Course: Capstone Database Engineer

Final Project Summary

Database Creation, Reading, Updating, and Deletion (CRUD) using .csv file, MySQL, MySQL WorkBench,Python, Jupyter Notebook, GitHub, and Visual Studio Code

Step 0:
Observe and transform the dataset

Step 1:
Create an ER diagram

Step 2:
Create the database structure

Step 3:
Insert data into the database structure

Step 4:
Insert data into the Tables

Step 5:
Create a stored procedure that returns the maximum quantity in an order.

Step 6:
Create a stored procedure that manages booking.

Step 7:
Create a stored procedure that inserts a booking into the database.

Step 8:
Create a stored procedure that updates a booking into the database.

Step 9:
Create a stored procedure that deletes a booking into the database.

Step 10:
Create a GitHub repository to store the code and then automatically manage its versions (adding and removing) with Git.

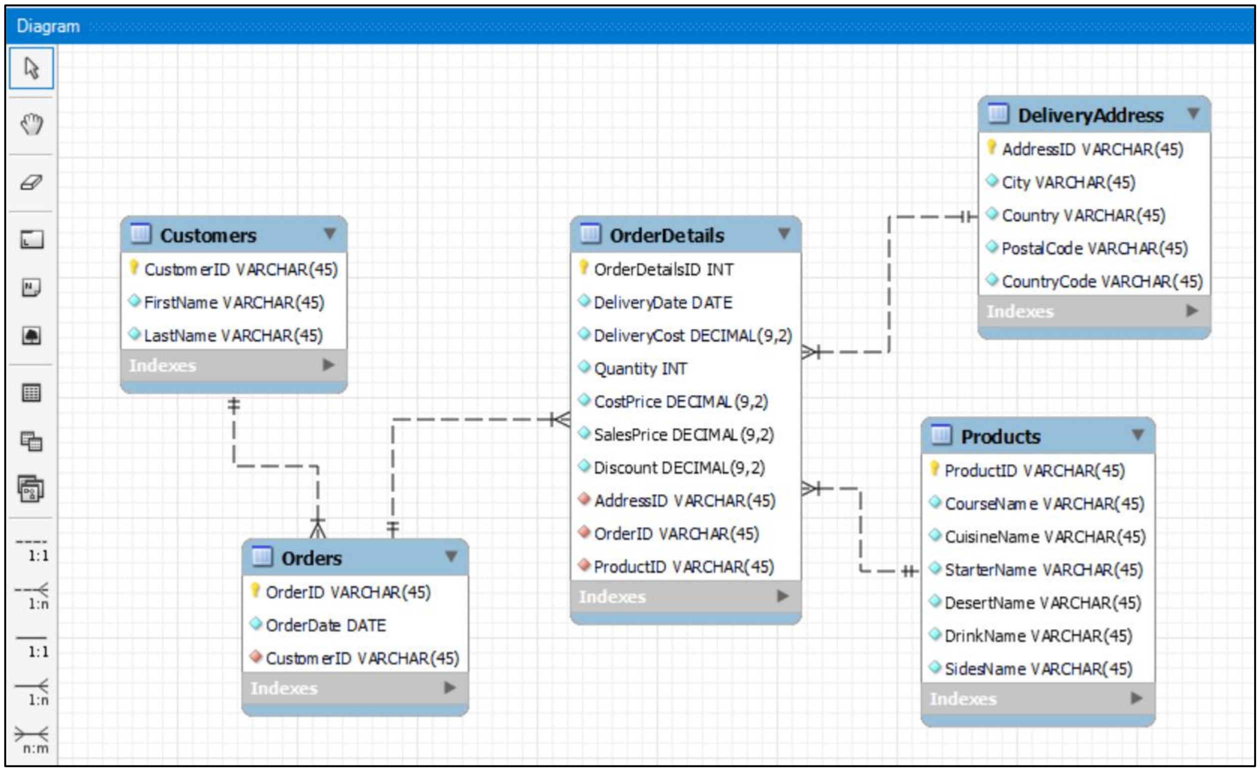
Step 0:
Observe and transform the dataset

First, I observed the Excel file dataset given that includes the data collected for Little Lemon Organization, as stated in the scenario, in order to understand this data. I checked and understood for example the column names the most appropriate datatype to assign to them. I converted the Excel file from the format .xlsx to the format .csv (Comma-Separated Values) because in this format I will be able to read it directly with Python.

Second, I used the Python Library Pandas to aggregate the data and understand them better. I understood for example that there are no empty cell and no duplicated rows. And each Customer ID corresponds to 21 orders that, however, don’t have for example the same Postal Code, Cost, Sales, and Discount. I also understood that some data that can be grouped in a Table missed a Primary key.

Step 1:
Create an ER diagram

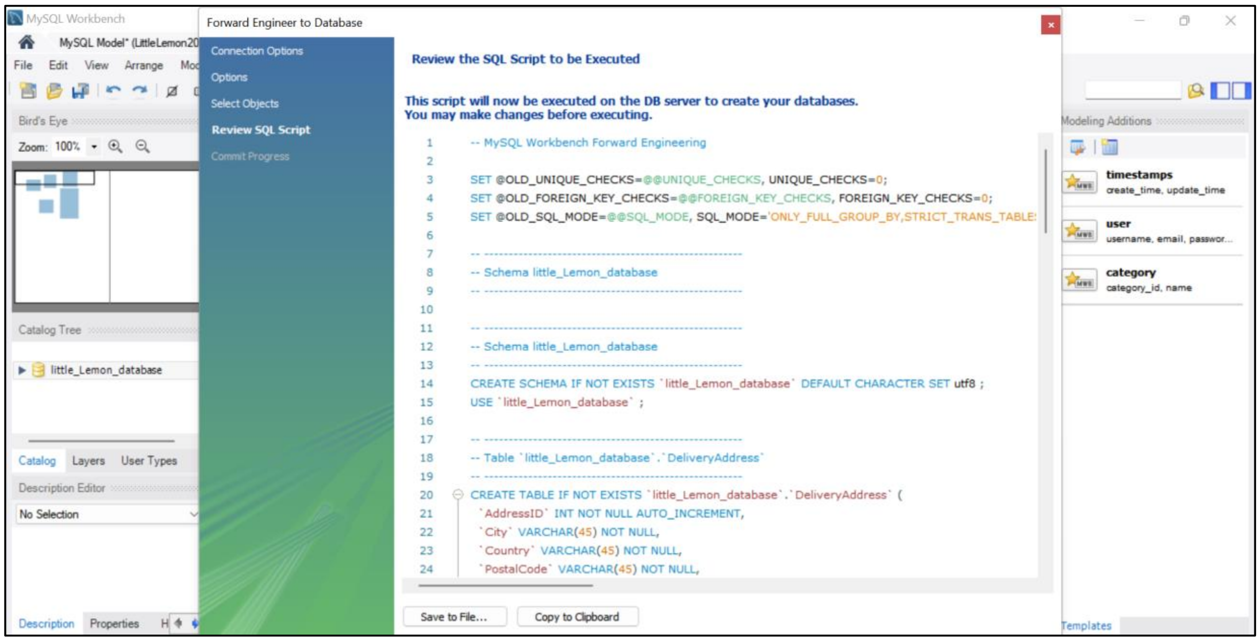
At the end of Step 1, I understood the data enough the data collected by Little Lemon to design the most appropriate model of the database that can be used to save them. I used MySQL Workbench to create a new model that includes relevant entities and attributes as shown below:



Step 2:

Create the database structure

I used the feature Forward Engineer of MySQL Workbench to implement the data model, created in Step 2 inside MySQL server, and then create the corresponding database and tables, as shown below.

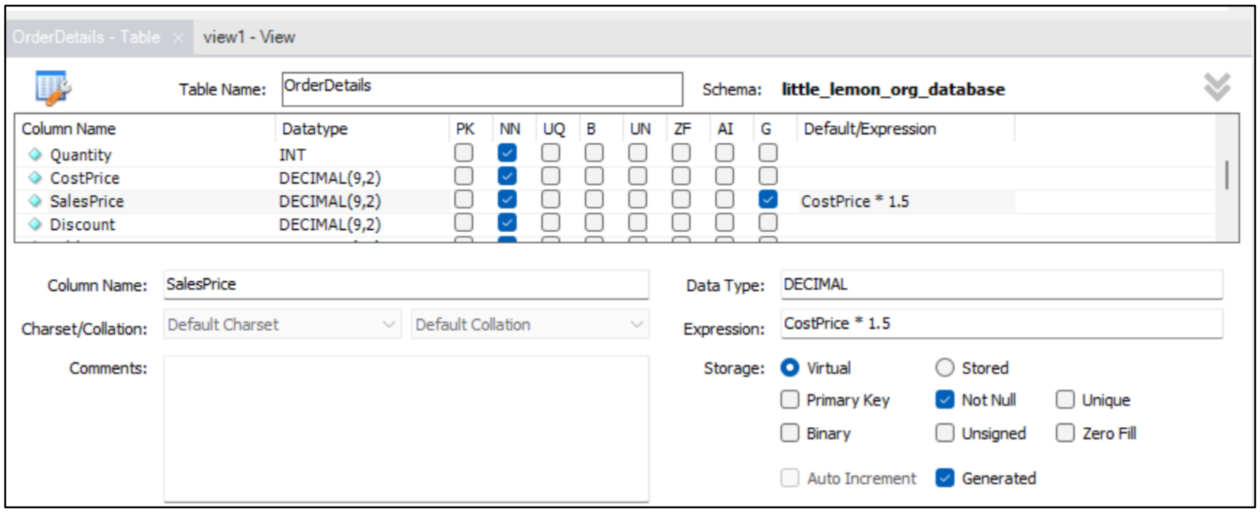


Review the SQL Script to be Executed

This script will now be executed on the DB server to create your databases.
You may make changes before executing.

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLE

-- -----------------------------------------------------
-- Schema little_Lemon_database
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema little_Lemon_database
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `little_Lemon_database` DEFAULT CHARACTER SET utf8 ;
USE `little_Lemon_database` ;

