Table of Contents

[Brief description: 4](#_Toc404555563)

Explicit [Requirements: 4](#_Toc404555564)

[Task 1: 5](#_Toc404555565)

[Entity – Relational Diagram 5](#_Toc404555566)

[Task 2:](#_Toc404555567) 6

[Transforming ERD to Relational schema:](#_Toc404555568) 6

[Step 1: Mapping of regular entities. 6](#_Toc404555569)

[Step 2: Mapping of weak entities. 7](#_Toc404555570)

[Step 3: Mapping of Binary 1:1 Relationship Types. 7](#_Toc404555571)

[Step 4: Mapping of Binary 1:N relationship types. 7](#_Toc404555572)

[Step 5: Mapping of Binary M:N Relationship Types. 8](#_Toc404555573)

[Step 6: Mapping of Multivalued attributes. 8](#_Toc404555574)

[Step 7: Mapping of N-ary Relationship Types. 8](#_Toc404555575)

[Final Schema: 9](#_Toc404555576)

[Task 3: 10](#_Toc404555577)

[Creation of Relations: 10](#_Toc404555578)

[Schema: 10](#_Toc404555579)

[Database: 10](#_Toc404555580)

[Location: 10](#_Toc404555581)

[Neighbourhood: 10](#_Toc404555582)

[Meterological\_Stations: 10](#_Toc404555583)

[Weather\_Observations: 10](#_Toc404555584)

[Events: 11](#_Toc404555585)

[Obs\_Precipitation: 11](#_Toc404555586)

[Reverse Engineered Schema Diagram: 11](#_Toc404555587)

[Task 3.1: 12](#_Toc404555588)

[Insert Statements: 12](#_Toc404555589)

[Inserting data to Location table: 12](#_Toc404555590)

[Inserting data to Meterological\_Stations table: 12](#_Toc404555591)

[Inserting data to Weather\_observations table: 12](#_Toc404555592)

[Inserting data to Events table: 12](#_Toc404555593)

[Inserting data to Obs\_precipitaion table: 12](#_Toc404555594)

[Inserting data to Neighbourhood table: 12](#_Toc404555595)

[Task 4: 13](#_Toc404555597)

[Query 1: 13](#_Toc404555598)

SQL [Query: 13](#_Toc404555599)

[Output: 13](#_Toc404555600)

[Query 2: 14](#_Toc404555601)

SQL [Query: 14](#_Toc404555602)

[Output: 15](#_Toc404555603)

[Query 3: 16](#_Toc404555604)

SQL [Query: 16](#_Toc404555605)

[Output: 16](#_Toc404555608)

[Query 4 17](#_Toc404555609)

SQL [Query: 17](#_Toc404555610)

[Output: 18](#_Toc404555611)

[Query 5: 18](#_Toc404555612)

SQL [Query: 18](#_Toc404555613)

[Output: 19](#_Toc404555614)

[Query 6: 20](#_Toc404555615)

SQL [Query: 20](#_Toc404555616)

[Output: 21](#_Toc404555617)

[Query 7: 22](#_Toc404555618)

SQL [Query: 22](#_Toc404555619)

[Output: 23](#_Toc404555617)

[Query 8: 24](#_Toc404555618)

SQL [Query: 24](#_Toc404555619)

[Output: 24](#_Toc404555617)

[Query 9: 25](#_Toc404555618)

SQL [Query: 25](#_Toc404555619)

[Output: 26](#_Toc404555617)

[Query 10: 27](#_Toc404555618)

SQL [Query: 27](#_Toc404555619)

[Output: 28](#_Toc404555617)

[Task 5: 28](#_Toc404555620)

[Functionality 1 29](#_Toc404555621)

[Web page: 29](#_Toc404555622)

[References: 30](#_Toc404555620)

# Brief description:

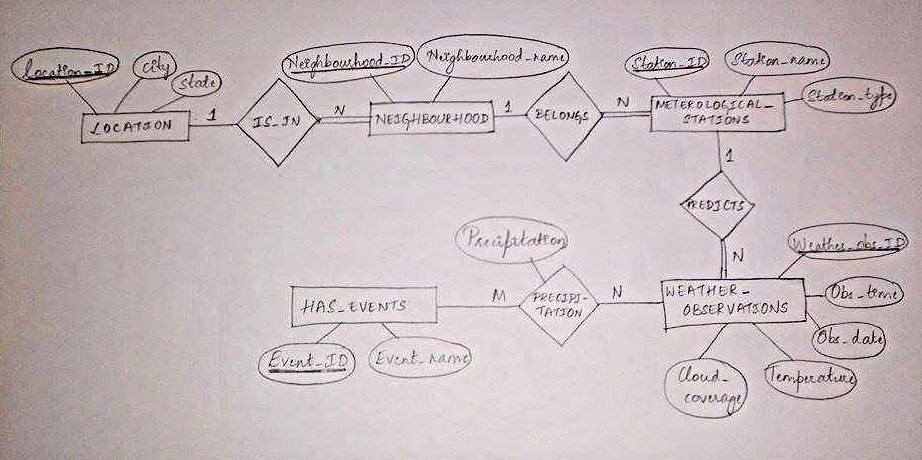
Measuring, capturing and navigating through weather observations from weather stations is very useful and comparing them with climatic values is not only interesting, but also useful for many reasons, like for example to identify if a specific weather observation is unusual for a specific location. We need to create a database based on the following mini world requirements and furthermore integrate this to a simple web application.

# Explicit requirements

* There are many meteorological stations (private or official) in each one of the three cities – Houston, Dallas and Chicago which perform measurements.
* each city can have many neighborhoods. We need to capture the neighborhood where the weather station is located at.
* Each station has specific weather observations captured every hour of a calendar date.
* In each hourly observation there might happen zero or more(min: 0, max: n) of the following events: rain, snow, sleet.
* Also in each 1-hour period, the sky is covered with clouds from a 0/6 (no clouds at all) to 6/6 (full cloud coverage) and we need to capture this information. The cloud coverage and the events are two different, independent things. We need to capture the cloud coverage in every hourly observation.
* In each hourly observation, for each weather station, there may or may not beprecipitation. We need to capture the amount (in cm) for eachprecipitation type separately (rain, snow, sleet).
* Finally, each weather station only captures one temperature (in Celsius) measurement in each of the hourly observations.

# Task 1

Below provided is the appropriate Entity-Relational Diagram (ERD) which conceptualizes the above process.



