## 

## 

**Experiment 5:**

**Data Modeling and Database Systems**

CPE106L (Software Design Laboratory)

**Manzanero, Jan Adrian C.**

Group No.: **6**

Section: **B1**

## **PreLab**



**Readings, Insights, and Reflection**

SQLite is a relational database system that provides a lightweight and self-contained solution for data storage. It organizes data into tables, which are like filing cabinets with categorized drawers. Each table focuses on a specific subject, such as customers or trips. The rows within a table function like individual folders, holding detailed records for a single customer or trip. Columns define the attributes associated with each record, similar to folder labels. SQL databases can be used to efficiently track records for stores, properties, and services.

In SQLite databases, primary keys (PK) and foreign keys (FK) work together to establish structure and enforce data relationships between tables. A primary key acts as a unique identifier for each row within a table, whereas a foreign key establishes relationships between two tables by referencing the primary key of another table. By using primary and foreign keys, one can construct a robust database schema in SQLite that maintains data consistency and simplifies queries.

Python bridges the gap between your programs and databases like SQLite. To manage them, import a library (e.g., sqlite3 for SQLite) and use it to connect. With the connection, you can execute SQL statements (retrieve, add, modify, or delete data) through a cursor. Python lets you process retrieved data (e.g., iterate through rows) and handle potential errors. This empowers you to build database applications that interact with various systems, storing, retrieving, manipulating, and analyzing data efficiently within your Python projects.

**Answer to Questions**

1. Data Manipulation Language (DML) statements in SQL are used to manage data within existing tables in a database, such as SELECT, INSERT, UPDATE, and DELETE. Data Definition Language (DDL) statements in SQL are used to define the structure of the database, such as CREATE TABLE, ALTER TABLE, and DROP TABLE.
2. SQLite functions can be categorized into several categories: aggregate (COUNT(), SUM(), AVG()), string (UPPER(), LOWER(), LENGTH()), date and time (DATETIME(), CURRENT\_DATE(), CURRENT\_TIME()), math (ABS(), ROUND(), POW()), and data conversion (INT(), TEXT(), REAL()).
3. To check if SQLite is installed in the Linux terminal, the user can type the command sqlite3 in the terminal. If it is installed, the terminal will prompt the version of the SQLite that has been installed. Otherwise, it will not return a prompt.

## **InLab**



**Objectives**

* To define SQL databases using DDL and DML SQLite commands.
* To display the contents of SQL Databases using DB Browser.
* To apply SQL databases in real-life scenarios.

**Tools Used**

* GitHub
* Oracle Virtual Machine VirtualBox Manager
* Visual Studio Code
* SQLite
* DB Browser

**Procedure**

**Part 1: Installing SQLite and Chinook Database**

The students must first install the needed files before doing the machine problems.

A screenshot of a computer

Description automatically generated

*Figure 1.1. Installing SQLite in the student’s VM using the Linux terminal.*

A screenshot of a computer

Description automatically generated

*Figure 1.2. SQL database scripts to be used for the machine problems.*

**Part 2: Creation of the Database using SQL Scripts**

Snippets of the SQL scripts used are shown below.

A screenshot of a computer

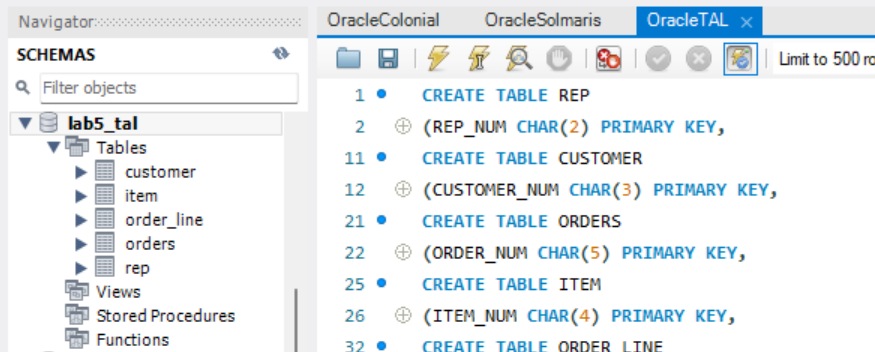
Description automatically generated

*Figure 2.1. Creation of the database for Colonial.*

A screenshot of a computer

Description automatically generated

*Figure 2.2. Creation of the database for Solmaris.*



*Figure 2.3. Creation of the database for TAL.*

**Part 3: Displaying Database Using SQL Commands**

The tables in the database earlier were already populated with data from the SQL scripts given to the students. These records are for different objects of the databases. For the Colonial database, there are tables for customers, trips, trip guides, assigned trips for the trip guides, and reserved dates for the trips. For the Solmaris database, there are tables for the condos, the owners, the locations, the types of services that they can avail, and the service requests of the owners. For the TAL database, there are tables for the customers, the items, and the representatives, as well as for their orders and order lines.

