# Creating an Effective DBMS for Luxury-Oriented Scenic Tours

**CSC 411 Group Project**

Daniel Easley

Corbin Cody

Eduardo Escobar

Heather Broome

John Doligale 5th

## Abstract:

Luxury-Oriented Scenic Tours, or LOST, is a system that provides guided tours to visitors of the Washington D.C. area. While researching the workings of LOST, we discovered that the problem with this system is that it has grown very quickly and cannot keep up with the various information needs of the company. This led us to ask how an effective database management system can be created to address and resolve this issue. The present report shows the process of providing a solution to LOST. It has an Entity-Relationship diagram, Schema diagram, and DBMS implementation using MySQL.

## Entity Relationship Diagram:

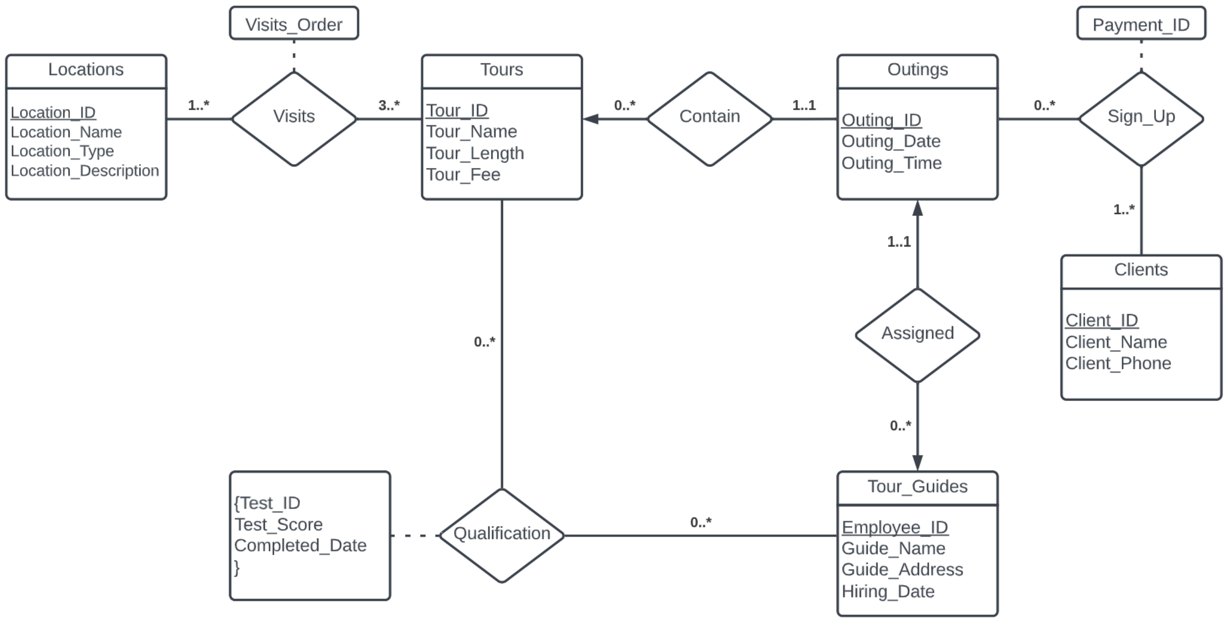


**Figure 1. E-R Diagram.**

The first step in this process was to create an entity relationship diagram. In the solution for the project, there is an entity called “Locations” that has a relationship “Visits” with the entity “Tours.” The attribute for that relationship is “Order.”

The “Clients” entity sends one to many clients to use the “Sign_Up” relation to sign up for tours. The clients then pay if they choose to go on an outing.

In order to go on a tour, there must be a tour guide assigned. Using the “Assigned” relation, zero to many tour guides can be sent for qualification. If the tour guide passes the qualification test, they are then given a tour with the “Tours” entity.

Clients who decided to sign up for an outing can be sent on one outing, which will contain either no tours or many. The “Tours” entity then visits using 3 or more locations the “Visits” relation. The visit order is then determined using “Visits_Order”, and the tour visits one or more locations using the “Locations” relation.