* We need to create a entity type which records all the hourly observations for all stations .This is depicted in WEATHER\_OBSERVATIONS entity type .
* Each observation may have an event associated with it .This relationship should be captured by creating a new entity type which is done by creating HAS\_EVENTS entity type.
* Each observation is recorded by a specific weather station . The weather station that a observation belongs to is shown in METROLOGICAL\_STATIONS.
* Each Metrological station belongs to a neighborhood which is mapped using NEIGHBOURHOOD entity type .
* Each neighborhood belongs to a specific location which is depicted using LOCATION entity type .
* Notations followed are :
  + 1. Strong entities are represented using rectangles .
    2. Attributes of an entity are represented using ovals
    3. Relationships are represented using lines
    4. Partial participation is represented using single line
    5. Total participation is represented using double line

# Task 2

Applying the ERD to relational algorithm in order to transform the ERD to a relational schema and present the relational schema .

## Transforming ERD to Relational schema:

### Step 1: Mapping of regular entities.

For each strong entity type in the ER diagram we create a new relation that includes all the simple attributes of that entity. We choose one of the key attributes and a primary key for the new relation. In case of a composite key the attributes which form the key together would become the primary key. In our case the mapping of strong entities to relations is as follows.

* Entity LOCATION is a strong entity and is mapped to Location relation with Location\_ID as primary key. This is shown below.

|  |  |  |
| --- | --- | --- |
| LOCATION | | |
| LOCATION\_ID | CITY | STATE |

* Entity NEIGHBOURHOOD reveals which city each neighbourhood belongs to , hence it is a strong entity and is mapped to NEIGHBOURHOOD relation with NEIGHBOURHOOD\_ID as its primary key. This is as shown below.

|  |  |
| --- | --- |
| NEIGHBOURHOOD | |
| NEIGHBOURHOOD\_ID | NEIGHBOURHOOD\_NAME |

* Entity METEROLOGICAL\_STATIONS shows the different stations that belong to a neighbourhood and is a strong entity, it is mapped to METEROLOGICAL\_STATION relation with STATION\_ID as a primary key. This is as shown below.

|  |  |  |
| --- | --- | --- |
| METEROLOGICAL\_STATIONS | | |
| STATION\_ID | STATION\_NAME | STATION\_TYPE |

* Entity WEATHER\_OBSERVATIONS shows all the different hourly weather observations for each station and is a strong entity this is mapped to WEATHER\_OBSERVATIONS relation with WEATHER\_OBS\_ID as its primary key. This is as shown below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| WEATHER\_OBSERVATIONS | | | | |
| WEATHER\_OBS\_ID | OBS\_DATE | OBS\_TIME | TEMPERATURE | CLOUD\_COVERAGE |

* Entity HAS\_EVENTS shows all the different events that can occur and is a strong entity this is mapped to HAS\_EVENTS relation with Event\_ID as its primary key. This is as shown below.

|  |  |
| --- | --- |
| HAS\_EVENTS | |
| EVENT\_ID | EVENT\_NAME |

### Step 2: Mapping of weak entities.

In case of each weak entity type W with owner entity type E, create a relation R that includes all simple attributes (or simple components of composite attributes) of was attributes of R. Include as foreign key attribute(s) in R the primary key attribute(s) of the relation(s) that corresponds to the own entity type(s). The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any.

In Our schema we do not have any weak entities. As all the entities have their independent existence and are not dependent on any other entities.

### Step 3: Mapping of Binary 1:1 Relationship Types.

For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. There are three possible approaches to this Foreign key approach, Merged relation approach and Cross – Reference or Relationship relation approach. In our schema we do not have any relations which have cardinality 1:1 hence moving on.

### Step 4: Mapping of Binary 1:N relationship types.

For each regular binary 1:N relationship type R, identify the relation S that represent the participating entity type at the N-side of the relationship type. Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R. Include any simple attribute of the 1:N relation type as attributes of S.

* In our schema we include Location\_ID attribute of Location relation in the neighbourhood relation as a foreign key reference as there is a 1:N cardinality between these two relations. This is as shown below.

|  |  |  |
| --- | --- | --- |
| NEIGHBOURHOOD | | |
| NEIGHBOURHOOD\_ID | NEIGHBOURHOOD\_NAME | LOCATION\_ID |

* In our schema we include NEIGHBOURHOOD\_ID attribute of NEIGHBOURHOOD relation in the METROLOGICAL\_STATIONS relation as a foreign key reference. This is done because there is a 1:N cardinality between NEIGHBOURHOOD and METEROLOGICAL STATIONS. This is as shown below.

|  |  |  |  |
| --- | --- | --- | --- |
| METEROLOGICAL\_STATIONS | | | |
| STATION\_ID | STATION\_NAME | STATION\_TYPE | NEIGHBOURHOOD\_ID |

* In our schema we include STATION\_ID attribute of METROLOGICAL\_STATIONS relation in the WEATHER\_OBSERVATIONS relation as a foreign key reference as there is a 1:N cardinality between these two relations. This is as shown below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| WEATHER\_OBSERVATIONS | | | | | |
| WEATHER\_OBS\_ID | OBS\_DATE | OBS\_TIME | TEMPERATURE | CLOUD\_COVERAGE | STATION\_ID |

### Step 5: Mapping of Binary M:N Relationship Types.

For each regular binary M:N relationship type R, create a new relations to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S. Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of S.

In our schema there is a many-to-many relationship between weather observations and events . Hence a new mapping table OBS\_PRECIPITATION is created with event\_id of HAS\_EVENTS relation and weather\_obs\_id of WEATHER\_OBSERVATIONS relation as a composite key .This is shown below. As precipitation belongs to this relationship it is included as a attribute in the lookup table.

|  |  |  |
| --- | --- | --- |
| OBS\_PRECIPITATION | | |
| WEATHER\_OBS\_ID | EVENT\_ID | PRECIPITATION |

### Step 6: Mapping of Multivalued attributes.

For each multivalued attribute A, create a new relation R. This relation R will include an attribute corresponding to A, plus the primary key attribute K (as a foreign key in R) of the relation that represents the entity type that has A as an attribute. The primary key of R is the combination of A and K. If the multivalued attribute is composite, we include its simple components. In our schema we do not have any relations which have multivalued attributes hence moving on.