-- -----------------------------------------------------
-- Table `little_Lemon_database`.`DeliveryAddress`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `little_Lemon_database`.`DeliveryAddress` (
  `AddressID` INT NOT NULL AUTO_INCREMENT,
  `City` VARCHAR(45) NOT NULL,
  `Country` VARCHAR(45) NOT NULL,
  `PostalCode` VARCHAR(45) NOT NULL,
```

This code includes a virtual column named SalesPrice in the table OrderDetails, as shown below.



OrderDetails - Table | view1 - View

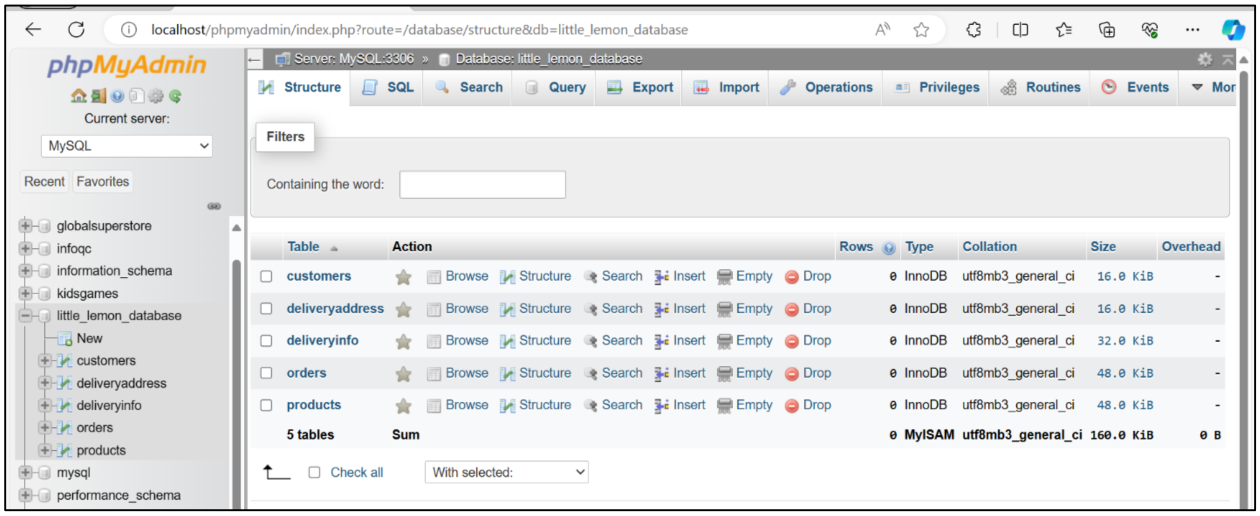
Table Name: OrderDetails   Schema: little_lemon_org_database

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
Quantity	INT	☐	✓	☐	☐	☐	☐	☐	☐	
CostPrice	DECIMAL(9,2)	☐	✓	☐	☐	☐	☐	☐	☐	
SalesPrice	DECIMAL(9,2)	☐	✓	☐	☐	☐	☐	☐	✓	CostPrice * 1.5
Discount	DECIMAL(9,2)	☐	✓	☐	☐	☐	☐	☐	☐	

Column Name: SalesPrice   Data Type: DECIMAL
Charset/Collation: Default Charset / Default Collation   Expression: CostPrice * 1.5
Comments:
Storage: [✓ Virtual] Stored
Primary Key [✓ Not Null] Unique
Binary Unsigned Zero Fill
Auto Increment [✓ Generated]

```
`SalesPrice` DECIMAL(9,2) GENERATED ALWAYS AS (CostPrice * 1.5) VIRTUAL,
```

Now, using phpMyAdmin, I can verify that the database was successfully created in MySQL, as shown below:



localhost/phpmyadmin/index.php?route=/database/structure&db=little_lemon_database

Server: MySQL:3306 » Database: little_lemon_database

Current server: MySQL

Databases: globalsuperstore, infoqc, information_schema, kidsgames, little_lemon_database (New, customers, deliveryaddress, deliveryinfo, orders, products), mysql, performance_schema

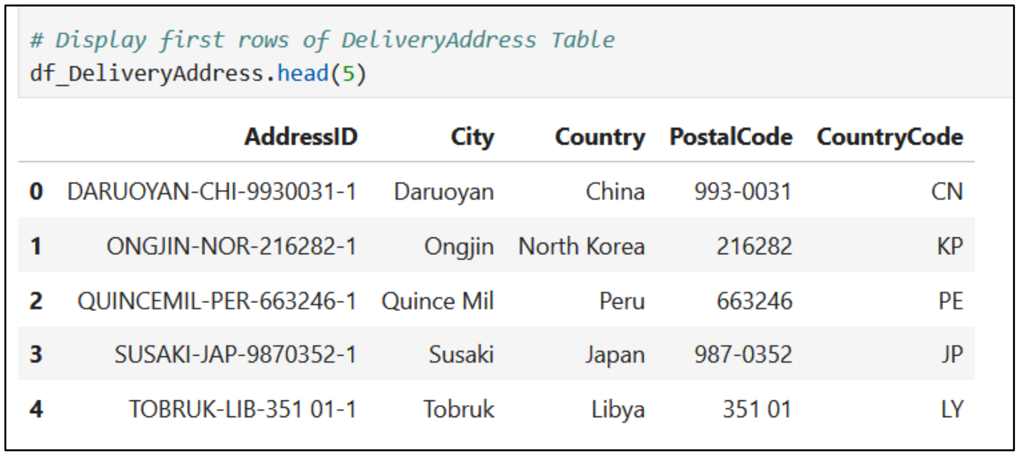
Table	Action	Rows	Type	Collation	Size	Overhead
customers	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb3_general_ci	16.0 KiB	-
deliveryaddress	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb3_general_ci	16.0 KiB	-
deliveryinfo	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb3_general_ci	32.0 KiB	-
orders	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb3_general_ci	48.0 KiB	-
products	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb3_general_ci	48.0 KiB	-
5 tables	Sum	0	MyISAM	utf8mb3_general_ci	160.0 KiB	0 B

Step 3:
Insert data into the database structure

I used Jupyter Notebook and the Python Library Pandas to transform the data and create a data frame that includes data for each of the Tables created in Step 3.

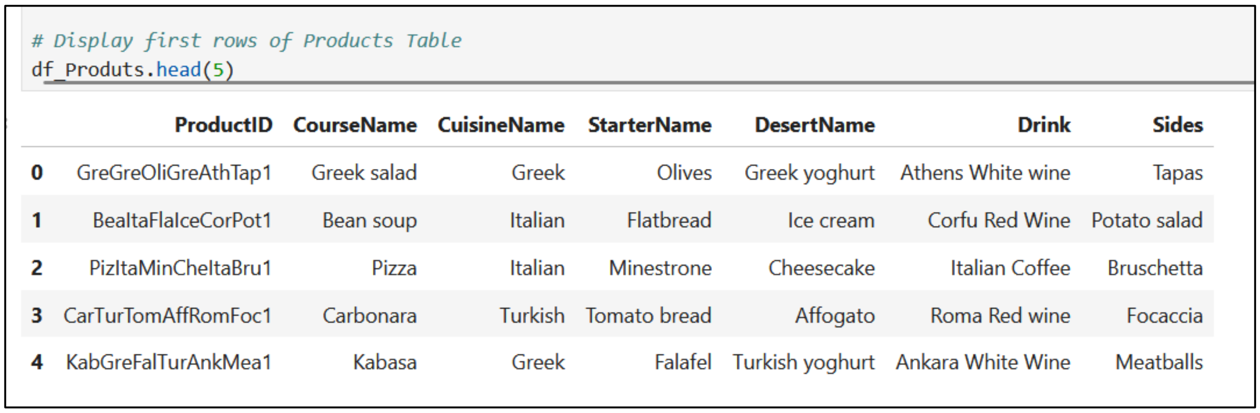
For example, I extract the data from the column Name to create 2 new columns named FirstName (first word of the column Name) and LastName (Words that are not the first word of the column Name). I created some missing primary keys.

Below is shown the first 5 rows of the data frame created for each table.



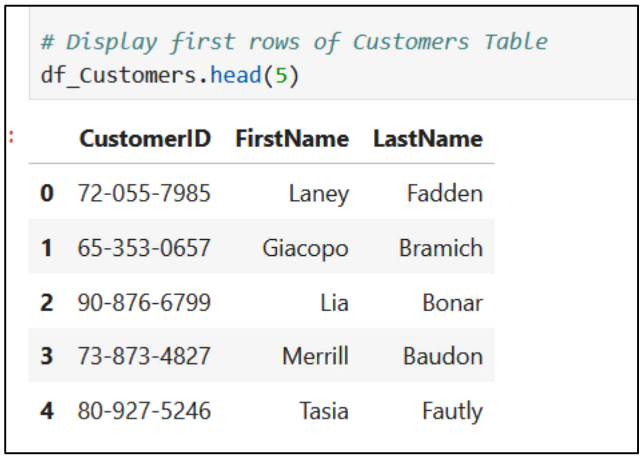
```
# Display first rows of DeliveryAddress Table
df_DeliveryAddress.head(5)
```

	AddressID	City	Country	PostalCode	CountryCode
0	DARUOYAN-CHI-9930031-1	Daruoyan	China	993-0031	CN
1	ONGJIN-NOR-216282-1	Ongjin	North Korea	216282	KP
2	QUINCEMIL-PER-663246-1	Quince Mil	Peru	663246	PE
3	SUSAKI-JAP-9870352-1	Susaki	Japan	987-0352	JP
4	TOBRUK-LIB-351 01-1	Tobruk	Libya	351 01	LY



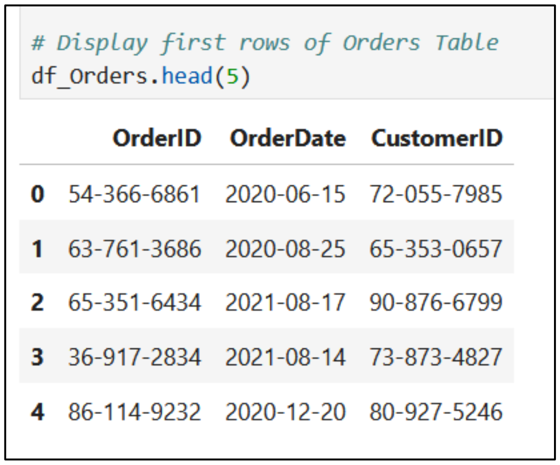
```
# Display first rows of Products Table
df_Produts.head(5)
```

	ProductID	CourseName	CuisineName	StarterName	DesertName	Drink	Sides
0	GreGreOliGreAthTap1	Greek salad	Greek	Olives	Greek yoghurt	Athens White wine	Tapas
1	BealtaFlaIceCorPot1	Bean soup	Italian	Flatbread	Ice cream	Corfu Red Wine	Potato salad
2	PizItaMinCheltaBru1	Pizza	Italian	Minestrone	Cheesecake	Italian Coffee	Bruschetta
3	CarTurTomAffRomFoc1	Carbonara	Turkish	Tomato bread	Affogato	Roma Red wine	Focaccia
4	KabGreFalTurAnkMea1	Kabasa	Greek	Falafel	Turkish yoghurt	Ankara White Wine	Meatballs



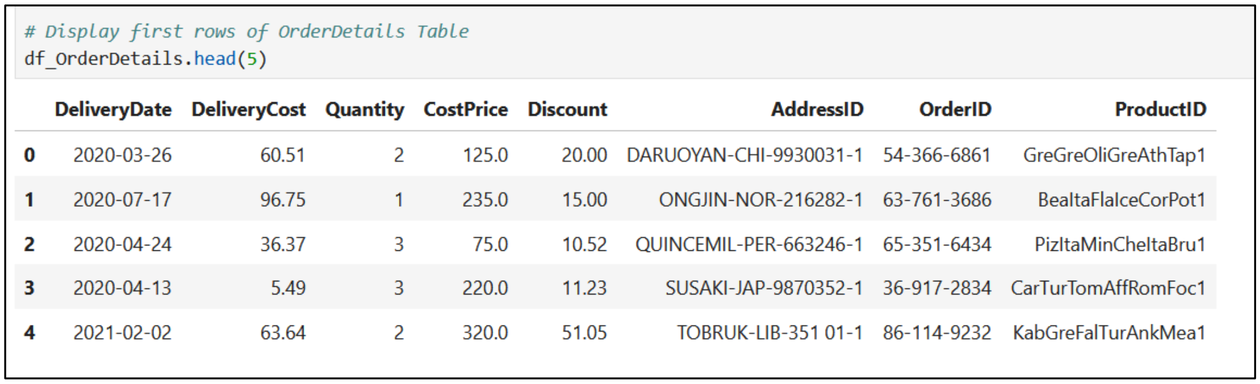
```python
# Display first rows of Customers Table
df_Customers.head(5)
```

	CustomerID	FirstName	LastName
0	72-055-7985	Laney	Fadden
1	65-353-0657	Giacopo	Bramich
2	90-876-6799	Lia	Bonar
3	73-873-4827	Merrill	Baudon
4	80-927-5246	Tasia	Fautly