**Schema Diagram:**

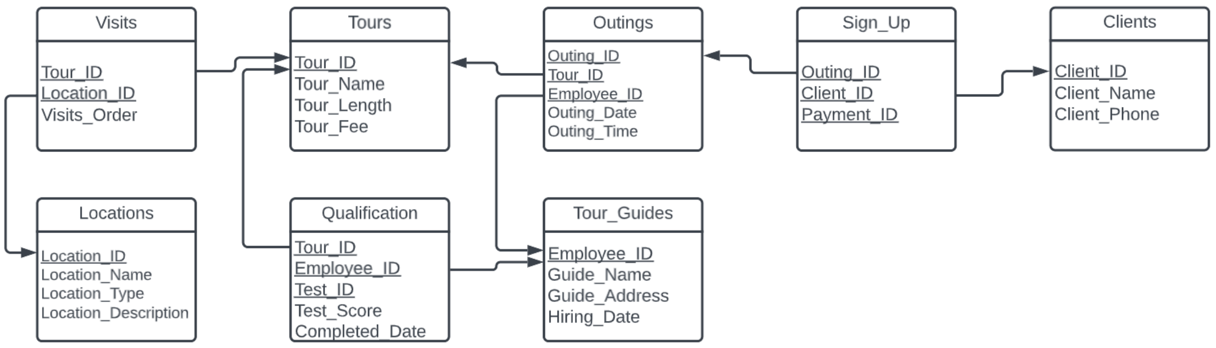


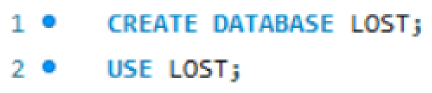
**Figure 2. Detailed Schema Diagram.**

The second step of the process was to create a detailed schema diagram using our entities and relationships. Each have their designated primary key within them defined.

The “Sign_Up” relationship sends the sign up information to clients within the “Clients” entity and within the “Outings” entity. Outings are then sent to possible tour guides in “Tour_Guides”, but they must first pass the “Qualification” relationship in order to be assigned.

If a tour guide has passed the Qualification relationship, they are then given a tour within “Tours” that they will guide on. The “Visits” relation tells how many tours will be taken and how many locations that will be visited. The IDs of each entity are stored within the database for their possibilities.

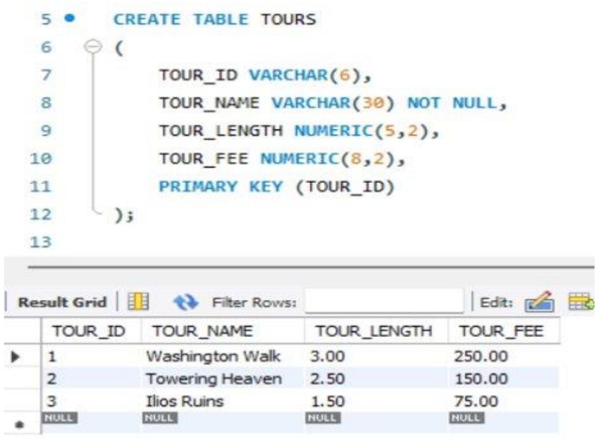
## Implementation Procedure:



```sql
CREATE DATABASE LOST;
USE LOST;
```

**Figure 3. Generating database.**

Using "CREATE DATABASE", we are able to generate the database named "LOST", and "USE" the database to generate our tables and input our information.

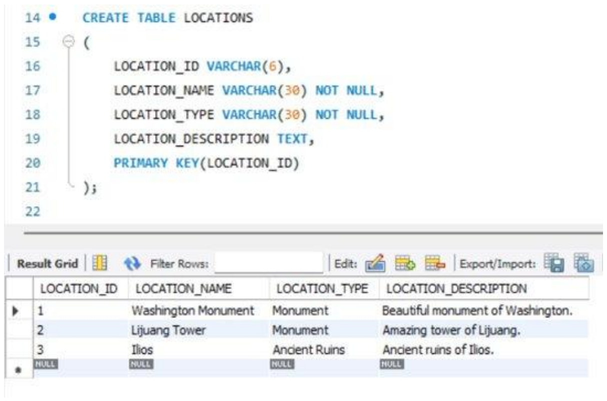


```sql
CREATE TABLE TOURS
(
    TOUR_ID VARCHAR(6),
    TOUR_NAME VARCHAR(30) NOT NULL,
    TOUR_LENGTH NUMERIC(5,2),
    TOUR_FEE NUMERIC(8,2),
    PRIMARY KEY (TOUR_ID)
);
```