### Step 7: Mapping of N-ary Relationship Types.

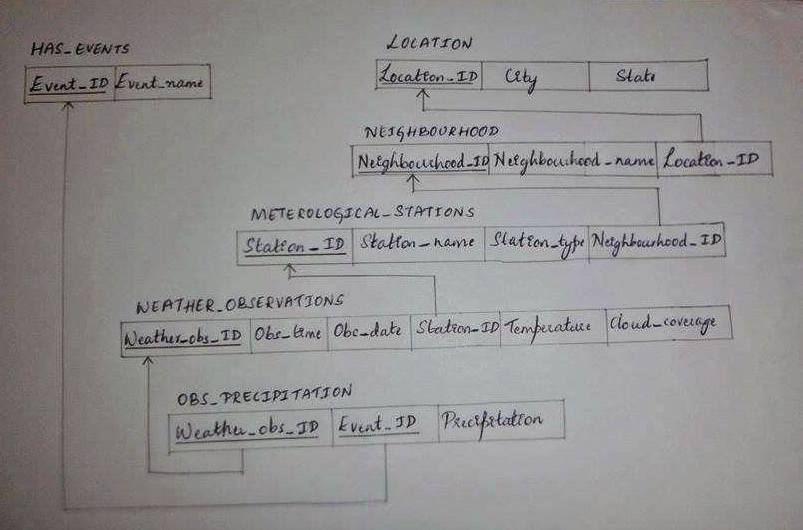
For each n-ary relationship type R, where n>2, create a new relationship relations to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types. Also include any simple attributes of the n-ary relationship (or simple components of composite attributes) as attributes of S. In our schema we do not have any n-ary relationship .Hence moving on.

### Final Schema:

The final schema obtained after applying 7 steps of ER to relation translation is as shown below. It has the following relations.

* Location
* Neighbourhood
* Meterological\_stations
* Weather\_observations
* Has\_events
* Obs\_precipitation

1. The final schema after the completion of the algorithm



# Task 3

Writing appropriate CREATE SQL code to design our database schema. Defining all the appropriate constraints, including ON DELETE and ON UPDATE constraints.

## Creation of Relations:

The CREATE SQL Codes for the aforementioned schema is as follows:

### SCHEMA:

CREATE SCHEMA weather\_project;

USE weather\_project ;

**DATABASE:**

### LOCATION:

CREATE TABLE LOCATION(LOCATION\_ID INT ,CITY VARCHAR(40) NOT NULL , STATE VARCHAR(40) ,PRIMARY KEY(LOCATION\_ID));

**NEIGHBOURHOOD:**

CREATE TABLE NEIGHBOURHOOD(NEIGHBOURHOOD\_ID INT,NEIGHBOURHOOD\_NAME VARCHAR(40) NOT NULL,LOCATION\_ID INT,PRIMARY KEY(NEIGHBOURHOOD\_ID),FOREIGN KEY(LOCATION\_ID) REFERENCES LOCATION(LOCATION\_ID) ON DELETE CASCADE ON UPDATE CASCADE);

**METEROLOGICAL\_STATIONS:**

CREATE TABLE METEROLOGICAL\_STATIONS(STATION\_ID INT PRIMARY KEY ,STATION\_NAME VARCHAR(40) NOT NULL,STATION\_TYPE VARCHAR(40) NOT NULL,NEIGHBOURHOOD\_ID INT ,FOREIGN KEY(NEIGHBOURHOOD\_ID) REFERENCES NEIGHBOURHOOD(NEIGHBOURHOOD\_ID) ON DELETE CASCADE ON UPDATE CASCADE );

**WEATHER\_OBSERVATIONS:**

CREATE TABLE WEATHER\_OBSERVATIONS(WEATHER\_OBS\_ID INT PRIMARY KEY ,OBS\_TIME time NOT NULL,OBS\_DATE date NOT NULL,STATION\_ID INT ,TEMPERATURE DOUBLE NOT NULL ,CLOUD\_COVERAGE FLOAT ,FOREIGN KEY (STATION\_ID) REFERENCES METEROLOGICAL\_STATIONS(STATION\_ID) ON DELETE CASCADE ON UPDATE CASCADE);

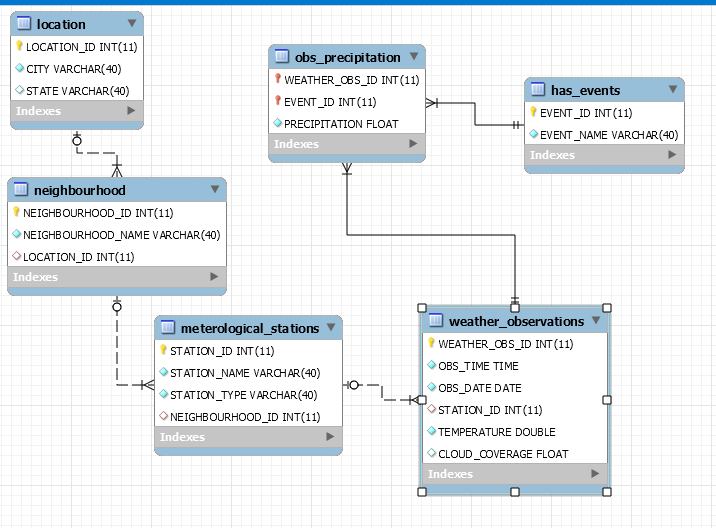
**EVENTS:**

CREATE TABLE HAS\_EVENTS(EVENT\_ID INT PRIMARY KEY ,EVENT\_NAME VARCHAR(40) NOT NULL);

**OBS\_PRECIPITATION**

CREATE TABLE OBS\_PRECIPITATION(WEATHER\_OBS\_ID INT ,EVENT\_ID INT,PRECIPITATION FLOAT NOT NULL,PRIMARY KEY(WEATHER\_OBS\_ID,EVENT\_ID),FOREIGN KEY(WEATHER\_OBS\_ID) REFERENCES WEATHER\_OBSERVATIONS(WEATHER\_OBS\_ID)ON DELETE CASCADE ON UPDATE CASCADE,FOREIGN KEY(EVENT\_ID) REFERENCES HAS\_EVENTS(EVENT\_ID) ON DELETE CASCADE ON UPDATE CASCADE);

**The reverse engineered schema** :

****

# Task 3.1

Inserting real data from various stations for the three cities.

\*\* On local system insert command was used for inserting data into tables .Later ,the data was converted to csv file for inserting data into tables on the server .The following command's are the command used for uploading data to server .

\* The insert commands are delivered as a sql script

**Inserting data to Location table:**

LOAD DATA INFILE '/home/a/ax/axc0302/public\_html/LOCATION’ INTO TABLE LOCATION FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' IGNORE 1 LINES;

**Inserting data to Meterological\_Stations table:**

LOAD DATA INFILE '/home/a/ax/axc0302/public\_html/ METEROLOGICAL\_STATIONS’ INTO TABLE METEROLOGICAL\_STATIONS FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' IGNORE 1 LINES;

**Inserting data to Weather\_observations table:**

LOAD DATA INFILE '/home/a/ax/axc0302/public\_html/ WEATHER\_OBSERVATIONS’ INTO TABLE WEATHER\_OBSERVATIONS FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' IGNORE 1 LINES;

**Inserting data to Events table:**

LOAD DATA INFILE '/home/a/ax/axc0302/public\_html/ HAS\_EVENTS’ INTO TABLE HAS\_EVENTS FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' IGNORE 1 LINES;