```python
# Display first rows of Orders Table
df_Orders.head(5)
```

	OrderID	OrderDate	CustomerID
0	54-366-6861	2020-06-15	72-055-7985
1	63-761-3686	2020-08-25	65-353-0657
2	65-351-6434	2021-08-17	90-876-6799
3	36-917-2834	2021-08-14	73-873-4827
4	86-114-9232	2020-12-20	80-927-5246



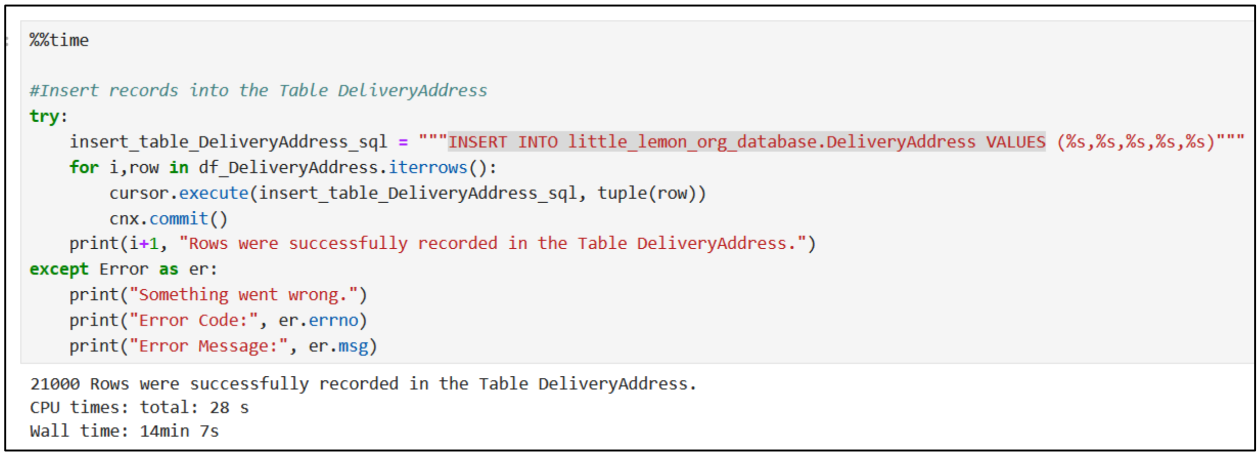
```python
# Display first rows of OrderDetails Table
df_OrderDetails.head(5)
```

	DeliveryDate	DeliveryCost	Quantity	CostPrice	Discount	AddressID	OrderID	ProductID
0	2020-03-26	60.51	2	125.0	20.00	DARUOYAN-CHI-9930031-1	54-366-6861	GreGreOliGreAthTap1
1	2020-07-17	96.75	1	235.0	15.00	ONGJIN-NOR-216282-1	63-761-3686	BealtaFlaIceCorPot1
2	2020-04-24	36.37	3	75.0	10.52	QUINCEMIL-PER-663246-1	65-351-6434	PizItaMinCheltaBru1
3	2020-04-13	5.49	3	220.0	11.23	SUSAKI-JAP-9870352-1	36-917-2834	CarTurTomAffRomFoc1
4	2021-02-02	63.64	2	320.0	51.05	TOBRUK-LIB-351 01-1	86-114-9232	KabGreFalTurAnkMea1

Step 4:

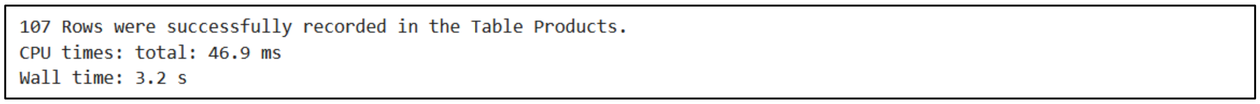
Insert data into the Tables

I used Jupyter Notebook and the Python's Library mysql.connector to connect to the database structure created in Step 2 in order to insert the data prepared in the data frames in Step 4 into the tables. Below is shown the confirmation received at the end of the insertion process in each table.

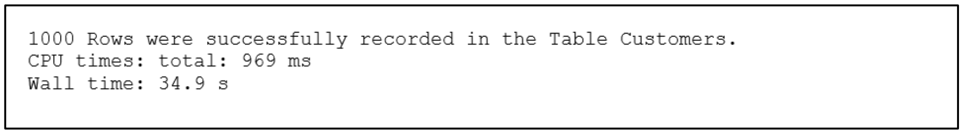


```
%%time

#Insert records into the Table DeliveryAddress
try:
    insert_table_DeliveryAddress_sql = """INSERT INTO little_lemon_org_database.DeliveryAddress VALUES (%s,%s,%s,%s,%s)"""
    for i,row in df_DeliveryAddress.iterrows():
        cursor.execute(insert_table_DeliveryAddress_sql, tuple(row))
        cnx.commit()
    print(i+1, "Rows were successfully recorded in the Table DeliveryAddress.")
except Error as er:
    print("Something went wrong.")
    print("Error Code:", er.errno)
    print("Error Message:", er.msg)
```

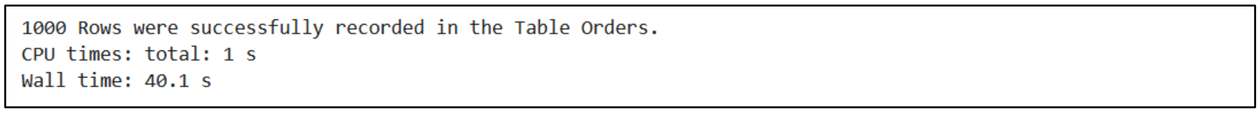
```
21000 Rows were successfully recorded in the Table DeliveryAddress.
CPU times: total: 28 s
Wall time: 14min 7s
```



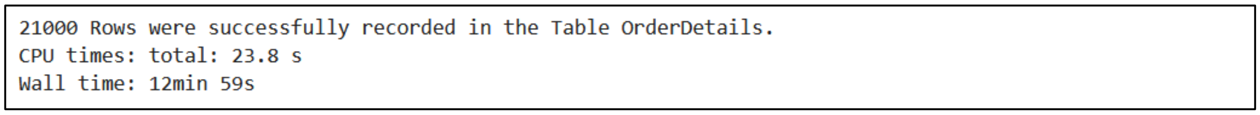
```
107 Rows were successfully recorded in the Table Products.
CPU times: total: 46.9 ms
Wall time: 3.2 s
```



```
1000 Rows were successfully recorded in the Table Customers.
CPU times: total: 969 ms
Wall time: 34.9 s
```



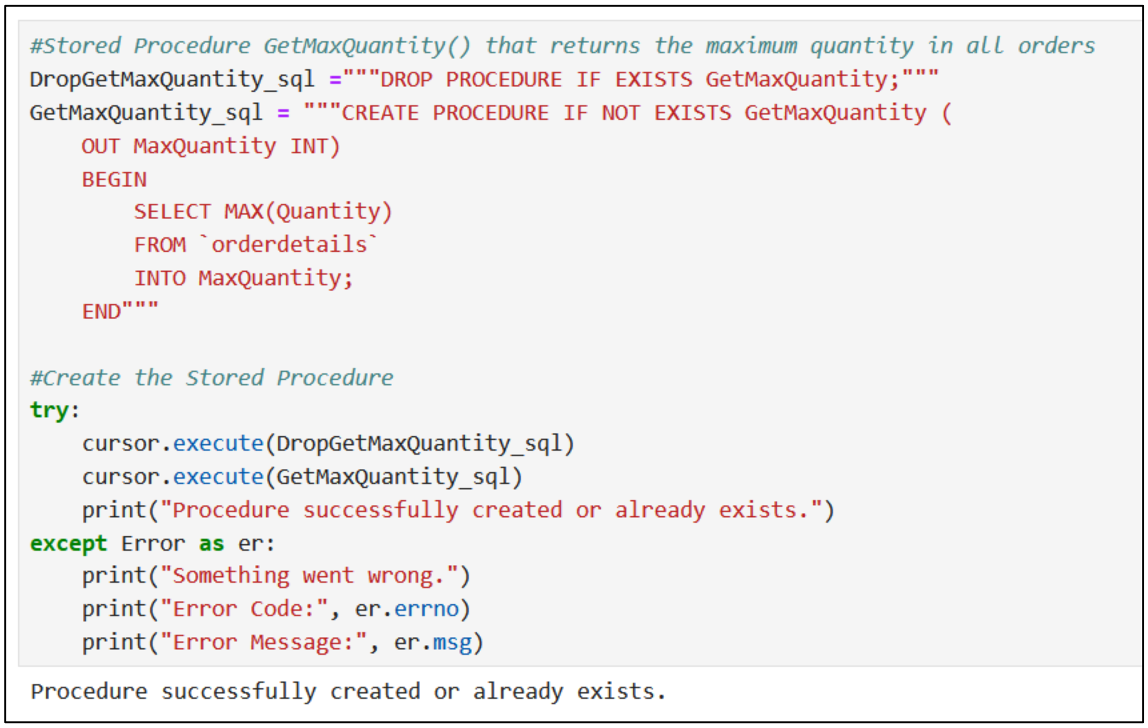
```
1000 Rows were successfully recorded in the Table Orders.
CPU times: total: 1 s
Wall time: 40.1 s
```



```
21000 Rows were successfully recorded in the Table OrderDetails.
CPU times: total: 23.8 s
Wall time: 12min 59s
```

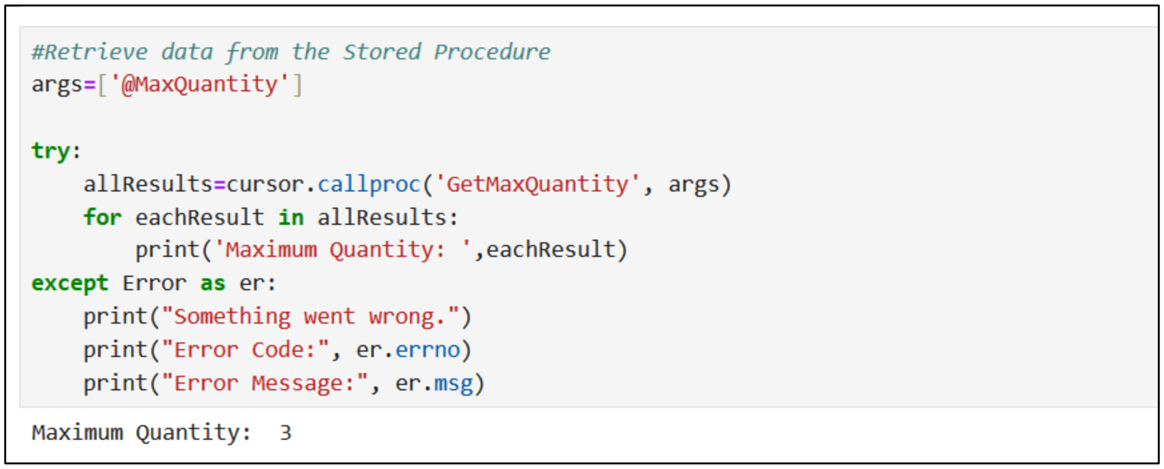
Step 5:
Create a stored procedure that returns the maximum quantity in an order.

Procedure GetMaxQuantity() that returns the maximum quantity in all orders



```
#Stored Procedure GetMaxQuantity() that returns the maximum quantity in all orders
DropGetMaxQuantity_sql ="""DROP PROCEDURE IF EXISTS GetMaxQuantity;"""
GetMaxQuantity_sql = """CREATE PROCEDURE IF NOT EXISTS GetMaxQuantity (
    OUT MaxQuantity INT)
    BEGIN
        SELECT MAX(Quantity)
        FROM `orderdetails`
        INTO MaxQuantity;
    END"""