| TOUR_ID | TOUR_NAME | TOUR_LENGTH | TOUR_FEE |
|---|---|---|---|
| 1 | Washington Walk | 3.00 | 250.00 |
| 2 | Towering Heaven | 2.50 | 150.00 |
| 3 | Ilios Ruins | 1.50 | 75.00 |
| NULL | NULL | NULL | NULL |

In the "TOURS" create table, it is given the values of a "TOUR_ID" of a random value up to 6. The name and the length of the tour are then assigned using "TOUR_NAME" and "TOUR_LENGTH". Finally the fee is decided from "TOUR_FEE". The primary key is then stored and sent into other tables.

**Figure 4. Creating Tours Relation.**



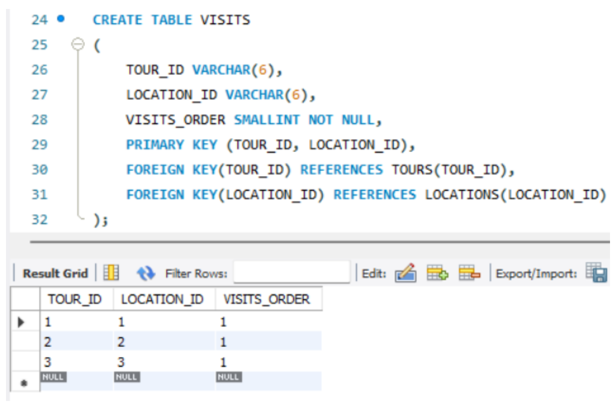
```sql
CREATE TABLE LOCATIONS
(
    LOCATION_ID VARCHAR(6),
    LOCATION_NAME VARCHAR(30) NOT NULL,
    LOCATION_TYPE VARCHAR(30) NOT NULL,
    LOCATION_DESCRIPTION TEXT,
    PRIMARY KEY(LOCATION_ID)
);
```

| LOCATION_ID | LOCATION_NAME | LOCATION_TYPE | LOCATION_DESCRIPTION |
|---|---|---|---|
| 1 | Washington Monument | Monument | Beautiful monument of Washington. |
| 2 | Lijuang Tower | Monument | Amazing tower of Lijuang. |
| 3 | Ilios | Ancient Ruins | Ancient ruins of Ilios. |
| NULL | NULL | NULL | NULL |

The "LOCATIONS" table is created and is assigned their "LOCATION_ID", "LOCATION_NAME", and "LOCATION TYPE". These display the ID, the name of the location, and the type of location that is being visited. There is also a text description of the location that is sent using "LOCATION_DESCRIPTION". The primary key returned is the "LOCATION_ID".

**Figure 5. Creating Locations Relation.**

Next, the “VISITS” table is created, and a tour is given a “TOUR_ID”. It will then give the “LOCATION_ID” of the location, the order in which they will be visited via “VISITS_ORDER”. The primary keys of this table are “TOUR_ID” and “LOCATION_ID”, and the foreign keys of “TOUR_ID” from the “TOURS” table and “LOCATION_ID” from the “LOCATIONS” table.