**Inserting data to Obs\_precipitaion table:**

LOAD DATA INFILE '/home/a/ax/axc0302/public\_html/ OBS\_PRECIPITATION’ INTO TABLE OBS\_PRECIPITATION FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' IGNORE 1 LINES;

**Inserting data to Neighbourhood table:**

LOAD DATA INFILE '/home/a/ax/axc0302/public\_html/ NEIGHBOURHOOD’ INTO TABLE NEIGHBOURHOOD FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' IGNORE 1 LINES;

# Task 4

Developing appropriate queries for the following:

## Question 1

Weather station reporting the lowest average temperature and the city it is located

### SQL Query:

### SELECT

### D.STATION\_NAME,

### ROUND(AVG(A.TEMPERATURE), 3) AS AVERAGE\_TEMPERATURE,

### C.CITY

### FROM

### weather\_observations AS A,

### neighbourhood AS B,

### LOCATION AS C,

### meterological\_stations AS D

### WHERE

### A.STATION\_ID = D.STATION\_ID

### AND D.NEIGHBOURHOOD\_ID = B.NEIGHBOURHOOD\_ID

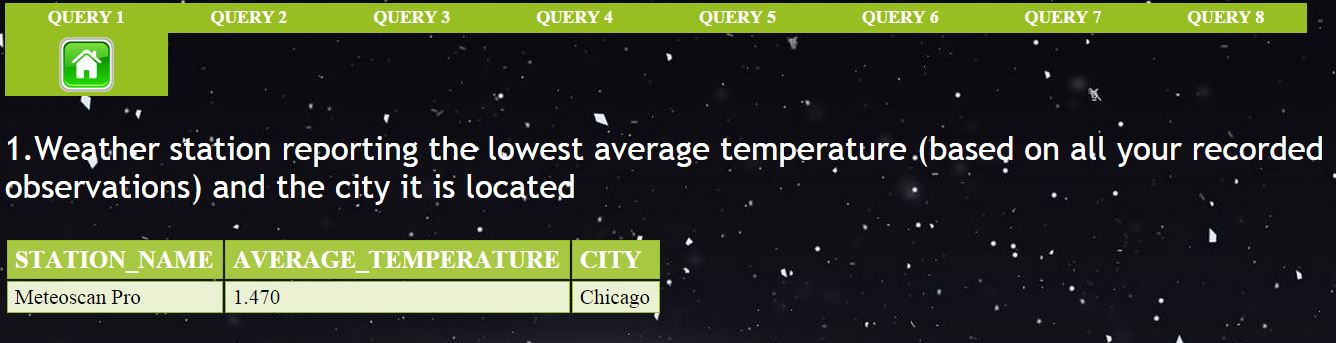
### AND B.LOCATION\_ID = C.LOCATION\_ID

### GROUP BY D.STATION\_NAME

### ORDER BY A.TEMPERATURE

### LIMIT 1;

### Result:



## Question 2

Weather station the highest frequency of sleet events and the city it is located

### SQL Query:

SELECT

B.STATION\_NAME,

D.CITY,

COUNT(F.EVENT\_NAME) AS count\_of\_sleet\_events,

F.EVENT\_NAME

FROM

weather\_observations AS A,

meterological\_stations AS B,

neighbourhood AS C,

location AS D,

obs\_precipitation AS E,

has\_events AS F

WHERE

A.STATION\_ID = B.STATION\_ID

AND B.NEIGHBOURHOOD\_ID = C.NEIGHBOURHOOD\_ID

AND C.LOCATION\_ID = D.LOCATION\_ID

AND A.WEATHER\_OBS\_ID = E.WEATHER\_OBS\_ID

AND E.EVENT\_ID = F.EVENT\_ID

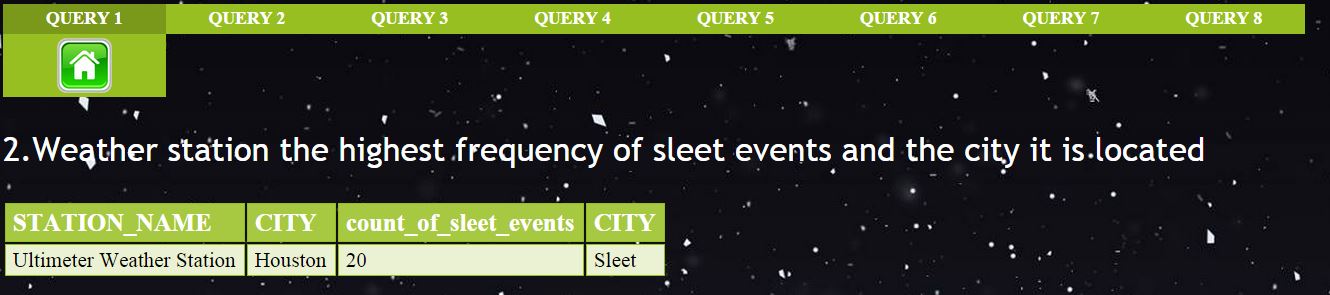
AND F.EVENT\_ID = 300

GROUP BY B.STATION\_NAME

ORDER BY COUNT(F.EVENT\_NAME) DESC

LIMIT 1;

### Result:



## Question 3

The number of weather stations in each neighborhood.

### SQL Query:

### SELECT

### b.NEIGHBOURHOOD\_NAME,

### COUNT(a.NEIGHBOURHOOD\_ID) AS NUMBER\_OF\_WEATHER\_STATIONS,

### c.CITY

### FROM

### meterological\_stations AS a,

### neighbourhood AS b,

### location AS c

### WHERE

### a.NEIGHBOURHOOD\_ID = b.NEIGHBOURHOOD\_ID

### AND b.LOCATION\_ID = c.LOCATION\_ID

### GROUP BY b.NEIGHBOURHOOD\_NAME;

### Result:



## Question 4

Hourly observation periods with no precipitation (no rain, sleet or snow) for Dallas.