#Create the Stored Procedure
try:
    cursor.execute(DropGetMaxQuantity_sql)
    cursor.execute(GetMaxQuantity_sql)
    print("Procedure successfully created or already exists.")
except Error as er:
    print("Something went wrong.")
    print("Error Code:", er.errno)
    print("Error Message:", er.msg)
```

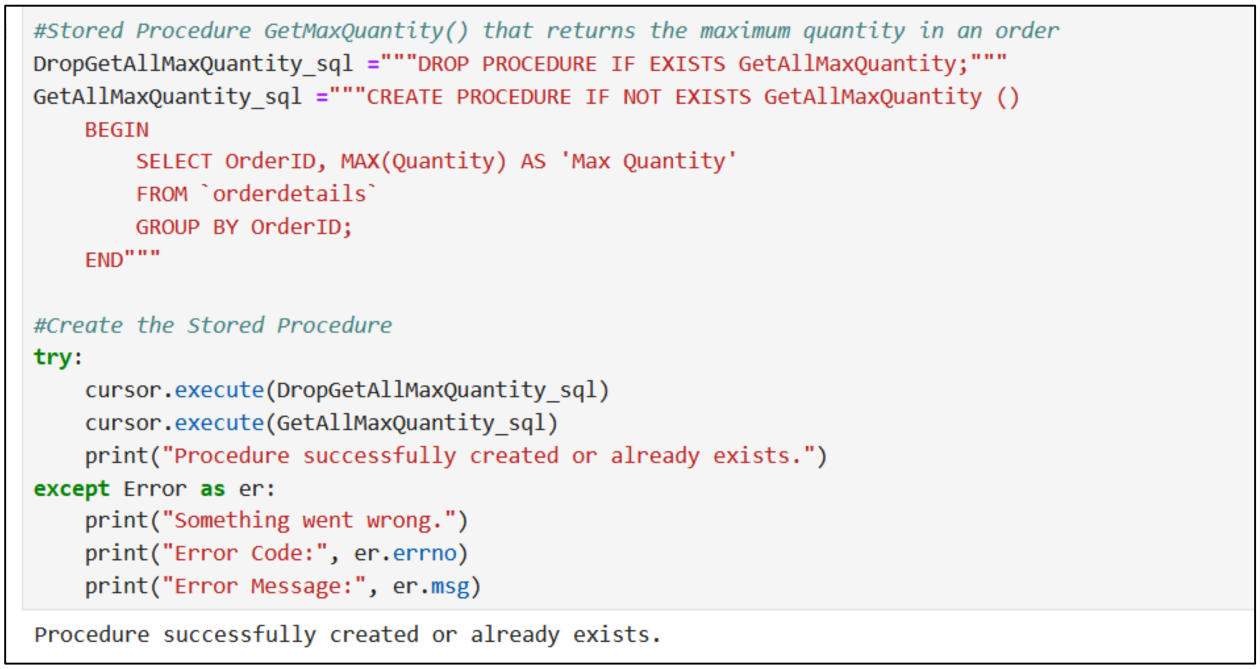
```
Procedure successfully created or already exists.
```



```
#Retrieve data from the Stored Procedure
args=['@MaxQuantity']

try:
    allResults=cursor.callproc('GetMaxQuantity', args)
    for eachResult in allResults:
        print('Maximum Quantity: ',eachResult)
except Error as er:
    print("Something went wrong.")
    print("Error Code:", er.errno)
    print("Error Message:", er.msg)
```

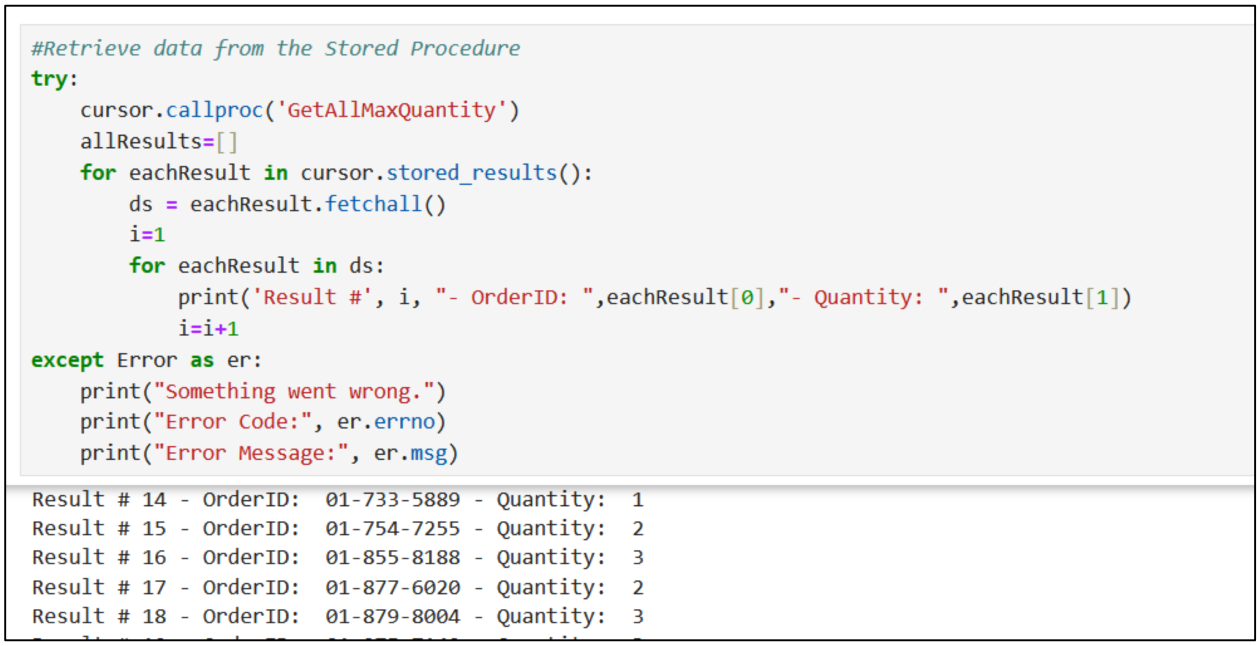
```
Maximum Quantity:  3
```

Procedure GetMaxQuantity() that returns the maximum quantity in an order



```
#Stored Procedure GetMaxQuantity() that returns the maximum quantity in an order
DropGetAllMaxQuantity_sql ="""DROP PROCEDURE IF EXISTS GetAllMaxQuantity;"""
GetAllMaxQuantity_sql ="""CREATE PROCEDURE IF NOT EXISTS GetAllMaxQuantity ()
    BEGIN
        SELECT OrderID, MAX(Quantity) AS 'Max Quantity'
        FROM `orderdetails`
        GROUP BY OrderID;
    END"""

#Create the Stored Procedure
try:
    cursor.execute(DropGetAllMaxQuantity_sql)
    cursor.execute(GetAllMaxQuantity_sql)
    print("Procedure successfully created or already exists.")
except Error as er:
    print("Something went wrong.")
    print("Error Code:", er.errno)
    print("Error Message:", er.msg)
```

```
Procedure successfully created or already exists.
```

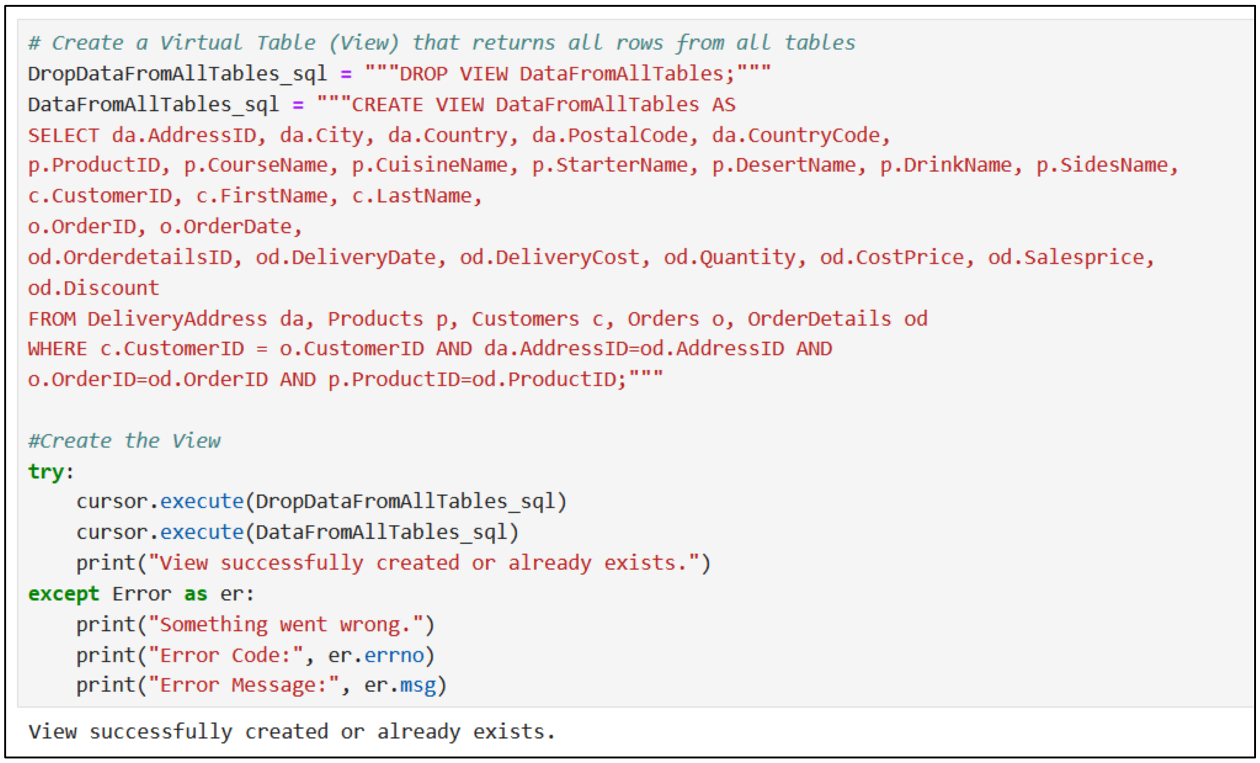


```
#Retrieve data from the Stored Procedure
try:
    cursor.callproc('GetAllMaxQuantity')
    allResults=[]
    for eachResult in cursor.stored_results():
        ds = eachResult.fetchall()
        i=1
        for eachResult in ds:
            print('Result #', i, "- OrderID: ",eachResult[0],"- Quantity: ",eachResult[1])
            i=i+1
except Error as er:
    print("Something went wrong.")
    print("Error Code:", er.errno)
    print("Error Message:", er.msg)
```