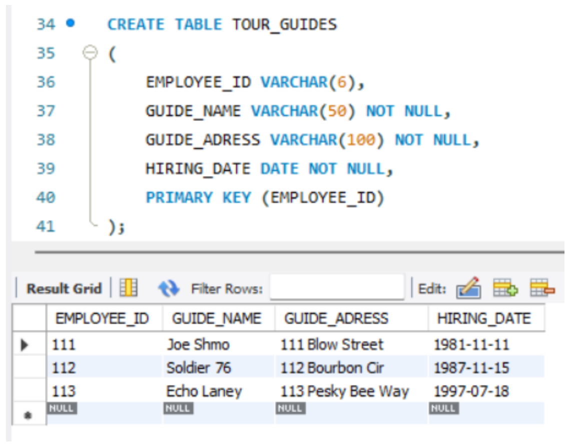


```sql
CREATE TABLE VISITS
(
    TOUR_ID VARCHAR(6),
    LOCATION_ID VARCHAR(6),
    VISITS_ORDER SMALLINT NOT NULL,
    PRIMARY KEY (TOUR_ID, LOCATION_ID),
    FOREIGN KEY(TOUR_ID) REFERENCES TOURS(TOUR_ID),
    FOREIGN KEY(LOCATION_ID) REFERENCES LOCATIONS(LOCATION_ID)
);
```

| TOUR_ID | LOCATION_ID | VISITS_ORDER |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 2 | 1 |
| 3 | 3 | 1 |
| NULL | NULL | NULL |

**Figure 6. Creating Visits Relation.**

The “TOUR_GUIDES” table is created, with a guide given their “EMPLOYEE_ID”, their “GUIDE_NAME”, where they live via “GUIDE_ADDRESS”, and the date they were hired using “HIRING_DATE”. The primary key of this table is “EMPLOYEE_ID”.

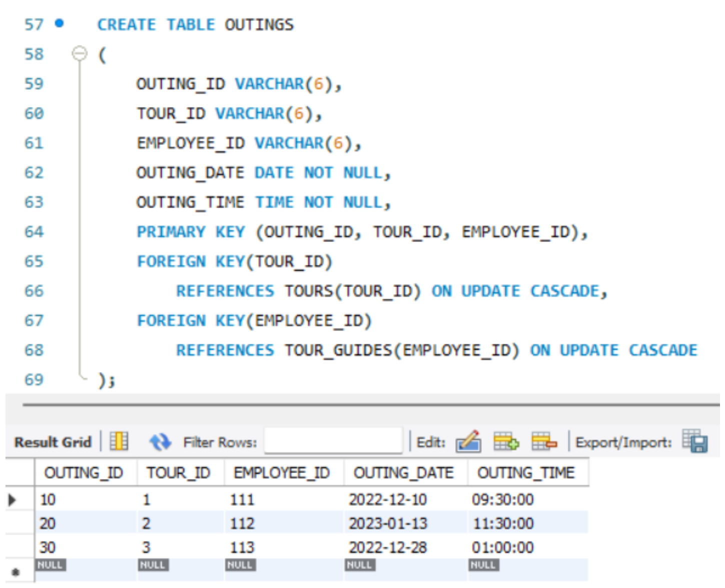


```sql
CREATE TABLE TOUR_GUIDES
(
    EMPLOYEE_ID VARCHAR(6),
    GUIDE_NAME VARCHAR(50) NOT NULL,
    GUIDE_ADRESS VARCHAR(100) NOT NULL,
    HIRING_DATE DATE NOT NULL,
    PRIMARY KEY (EMPLOYEE_ID)
);
```

| EMPLOYEE_ID | GUIDE_NAME | GUIDE_ADRESS | HIRING_DATE |
|---|---|---|---|
| 111 | Joe Shmo | 111 Blow Street | 1981-11-11 |
| 112 | Soldier 76 | 112 Bourbon Cir | 1987-11-15 |
| 113 | Echo Laney | 113 Pesky Bee Way | 1997-07-18 |
| NULL | NULL | NULL | NULL |

**Figure 7. Creating Tour_Guides Relation.**

The “OUTINGS” table describes the outings in which clients will go on. The “OUTING_ID” assigns which ID the outing will be. The “TOUR_ID” assigns which tour will be taken, and it is given an “OUTING_DATE” for which day and “OUTING_TIME” gives the time that the tour will take place. There is also an employee assigned using “EMPLOYEE_ID”.

