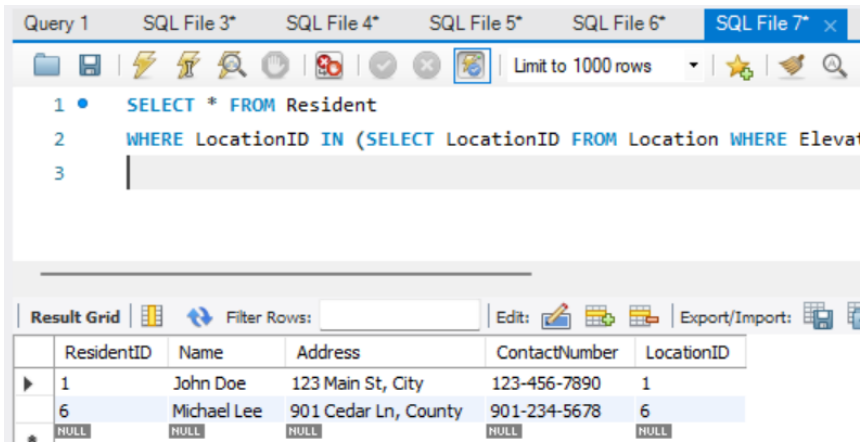
```
SELECT * FROM PreventiveMeasure
WHERE ImplementationDate > '2022-01-01';
```



MeasureID	LocationID	MeasureType	Description	ImplementationDate
1	1	Structural	Reinforcement of riverbanks	2022-01-15
2	2	Early Warning	Installation of flood sensors	2022-02-20
3	3	Community	Conducting awareness campaigns	2022-03-25
4	4	Infrastructure	Upgrading drainage systems	2022-04-30
5	5	Evacuation	Establishment of evacuation routes	2022-05-10
6	6	Monitoring	Implementing real-time monitoring	2022-06-15
7	7	Preparedness	Training emergency response teams	2022-07-20
8	8	Communication	Enhancing public communication channels	2022-08-25
9	9	Equipment	Procurement of emergency supplies	2022-09-05
10	10	Technology	Implementing GIS-based flood modeling	2022-10-10

28) Retrieve residents' contact information for low elevation areas:

```
SELECT * FROM Resident
WHERE LocationID IN (SELECT LocationID FROM Location WHERE
    Elevation < 15);
```



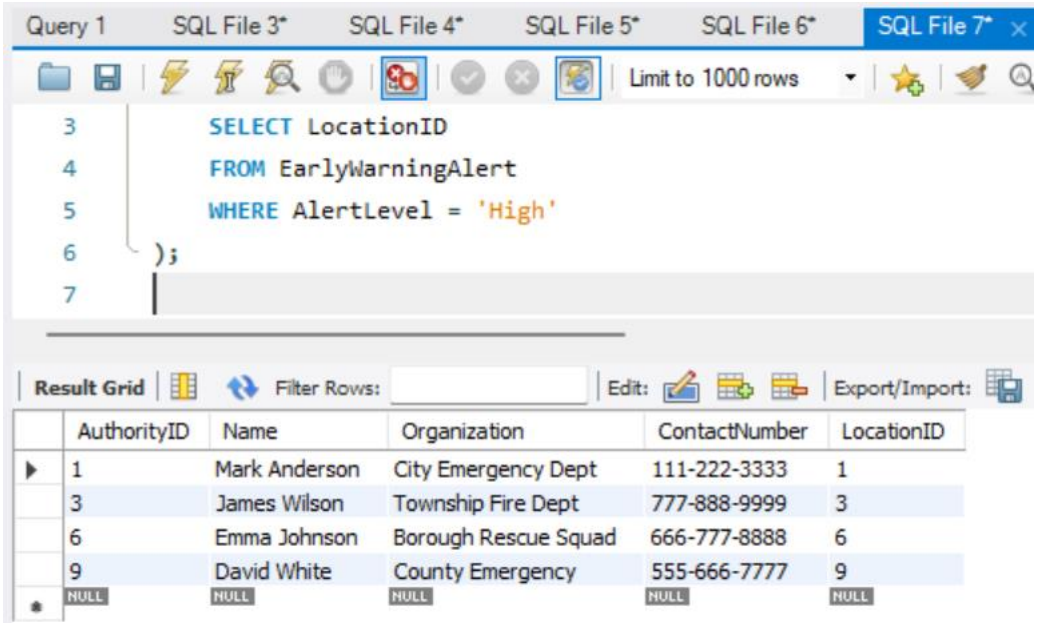
ResidentID	Name	Address	ContactNumber	LocationID
1	John Doe	123 Main St, City	123-456-7890	1
6	Michael Lee	901 Cedar Ln, County	901-234-5678	6

29) Retrieve authorities responsible for issuing high alert early warning alerts:

```
SELECT * FROM Authority
WHERE LocationID IN (
    SELECT LocationID
    FROM EarlyWarningAlert
```

WHERE AlertLevel = 'High'

);

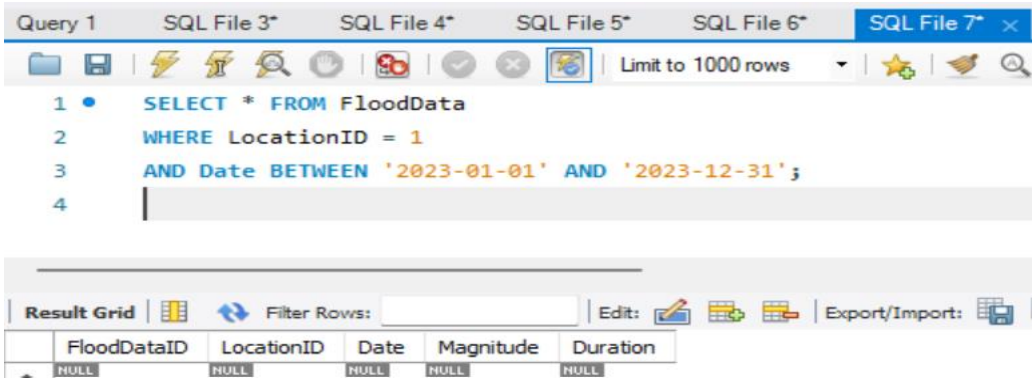


30) Retrieve flood data for a specific location and date range:

SELECT * FROM FloodData

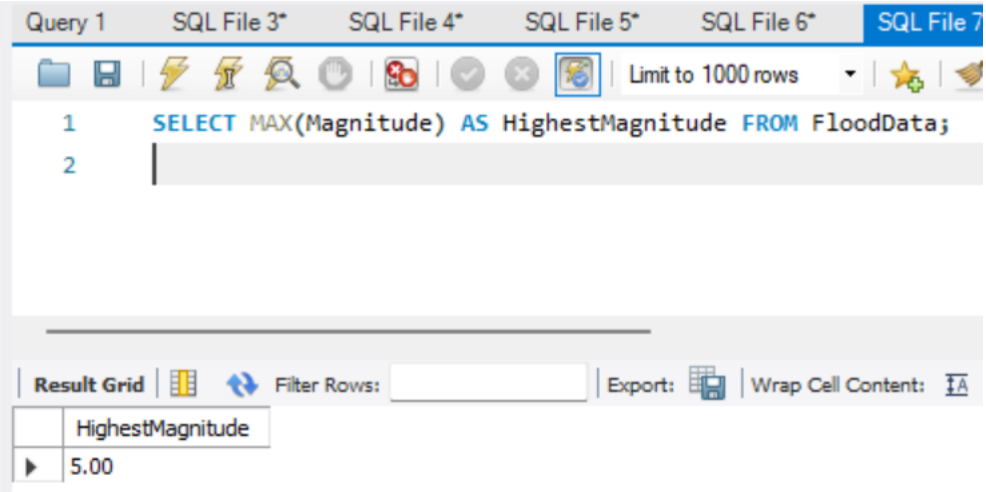
WHERE LocationID = 1

AND Date BETWEEN '2023-01-01' AND '2023-12-31';



31) Retrieve the highest magnitude flood recorded:

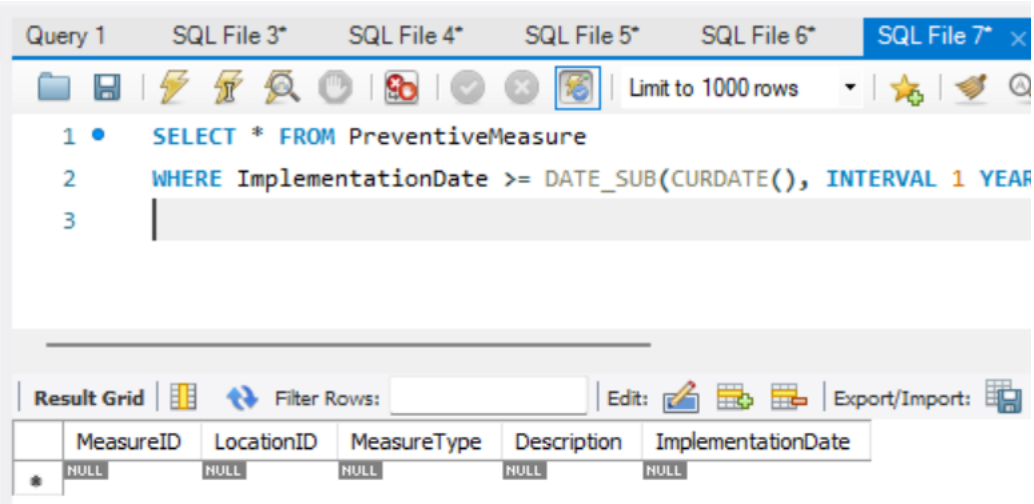
SELECT MAX(Magnitude) AS HighestMagnitude FROM FloodData;



32) Retrieve preventive measures with implementation dates within the last year:

SELECT * FROM PreventiveMeasure

WHERE ImplementationDate >= DATE_SUB(CURDATE(), INTERVAL 1 YEAR);



33) Retrieve residents' contact information for regions with flood occurrences:

SELECT DISTINCT r.Name, r.Address, r.ContactNumber

FROM Resident r

JOIN Location l ON r.LocationID = l.LocationID

JOIN FloodData f ON l.LocationID = f.LocationID;

```
SELECT DISTINCT r.Name, r.Address, r.ContactNumber
FROM Resident r
JOIN Location l ON r.LocationID = l.LocationID
JOIN FloodData f ON l.LocationID = f.LocationID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell C

Name	Address	ContactNumber
John Doe	123 Main St, City	123-456-7890
Jane Smith	456 Oak Ave, Town	456-789-0123
David Johnson	789 Elm Rd, Village	789-012-3456
Sarah Brown	321 Pine Blvd, Suburb	321-654-9870
Emily Wilson	567 Maple Dr, Hamlet	567-890-1234
Michael Lee	901 Cedar Ln, County	901-234-5678
Laura Miller	234 Birch St, Township	234-567-8901
Kevin Clark	678 Walnut Ave, Parish	678-901-2345
Amanda White	890 Spruce Rd, Borough	890-123-4567
Eric Taylor	345 Oakwood Dr, Manor	345-678-9012

34) Count the number of floods per year:

```
SELECT YEAR(Date) AS Year, COUNT(*) AS FloodCount
FROM FloodData
GROUP BY YEAR(Date);
```

Query 1 | SQL File 3* | SQL File 4* | SQL File 5* | SQL File 6* | SQL File 7*

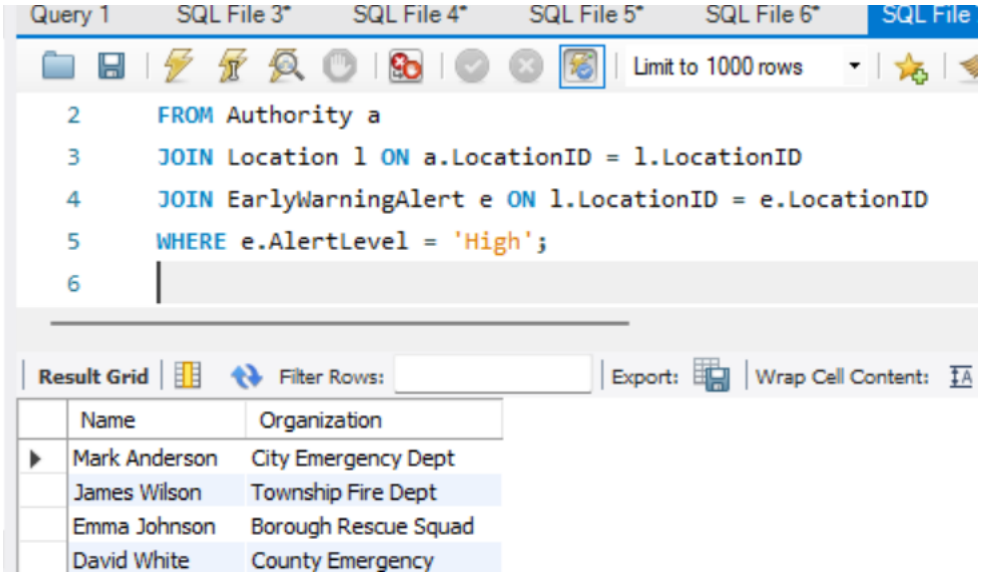
```
SELECT YEAR(Date) AS Year, COUNT(*) AS FloodCount
FROM FloodData
GROUP BY YEAR(Date);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Year	FloodCount
2022	10

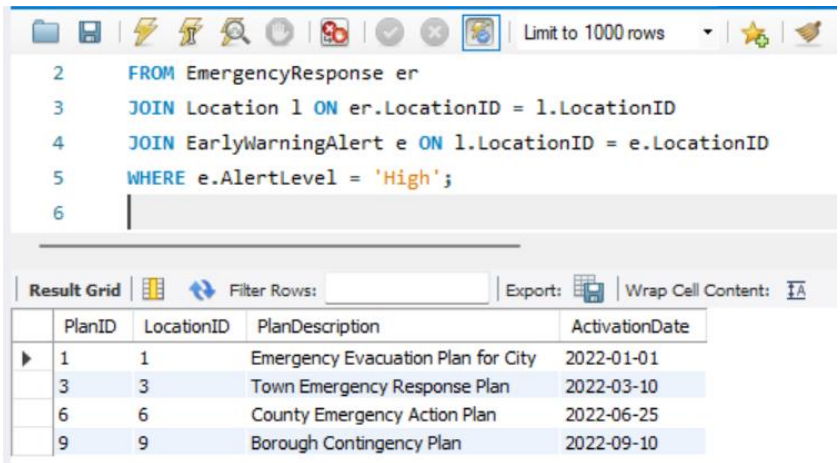
35) Retrieve authorities for regions with flood occurrences and high alert levels:

```
SELECT DISTINCT a.Name, a.Organization
FROM Authority a
JOIN Location l ON a.LocationID = l.LocationID
JOIN EarlyWarningAlert e ON l.LocationID = e.LocationID
WHERE e.AlertLevel = 'High';
```



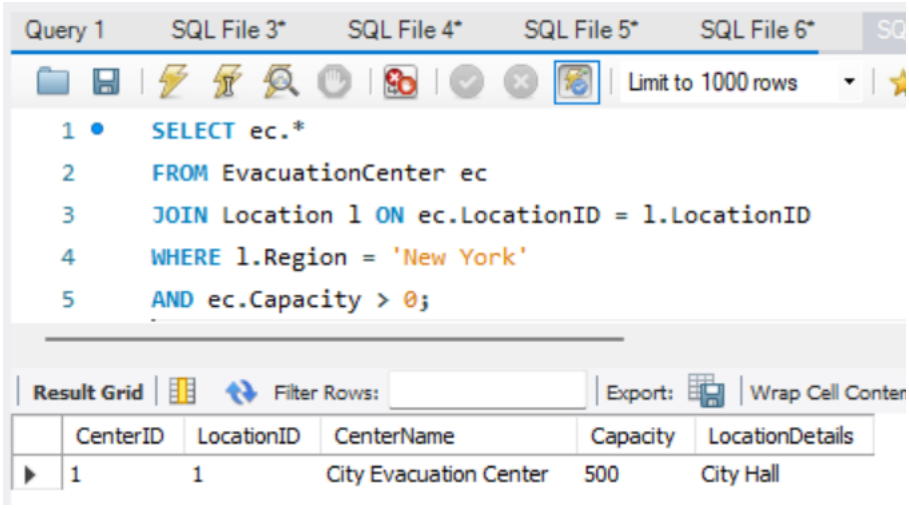
36) Retrieve emergency response plans for regions with flood occurrences and high alert levels:

```
SELECT DISTINCT er.*
FROM EmergencyResponse er
JOIN Location l ON er.LocationID = l.LocationID
JOIN EarlyWarningAlert e ON l.LocationID = e.LocationID
WHERE e.AlertLevel = 'High';
```



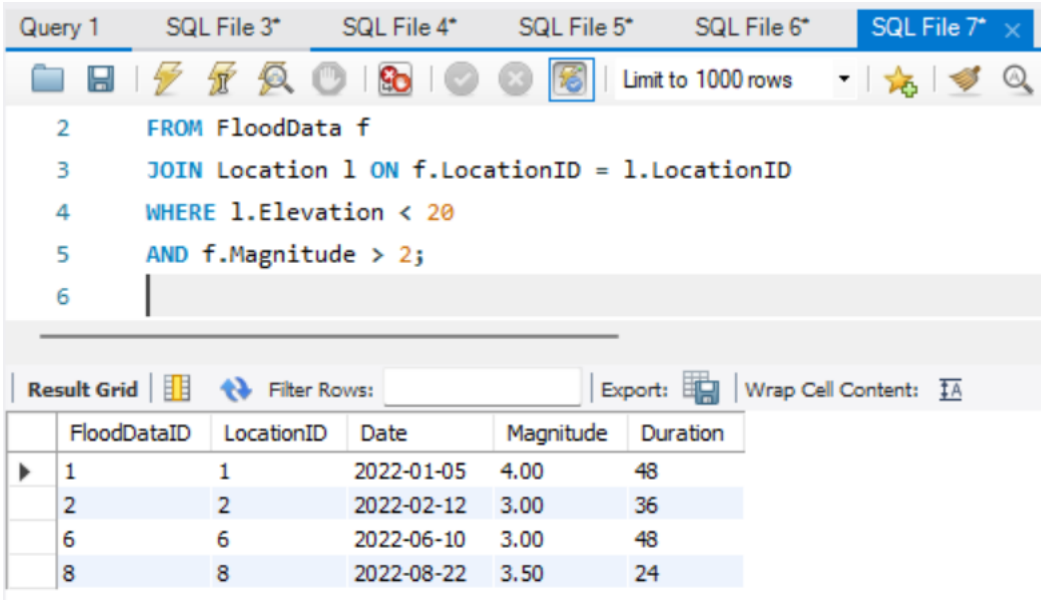
37) Retrieve evacuation centers with available capacity for a specific region:

```
SELECT ec.*
FROM EvacuationCenter ec
JOIN Location l ON ec.LocationID = l.LocationID
WHERE l.Region = 'New York'
AND ec.Capacity > 0;
```



38) Retrieve flood data for regions with elevation less than 20 and magnitude greater than 2:

```
SELECT f.*
FROM FloodData f
JOIN Location l ON f.LocationID = l.LocationID
WHERE l.Elevation < 20
AND f.Magnitude > 2;
```



39)FINAL QUERY:

```
-- Query 1: Analyze historical flood data
SELECT LocationID, AVG(Magnitude) AS AverageMagnitude, MAX(Magnitude)
AS MaxMagnitude, MIN(Magnitude) AS MinMagnitude,
      AVG(Duration) AS AverageDuration, COUNT(*) AS TotalFloods
FROM FloodData
GROUP BY LocationID;

-- Query 2: Determine areas with the highest flood risk
SELECT LocationID, Region, Elevation, AVG(Magnitude) AS AverageMagnitude
FROM Location
JOIN FloodData ON Location.LocationID = FloodData.LocationID
GROUP BY LocationID
HAVING AVG(Magnitude) > 4; -- Example threshold for high flood risk

-- Query 3: Generate early warning alerts
SELECT LocationID, DateIssued, AlertLevel, RecommendedActions
FROM EarlyWarningAlert
WHERE DateIssued >= CURDATE()
ORDER BY DateIssued;

-- Query 4: Retrieve preventive measures
SELECT LocationID, MeasureType, Description, ImplementationDate
FROM PreventiveMeasure
WHERE LocationID IN (SELECT LocationID FROM Location WHERE Elevation
< 15);

-- Query 5: Retrieve emergency response plans
SELECT LocationID, PlanDescription, ActivationDate
FROM EmergencyResponse
WHERE LocationID IN (SELECT LocationID FROM Location WHERE Elevation
< 15);

-- Query 6: Identify evacuation centers
SELECT LocationID, CenterName, Capacity, LocationDetails
FROM EvacuationCenter
WHERE LocationID IN (SELECT LocationID FROM Location WHERE Elevation
< 15);
```