```
Result # 14 - OrderID:  01-733-5889 - Quantity:  1
Result # 15 - OrderID:  01-754-7255 - Quantity:  2
Result # 16 - OrderID:  01-855-8188 - Quantity:  3
Result # 17 - OrderID:  01-877-6020 - Quantity:  2
Result # 18 - OrderID:  01-879-8004 - Quantity:  3
```

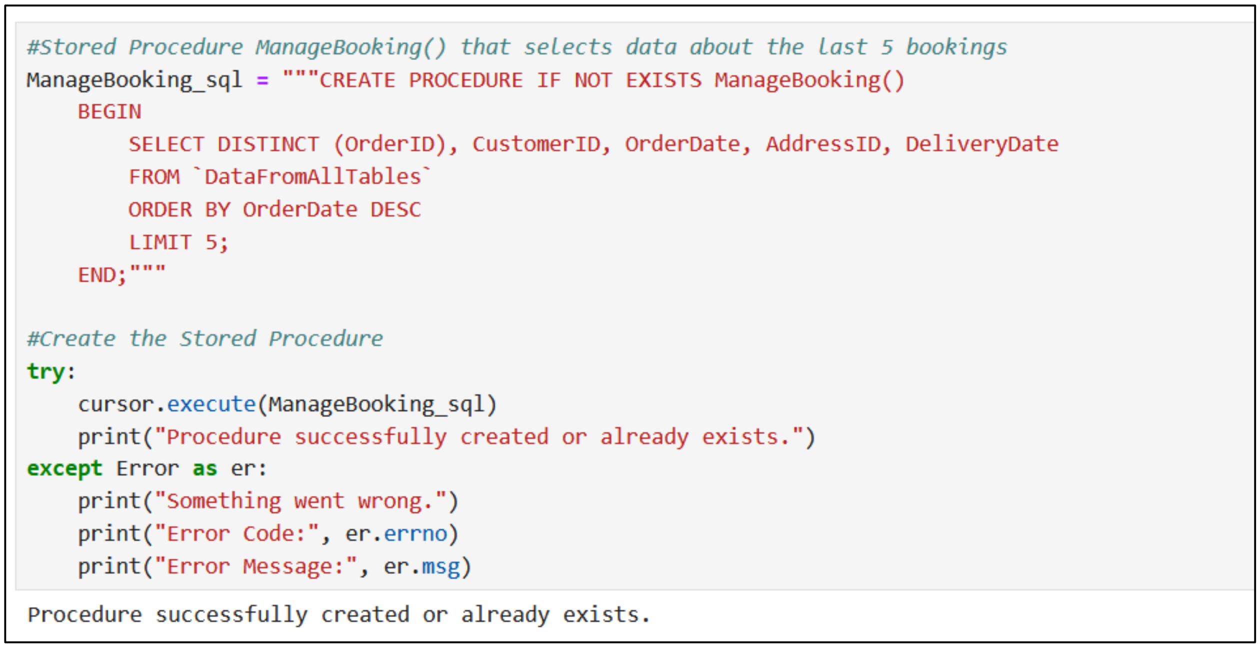
Step 6:

Create a stored procedure that manages booking.



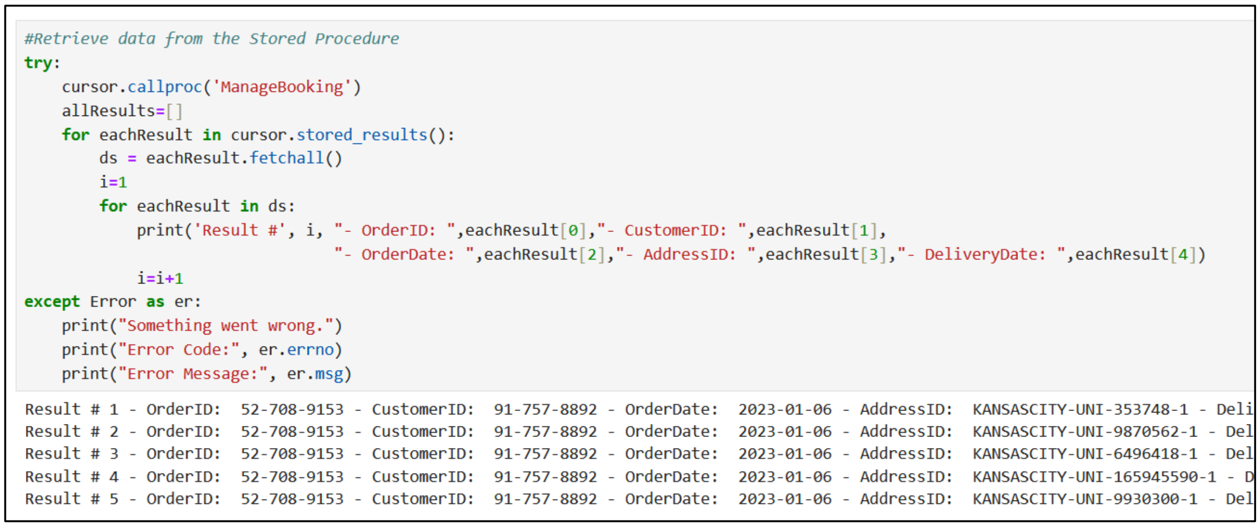
```
# Create a Virtual Table (View) that returns all rows from all tables
DropDataFromAllTables_sql = """DROP VIEW DataFromAllTables;"""
DataFromAllTables_sql = """CREATE VIEW DataFromAllTables AS
SELECT da.AddressID, da.City, da.Country, da.PostalCode, da.CountryCode,
p.ProductID, p.CourseName, p.CuisineName, p.StarterName, p.DesertName, p.DrinkName, p.SidesName,
c.CustomerID, c.FirstName, c.LastName,
o.OrderID, o.OrderDate,
od.OrderdetailsID, od.DeliveryDate, od.DeliveryCost, od.Quantity, od.CostPrice, od.Salesprice,
od.Discount
FROM DeliveryAddress da, Products p, Customers c, Orders o, OrderDetails od
WHERE c.CustomerID = o.CustomerID AND da.AddressID=od.AddressID AND
o.OrderID=od.OrderID AND p.ProductID=od.ProductID;"""

#Create the View
try:
    cursor.execute(DropDataFromAllTables_sql)
    cursor.execute(DataFromAllTables_sql)
    print("View successfully created or already exists.")
except Error as er:
    print("Something went wrong.")
    print("Error Code:", er.errno)
    print("Error Message:", er.msg)
```

```
View successfully created or already exists.
```



```
#Stored Procedure ManageBooking() that selects data about the last 5 bookings
ManageBooking_sql = """CREATE PROCEDURE IF NOT EXISTS ManageBooking()
    BEGIN
        SELECT DISTINCT (OrderID), CustomerID, OrderDate, AddressID, DeliveryDate
        FROM `DataFromAllTables`
        ORDER BY OrderDate DESC
        LIMIT 5;
    END;"""

#Create the Stored Procedure
try:
    cursor.execute(ManageBooking_sql)
    print("Procedure successfully created or already exists.")
except Error as er:
    print("Something went wrong.")
    print("Error Code:", er.errno)
    print("Error Message:", er.msg)
```

```
Procedure successfully created or already exists.
```

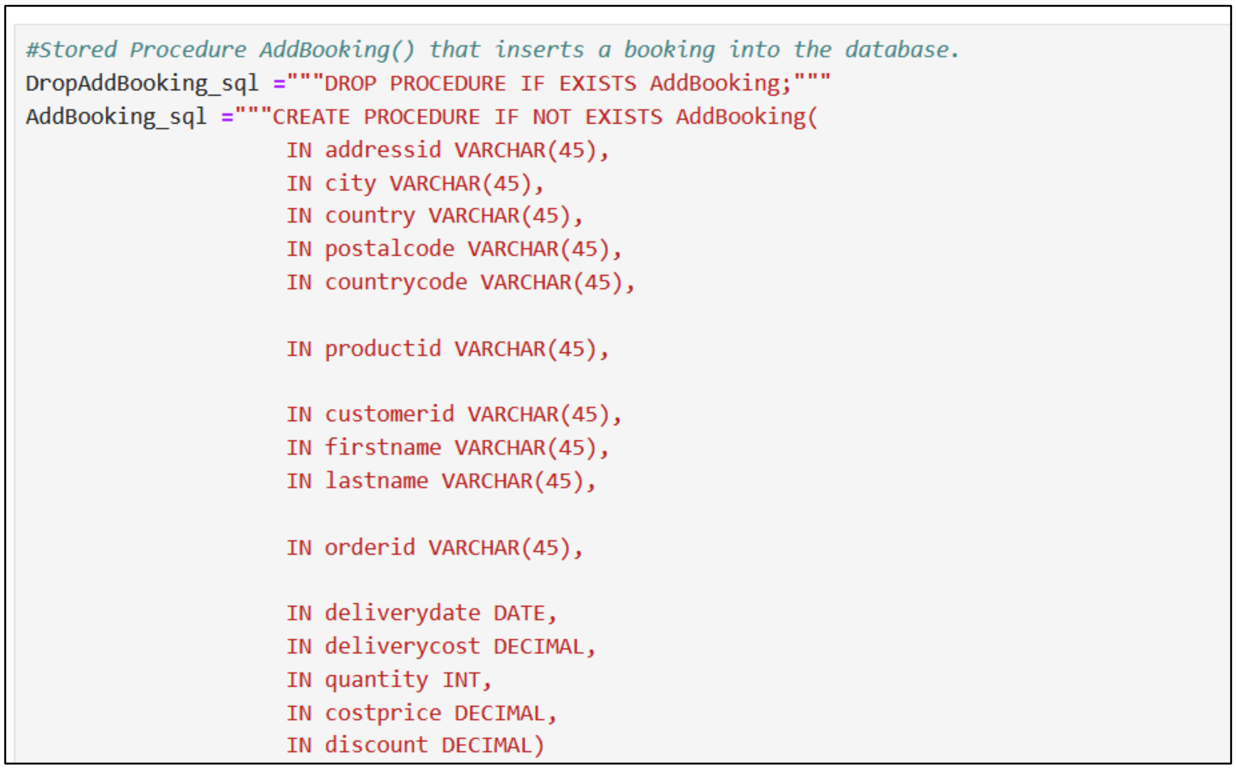