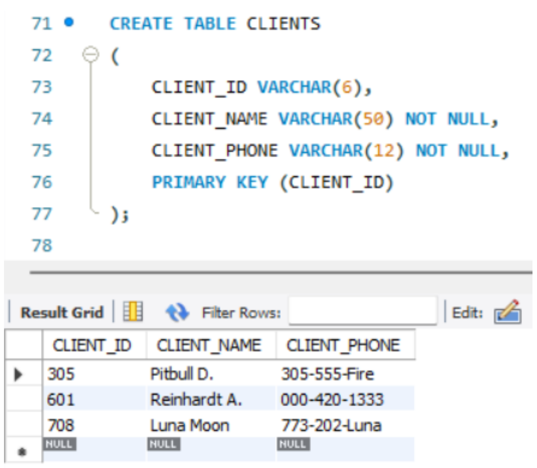


```sql
CREATE TABLE OUTINGS
(
    OUTING_ID VARCHAR(6),
    TOUR_ID VARCHAR(6),
    EMPLOYEE_ID VARCHAR(6),
    OUTING_DATE DATE NOT NULL,
    OUTING_TIME TIME NOT NULL,
    PRIMARY KEY (OUTING_ID, TOUR_ID, EMPLOYEE_ID),
    FOREIGN KEY(TOUR_ID)
        REFERENCES TOURS(TOUR_ID) ON UPDATE CASCADE,
    FOREIGN KEY(EMPLOYEE_ID)
        REFERENCES TOUR_GUIDES(EMPLOYEE_ID) ON UPDATE CASCADE
);
```

| OUTING_ID | TOUR_ID | EMPLOYEE_ID | OUTING_DATE | OUTING_TIME |
|---|---|---|---|---|
| 10 | 1 | 111 | 2022-12-10 | 09:30:00 |
| 20 | 2 | 112 | 2023-01-13 | 11:30:00 |
| 30 | 3 | 113 | 2022-12-28 | 01:00:00 |
| NULL | NULL | NULL | NULL | NULL |

**Figure 8. Creating Outings Relation.**

The primary key would be “OUTING_ID”, “TOUR_ID”, and “EMPLOYEE ID. The foreign keys would be “TOUR_ID”, which references the “TOURS” table and cascades on update, and “EMPLOYEE_ID”, which references “TOUR_GUIDES” and cascades on update as well.

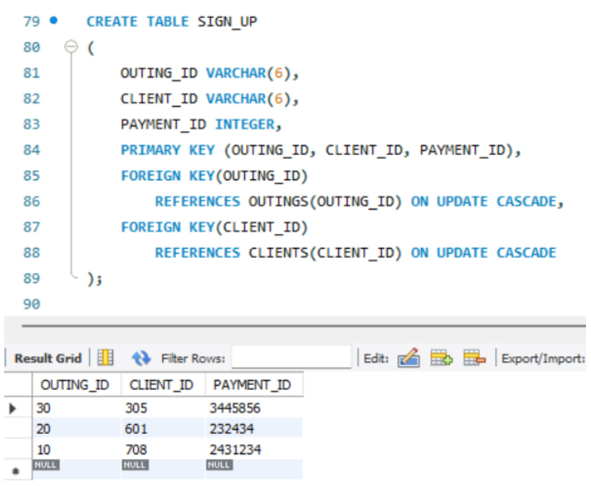


```
CREATE TABLE CLIENTS
(
    CLIENT_ID VARCHAR(6),
    CLIENT_NAME VARCHAR(50) NOT NULL,
    CLIENT_PHONE VARCHAR(12) NOT NULL,
    PRIMARY KEY (CLIENT_ID)
);
```

| CLIENT_ID | CLIENT_NAME | CLIENT_PHONE |
|---|---|---|
| 305 | Pitbull D. | 305-555-Fire |
| 601 | Reinhardt A. | 000-420-1333 |
| 708 | Luna Moon | 773-202-Luna |
| NULL | NULL | NULL |

The “CLIENTS” table is created next, and holds information for the client. It contains “CLIENT_ID” for their ID, their name with “CLIENT_NAME”, and their phone number using “CLIENT_PHONE”. The primary key of this relation would be “CLIENT_ID”.