These tables can be displayed in the DB Browser and are shown in the screenshots below.

A screenshot of a computer

Description automatically generated

*Figure 3.1. Displaying the table TRIP of the Colonial database.*

A screenshot of a computer

Description automatically generated

*Figure 3.2. Displaying the table SERVICE\_REQUESTS of the Solmaris database.*

A screenshot of a computer

Description automatically generated

*Figure 3.3. Displaying the table CUSTOMER of the TAL database.*

## **PostLab**



[GitHub Repository Branch](https://github.com/mepue-mels/CPE106L_B1_Labs/tree/manzanero/Lab5)

**Machine Problem 1:**

a)

CUSTOMER(customer\_num, last\_name, first\_name, address, city, state, postal\_code, phone, birth\_date)

b)

TRIP(trip\_id, trip\_name, max\_grp\_size, trip\_price)

c)

CUSTOMER(customer\_num, last\_name, first\_name)

RESERVATION(customer\_num -> CUSTOMER.customer\_num, trip\_id -> TRIP.trip\_id, trip\_name -> TRIP.trip\_name, trip\_date)

d)

RESERVATION(trip\_date, trip\_id -> TRIP.trip\_id, trip\_name -> TRIP.trip\_name, customer\_num -> CUSTOMER.customer\_num)

CUSTOMER(customer\_num, last\_name, first\_name)

+-------------------+ (One-to-Many)

| CUSTOMER | ------------+------------------------------------------+

+-------------------+ | RESERVATION |

| CUSTOMER\_NUM (PK) | <---------> | (Entity) |

| LAST\_NAME | | RESERVATION\_ID (PK) | varchar(7)

| FIRST\_NAME | | TRIP\_ID (FK)->TRIP.TRIP\_ID | decimal(3,0)

| ADDRESS | | TRIP\_DATE | char(20)

| CITY | | NUM\_PERSONS | decimal(3,0)

| STATE | | TRIP\_PRICE | decimal(6,2)

| POSTAL\_CODE | | OTHER\_FEES | decimal(6,2)

| PHONE | | CUSTOMER\_NUM (FK)->CUSTOMER.CUSTOMER\_NUM | char(4)

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

(Many-to-Many)

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

| GUIDE | | TRIP | (Many-to-Many)

+-------------------+ | TRIP\_ID (PK) | decimal(3,0) (PK)

| GUIDE\_NUM (PK) | <------> | TRIP\_NAME | char(75)

| LAST\_NAME | | START\_LOCATION | char(50)

| FIRST\_NAME | | STATE | char(2)

| ADDRESS | | DISTANCE | decimal(4,0)

| CITY | | MAX\_GRP\_SIZE | decimal(4,0)

| STATE | | TYPE | char(20)

| POSTAL\_CODE | | SEASON | char(20)

| PHONE\_NUM | +----------------+

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

**Machine Problem 2:**

a)

OWNER(owner\_num, first\_name, middle\_initial, last\_name, address, city, state, postal\_code, phone, email)

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

| OWNER | (Entity)

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

| owner\_num (PK) | char(5)

| first\_name | char(25)

| middle\_initial | char(1)

| last\_name | char(25)

| address | char(25)

| city | char(25)

| state | char(2)

| postal\_code | char(5)

| phone | char(12)

| email | char(50)

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

b)

LOCATION(location\_num, location\_name, address, city, state, postal\_code)

CONDO\_UNIT(location\_num -> LOCATION.location\_num, unit\_num, sqr\_ft, bdrms, baths, max\_people, condo\_fee)

+-------------------+ (One-to-Many)

| LOCATION |

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

| location\_num (PK) | decimal(2,0) <---->|

| location\_name | char(25) | CONDO\_UNIT |

| address | char(25) | (Entity) |

| city | char(25) +--------------+

| state | char(2) | unit\_num (PK)| char(3)

| postal\_code | char(5) | sqr\_ft | decimal(5,0)