```
#Retrieve data from the Stored Procedure
try:
    cursor.callproc('ManageBooking')
    allResults=[]
    for eachResult in cursor.stored_results():
        ds = eachResult.fetchall()
        i=1
        for eachResult in ds:
            print('Result #', i, "- OrderID: ",eachResult[0],"- CustomerID: ",eachResult[1],
                                 "- OrderDate: ",eachResult[2],"- AddressID: ",eachResult[3],"- DeliveryDate: ",eachResult[4])
            i=i+1
except Error as er:
    print("Something went wrong.")
    print("Error Code:", er.errno)
    print("Error Message:", er.msg)
```

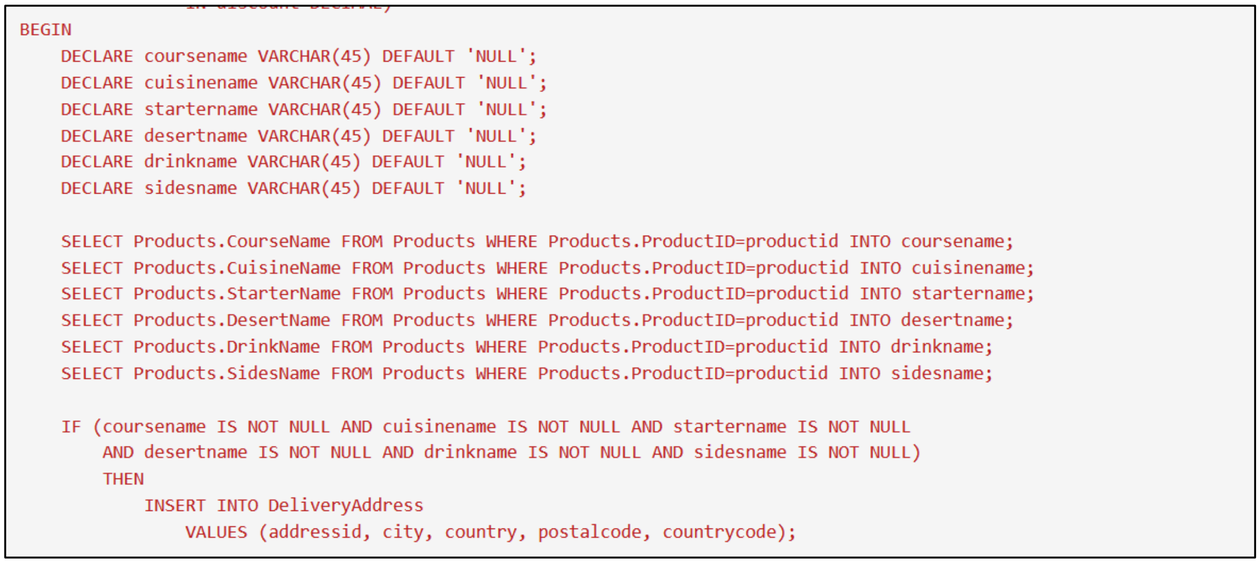
```
Result # 1 - OrderID:  52-708-9153 - CustomerID:  91-757-8892 - OrderDate:  2023-01-06 - AddressID:  KANSASCITY-UNI-353748-1 - Deli
Result # 2 - OrderID:  52-708-9153 - CustomerID:  91-757-8892 - OrderDate:  2023-01-06 - AddressID:  KANSASCITY-UNI-9870562-1 - Del
Result # 3 - OrderID:  52-708-9153 - CustomerID:  91-757-8892 - OrderDate:  2023-01-06 - AddressID:  KANSASCITY-UNI-6496418-1 - Del
Result # 4 - OrderID:  52-708-9153 - CustomerID:  91-757-8892 - OrderDate:  2023-01-06 - AddressID:  KANSASCITY-UNI-165945590-1 - D
Result # 5 - OrderID:  52-708-9153 - CustomerID:  91-757-8892 - OrderDate:  2023-01-06 - AddressID:  KANSASCITY-UNI-9930300-1 - Del
```

Step 7:

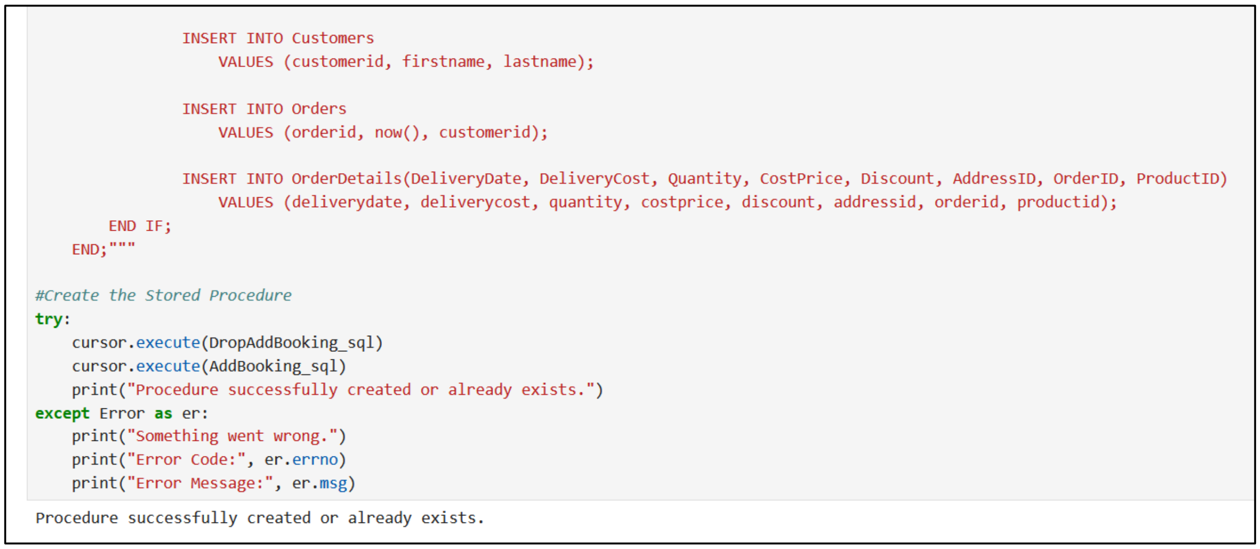
Create a stored procedure that inserts a booking into the database.



```
#Stored Procedure AddBooking() that inserts a booking into the database.
DropAddBooking_sql ="""DROP PROCEDURE IF EXISTS AddBooking;"""
AddBooking_sql ="""CREATE PROCEDURE IF NOT EXISTS AddBooking(
                    IN addressid VARCHAR(45),
                    IN city VARCHAR(45),
                    IN country VARCHAR(45),
                    IN postalcode VARCHAR(45),
                    IN countrycode VARCHAR(45),

                    IN productid VARCHAR(45),

                    IN customerid VARCHAR(45),
                    IN firstname VARCHAR(45),
                    IN lastname VARCHAR(45),

                    IN orderid VARCHAR(45),

                    IN deliverydate DATE,
                    IN deliverycost DECIMAL,
                    IN quantity INT,
                    IN costprice DECIMAL,
                    IN discount DECIMAL)
```

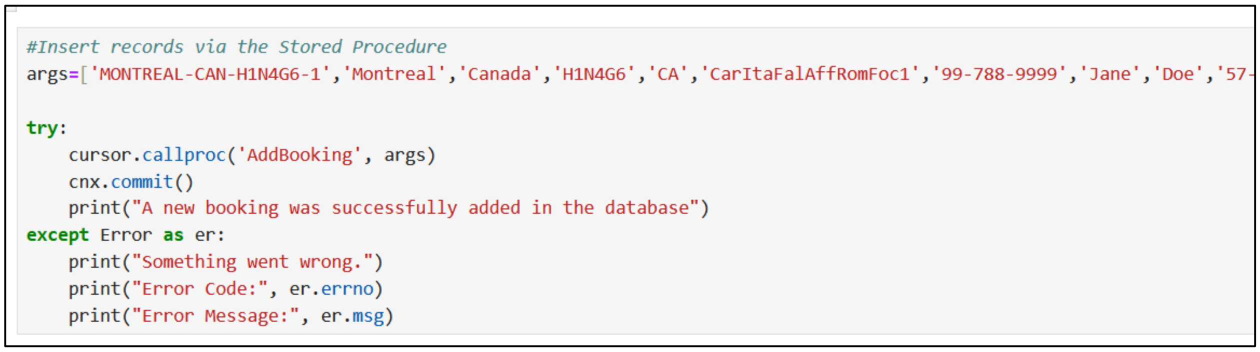


```
BEGIN
    DECLARE coursename VARCHAR(45) DEFAULT 'NULL';
    DECLARE cuisinename VARCHAR(45) DEFAULT 'NULL';
    DECLARE startername VARCHAR(45) DEFAULT 'NULL';
    DECLARE desertname VARCHAR(45) DEFAULT 'NULL';
    DECLARE drinkname VARCHAR(45) DEFAULT 'NULL';
    DECLARE sidesname VARCHAR(45) DEFAULT 'NULL';

    SELECT Products.CourseName FROM Products WHERE Products.ProductID=productid INTO coursename;
    SELECT Products.CuisineName FROM Products WHERE Products.ProductID=productid INTO cuisinename;
    SELECT Products.StarterName FROM Products WHERE Products.ProductID=productid INTO startername;
    SELECT Products.DesertName FROM Products WHERE Products.ProductID=productid INTO desertname;
    SELECT Products.DrinkName FROM Products WHERE Products.ProductID=productid INTO drinkname;
    SELECT Products.SidesName FROM Products WHERE Products.ProductID=productid INTO sidesname;

    IF (coursename IS NOT NULL AND cuisinename IS NOT NULL AND startername IS NOT NULL
        AND desertname IS NOT NULL AND drinkname IS NOT NULL AND sidesname IS NOT NULL)
        THEN
            INSERT INTO DeliveryAddress
                VALUES (addressid, city, country, postalcode, countrycode);
```



```
                INSERT INTO Customers
                    VALUES (customerid, firstname, lastname);

                INSERT INTO Orders
                    VALUES (orderid, now(), customerid);

                INSERT INTO OrderDetails(DeliveryDate, DeliveryCost, Quantity, CostPrice, Discount, AddressID, OrderID, ProductID)
                    VALUES (deliverydate, deliverycost, quantity, costprice, discount, addressid, orderid, productid);
        END IF;
    END;"""

#Create the Stored Procedure
try:
    cursor.execute(DropAddBooking_sql)
    cursor.execute(AddBooking_sql)
    print("Procedure successfully created or already exists.")
except Error as er:
    print("Something went wrong.")
    print("Error Code:", er.errno)
    print("Error Message:", er.msg)
```

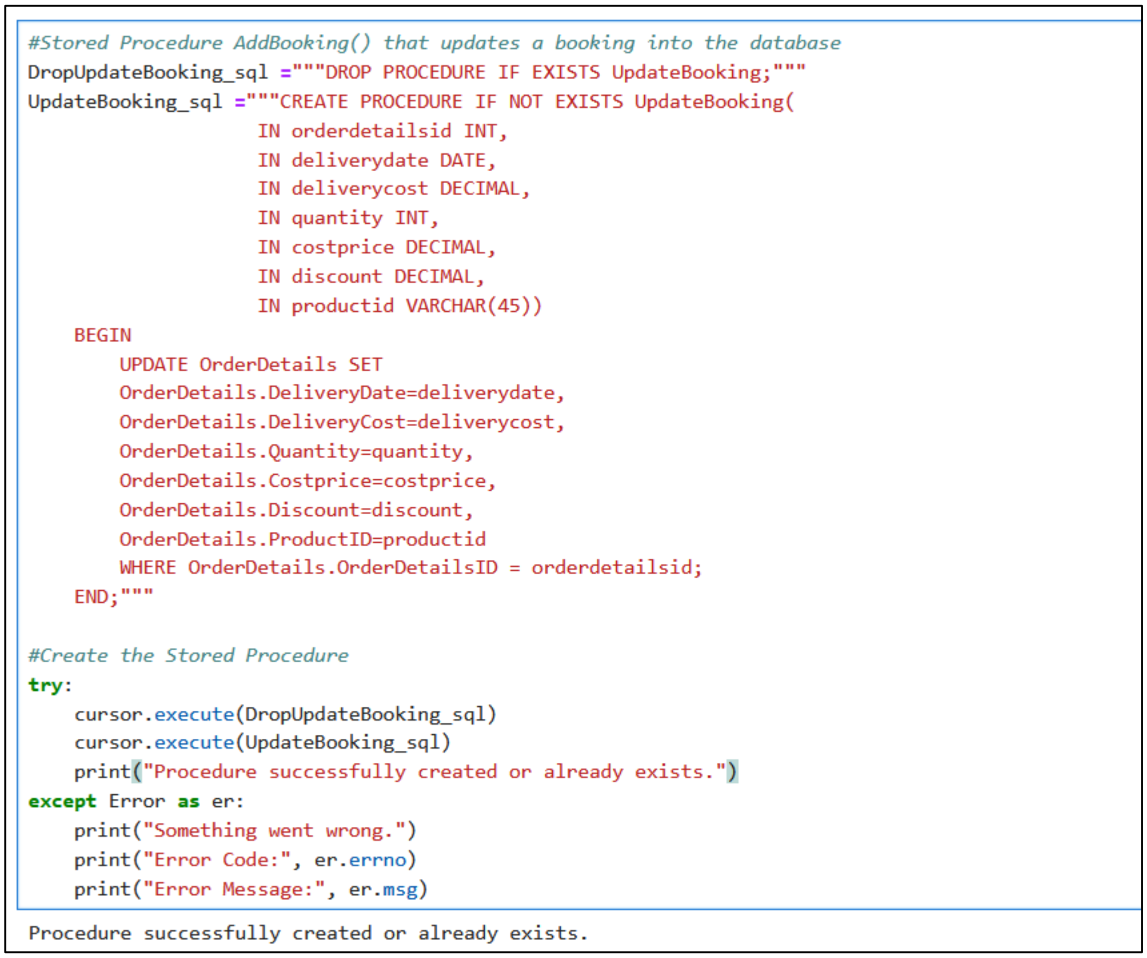
```
Procedure successfully created or already exists.
```