**Figure 9. Creating Clients Relation.**



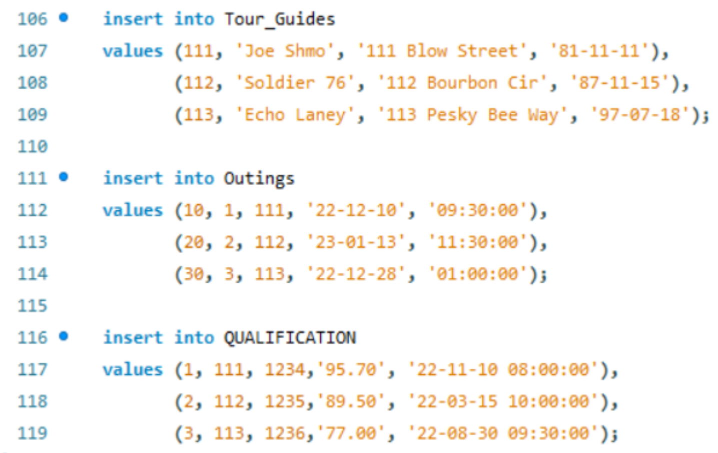
```
CREATE TABLE SIGN_UP
(
    OUTING_ID VARCHAR(6),
    CLIENT_ID VARCHAR(6),
    PAYMENT_ID INTEGER,
    PRIMARY KEY (OUTING_ID, CLIENT_ID, PAYMENT_ID),
    FOREIGN KEY(OUTING_ID)
        REFERENCES OUTINGS(OUTING_ID) ON UPDATE CASCADE,
    FOREIGN KEY(CLIENT_ID)
        REFERENCES CLIENTS(CLIENT_ID) ON UPDATE CASCADE
);
```

| OUTING_ID | CLIENT_ID | PAYMENT_ID |
|---|---|---|
| 30 | 305 | 3445856 |
| 20 | 601 | 232434 |
| 10 | 708 | 2431234 |
| NULL | NULL | NULL |

When a client signs up for an outing, their information is sent into the “SIGN_UP” table. It returns the “OUTING_ID”, which is the outing the client will take. Next, it returns the “CLIENT_ID” of what client is going on the outing, and finally it will take the client’s “PAYMENT_ID”. The primary keys would be “OUTING_ID”, “CLIENT_ID”, and “PAYMENT_ID”.  The foreign keys are the “OUTING_ID”, which references the “OUTINGS” table and cascades on update, and the “CLIENT_ID”, which references the “CLIENTS” table and cascades on update.

**Figure 10. Creating Sign_Up Relation.**

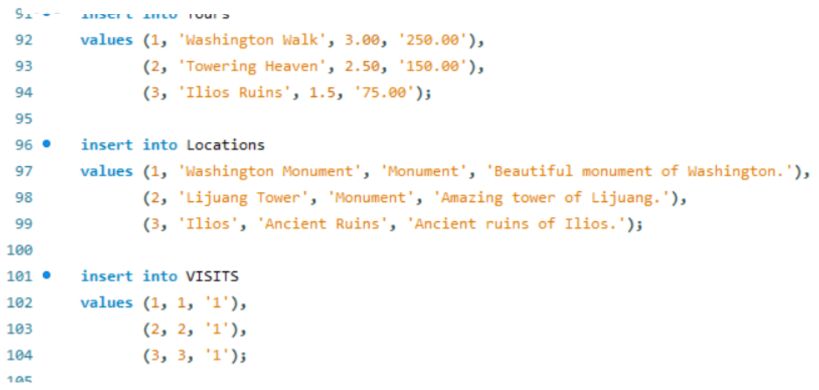
The statements “insert into Tours_Guides values”, “insert into Outings values”, and “insert into QUALIFICATION values” are reserved statements with the purpose of providing the values of



```sql
insert into Tour_Guides
values (111, 'Joe Shmo', '111 Blow Street', '81-11-11'),
       (112, 'Soldier 76', '112 Bourbon Cir', '87-11-15'),
       (113, 'Echo Laney', '113 Pesky Bee Way', '97-07-18');

insert into Outings
values (10, 1, 111, '22-12-10', '09:30:00'),
       (20, 2, 112, '23-01-13', '11:30:00'),
       (30, 3, 113, '22-12-28', '01:00:00');

insert into QUALIFICATION
values (1, 111, 1234,'95.70', '22-11-10 08:00:00'),
       (2, 112, 1235,'89.50', '22-03-15 10:00:00'),
       (3, 113, 1236,'77.00', '22-08-30 09:30:00');
```

the variables of the following table "Tours_Guides", row by row. For which the LOST organization has assigned them.