OUTPUT:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

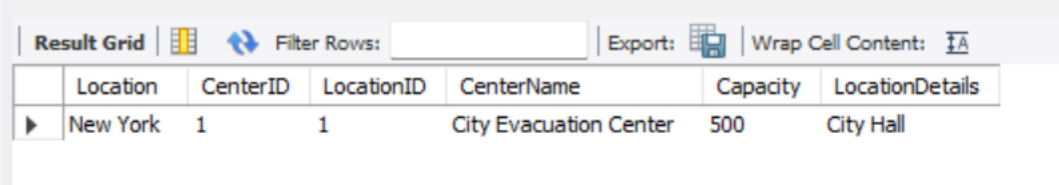
LocationID	AverageMagnitude	MaxMagnitude	MinMagnitude	AverageDuration	TotalFloods
1	4.000000	4.00	4.00	48.0000	1
2	3.000000	3.00	3.00	36.0000	1
3	5.000000	5.00	5.00	72.0000	1
4	3.500000	3.50	3.50	24.0000	1
5	4.500000	4.50	4.50	60.0000	1
6	3.000000	3.00	3.00	48.0000	1
7	4.000000	4.00	4.00	36.0000	1
8	3.500000	3.50	3.50	24.0000	1
9	5.000000	5.00	5.00	72.0000	1
10	4.500000	4.50	4.50	60.0000	1

The SQL queries aim to analyze historical flood data, determine areas with high flood risk, generate early warning alerts, retrieve preventive measures, retrieve emergency response plans, and identify evacuation centers based on location.

```
-- Analyzing Historical Flood Data
SELECT l.Region AS Location, AVG(fd.Magnitude) AS AverageMagnitude
FROM Location l
JOIN FloodData fd ON l.LocationID = fd.LocationID
GROUP BY l.Region;

-- Determining Areas with High Flood Risk
SELECT l.Region AS Location, AVG(fd.Magnitude) AS AverageMagnitude
FROM Location l
JOIN FloodData fd ON l.LocationID = fd.LocationID
GROUP BY l.Region
HAVING AVG(fd.Magnitude) > 4;

-- Generating Early Warning Alerts
SELECT l.Region AS Location, ewa.*
FROM Location l
JOIN EarlyWarningAlert ewa ON l.LocationID = ewa.LocationID
WHERE l.LocationID = 1; -- Example: LocationID 1

-- Retrieving Preventive Measures
SELECT l.Region AS Location, pm.*
FROM Location l
JOIN PreventiveMeasure pm ON l.LocationID = pm.LocationID
WHERE l.LocationID = 1; -- Example: LocationID 1

-- Retrieving Emergency Response Plans
SELECT l.Region AS Location, er.*
FROM Location l
JOIN EmergencyResponse er ON l.LocationID = er.LocationID
WHERE l.LocationID = 1; -- Example: LocationID 1
```

-- Identifying Evacuation Centers

```
SELECT l.Region AS Location, ec.*
FROM Location l
JOIN EvacuationCenter ec ON l.LocationID = ec.LocationID
WHERE l.LocationID = 1; -- Example: LocationID 1
```

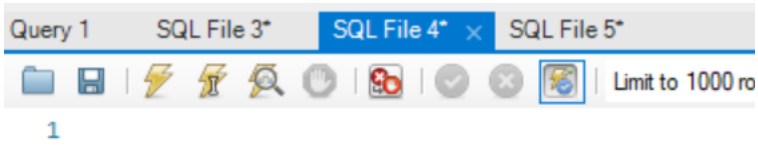


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Location	CenterID	LocationID	CenterName	Capacity	LocationDetails
New York	1	1	City Evacuation Center	500	City Hall

40) Query to Retrieve Historical Flood Data:

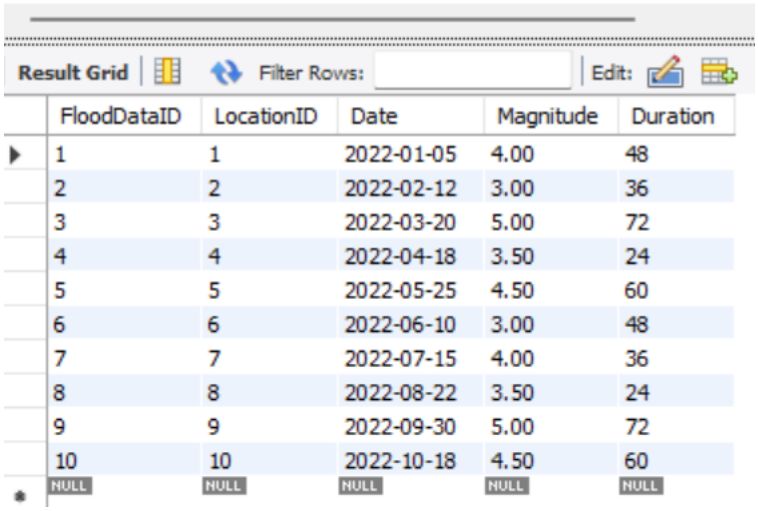
```
SELECT * FROM FloodData;
```



Query 1 | SQL File 3* | SQL File 4* | SQL File 5*

Limit to 1000 ro

```
1
2
3 •   SELECT * FROM FloodData;
4
```



Result Grid | Filter Rows: | Edit:

FloodDataID	LocationID	Date	Magnitude	Duration
1	1	2022-01-05	4.00	48
2	2	2022-02-12	3.00	36
3	3	2022-03-20	5.00	72
4	4	2022-04-18	3.50	24
5	5	2022-05-25	4.50	60
6	6	2022-06-10	3.00	48
7	7	2022-07-15	4.00	36
8	8	2022-08-22	3.50	24
9	9	2022-09-30	5.00	72
10	10	2022-10-18	4.50	60
NULL	NULL	NULL	NULL	NULL

FINAL RESULT OF THIS PROJECT:

```
-- 1. Analyzing Historical Flood Data for All Cities
SELECT fd.*, l.Region AS Location
FROM FloodData fd
JOIN Location l ON fd.LocationID = l.LocationID;

-- 2. Average Magnitude of Floods in Each Region
SELECT l.Region AS Location, AVG(fd.Magnitude) AS AverageMagnitude
```

```
FROM Location l
JOIN FloodData fd ON l.LocationID = fd.LocationID
GROUP BY l.Region;

-- 3. Early Warning Alerts for All Cities
SELECT ewa.*, l.Region AS Location
FROM EarlyWarningAlert ewa
JOIN Location l ON ewa.LocationID = l.LocationID;

-- 4. Preventive Measures for All Cities
SELECT pm.*, l.Region AS Location
FROM PreventiveMeasure pm
JOIN Location l ON pm.LocationID = l.LocationID;

-- 5. Emergency Response Plans for All Cities
SELECT er.*, l.Region AS Location
FROM EmergencyResponse er
JOIN Location l ON er.LocationID = l.LocationID;

-- 6. Evacuation Centers for All Cities
SELECT ec.*, l.Region AS Location
FROM EvacuationCenter ec
JOIN Location l ON ec.LocationID = l.LocationID;

-- 7. Query 7 Description
SELECT * FROM YourTableName WHERE YourCondition;


-- 8. Retrieve Emergency Response Logs for All Cities
SELECT erl.*, l.Region AS Location
FROM EmergencyResponseLog erl
JOIN Location l ON erl.LocationID = l.LocationID;

-- 9. Retrieve Authorities for All Cities
SELECT a.*, l.Region AS Location
FROM Authority a
JOIN Location l ON a.LocationID = l.LocationID;

-- 10. Flood Data for All Cities
SELECT fd.*, l.Region AS Location
FROM FloodData fd
JOIN Location l ON fd.LocationID = l.LocationID;

OUTPUT:
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

FloodDataID	LocationID	Date	Magnitude	Duration	Location
1	1	2022-01-05	4.00	48	New York
2	2	2022-02-12	3.00	36	Los Angeles
3	3	2022-03-20	5.00	72	London
4	4	2022-04-18	3.50	24	Paris
5	5	2022-05-25	4.50	60	Tokyo
6	6	2022-06-10	3.00	48	San Francisco
7	7	2022-07-15	4.00	36	Madrid
8	8	2022-08-22	3.50	24	Athens
9	9	2022-09-30	5.00	72	Rome
10	10	2022-10-18	4.50	60	Moscow

Result 26 | Result 27 | Result 28 | Result 29 | Result 30 | Result 31

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Location	AverageMagnitude
New York	4.000000
Los Angeles	3.000000
London	5.000000
Paris	3.500000
Tokyo	4.500000
San Francisco	3.000000
Madrid	4.000000
Athens	3.500000
Rome	5.000000
Moscow	4.500000

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

AlertID	LocationID	DateIssued	AlertLevel	RecommendedActions	Location
1	1	2022-01-03	High	Evacuate	New York
2	2	2022-02-10	Moderate	Stay Alert	Los Angeles
3	3	2022-03-18	High	Prepare	London
4	4	2022-04-15	Low	Monitor	Paris
5	5	2022-05-22	Moderate	Prepare	Tokyo
6	6	2022-06-08	High	Evacuate	San Francisco
7	7	2022-07-13	Low	Monitor	Madrid
8	8	2022-08-20	Moderate	Prepare	Athens
9	9	2022-09-28	High	Evacuate	Rome
10	10	2022-10-15	Low	Monitor	Moscow

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

MeasureID	LocationID	MeasureType	Description	ImplementationDate	Location
1	1	Structural	Reinforcement of riverbanks	2022-01-15	New York
2	2	Early Warning	Installation of flood sensors	2022-02-20	Los Angeles
3	3	Community	Conducting awareness campaigns	2022-03-25	London
4	4	Infrastructure	Upgrading drainage systems	2022-04-30	Paris
5	5	Evacuation	Establishment of evacuation routes	2022-05-10	Tokyo
6	6	Monitoring	Implementing real-time monitoring	2022-06-15	San Francisco
7	7	Preparedness	Training emergency response teams	2022-07-20	Madrid
8	8	Communication	Enhancing public communication channels	2022-08-25	Athens
9	9	Equipment	Procurement of emergency supplies	2022-09-05	Rome
10	10	Technology	Implementing GIS-based flood modeling	2022-10-10	Moscow

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

PlanID	LocationID	PlanDescription	ActivationDate	Location
1	1	Emergency Evacuation Plan for City	2022-01-01	New York
2	2	Disaster Preparedness Plan for County	2022-02-05	Los Angeles
3	3	Town Emergency Response Plan	2022-03-10	London
4	4	Parish Crisis Management Plan	2022-04-15	Paris
5	5	Hamlet Disaster Recovery Plan	2022-05-20	Tokyo
6	6	County Emergency Action Plan	2022-06-25	San Francisco
7	7	Township Disaster Management Plan	2022-07-30	Madrid
8	8	Village Emergency Preparedness Strategy	2022-08-05	Athens
9	9	Borough Contingency Plan	2022-09-10	Rome
10	10	Manor Disaster Response Framework	2022-10-15	Moscow

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

CenterID	LocationID	CenterName	Capacity	LocationDetails	Location
1	1	City Evacuation Center	500	City Hall	New York
2	2	County Shelter	300	Community Center	Los Angeles
3	3	Town Emergency Center	200	Town Hall	London
4	4	Parish Safe Haven	400	Parish School	Paris
5	5	Hamlet Evacuation Site	250	Hamlet Park	Tokyo
6	6	County Relief Center	350	County Fairgrounds	San Francisco
7	7	Township Refuge	150	Township Library	Madrid
8	8	Village Shelter	280	Village Hall	Athens
9	9	Borough Emergency Hub	180	Borough Community Center	Rome
10	10	Manor Safe Zone	200	Manor Civic Center	Moscow

RESULT:

Thus, working with queries related to project on MySQL is successfully verified.

Ex-8	WORKING WITH BASIC PL/SQL PROGRAMMING

AIM:

To work with basic PL/SQL programming on MySQL.

1. Write a PL/SL program to check given number is prime or not.