```
#Insert records via the Stored Procedure
args=['MONTREAL-CAN-H1N4G6-1','Montreal','Canada','H1N4G6','CA','CarItaFalAffRomFoc1','99-788-9999','Jane','Doe','57-

try:
    cursor.callproc('AddBooking', args)
    cnx.commit()
    print("A new booking was successfully added in the database")
except Error as er:
    print("Something went wrong.")
    print("Error Code:", er.errno)
    print("Error Message:", er.msg)
```

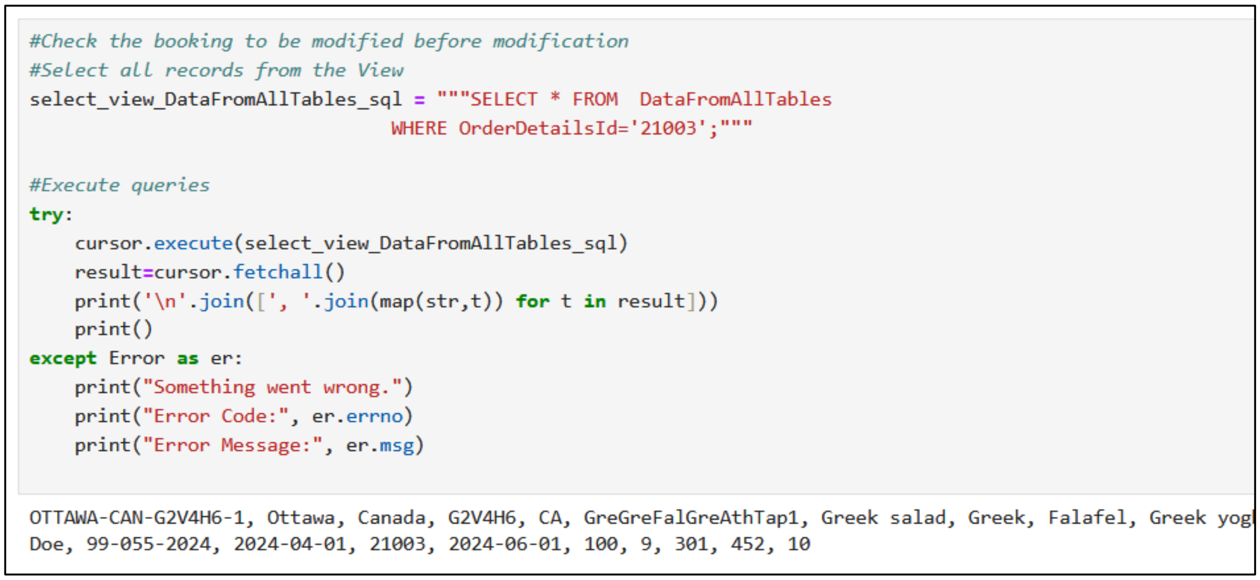
Step 8:

Create a stored procedure that updates a booking into the database.



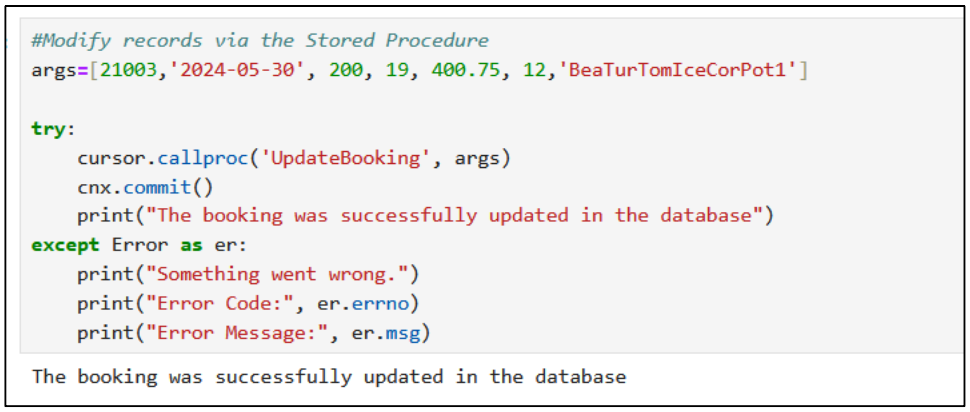
```
#Stored Procedure AddBooking() that updates a booking into the database
DropUpdateBooking_sql ="""DROP PROCEDURE IF EXISTS UpdateBooking;"""
UpdateBooking_sql ="""CREATE PROCEDURE IF NOT EXISTS UpdateBooking(
                    IN orderdetailsid INT,
                    IN deliverydate DATE,
                    IN deliverycost DECIMAL,
                    IN quantity INT,
                    IN costprice DECIMAL,
                    IN discount DECIMAL,
                    IN productid VARCHAR(45))
    BEGIN
        UPDATE OrderDetails SET
        OrderDetails.DeliveryDate=deliverydate,
        OrderDetails.DeliveryCost=deliverycost,
        OrderDetails.Quantity=quantity,
        OrderDetails.Costprice=costprice,
        OrderDetails.Discount=discount,
        OrderDetails.ProductID=productid
        WHERE OrderDetails.OrderDetailsID = orderdetailsid;
    END;"""

#Create the Stored Procedure
try:
    cursor.execute(DropUpdateBooking_sql)
    cursor.execute(UpdateBooking_sql)
    print("Procedure successfully created or already exists.")
except Error as er:
    print("Something went wrong.")
    print("Error Code:", er.errno)
    print("Error Message:", er.msg)
```

```
Procedure successfully created or already exists.
```



```
#Check the booking to be modified before modification
#Select all records from the View
select_view_DataFromAllTables_sql = """SELECT * FROM  DataFromAllTables
                                WHERE OrderDetailsId='21003';"""

#Execute queries
try:
    cursor.execute(select_view_DataFromAllTables_sql)
    result=cursor.fetchall()
    print('\n'.join([', '.join(map(str,t)) for t in result]))
    print()
except Error as er:
    print("Something went wrong.")
    print("Error Code:", er.errno)
    print("Error Message:", er.msg)
```

```
OTTAWA-CAN-G2V4H6-1, Ottawa, Canada, G2V4H6, CA, GreGreFalGreAthTap1, Greek salad, Greek, Falafel, Greek yog
Doe, 99-055-2024, 2024-04-01, 21003, 2024-06-01, 100, 9, 301, 452, 10
```

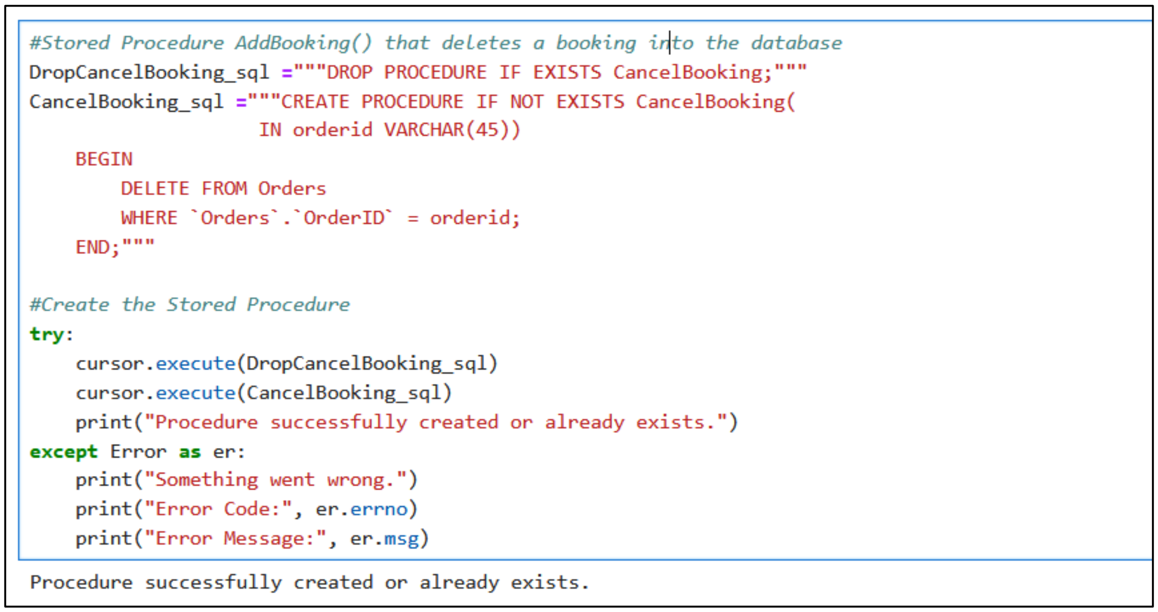


```
#Modify records via the Stored Procedure
args=[21003,'2024-05-30', 200, 19, 400.75, 12,'BeaTurTomIceCorPot1']

try:
    cursor.callproc('UpdateBooking', args)
    cnx.commit()
    print("The booking was successfully updated in the database")
except Error as er:
    print("Something went wrong.")
    print("Error Code:", er.errno)
    print("Error Message:", er.msg)
```

```
The booking was successfully updated in the database
```

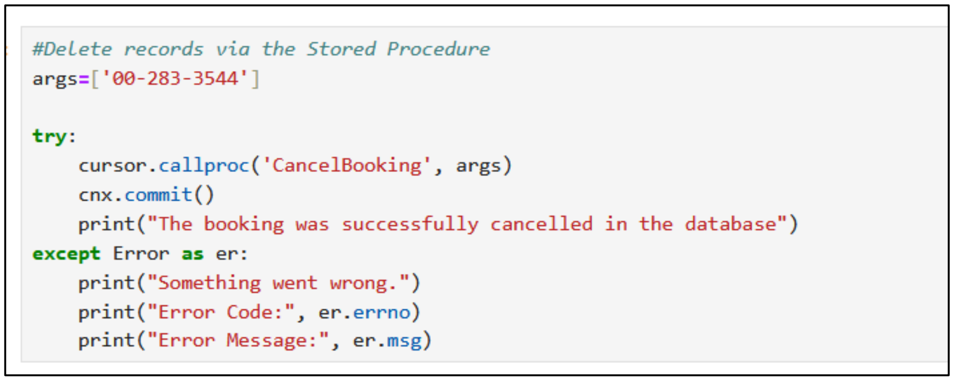
Step 9:

Create a stored procedure that deletes a booking into the database.