### SQL Query:

SELECT

A.WEATHER\_OBS\_ID, A.OBS\_TIME, A.OBS\_DATE, Z.CITY

FROM

weather\_observations AS A,

location AS Z

WHERE

Z.CITY = 'Dallas'

AND NOT EXISTS( SELECT

\*

FROM

obs\_precipitation AS B,

meterological\_stations AS C,

neighbourhood AS D,

location AS E

WHERE

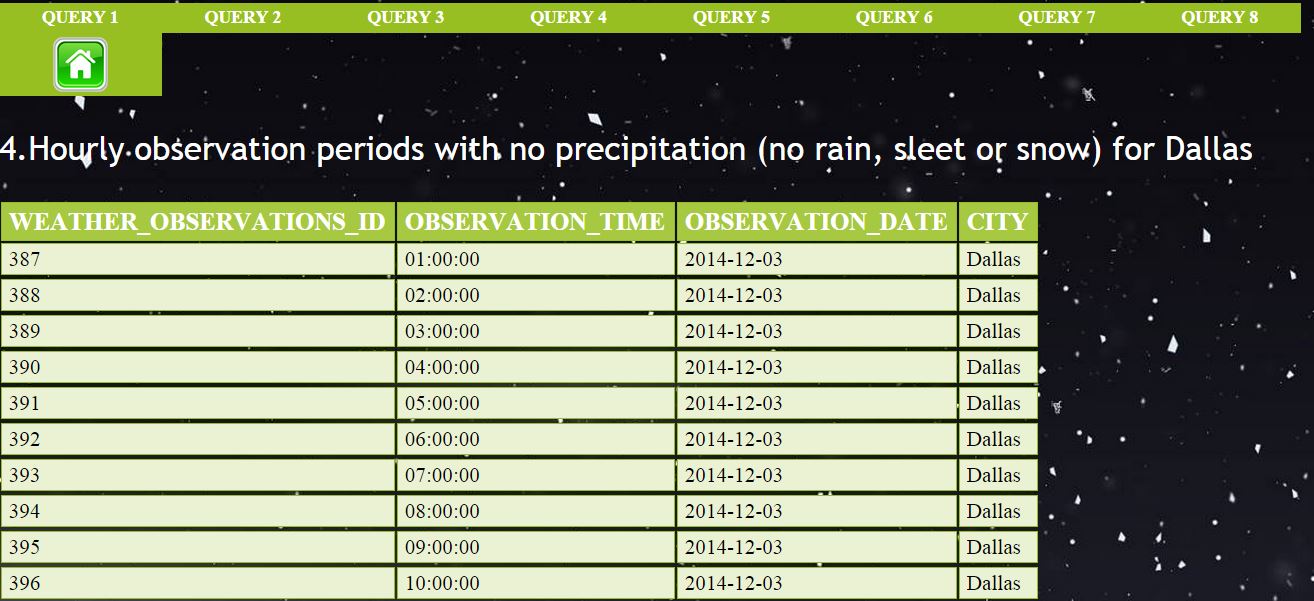
A.WEATHER\_OBS\_ID = B.WEATHER\_OBS\_ID

AND A.STATION\_ID = C.STATION\_ID

AND C.NEIGHBOURHOOD\_ID = D.NEIGHBOURHOOD\_ID

AND Z.LOCATION\_ID = E.LOCATION\_ID);

### Result:



## Question 5

City with most meteorological stations.

### SQL Query:

SELECT

C.CITY, COUNT(A.STATION\_ID) AS NUM\_OF\_METREOLOGICAL\_STATIONS

FROM

meterological\_stations AS a,

neighbourhood AS b,

location AS c

WHERE

a.NEIGHBOURHOOD\_ID = b.NEIGHBOURHOOD\_ID

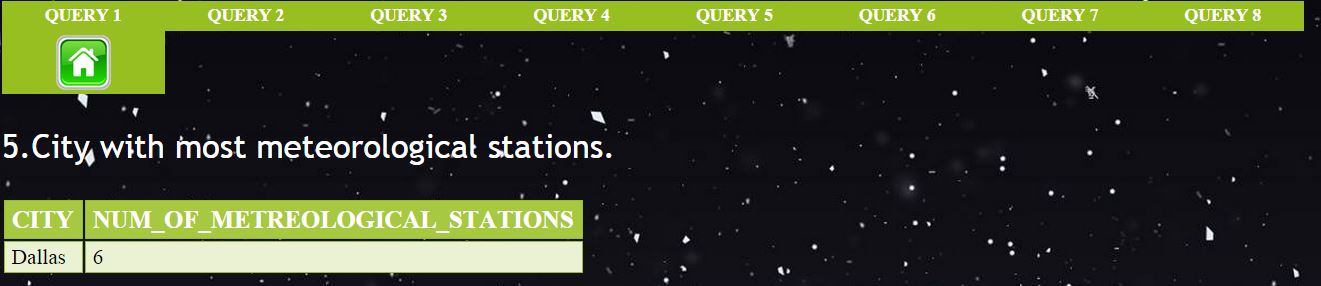
AND b.LOCATION\_ID = c.LOCATION\_ID

GROUP BY C.CITY

ORDER BY COUNT(A.STATION\_ID) DESC

LIMIT 1;

### Result:



## Question 6

Those meteorological stations, which, for Chicago, provide constantly wrong measurements .