```
mysql> DELIMITER //
mysql> CREATE FUNCTION IsPrime(N INT) RETURNS VARCHAR(50) READS SQL DATA
    -> BEGIN
    ->     DECLARE I INT;
    ->     DECLARE TEMP INT;
    ->     SET I = 2;
    ->     SET TEMP = 0;
    ->
    ->     WHILE I <= N/2 DO
    ->         IF N % I = 0 THEN
    ->             SET TEMP = 1;
    ->             RETURN 'IT IS NOT A PRIME NUMBER';
    ->         END IF;
    ->         SET I = I + 1;
    ->     END WHILE;
    ->
    ->     IF TEMP = 1 THEN
    ->         RETURN 'IT IS NOT A PRIME NUMBER';
    ->     ELSE
    ->         RETURN 'IT IS A PRIME NUMBER';
    ->     END IF;
    ->
    -> END//
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> SELECT IsPrime(7);
+---------------------+
| IsPrime(7)          |
+---------------------+
| IT IS A PRIME NUMBER |
+---------------------+
1 row in set (0.01 sec)
```

2. To Find Factorial

```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION CalculateFactorial(n INT) RETURNS INT DETERMINISTIC
    -> BEGIN
    ->   DECLARE result INT DEFAULT 1;
    ->   DECLARE i INT DEFAULT 1;
    ->
    ->   WHILE i <= n DO
    ->     SET result = result * i;
    ->     SET i = i + 1;
    ->   END WHILE;
    ->
    ->   RETURN result;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT CalculateFactorial(5);
+---------------------+
| CalculateFactorial(5) |
+---------------------+
|                 120 |
+---------------------+
1 row in set (0.00 sec)
```

3. Write a PL/SL program to reverser a given number

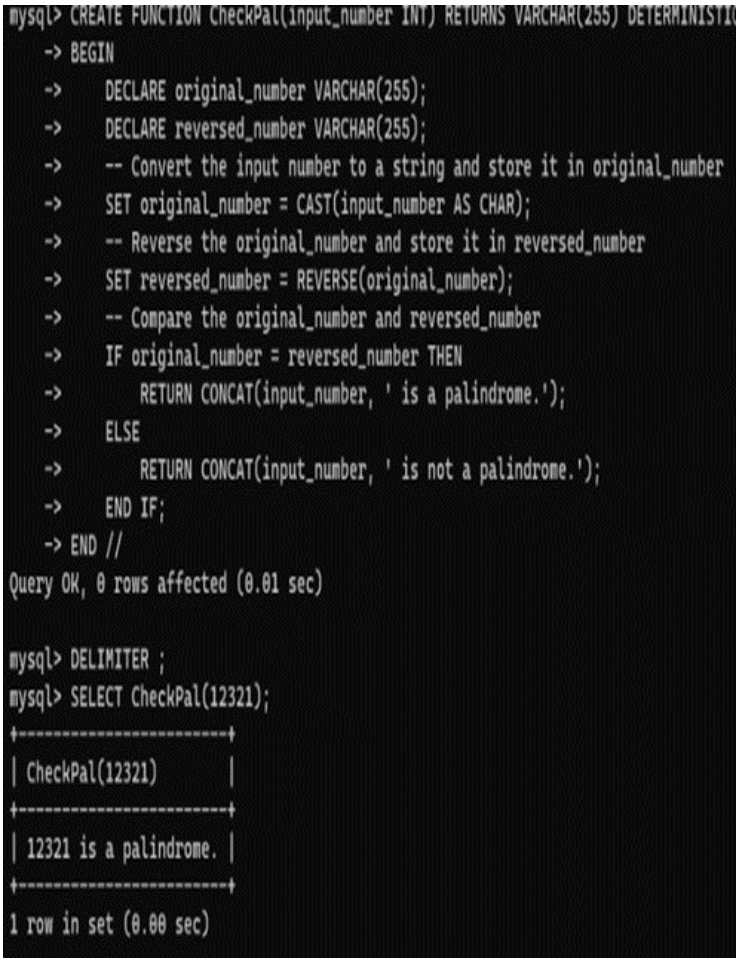
```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION ReverseNumber(n INT) RETURNS INT DETERMINISTIC
    -> BEGIN
    ->   DECLARE reversed INT DEFAULT 0;
    ->   DECLARE digit INT;
    ->
    ->   WHILE n > 0 DO
    ->     SET digit = n % 10;
    ->     SET reversed = reversed * 10 + digit;
    ->     SET n = FLOOR(n / 10);
    ->   END WHILE;
    ->
    ->   RETURN reversed;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
```

```
mysql> Select ReverseNumber(9786);
+----------------------+
| ReverseNumber(9786) |
+----------------------+
|                 6879 |
+----------------------+
1 row in set (0.00 sec)
```

4. Write a PL/SL program to generate a Fibonacci series

```
mysql> CREATE FUNCTION FibonacciSeries(terms INT) RETURNS VARCHAR(255) DETERMINISTIC
    -> BEGIN
    ->   DECLARE result VARCHAR(255) DEFAULT '0, 1';
    ->   DECLARE i INT DEFAULT 2;
    ->   DECLARE a INT DEFAULT 0;
    ->   DECLARE b INT DEFAULT 1;
    ->   DECLARE next_term INT;
    ->
    ->   IF terms <= 0 THEN
    ->     RETURN 'Invalid input';
    ->   ELSEIF terms = 1 THEN
    ->     RETURN '0';
    ->   ELSEIF terms = 2 THEN
    ->     RETURN '0, 1';
    ->   END IF;
    ->
    ->   WHILE i < terms DO
    ->     SET next_term = a + b;
    ->     SET result = CONCAT(result, ', ', next_term);
    ->     SET a = b;
    ->     SET b = next_term;
    ->     SET i = i + 1;
    ->   END WHILE;
    ->
    ->   RETURN result;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT FibonacciSeries(10);
+--------------------------------+
| FibonacciSeries(10)            |
+--------------------------------+
| 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 |
+--------------------------------+
1 row in set (0.00 sec)
```

5. Write a PL/SL program to check given number is palindrome or not



```
mysql> CREATE FUNCTION CheckPal(input_number INT) RETURNS VARCHAR(255) DETERMINISTIC
    -> BEGIN
    ->     DECLARE original_number VARCHAR(255);
    ->     DECLARE reversed_number VARCHAR(255);
    ->     -- Convert the input number to a string and store it in original_number
    ->     SET original_number = CAST(input_number AS CHAR);
    ->     -- Reverse the original_number and store it in reversed_number
    ->     SET reversed_number = REVERSE(original_number);
    ->     -- Compare the original_number and reversed_number
    ->     IF original_number = reversed_number THEN
    ->         RETURN CONCAT(input_number, ' is a palindrome.');
    ->     ELSE
    ->         RETURN CONCAT(input_number, ' is not a palindrome.');
    ->     END IF;
    -> END //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> SELECT CheckPal(12321);
+----------------------+
| CheckPal(12321)      |
+----------------------+
| 12321 is a palindrome. |
+----------------------+
1 row in set (0.00 sec)
```

6. CalculateSal

```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION CalculateSavings(salary INT, shifts INT) RETURNS INT DETERMINISTIC
    -> BEGIN
    ->   DECLARE savings INT;
    ->   DECLARE food_expense INT;
    ->   DECLARE travel_expense INT;
    ->   DECLARE shift_earnings INT;
    ->
    ->   -- Check if the salary is too large
    ->   IF salary > 8000 THEN
    ->     RETURN -1; -- Salary too large
    ->   END IF;
    ->
    ->   -- Check if the shifts are too small
    ->   IF shifts < 0 THEN
    ->     RETURN -2; -- Shifts too small
    ->   END IF;
    ->
    ->   -- Check if the salary is too small
    ->   IF salary < 0 THEN
    ->     RETURN -3; -- Salary too small
    ->   END IF;
    ->
    ->   -- Calculate the amount spent on food and travel
    ->   SET food_expense = salary * 20 / 100;
    ->   SET travel_expense = salary * 30 / 100;
    ->
    ->   -- Calculate the amount earned from shifts
    ->   SET shift_earnings = shifts * salary * 2 / 100;
    ->
    ->   -- Calculate savings
    ->   SET savings = salary - food_expense - travel_expense + shift_earnings;
    ->
    ->   RETURN savings;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT CalculateSavings(5000, 5);
+--------------------------+
| CalculateSavings(5000, 5) |
+--------------------------+
|                     3000 |
+--------------------------+
1 row in set (0.00 sec)
```

7. Product of Digits