```
#Stored Procedure AddBooking() that deletes a booking into the database
DropCancelBooking_sql ="""DROP PROCEDURE IF EXISTS CancelBooking;"""
CancelBooking_sql ="""CREATE PROCEDURE IF NOT EXISTS CancelBooking(
                    IN orderid VARCHAR(45))
    BEGIN
        DELETE FROM Orders
        WHERE `Orders`.`OrderID` = orderid;
    END;"""

#Create the Stored Procedure
try:
    cursor.execute(DropCancelBooking_sql)
    cursor.execute(CancelBooking_sql)
    print("Procedure successfully created or already exists.")
except Error as er:
    print("Something went wrong.")
    print("Error Code:", er.errno)
    print("Error Message:", er.msg)
```

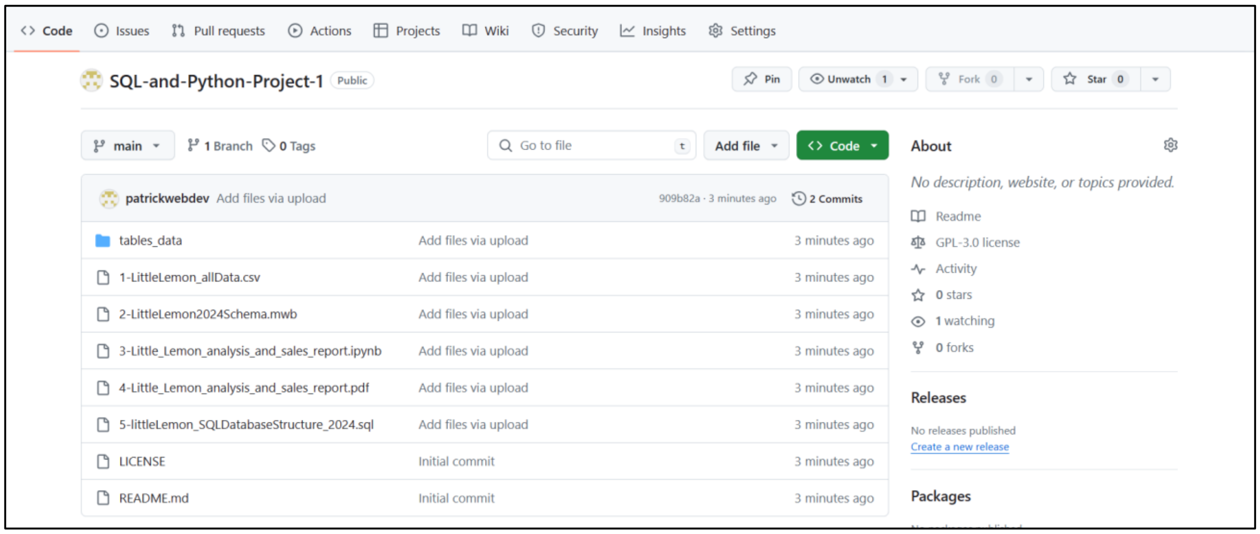
```
Procedure successfully created or already exists.
```



```
#Delete records via the Stored Procedure
args=['00-283-3544']

try:
    cursor.callproc('CancelBooking', args)
    cnx.commit()
    print("The booking was successfully cancelled in the database")
except Error as er:
    print("Something went wrong.")
    print("Error Code:", er.errno)
    print("Error Message:", er.msg)
```

Step 10:
Create a GitHub repository to store the project and then automatically manage the versions of the Python and SQL code (e.g., adding and removing) with Git.

Screenshots of the repository and files within it are shown below.



patrickwebdev Add files via upload — 909b82a · 3 minutes ago — 2 Commits

tables_data	Add files via upload	3 minutes ago
1-LittleLemon_allData.csv	Add files via upload	3 minutes ago
2-LittleLemon2024Schema.mwb	Add files via upload	3 minutes ago
3-Little_Lemon_analysis_and_sales_report.ipynb	Add files via upload	3 minutes ago
4-Little_Lemon_analysis_and_sales_report.pdf	Add files via upload	3 minutes ago
5-littleLemon_SQLDatabaseStructure_2024.sql	Add files via upload	3 minutes ago
LICENSE	Initial commit	3 minutes ago
README.md	Initial commit	3 minutes ago

About

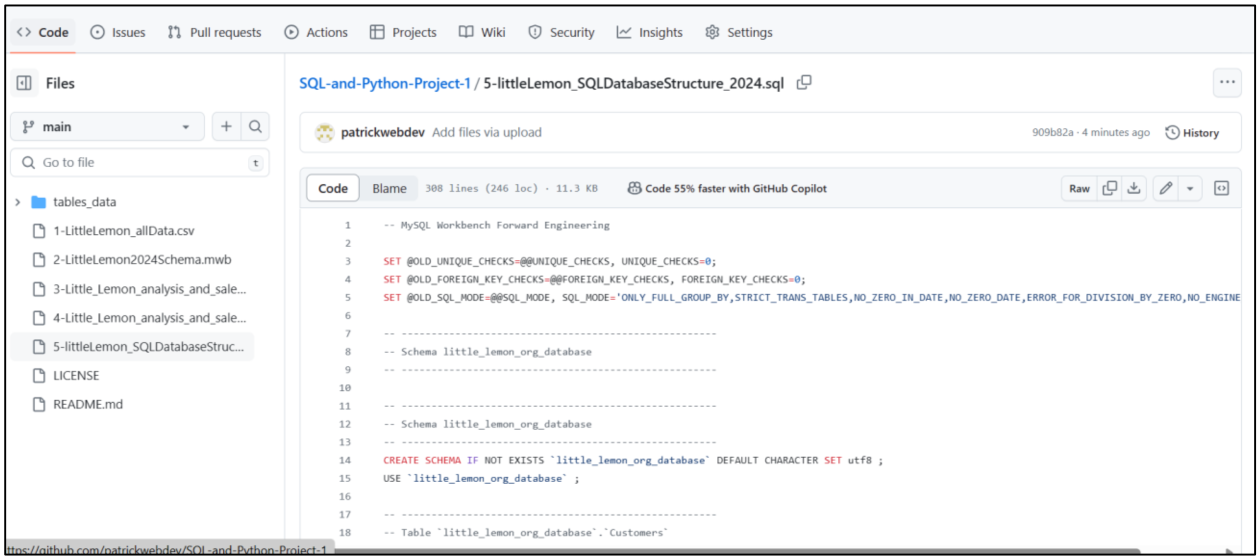
No description, website, or topics provided.

- Readme
- GPL-3.0 license
- Activity
- 0 stars
- 1 watching
- 0 forks

Releases

No releases published
Create a new release

Packages



- tables_data
- 1-LittleLemon_allData.csv
- 2-LittleLemon2024Schema.mwb
- 3-Little_Lemon_analysis_and_sale...
- 4-Little_Lemon_analysis_and_sale...
- 5-littleLemon_SQLDatabaseStruc...
- LICENSE
- README.md

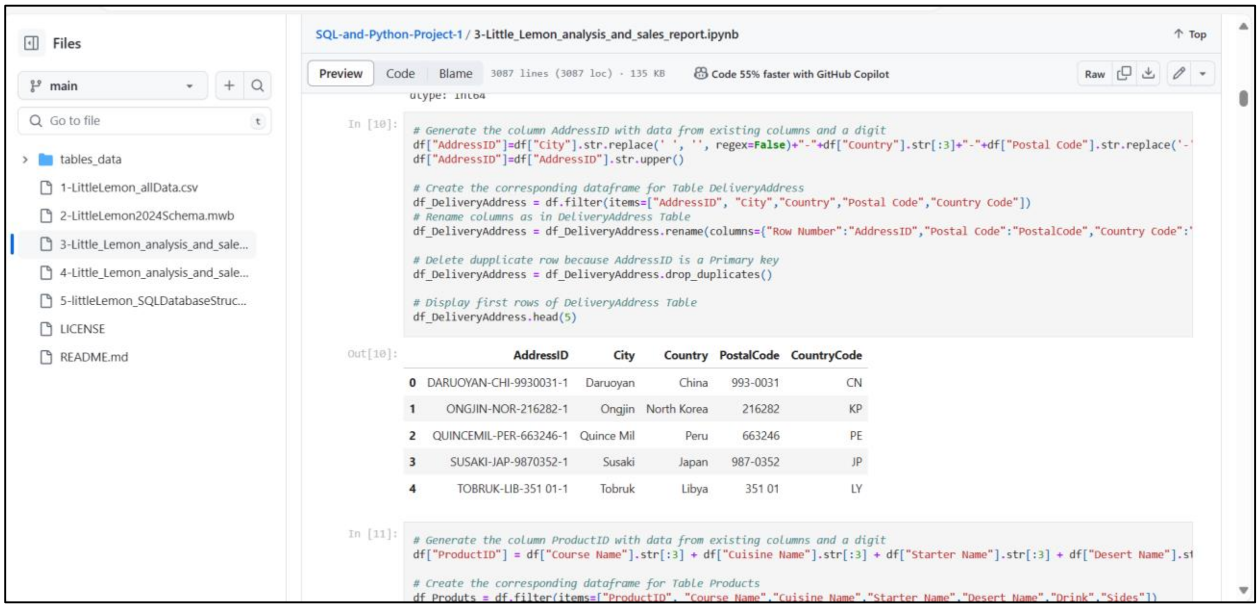
SQL-and-Python-Project-1 / 5-littleLemon_SQLDatabaseStructure_2024.sql

patrickwebdev Add files via upload — 909b82a · 4 minutes ago — History

Code Blame 308 lines (246 loc) · 11.3 KB — Code 55% faster with GitHub Copilot — Raw

```sql
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE

-- -----------------------------------------------------
-- Schema little_lemon_org_database
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema little_lemon_org_database
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `little_lemon_org_database` DEFAULT CHARACTER SET utf8 ;
USE `little_lemon_org_database` ;

-- -----------------------------------------------------
-- Table `little_lemon_org_database`.`Customers`
```

ttps://github.com/patrickwebdev/SQL-and-Python-Project-1


Files
main
Go to file
tables_data
1-LittleLemon_allData.csv
2-LittleLemon2024Schema.mwb
3-Little_Lemon_analysis_and_sale...
4-Little_Lemon_analysis_and_sale...
5-littleLemon_SQLDatabaseStruc...
LICENSE
README.md
SQL-and-Python-Project-1 / 3-Little_Lemon_analysis_and_sales_report.ipynb
Top
Preview
Code
Blame
3087 lines (3087 loc) · 135 KB
Code 55% faster with GitHub Copilot
Raw

```
dtype: int64
```

In [10]:

```
# Generate the column AddressID with data from existing columns and a digit
df["AddressID"]=df["City"].str.replace(' ', '', regex=False)+"-"+df["Country"].str[:3]+"-"+df["Postal Code"].str.replace('-'
df["AddressID"]=df["AddressID"].str.upper()

# Create the corresponding dataframe for Table DeliveryAddress
df_DeliveryAddress = df.filter(items=["AddressID", "City","Country","Postal Code","Country Code"])
# Rename columns as in DeliveryAddress Table
df_DeliveryAddress = df_DeliveryAddress.rename(columns={"Row Number":"AddressID","Postal Code":"PostalCode","Country Code":'

# Delete dupplicate row because AddressID is a Primary key
df_DeliveryAddress = df_DeliveryAddress.drop_duplicates()

# Display first rows of DeliveryAddress Table
df_DeliveryAddress.head(5)
```

Out[10]:

	AddressID	City	Country	PostalCode	CountryCode
0	DARUOYAN-CHI-9930031-1	Daruoyan	China	993-0031	CN
1	ONGJIN-NOR-216282-1	Ongjin	North Korea	216282	KP
2	QUINCEMIL-PER-663246-1	Quince Mil	Peru	663246	PE
3	SUSAKI-JAP-9870352-1	Susaki	Japan	987-0352	JP
4	TOBRUK-LIB-351 01-1	Tobruk	Libya	351 01	LY

In [11]:

```
# Generate the column ProductID with data from existing columns and a digit
df["ProductID"] = df["Course Name"].str[:3] + df["Cuisine Name"].str[:3] + df["Starter Name"].str[:3] + df["Desert Name"].st

# Create the corresponding dataframe for Table Products
df_Produts = df.filter(items=["ProductID", "Course Name","Cuisine Name","Starter Name","Desert Name","Drink","Sides"])
```

List of attachments

#	Description	Name
1	.csv file including the dataset given	tables_data\LittleLemon_allData.csv
2	ERP Model created with MySQL Workbench	LittleLemon2024Schema.mwb
3	Python program using Jupyter Notebook	Little_Lemon_analysis_and_sales_report.ipynb
4	Print of the execution of the Python program using Jupyter Notebook	Little_Lemon_analysis_and_sales_report.pdf
5	Raw sql code created with Visual Studio Code	littleLemon_SQLDatabaseStructure_2024.sql