+-------------------+ | bdrms | decimal(2,0)

| baths | decimal(2,0)

| max\_people | decimal(2,0)

| condo\_fee | decimal(6,2)

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

c)

OWNER(owner\_num, first\_name, middle\_initial, last\_name, address, city, state, postal\_code, phone)

CONDO\_UNIT(start\_date, end\_date, condo\_fee)

+-------------------+ (One-to-Many)

| OWNER |

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

| owner\_num (PK) | char(5) <---------> |

| first\_name | char(25) | CONDO\_UNIT |

| middle\_initial | char(1) | (Entity) |

| last\_name | char(25) +-------------+

| address | char(25) | start\_date | date

| city | char(25) | end\_date | date

| state | char(2) | condo\_fee | decimal(6,2)

| postal\_code | char(5) +-------------+

| phone | char(12)

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

**Machine Problem 3:**

a)

CREATE TABLE ADVENTURE\_TRIP (

TRIP\_ID INTEGER PRIMARY KEY,

TRIP\_NAME TEXT,

START\_LOCATION TEXT,

STATE TEXT,

DISTANCE INTEGER,

MAX\_GRP\_SIZE INTEGER,

TYPE TEXT,

SEASON TEXT

);

b)

INSERT INTO `ADVENTURE\_TRIP` (TRIP\_ID, TRIP\_NAME, START\_LOCATION, STATE, DISTANCE, MAX\_GRP\_SIZE, TYPE, SEASON)

VALUES('45','Jay Peak','Jay','VT','8','8','Hiking','Summer');

c)

DROP TABLE `ADVENTURE\_TRIP`;

d)

CREATE TABLE Guide (

Guide\_Num CHAR(4) PRIMARY KEY,

Last\_Name CHAR(15) NOT NULL,

First\_Name CHAR(15) NOT NULL,

Address CHAR(25),

City CHAR(25),

State CHAR(2),

Postal\_Code CHAR(5),

Phone\_Num CHAR(12),

Hire\_Date CHAR(20)

);

CREATE TABLE Customer (

Customer\_Num CHAR(4) PRIMARY KEY,

Last\_Name CHAR(30) NOT NULL,

First\_Name CHAR(30),

Address CHAR(35),

City CHAR(35),

State CHAR(2),

Postal\_Code CHAR(5),

Phone CHAR(12)

);

CREATE TABLE Reservation (

Reservation\_ID CHAR(7) PRIMARY KEY,

Trip\_ID INTEGER NOT NULL,

Trip\_Date CHAR(20) NOT NULL,

Num\_Persons INTEGER NOT NULL,

Trip\_Price DECIMAL(6,2) NOT NULL,

Other\_Fees DECIMAL(6,2) NOT NULL,

Customer\_Num CHAR(4) NOT NULL,

FOREIGN KEY (Customer\_Num) REFERENCES Customer(Customer\_Num));

CREATE TABLE Trip (

Trip\_ID INTEGER PRIMARY KEY,

Trip\_Name CHAR(75) NOT NULL,

Start\_Location CHAR(50) NOT NULL,

State CHAR(2),

Distance INTEGER NOT NULL,

Max\_Grp\_Size INTEGER NOT NULL,

Type CHAR(20) NOT NULL,

Season CHAR(20) NOT NULL);

CREATE TABLE Trip\_Guides (

Trip\_ID INTEGER NOT NULL,

Guide\_Num CHAR(4) NOT NULL,

PRIMARY KEY (Trip\_ID, Guide\_Num),

FOREIGN KEY (Trip\_ID) REFERENCES Trip(Trip\_ID),

FOREIGN KEY (Guide\_Num) REFERENCES Guide(Guide\_Num));

e)

A screenshot of a computer

Description automatically generated

*Figure 4.1. Colonial Adventure Tours Database CUSTOMER Table.*

A screenshot of a computer

Description automatically generated

*Figure 4.2. Colonial Adventure Tours Database GUIDE Table.*

A screenshot of a computer

Description automatically generated

*Figure 4.3. Colonial Adventure Tours Database RESERVATION Table.*

A screenshot of a computer

Description automatically generated

*Figure 4.4. Colonial Adventure Tours Database TRIP Table.*

A screenshot of a computer

Description automatically generated

*Figure 4.5. Colonial Adventure Tours Database TRIP\_GUIDES Table.*