```
mysql> DELIMITER //
mysql> CREATE FUNCTION productDigits(input_num INT) RETURNS VARCHAR(255) DETERMINISTIC
    -> BEGIN
    ->     DECLARE product INT;
    ->     DECLARE digit INT;
    ->     -- Check for invalid input conditions
    ->     IF input_num < 0 OR input_num > 32767 THEN
    ->         RETURN 'Invalid Input'; -- Return the message as a string
    ->     END IF;
    ->     SET product = 1;
    ->     -- Calculate the product of digits
    ->     WHILE input_num > 0 DO
    ->         SET digit = input_num % 10;
    ->         SET product = product * digit;
    ->         SET input_num = FLOOR(input_num / 10);
    ->     END WHILE;
    ->     RETURN CAST(product AS CHAR);
    -> END //
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> SELECT productDigits(45);
+------------------+
| productDigits(45) |
+------------------+
| 20               |
+------------------+
1 row in set (0.00 sec)
```

8. Power of Two

```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION IsPowerOfTwo(num INT) RETURNS VARCHAR(3) DETERMINISTIC
    -> BEGIN
    ->   IF num < 0 THEN
    ->     RETURN 'Number too small';
    ->   END IF;
    ->
    ->   IF num > 32767 THEN
    ->     RETURN 'Number too large';
    ->   END IF;
    ->
    ->   IF num > 0 AND (num & (num - 1)) = 0 THEN
    ->     RETURN 'Yes'; -- It's a power of 2
    ->   ELSE
    ->     RETURN 'No'; -- It's not a power of 2
    ->   END IF;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT IsPowerOfTwo(18);
+-----------------+
| IsPowerOfTwo(18) |
+-----------------+
| No              |
+-----------------+
1 row in set (0.00 sec)

mysql> SELECT IsPowerOfTwo(8);
+----------------+
| IsPowerOfTwo(8) |
+----------------+
| Yes            |
+----------------+
1 row in set (0.00 sec)
```

9. Decimal conversion

```
mysql> DELIMITER //
mysql> CREATE FUNCTION convertToDecimal(bin_num VARCHAR(10)) RETURNS VARCHAR(255) DETERMINISTIC
    -> BEGIN
    ->     DECLARE decimal_num INT DEFAULT 0;
    ->     DECLARE binary_digit CHAR(1);
    ->     DECLARE i INT;
    ->     -- Check for invalid input conditions
    ->     IF bin_num REGEXP '^[01]+$' = 0 OR LENGTH(bin_num) > 5 THEN
    ->         RETURN 'Invalid Input'; -- Return the message as a string
    ->     END IF;
    ->     -- Convert binary to decimal
    ->     SET i = LENGTH(bin_num);
    ->     WHILE i > 0 DO
    ->         SET binary_digit = SUBSTRING(bin_num, i, 1);
    ->         SET decimal_num = decimal_num + (binary_digit * POW(2, LENGTH(bin_num) - i));
    ->         SET i = i - 1;
    ->     END WHILE;
    ->     RETURN CAST(decimal_num AS CHAR);
    -> END //
Query OK, 0 rows affected (0.06 sec)

mysql> DELIMITER ;
mysql> SELECT  convertToDecimal(1001);
+----------------------+
| convertToDecimal(1001) |
+----------------------+
| 9                    |
+----------------------+
1 row in set (0.03 sec)
```

10. Arithmetic Operation

```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION PerformArithmeticOperation(input1 INT, input2 INT, operation INT) RETURNS INT DETERMINISTIC
    -> BEGIN
    ->   -- Check if input1 or input2 is negative or greater than 32767
    ->   IF input1 < 0 OR input1 > 32767 OR input2 < 0 OR input2 > 32767 THEN
    ->     RETURN -1;
    ->   END IF;
    ->
    ->   -- Check if the operation choice is in the range 1 to 4
    ->   IF operation < 1 OR operation > 4 THEN
    ->     RETURN -1;
    ->   END IF;
    ->
    ->   -- Perform the specified arithmetic operation
    ->   CASE operation
    ->     WHEN 1 THEN RETURN input1 + input2;
    ->     WHEN 2 THEN RETURN input1 - input2;
    ->     WHEN 3 THEN RETURN input1 * input2;
    ->     WHEN 4 THEN
    ->       -- Check if input2 is zero to avoid division by zero
    ->       IF input2 = 0 THEN
    ->         RETURN -1; -- Division by zero, return -1
    ->       ELSE
    ->         RETURN input1 / input2; -- Calculate the quotient
    ->       END IF;
    ->   END CASE;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT PerformArithmeticOperation(10, 5, 2);
+-------------------------------------+
| PerformArithmeticOperation(10, 5, 2) |
+-------------------------------------+
|                                   5 |
+-------------------------------------+
1 row in set (0.00 sec)
```

11. Digit Factorial

```
mysql> DELIMITER //
mysql> CREATE FUNCTION DigitFactorial(inputNumber INT) RETURNS VARCHAR(1000) DETERMINISTI
    -> BEGIN
    ->     DECLARE n INT DEFAULT inputNumber;
    ->     DECLARE factorial INT;
    ->     DECLARE result VARCHAR(1000) DEFAULT '';
    ->
    ->     WHILE n > 0 DO
    ->         SET factorial = 1;
    ->         SET @digit = n % 10;
    ->
    ->         IF @digit > 1 THEN
    ->             SET @i = 2;
    ->             WHILE @i <= @digit DO
    ->                 SET factorial = factorial * @i;
    ->                 SET @i = @i + 1;
    ->             END WHILE;
    ->         END IF;
    ->
    ->         SET result = CONCAT(result, factorial, ',');
    ->         SET n = FLOOR(n / 10);
    ->     END WHILE;
    ->
    ->     RETURN SUBSTRING(result, 1, LENGTH(result) - 1); -- Remove trailing comma
    -> END//
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> SELECT DigitFactorial(123);
+--------------------+
| DigitFactorial(123) |
+--------------------+
| 6,2,1              |
+--------------------+
1 row in set (0.00 sec)
```

12. Sum of Odd Digits

```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION SumOddDigits(input_num INT) RETURNS INT DETERMINISTIC
    -> BEGIN
    ->   DECLARE digit INT;
    ->   DECLARE sum_odd_digits INT DEFAULT 0;
    ->   DECLARE num INT;
    ->
    ->   -- Check if input_num is less than zero or greater than 32767
    ->   IF input_num < 0 OR input_num > 32767 THEN
    ->     RETURN -1;
    ->   END IF;
    ->
    ->   SET num = input_num;
    ->
    ->   -- Loop to extract and sum the odd digits
    ->   WHILE num > 0 DO
    ->     SET digit = num % 10;
    ->
    ->     IF digit % 2 <> 0 THEN
    ->       SET sum_odd_digits = sum_odd_digits + digit;
    ->     END IF;
    ->
    ->     SET num = FLOOR(num / 10);
    ->   END WHILE;
    ->
    ->   RETURN sum_odd_digits;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT SumOddDigits(15672);
+--------------------+
| SumOddDigits(15672) |
+--------------------+
|                 13 |
+--------------------+
1 row in set (0.00 sec)
```

13. Generate New number

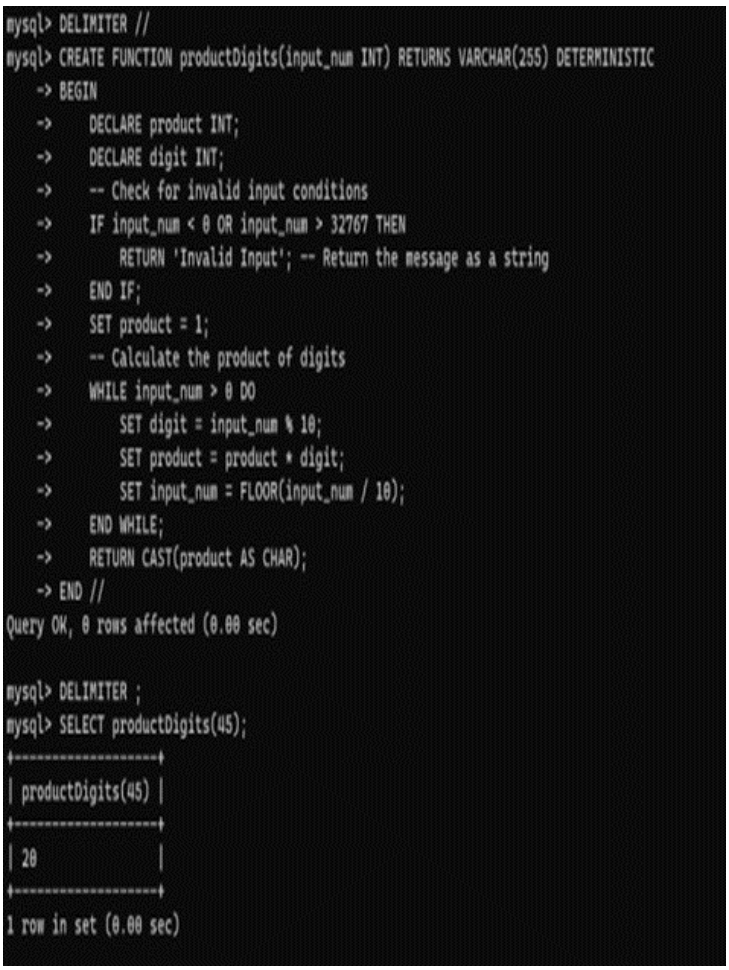
```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION GenerateNewNumber(input_num INT) RETURNS INT DETERMINISTIC
    -> BEGIN
    ->   DECLARE new_num INT DEFAULT 0;
    ->   DECLARE digit INT;
    ->   DECLARE multiplier INT DEFAULT 1;
    ->
    ->   -- Check if the input_num is less than zero or greater than 32767
    ->   IF input_num < 0 OR input_num > 32767 THEN
    ->     RETURN -1;
    ->   END IF;
    ->
    ->   -- Loop through the digits of the input_num
    ->   WHILE input_num > 0 DO
    ->     SET digit = input_num % 10;
    ->
    ->     IF digit % 2 = 0 THEN
    ->       -- Even digit, replace with the next even digit
    ->       SET digit = (digit + 2) % 10;
    ->     ELSE
    ->       -- Odd digit, replace with the next odd digit
    ->       SET digit = (digit + 2) % 9;
    ->     END IF;
    ->
    ->     -- Add the modified digit to the new_num
    ->     SET new_num = new_num + digit * multiplier;
    ->
    ->     -- Move to the next digit
    ->     SET input_num = FLOOR(input_num / 10);
    ->     SET multiplier = multiplier * 10;
    ->   END WHILE;
    ->
    ->   RETURN new_num;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> SELECT GenerateNewNumber(123);
+--------------------------+
| GenerateNewNumber(123) |
+--------------------------+
|                      345 |
+--------------------------+
1 row in set (0.00 sec)

mysql> SELECT GenerateNewNumber(32768);
+----------------------------+
| GenerateNewNumber(32768) |
+----------------------------+
|                         -1 |
+----------------------------+
1 row in set (0.00 sec)
```

14. Perfect Number