### SQL Query:

### SELECT

### ROUND(AVG(TEMPERATURE), 2) avg\_temp, wo.STATION\_ID

### FROM

### location lo,

### neighbourhood ne,

### meterological\_stations ms,

### weather\_observations wo

### WHERE

### wo.STATION\_ID = ms.STATION\_ID

### AND ms.NEIGHBOURHOOD\_ID = ne.NEIGHBOURHOOD\_ID

### AND ne.LOCATION\_ID = lo.LOCATION\_ID

### AND CITY = 'chicago'

### GROUP BY wo.STATION\_ID

### HAVING (avg\_temp - (SELECT

### ROUND(AVG(temp), 2)

### FROM

### (SELECT

### ROUND(AVG(temperature), 2) temp

### FROM

### location l, neighbourhood n, meterological\_stations m, weather\_observations w

### WHERE

### w.STATION\_ID = m.STATION\_ID

### AND m.NEIGHBOURHOOD\_ID = n.NEIGHBOURHOOD\_ID

### AND n.LOCATION\_ID = l.LOCATION\_ID

### AND CITY = 'chicago'

### GROUP BY w.station\_id) a) > 10)

### OR (avg\_temp - (SELECT

### ROUND(AVG(temp), 2)

### FROM

### (SELECT

### ROUND(AVG(temperature), 2) temp

### FROM

### location l, neighbourhood n, meterological\_stations m, weather\_observations w

### WHERE

### w.STATION\_ID = m.STATION\_ID

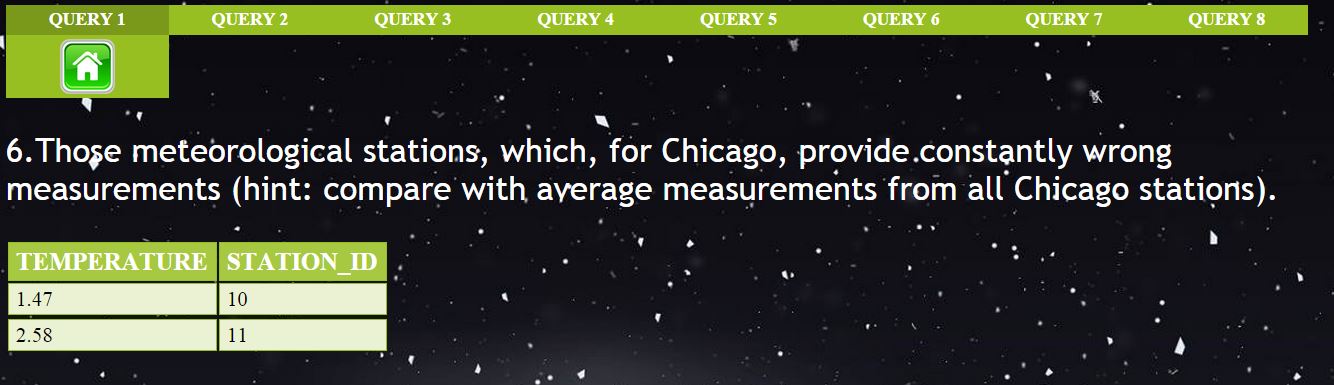
### AND m.NEIGHBOURHOOD\_ID = n.NEIGHBOURHOOD\_ID

### AND n.LOCATION\_ID = l.LOCATION\_ID

### AND CITY = 'chicago'

### GROUP BY w.station\_id) a) < - 10);

### Result:



## Question 7

The total amount of precipitation per month of each city, based on the average measurements of the weather stations located in that city.

### SQL Query:

SELECT

MONTHNAME(A.OBS\_DATE) AS MONTH,

YEAR(A.OBS\_DATE) AS YEAR,

ROUND(AVG(PRECIPITATION), 3) AS AVERAGE\_PRECIPITATION,

CITY

FROM

weather\_observations AS A,

obs\_precipitation AS B,

meterological\_stations AS C,

neighbourhood AS D,

location AS E

WHERE

A.STATION\_ID = C.STATION\_ID

AND C.NEIGHBOURHOOD\_ID = D.NEIGHBOURHOOD\_ID

AND D.LOCATION\_ID = E.LOCATION\_ID

AND A.WEATHER\_OBS\_ID = B.WEATHER\_OBS\_ID

GROUP BY MONTH(A.OBS\_DATE) , YEAR(A.OBS\_DATE) , E.CITY;

### Result:



## Question 8

The average daily temperature range in each station (max-min temperature per day)

### SQL Query:

### SELECT

### B.STATION\_NAME,

### MAX(A.TEMPERATURE) AS MAX\_TEMPERATURE,

### MIN(A.TEMPERATURE) AS MIN\_TEMPERATURE,

### A.OBS\_DATE

### FROM

### weather\_observations AS A,

### meterological\_stations AS B

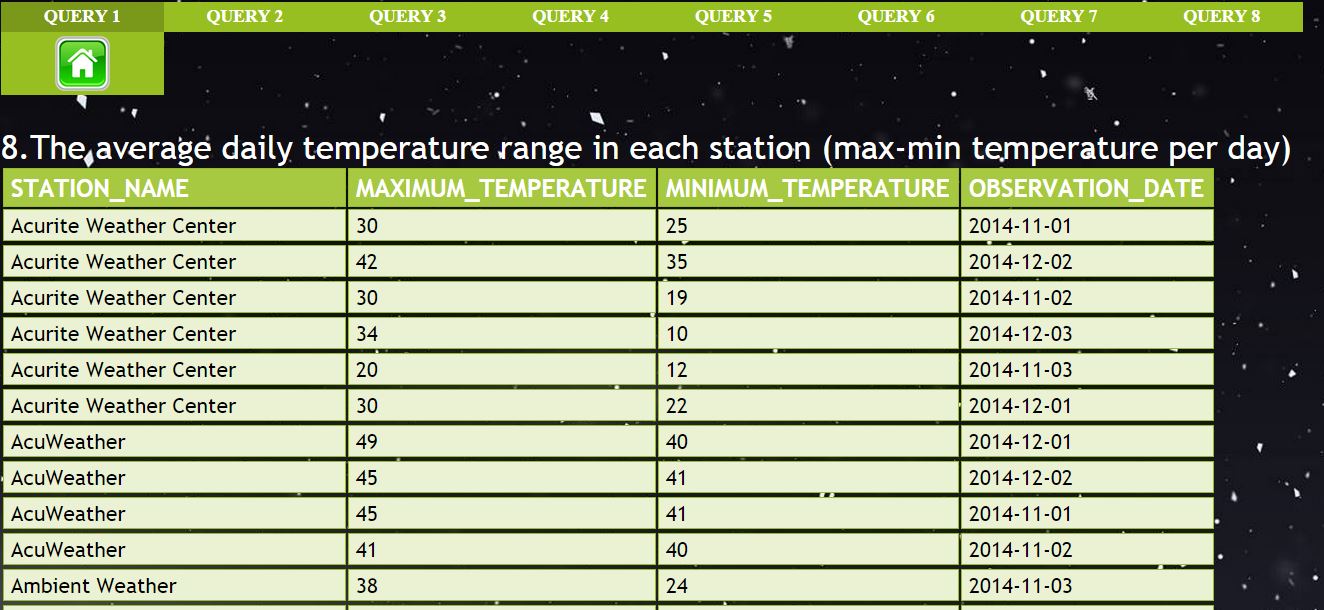
### WHERE

### A.STATION\_ID = B.STATION\_ID

### GROUP BY A.OBS\_DATE , B.STATION\_NAME

### ORDER BY B.STATION\_NAME;

### Result:



## Question 9(Extreme event 1)

Hourly observation periods with very large (>10 Celsius) of temperature from the average measurements for all cities