**Figure 11. Syntax for "insert into Tour_Guides values" Command.**

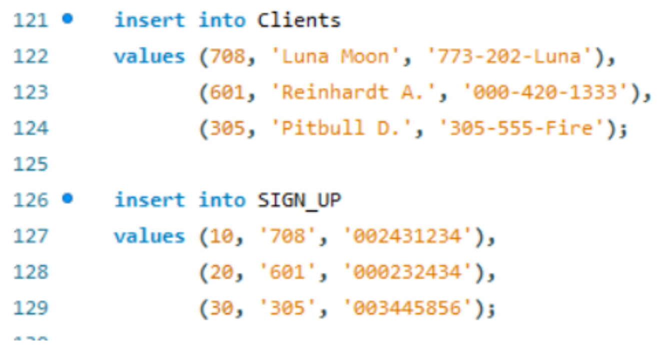


```sql
insert into Tours
values (1, 'Washington Walk', 3.00, '250.00'),
       (2, 'Towering Heaven', 2.50, '150.00'),
       (3, 'Ilios Ruins', 1.5, '75.00');

insert into Locations
values (1, 'Washington Monument', 'Monument', 'Beautiful monument of Washington.'),
       (2, 'Lijuang Tower', 'Monument', 'Amazing tower of Lijuang.'),
       (3, 'Ilios', 'Ancient Ruins', 'Ancient ruins of Ilios.');

insert into VISITS
values (1, 1, '1'),
       (2, 2, '1'),
       (3, 3, '1');
```

Using "insert into Tours values", "insert into Locations values", and "insert into VISITS", the LOST organization is able to input values into the Tours table. All tours, locations, and visits will be recorded in the table for the LOST organization to see.

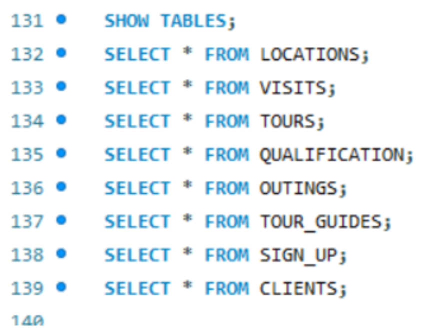
**Figure 12. Syntax for "insert into Tours values" Command.**

Using "inserts into Clients values" and "insert into SIGN_UP values", the LOST organization is able to insert values into the Clients and Sign Up tables. Each of the information is properly stored into the tables.



```sql
insert into Clients
values (708, 'Luna Moon', '773-202-Luna'),
       (601, 'Reinhardt A.', '000-420-1333'),
       (305, 'Pitbull D.', '305-555-Fire');

insert into SIGN_UP
values (10, '708', '002431234'),
       (20, '601', '000232434'),
       (30, '305', '003445856');
```

**Figure 13. Syntax for "insert into Clients values" Command**



```
SHOW TABLES;
SELECT * FROM LOCATIONS;
SELECT * FROM VISITS;
SELECT * FROM TOURS;
SELECT * FROM QUALIFICATION;
SELECT * FROM OUTINGS;
SELECT * FROM TOUR_GUIDES;
SELECT * FROM SIGN_UP;
SELECT * FROM CLIENTS;
```

To view the created tables, apply the "show tables" reserved keyword. Then use select functions to view tables with data from the desired table.

**Figure 14. Selecting Data**

**Conclusion:**

When looking at the LOST scenario, it seemed like a really difficult challenge to overcome. After drawing a schematic diagram, implementing the design of the entity-relationship diagram, and discussing the structure of the database, we discovered that this design is the best approach to handling this situation. Hence we followed through with our design and began constructing the database through MySQL by coding the tables established in our schema, and using commands to insert LOST's data of their tours into the respected values. Making this difficult challenge