```
mysql> DELIMITER //
mysql> CREATE FUNCTION FindPerfect(inputNumber INT) RETURNS VARCHAR(20) DETERMINISTIC
    -> BEGIN
    ->     DECLARE sumOfDivisors INT DEFAULT 0;
    ->     DECLARE divisor INT DEFAULT 1;
    ->
    ->     IF inputNumber < 0 OR inputNumber > 32767 THEN
    ->         RETURN 'Invalid Input';
    ->     ELSE
    ->         WHILE divisor < inputNumber DO
    ->             IF inputNumber % divisor = 0 THEN
    ->                 SET sumOfDivisors = sumOfDivisors + divisor;
    ->             END IF;
    ->             SET divisor = divisor + 1;
    ->         END WHILE;
    ->
    ->         IF sumOfDivisors = inputNumber THEN
    ->             RETURN 'yes';
    ->         ELSE
    ->             RETURN 'no';
    ->         END IF;
    ->     END IF;
    -> END//
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> SELECT FindPerfect(28);
+----------------+
| FindPerfect(28) |
+----------------+
| yes            |
+----------------+
1 row in set (0.00 sec)
```

15. Product of digits



```
mysql> DELIMITER //
mysql> CREATE FUNCTION productDigits(input_num INT) RETURNS VARCHAR(255) DETERMINISTIC
    -> BEGIN
    ->     DECLARE product INT;
    ->     DECLARE digit INT;
    ->     -- Check for invalid input conditions
    ->     IF input_num < 0 OR input_num > 32767 THEN
    ->         RETURN 'Invalid Input'; -- Return the message as a string
    ->     END IF;
    ->     SET product = 1;
    ->     -- Calculate the product of digits
    ->     WHILE input_num > 0 DO
    ->         SET digit = input_num % 10;
    ->         SET product = product * digit;
    ->         SET input_num = FLOOR(input_num / 10);
    ->     END WHILE;
    ->     RETURN CAST(product AS CHAR);
    -> END //
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> SELECT productDigits(45);
+------------------+
| productDigits(45) |
+------------------+
| 20               |
+------------------+
1 row in set (0.00 sec)
```

RESULT: Thus, working with basic PL/SQL programming on MySQL is successfully verified.

Ex-8	

WORKING WITH PL/SQL PROCEDURES

AIM: To work with PL/SQL Procedures on MySQL.

1) Create a new User

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE CreateUser(
    ->     IN firstName VARCHAR(255),
    ->     IN lastName VARCHAR(255),
    ->     IN gender CHAR(1),
    ->     IN dob DATE,
    ->     IN city VARCHAR(255),
    ->     IN email VARCHAR(255),
    ->     IN accountsHolding TEXT,
    ->     IN contactNumber VARCHAR(15)
    -> )
    -> BEGIN
    ->     INSERT INTO user (First_name, last_name, gender, dob, city, email_id, accounts_holding, contact_number)
    ->     VALUES (firstName, lastName, gender, dob, city, email, accountsHolding, contactNumber);
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> CALL CreateUser('Abirami', 'M', 'F', '2005-05-01', 'Salem', 'a@gmail.com', 'Saving', '789');
ERROR 1366 (HY000): Incorrect integer value: 'Saving' for column 'accounts_holding' at row 1
mysql> CALL CreateUser('Abirami', 'M', 'F', '2005-05-01', 'Salem', 'a@gmail.com', 3, 789);
Query OK, 1 row affected (0.01 sec)

mysql> select * from user;
+---------+------------+-----------+--------+------------+------------+-------------+------------------+----------------+
| user_id | First_name | last_name | gender | dob        | city       | email_id    | accounts_holding | contact_number |
+---------+------------+-----------+--------+------------+------------+-------------+------------------+----------------+
|     101 | Lokesh     | Kumar     | M      | 2004-01-01 | Chennai    | l@gmail.com |                1 |            123 |
|     202 | Divya      | Murugesan | F      | 2004-04-05 | Bangalore  | d@gmail.com |                2 |            890 |
|     303 | Nikhil     | Krishnan  | M      | 2004-05-02 | Coimbatore | s@gmail.com |                3 |            890 |
|     404 | Kavin      | V         | M      | 2004-05-01 | Coimbatore | k@gmail.com |                2 |            908 |
|     505 | Nikitha    | k         | M      | 2005-09-09 | Chennai    | b@gmail.com |                1 |            678 |
|    NULL | Abirami    | M         | F      | 2005-05-01 | Salem      | a@gmail.com |                3 |            789 |
+---------+------------+-----------+--------+------------+------------+-------------+------------------+----------------+
6 rows in set (0.00 sec)
```

2) Create a new account

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE AddAccount(
    ->     IN accountNumber INT,
    ->     IN holderName VARCHAR(255),
    ->     IN accountType VARCHAR(255),
    ->     IN balance DECIMAL(10, 2)
    -> )
    -> BEGIN
    ->     INSERT INTO account (Account_number, Holder_name, Account_type, Balance)
    ->     VALUES (accountNumber, holderName, accountType, balance);
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> CALL AddAccount(6, 'Abirami', 'Savings', 1000.00);
Query OK, 1 row affected (0.01 sec)

mysql> select * from account;
+----------------+-------------+--------------+--------------+
| Account_number | Holder_name | Account_type | Balance      |
+----------------+-------------+--------------+--------------+
|              1 | Lokesh      | Savings      |   10000.0000 |
|              2 | Divya       | Savings      |  200000.0000 |
|              3 | Nikhil      | Business     | 4000000.0000 |
|              4 | Kavin       | Current      |    3000.0000 |
|              5 | Nikitha     | Business     |   80000.0000 |
|              6 | Abirami     | Savings      |    1000.0000 |
+----------------+-------------+--------------+--------------+
6 rows in set (0.00 sec)
```

3) Create a new bill

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE AddBill(
    ->     IN userId INT,
    ->     IN name VARCHAR(255),
    ->     IN billType VARCHAR(255),
    ->     IN status VARCHAR(255),
    ->     IN dueDate DATE,
    ->     IN amount DECIMAL(10, 2)
    -> )
    -> BEGIN
    ->     INSERT INTO bill (User_id, Name, Bill_type, Status, Due_date, ammount)
    ->     VALUES (userId, name, billType, status, dueDate, amount);
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
```

```
mysql> select * from bill;
+---------+---------+---------+-----------+---------+------------+-----------+
| Bill_id | User_id | Name    | Bill_type | Status  | Due_date   | ammount   |
+---------+---------+---------+-----------+---------+------------+-----------+
|       1 |     101 | Lokesh  | GST       | Pending | 2023-11-09 |   3000.00 |
|       2 |     202 | Divya   | NONGST    | Pending | 2023-09-10 |  10000.00 |
|       3 |     303 | Nikhil  | GST       | Pending | 2023-11-07 | 100000.00 |
|       4 |     404 | Kavin   | GST       | Pending | 2023-11-20 |   1000.00 |
|       5 |     505 | Nikitha | NONGST    | Pending | 2023-03-01 |  20000.00 |
|    NULL |     606 | Abirami | NONGST    | Pending | 2023-11-01 |     50.00 |
+---------+---------+---------+-----------+---------+------------+-----------+
```

4) Create a new Investment

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE CreateInvestment3(
    ->     IN investmentId INT,
    ->     IN userId INT,
    ->     IN amountId INT,
    ->     IN investmentDate DATE,
    ->     IN investmentType VARCHAR(255)
    -> )
    -> BEGIN
    ->     INSERT INTO investment (Investment_id, User_id, Ammount_id, Date_of_investment, Investment_type)
    ->     VALUES (investmentId, userId, amountId, investmentDate, investmentType);
    -> END;
    ->
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> CALL CreateInvestment3(6, 606, 107, '2023-11-01', 'Stocks');
Query OK, 1 row affected (0.01 sec)

mysql> select * from investment;
+---------------+---------+------------+--------------------+-----------------+
| Investment_id | User_id | Ammount_id | Date_of_investment | Investment_type |
+---------------+---------+------------+--------------------+-----------------+
|             1 |     101 |        102 | 2023-10-01         | Stock           |
|             2 |     202 |        103 | 2023-10-10         | Mutual Funds    |
|             3 |     303 |        104 | 2023-10-15         | Bonds           |
|             4 |     404 |        105 | 2023-10-20         | Stock           |
|             5 |     505 |        106 | 2023-10-25         | Bonds           |
|             6 |     606 |        107 | 2023-11-01         | Stocks          |
+---------------+---------+------------+--------------------+-----------------+
6 rows in set (0.00 sec)
```

5) Create a new loan

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE CreateLoan3(
    ->     IN loadID INT,
    ->     IN accountNo INT,
    ->     IN amount DECIMAL(10, 2),
    ->     IN term INT,
    ->     IN rate DECIMAL(5, 2)
    -> )
    -> BEGIN
    ->     INSERT INTO loan (loan_id, accountno, ammount, term, rate)
    ->     VALUES (loadID, accountNo, amount, term, rate);
    -> END;
    ->
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> CALL CreateLoan3(6000, 6, 5000.00, 12, 5.5);
Query OK, 1 row affected (0.01 sec)

mysql> select * from loan;
+---------+-----------+------------+------+------+
| loan_id | accountno | ammount    | term | rate |
+---------+-----------+------------+------+------+
|    1000 |         1 |  600000.00 |    2 | 5.00 |
|    2000 |         2 |  100000.00 |    1 | 2.00 |
|    3000 |         3 | 1000000.00 |    5 | 3.00 |
|    4000 |         4 |   50000.00 |    3 | 2.00 |
|    5000 |         5 |  200000.00 |    2 | 3.00 |
|    6000 |         6 |    5000.00 |   12 | 5.50 |
+---------+-----------+------------+------+------+
6 rows in set (0.00 sec)
```

RESULT: Thus, working with PL/SQL Procedures on MySQL is successfully verified.