### SQL Query:

### SELECT

### city,

### TEMPERATURE,

### wo.OBS\_DATE,

### wo.OBS\_TIME,

### wo.WEATHER\_OBS\_ID AS observation\_id

### FROM

### location lo,

### neighbourhood ne,

### meterological\_stations ms,

### weather\_observations wo

### WHERE

### wo.STATION\_ID = ms.STATION\_ID

### AND ms.NEIGHBOURHOOD\_ID = ne.NEIGHBOURHOOD\_ID

### AND ne.LOCATION\_ID = lo.LOCATION\_ID

### AND TEMPERATURE - (SELECT

### ROUND(AVG(TEMPERATURE), 2)

### FROM

### location l,

### neighbourhood n,

### meterological\_stations m,

### weather\_observations w

### WHERE

### w.STATION\_ID = m.STATION\_ID

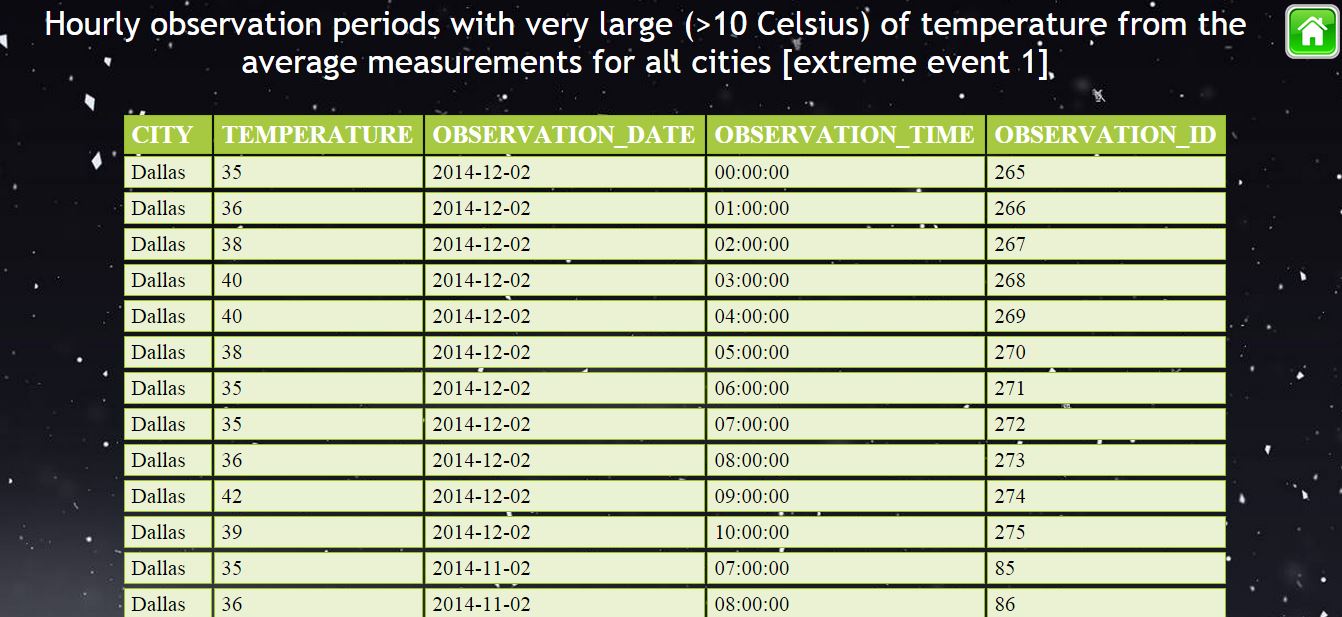
### AND m.NEIGHBOURHOOD\_ID = n.NEIGHBOURHOOD\_ID

### AND n.LOCATION\_ID = l.LOCATION\_ID

### AND l.city = lo.city

### GROUP BY CITY) > 10;

### Result:



## Question 10(Extreme event 2)

6-hour periods with precipitation >3 inches for all cities

### SQL Query:

### SELECT

### B.WEATHER\_OBS\_ID,

### OBS\_DATE,

### OBS\_TIME,

### PRECIPITATION,

### CITY,

### STATE

### FROM

### obs\_precipitation tt,

### WEATHER\_OBSERVATIONs AS B,

### METEROLOGICAL\_STATIONS AS C,

### NEIGHBOURHOOD AS D,

### LOCATION AS E

### WHERE

### precipitation > 9

### AND tt.weather\_obs\_id = B.weather\_obs\_id

### AND B.station\_id = C.station\_id

### AND C.neighbourhood\_id = D.neighbourhood\_id

### AND D.location\_id = e.location\_id

### AND tt.weather\_obs\_id IN (SELECT

### weather\_obs\_id

### FROM

### (SELECT

### tt.