Ex-10

WORKING WITH PL/SQL FUNCTIONS

AIM: To work with PL/SQL Functions on MySQL

1. Function to Calculate Total Account Balance for a User:

```
mysql> DELIMITER //
mysql> CREATE FUNCTION GetUserTotalBalance1(userId INT) RETURNS DECIMAL(10, 2) DETERMINISTIC
    -> BEGIN
    ->     DECLARE totalBalance DECIMAL(10, 2) DEFAULT 0;
    ->
    ->     SELECT COALESCE(SUM(Balance), 0) INTO totalBalance
    ->     FROM account
    ->     WHERE Account_number = userId;
    ->
    ->     RETURN totalBalance;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> SELECT GetUserTotalBalance1(1) AS UserTotalBalance;
+------------------+
| UserTotalBalance |
+------------------+
|         10000.00 |
+------------------+
1 row in set (0.00 sec)
```

2. Function to Calculate Total Bills Amount for a User

```
mysql> DELIMITER //
mysql> CREATE FUNCTION GetUserTotalBillsAmount(userId INT) RETURNS DECIMAL(10, 2) DETERMINIST
    -> BEGIN
    ->     DECLARE totalBillAmount DECIMAL(10, 2) DEFAULT 0;
    ->
    ->     SELECT COALESCE(SUM(ammount), 0) INTO totalBillAmount
    ->     FROM bill
    ->     WHERE User_id = userId;
    ->
    ->     RETURN totalBillAmount;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT GetUserTotalBillsAmount(101) AS UserTotalBillsAmount;
+----------------------+
| UserTotalBillsAmount |
+----------------------+
|              3000.00 |
+----------------------+
1 row in set (0.00 sec)
```

3. Function to Calculate Age from Date of Birth:

```
mysql> DELIMITER //
mysql> CREATE FUNCTION CalculateUserAge(dateOfBirth DATE) RETURNS INT DETERMINISTIC
    -> BEGIN
    ->     DECLARE userAge INT;
    ->
    ->     SET userAge = TIMESTAMPDIFF(YEAR, dateOfBirth, CURDATE());
    ->
    ->     RETURN userAge;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT CalculateUserAge('2004-01-01') AS UserAge;
+---------+
| UserAge |
+---------+
|      19 |
+---------+
1 row in set (0.00 sec)
```

4. Function to Retrieve User's Full Name

```
mysql> DELIMITER //
mysql> CREATE FUNCTION GetUserFullName(userId INT) RETURNS VARCHAR(255) DETERMINISTIC
    -> BEGIN
    ->     DECLARE userFullName VARCHAR(255);
    ->
    ->     SELECT CONCAT(First_name, ' ', last_name) INTO userFullName
    ->     FROM user
    ->     WHERE user_id = userId;
    ->
    ->     RETURN userFullName;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT GetUserFullName(303) AS FullName;
+-----------------+
| FullName        |
+-----------------+
| Nikhil Krishnan |
+-----------------+
1 row in set (0.00 sec)
```

5. Function to Check if a User has a Specific Account Type

```
mysql> DELIMITER //
mysql> CREATE FUNCTION UserHasAccountType(userId INT, accountTypeToCheck CHAR(20)) RETURNS BOOLEAN DETERMINISTIC
    -> BEGIN
    ->     DECLARE hasAccountType BOOLEAN;
    ->
    ->     SELECT COUNT(*) INTO hasAccountType
    ->     FROM account
    ->     WHERE Account_number = userId AND Account_type = accountTypeToCheck;
    ->
    ->     RETURN hasAccountType > 0;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> SELECT UserHasAccountType(101, 'Savings') AS HasSavingsAccount;
+-------------------+
| HasSavingsAccount |
+-------------------+
|                 0 |
+-------------------+
1 row in set (0.00 sec)
```

RESULT: Thus, working with PL/SQL Functions on MySQL is successfully verified.

Ex-11	WORKING WITH PL/SQL CURSORS

AIM: To work with PL/SQL Cursors on MySQL.

1. Cursor to Retrieve User's Bills

```
mysql>  DELIMITER //
mysql> CREATE PROCEDURE GetUserBills1(IN userId INT)
    -> BEGIN
    ->     DECLARE done INT DEFAULT 0;
    ->     DECLARE billId INT;
    ->     DECLARE billName VARCHAR(255);
    ->     DECLARE billType VARCHAR(255);
    ->     DECLARE billStatus VARCHAR(255);
    ->     DECLARE dueDate DATE;
    ->     DECLARE amount DECIMAL(10, 2);
    ->
    ->     DECLARE cur CURSOR FOR
    ->     SELECT Bill_id, Name, Bill_type, Status, Due_date, ammount
    ->     FROM bill
    ->     WHERE User_id = userId;
    ->
    ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    ->
    ->     OPEN cur;
    ->
    ->     read_loop: LOOP
    ->         FETCH cur INTO billId, billName, billType, billStatus, dueDate, amount;
    ->         IF done = 1 THEN
    ->             LEAVE read_loop;
    ->         END IF;
    ->
    ->         -- Process the retrieved data here
    ->         SELECT billId, billName, billType, billStatus, dueDate, amount;
    ->     END LOOP;
    ->
    ->     CLOSE cur;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> CALL GetUserBills1(101);
+--------+----------+----------+------------+------------+---------+
| billId | billName | billType | billStatus | dueDate    | amount  |
+--------+----------+----------+------------+------------+---------+
|      1 | Lokesh   | GST      | Pending    | 2023-11-09 | 3000.00 |
+--------+----------+----------+------------+------------+---------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.02 sec)
```

2. Cursor to Calculate Total Balance

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE CalculateTotalBalance()
    -> BEGIN
    ->     DECLARE done INT DEFAULT 0;
    ->     DECLARE totalBalance DECIMAL(10, 2) DEFAULT 0;
    ->     DECLARE accountBalance DECIMAL(10, 2);
    ->
    ->     DECLARE cur CURSOR FOR
    ->     SELECT Balance FROM account;
    ->
    ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    ->
    ->     OPEN cur;
    ->
    ->     read_loop: LOOP
    ->         FETCH cur INTO accountBalance;
    ->         IF done = 1 THEN
    ->             LEAVE read_loop;
    ->         END IF;
    ->
    ->         -- Calculate total balance
    ->         SET totalBalance = totalBalance + accountBalance;
    ->     END LOOP;
    ->
    ->     CLOSE cur;
    ->
    ->     -- Return the total balance
    ->     SELECT totalBalance;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> CALL CalculateTotalBalance();
+--------------+
| totalBalance |
+--------------+
|   4294000.00 |
+--------------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

3. Cursor to List Users with Overdue Bills

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE ListUsersWithOverdueBills1()
    -> BEGIN
    ->     DECLARE done INT DEFAULT 0;
    ->     DECLARE userId INT;
    ->     DECLARE firstName VARCHAR(255);
    ->     DECLARE lastName VARCHAR(255);
    ->
    ->     DECLARE cur CURSOR FOR
    ->     SELECT u.user_id, u.First_name, u.Last_name
    ->     FROM user u
    ->     JOIN bill b ON u.user_id = b.User_id
    ->     WHERE b.Status = 'Pending' AND b.Due_date < CURDATE();
    ->
    ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    ->
    ->     OPEN cur;
    ->
    ->     read_loop: LOOP
    ->         FETCH cur INTO userId, firstName, lastName;
    ->         IF done = 1 THEN
    ->             LEAVE read_loop;
    ->         END IF;
    ->
    ->         -- Process the retrieved user data
    ->         SELECT userId, firstName, lastName;
    ->     END LOOP;
    ->
    ->     CLOSE cur;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> CALL ListUsersWithOverdueBills1();
+--------+-----------+-----------+
| userId | firstName | lastName  |
+--------+-----------+-----------+
|    202 | Divya     | Murugesan |
+--------+-----------+-----------+
1 row in set (0.00 sec)

+--------+-----------+----------+
| userId | firstName | lastName |
+--------+-----------+----------+
|    505 | Nikitha   | k        |
+--------+-----------+----------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.02 sec)
```

4. Cursor to Retrieve Investment Details

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE GetUserInvestments(IN userId INT)
    -> BEGIN
    ->     DECLARE done INT DEFAULT 0;
    ->     DECLARE investmentId INT;
    ->     DECLARE investmentType VARCHAR(255);
    ->     DECLARE investmentDate DATE;
    ->
    ->     DECLARE cur CURSOR FOR
    ->     SELECT Investment_id, Investment_type, Date_of_investment
    ->     FROM Investment
    ->     WHERE User_id = userId;
    ->
    ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    ->
    ->     OPEN cur;
    ->
    ->     read_loop: LOOP
    ->         FETCH cur INTO investmentId, investmentType, investmentDate;
    ->         IF done = 1 THEN
    ->             LEAVE read_loop;
    ->         END IF;
    ->
    ->         -- Process the retrieved investment data here
    ->         SELECT investmentId, investmentType, investmentDate;
    ->     END LOOP;
    ->
    ->     CLOSE cur;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> CALL GetUserInvestments(303);
+--------------+----------------+----------------+
| investmentId | investmentType | investmentDate |
+--------------+----------------+----------------+
|            3 | Bonds          | 2023-10-15     |
+--------------+----------------+----------------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

RESULT: Thus, working with PL/SQL Cursors on MySQL is successfully verified

Ex-12	**WORKING WITH PL/SQL TRIGGERS**

AIM: To work with PL/SQL Triggers on MySQL.

1. Trigger to Update User Account Balance After Bill Payment:

```
mysql> DELIMITER //
mysql> CREATE TRIGGER update_balance_after_payment
    -> AFTER INSERT ON bill
    -> FOR EACH ROW
    -> BEGIN
    ->     UPDATE account
    ->     SET Balance = Balance - NEW.ammount
    ->     WHERE Account_number = NEW.User_id;
    -> END;
    -> //
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> SELECT * FROM account WHERE Account_number = 1;
+----------------+-------------+--------------+-----------+
| Account_number | Holder_name | Account_type | Balance   |
+----------------+-------------+--------------+-----------+
|              1 | Lokesh      | Savings      | 9950.0000 |
+----------------+-------------+--------------+-----------+
1 row in set (0.00 sec)
```

2. Trigger to Enforce a Maximum Investment Limit:

```
mysql> DELIMITER //
mysql> CREATE TRIGGER enforce_investment_limit
    -> BEFORE INSERT ON investment
    -> FOR EACH ROW
    -> BEGIN
    ->     DECLARE max_investment DECIMAL(10, 2) DEFAULT 10000.00;
    ->     IF NEW.Investment_type = 'Stocks' AND NEW.Ammount_id > max_investment THEN
    ->         SIGNAL SQLSTATE '45000'
    ->         SET MESSAGE_TEXT = 'Investment exceeds maximum limit';
    ->     END IF;
    -> END;
    -> //
Query OK, 0 rows affected (0.02 sec)

mysql> DELIMITER ;
mysql> INSERT INTO investment (Investment_type, Ammount_id) VALUES ('Stocks', 15000.00);
ERROR 1644 (45000): Investment exceeds maximum limit
```

3. Trigger to Calculate and Update Loan Repayments:

```
mysql> DELIMITER //
mysql> CREATE TRIGGER calculate_loan_repayment1
    -> BEFORE INSERT ON loan
    -> FOR EACH ROW
    -> BEGIN
    ->     SET NEW.ammount =NEW.loan_id+NEW.accountno+ NEW.ammount + (NEW.ammount * NEW.rate / 100) * NEW.term;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO loan (loan_id,accountno,ammount, rate, term) VALUES (7000,7,10000, 5, 2);
    -> delemitter;
    -> INSERT INTO loan (loan_id,accountno,ammount, rate, term) VALUES (7000,7,10000, 5, 2);
    -> //
Query OK, 1 row affected (0.01 sec)
```

```
mysql> select * from loan;
    -> //
+---------+-----------+------------+------+------+
| loan_id | accountno | ammount    | term | rate |
+---------+-----------+------------+------+------+
|    1000 |         1 |  600000.00 |    2 | 5.00 |
|    2000 |         2 |  100000.00 |    1 | 2.00 |
|    3000 |         3 | 1000000.00 |    5 | 3.00 |
|    4000 |         4 |   50000.00 |    3 | 2.00 |
|    5000 |         5 |  200000.00 |    2 | 3.00 |
|    6000 |         6 |    5000.00 |   12 | 5.50 |
|    7000 |         7 |   19107.00 |    2 | 5.00 |
+---------+-----------+------------+------+------+
7 rows in set (0.00 sec)
```

4. Trigger to Prevent Deletion of User Accounts with Active Bills:

```
mysql> DELIMITER //
mysql> CREATE TRIGGER prevent_account_deletion1
    -> BEFORE DELETE ON user
    -> FOR EACH ROW
    -> BEGIN
    ->     DECLARE unpaid_bills INT;
    ->     SELECT COUNT(*) INTO unpaid_bills
    ->     FROM bill
    ->     WHERE User_id = OLD.user_id AND Status = 'Pending';
    ->     IF unpaid_bills > 0 THEN
    ->         SIGNAL SQLSTATE '45000'
    ->         SET MESSAGE_TEXT = 'Cannot delete user with unpaid bills';
    ->     END IF;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql>
mysql> DELETE FROM user WHERE user_id = 101;
ERROR 1644 (45000): Cannot delete user with unpaid bills
```

RESULT: Thus, working with PL/SQL Triggers on MySQL is successfully verified.

Ex-13	PROJECT REPORT ANALYSIS

1. Introduction:

Flooding is a recurring natural disaster that poses significant threats to lives and properties worldwide. In response to the escalating risks associated with flooding incidents, the development of an advanced Flood Risk Prediction and Early Warning System is imperative. This project aims to address this need by leveraging data integration, predictive modeling, and advanced technologies to provide timely and accurate flood risk predictions and early warning alerts.

2. Objective:

The primary objective of the Flood Risk Prediction and Early Warning System project is to create a comprehensive platform that integrates historical flood data, terrain particulars, climate models, and predictive analytics to accurately forecast flood risks. The system will deliver proactive early warning alerts to residents and authorities, enabling them to take preemptive measures and mitigate potential damages effectively. Key objectives include data integration, database design and management, flood risk prediction, early warning system implementation, user-friendly interface development, and scalability and reliability enhancement.

3. Technologies Used:

- Programming Languages: Python, JavaScript
- Frameworks: Flask (Backend), React (Frontend)
- Database Management System: PostgreSQL
- Data Analysis and Modeling: Pandas, Scikit-learn
- Geographic Information System (GIS): ArcGIS, QGIS
- Cloud Services: AWS, Google Cloud Platform
- Security: SSL/TLS encryption, Authentication, Authorization

4. Features:

- **Data Integration:** Integration of historical flood data, terrain particulars, and climate models.
- **Flood Risk Prediction:** Utilization of predictive modeling techniques to forecast flood risks.
- **Early Warning System:** Provision of punctual and dependable early warning alerts to residents and authorities.

- **User-friendly Interface:** Development of an intuitive and easy-to-navigate interface for users.
- **Scalability and Reliability:** Implementation of scalable and reliable infrastructure to handle varying loads and ensure system availability.

5. Database Structure:

The database structure includes tables for storing historical flood data, terrain particulars, climate models, user information, early warning alerts, and system logs. Relationships between tables are established using primary and foreign keys to ensure data integrity and consistency.

6. Implementation:

- Frontend: Development of a responsive and intuitive frontend interface using React.js.
- Backend: Implementation of backend logic and APIs using Flask framework to handle data processing and communication with the frontend.
- Database Integration: Integration of PostgreSQL database to store and manage data efficiently.
- Security Measures: Implementation of SSL/TLS encryption, user authentication, and authorization mechanisms to ensure data security and user privacy.
- Maintenance: Regular monitoring, updates, and maintenance to ensure system reliability and performance optimizations.

7. Conclusion:

The Flood Risk Prediction and Early Warning System project aims to provide a comprehensive solution to address the challenges posed by flooding incidents. By integrating historical data, predictive analytics, and advanced technologies, the system offers proactive measures to mitigate risks and enhance community resilience. With its user-friendly interface and robust features, the system contributes significantly to reducing property damage and saving lives in flood-prone regions.

8. Future Scope and Enhancements:

- Integration of real-time data sources for more accurate predictions.
- Implementation of machine learning algorithms for adaptive forecasting.
- Enhancement of visualization tools for better data interpretation.
- Collaboration with government agencies and disaster management organizations for broader implementation and impact.

SAMPLE FRONTEND:

Flood Risk Prediction and Early Warning System

User and Location Details

Name:

Address:

Contact Number:

Region:

Elevation:

Submit

Flood Risk Prediction and Early Warning System

Name:

Address:

Contact Number:

Select Region:

Select Region

Select Region
New York
Los Angeles
London
Paris
Tokyo
San Francisco
Madrid
Athens
Rome
Moscow

Flood Risk Prediction and Early Warning System

Name:

Rangesh Suraj

Address:

Chennai

Contact Number:

8825714465

Select Region:

New York

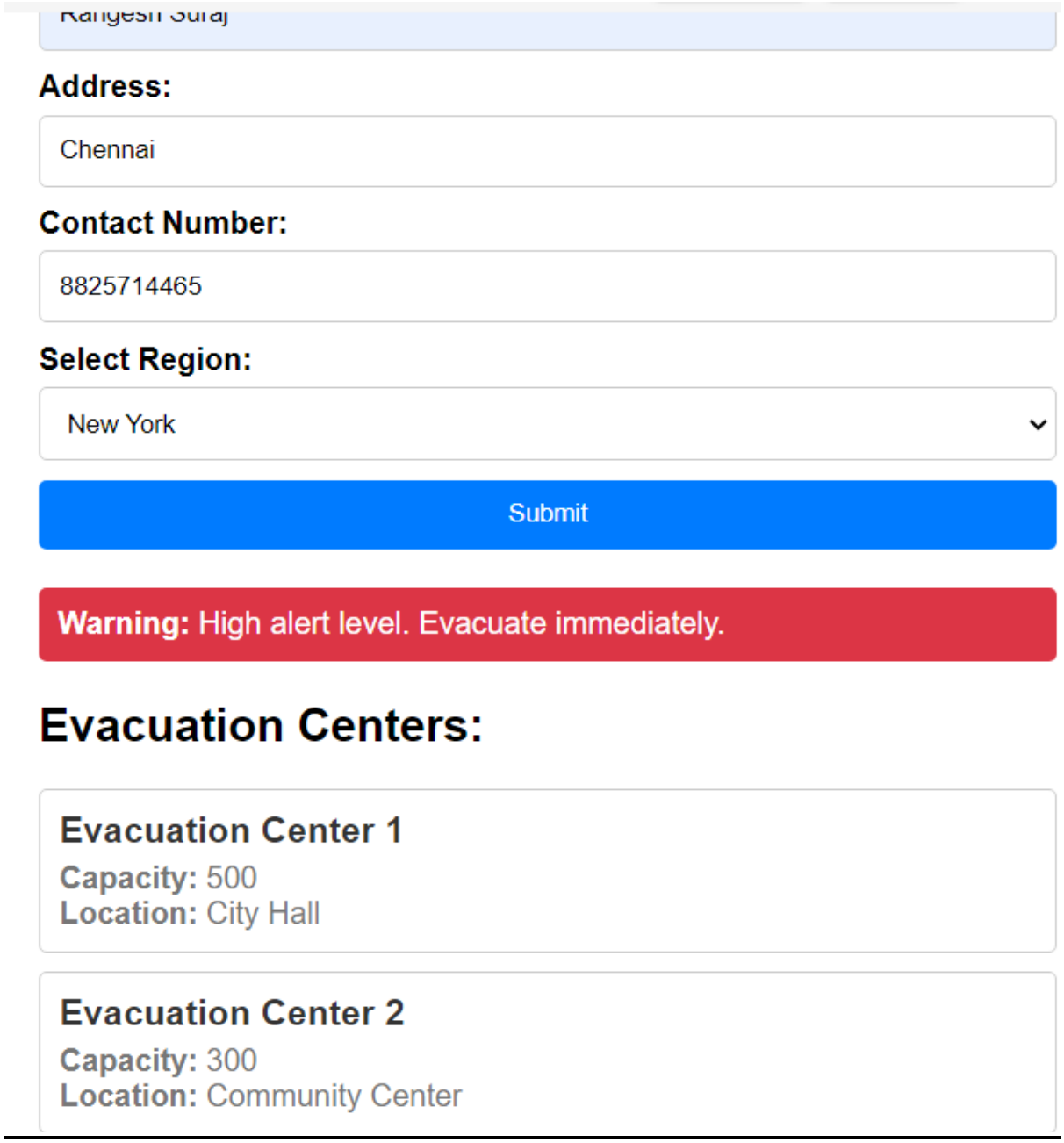
Submit

Warning: High alert level. Evacuate immediately.

Evacuation Centers:

Evacuation Center 1

Capacity: 500



- The user fills out the form with their details and selects the region from the dropdown.
- Upon submission, the system fetches the alert details based on the selected region.
- If the alert level is "High," a warning message is displayed along with the recommended actions.

- Evacuation centers are displayed below the warning message, providing users with information on where to evacuate in case of a high alert level.

- The design is clean and user-friendly, with proper alignment and color scheme to enhance readability and usability.