\*,

### (SELECT

### COUNT(weather\_obs\_id)

### FROM

### obs\_precipitation

### WHERE

### precipitation <= 9

### AND weather\_obs\_id < tt.weather\_obs\_id) AS cnt

### FROM

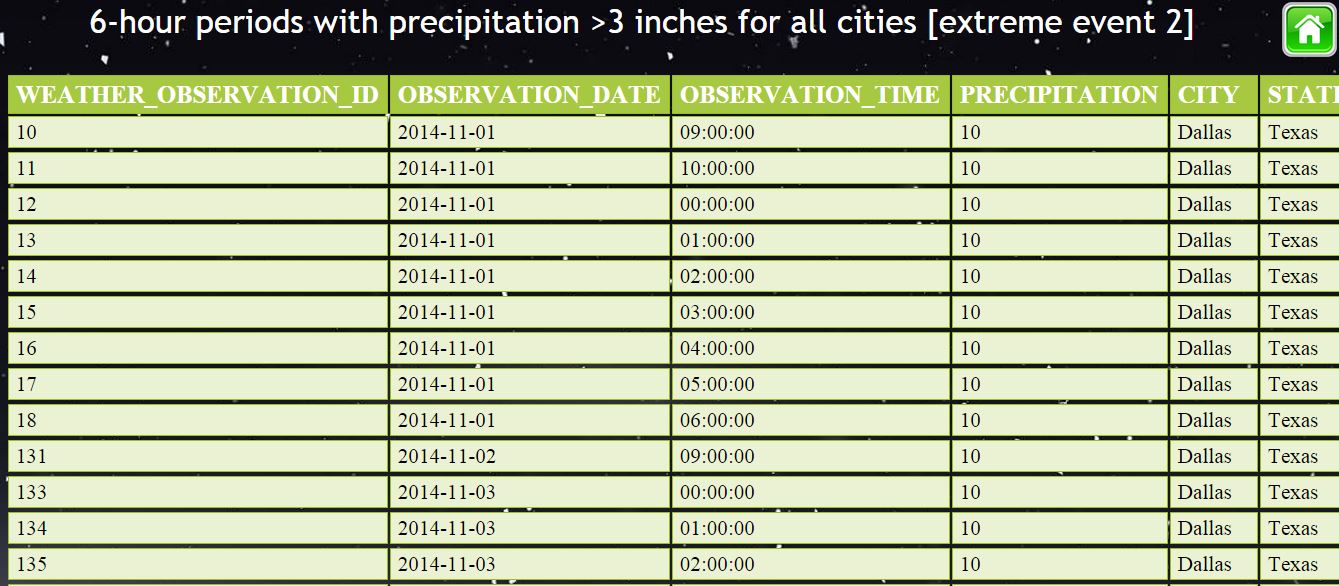
### obs\_precipitation tt

### WHERE

### precipitation > 9) t1)

### ORDER BY B.WEATHER\_OBS\_ID;

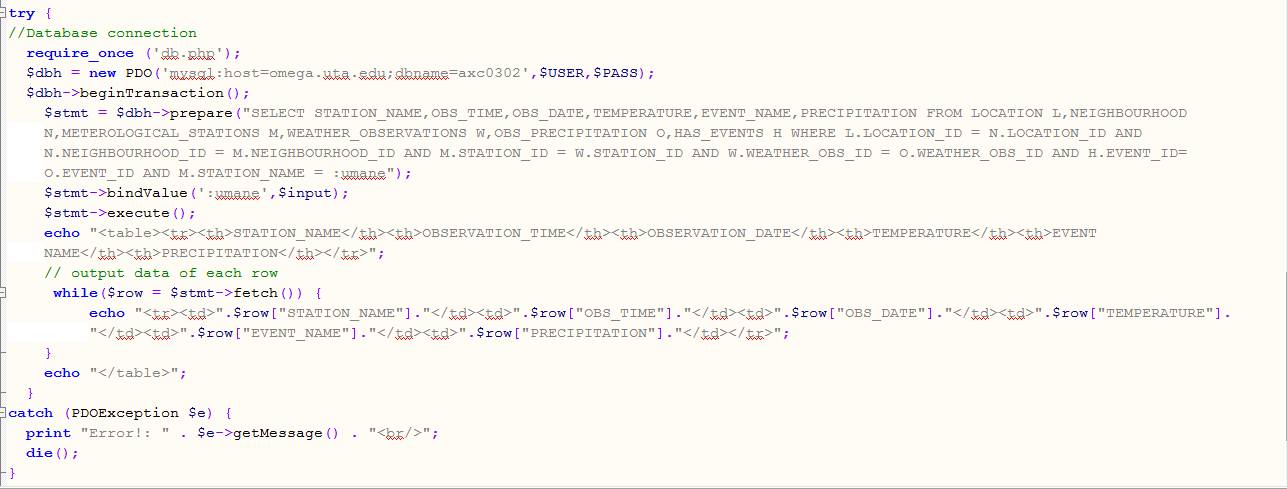
### Result:

****

# Task 5

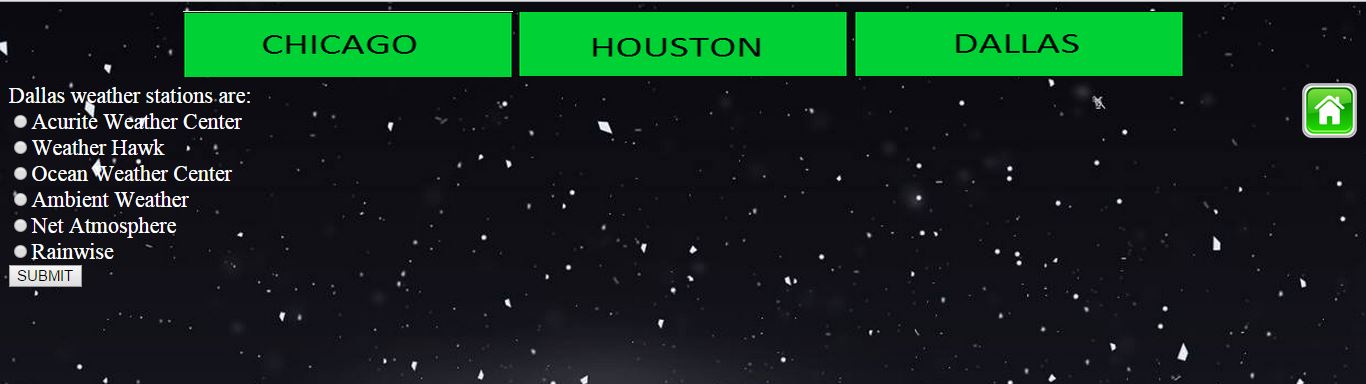
Developing a web application based on our database, and queries with the following functionalities.

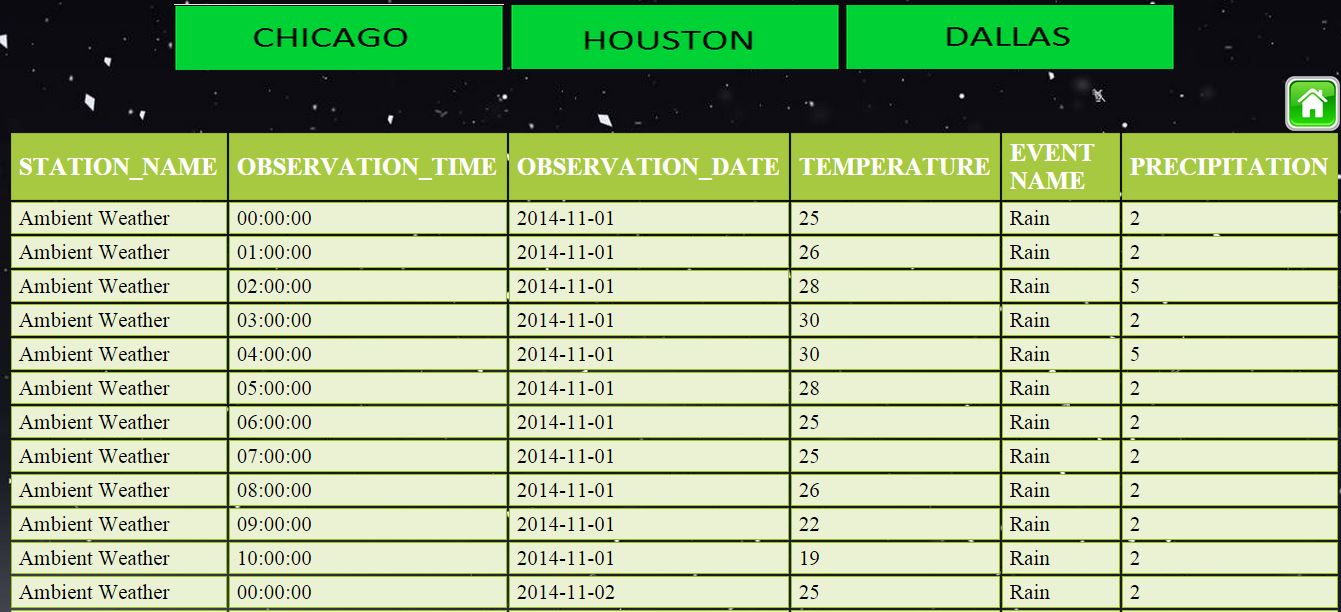
The PHP Data Objects (PDO) extension defines a lightweight, consistent interface for accessing databases in PHP. Each database driver that implements the PDO interface can expose database-specific features as regular extension functions. Note that you cannot perform any database functions using the PDO extension by itself; you must use a database-specific PDO driver to access a database server.



(i) Navigation to the various stations’ measurements for our cities: the user choses a city, then a weather station, and then he/she can navigate through the observations

Screenshots of the functionality implemented .





# 

# References:

Links and books used for reference are as below:

1. Requirement specification document of the project.
2. <http://php.net/manual/en/intro.pdo.php>
3. Fundamentals of Database Systems by Ramez Elmasri and Shamkant B. Navathe.