**Database Engineer Capstone Project**

# Content

This document contains the solutions to the Database Engineer Capstone Project exercises and tasks.

The name of the evidence files are after the arrow (--->) sign.

The evidence files are uploaded in Git.

Github link: <https://github.com/annareuss/db-capstone-project/tree/main>

**Below you can find the mapping between the tasks and the related evidences:**

Exercise-1: Committing the project (Task-1,2,3)

Task-1 Create a normalized ER diagram ---> LittleLemonDM.png

Task-2 Implement the Little Lemon data model ---> LittleLemonDB.sql

Task-3 Write ‘show databases’ SQL code ---> show\_databases\_result.png

Exercise-2: Create a virtual table to summarize data (Task-1,2,3)

Task-1 Create a virtual table ---> OrdersView.png

Task-2 Create a JOIN ---> join.png

Task-3 Use the ANY operator in a subquery ---> any.png

Exercise-3: Create optimized queries (Task-1,2,3)

Task-1 Create stored procedure GetMaxQuantity ---> stored\_procedure\_GetMaxQuantity.png

Task-2 Create prepared statement GetOrderDetail ---> prepared\_statement\_GetOrderDetail.png

Task-3 Create stored procedure CancelOrder ---> stored\_procedure\_CancelOrder.png

Exercise-4: Create SQL queries to check available bookings (Task-1,2,3)

Task-1 Insert into Bookings ---> insert\_into\_Bookings.png

Task-2 Create stored procedure CheckBooking ---> stored\_procedure\_CheckBooking.png

Task-3 Create stored procedure AddValidBooking ---> stored\_procedure\_AddValidBooking.png

Exercise-5: Create SQL queries to add and update bookings (Task-1,2,3)

Task-1 Create stored procedure AddBooking ---> stored\_procedure\_AddBooking.png

Task-2 Create stored procedure UpdateBooking ---> stored\_procedure\_UpdateBooking.png

Task-3 Create stored procedure CancelBooking ---> stored\_procedure\_CancelBooking.png

Exercise-6: Set up the Tableau Workspace (Task-1,2,3)

Task-1 Tableau connect and filter the source ---> Tableau\_connect\_and\_filter.png

Task-2 Tableau split columns ---> Tableau\_split\_columns.png

Task-3 Tableau calculated column ---> Tableau\_calculated\_column.png

Exercise-7: Create interactive dashboard (Task-1,2,3,4,5)

Task-1 Tableau Customer Sales bar chart ---> Tableau\_CustomerSales.png

Task-2 Tableau Profit Chart line chart ---> Tableau\_ProfitChart.png

Task-3 Tableau Sales Bubble Chart ---> Tableau\_SalesBubbleChart.png

Task-4 Tableau Cuisine Sales and Profits ---> Tableau\_CuisineSalesAndProfits.png

Task-5 Tableau Interactive dashboard ---> Tableau\_interactive\_dashboard.png

Exercise-8: Database Client Set up the client project (Task-1,2,3)

Task-1 Database Client install python ---> DatabaseClient\_install\_python.png

Task-2 Database Client install jupyter ---> DatabaseClient\_install\_jupyíter.png

Task-3 Database Client install mysql-connector ---> DatabaseClient\_install\_mysql-conn.png

Exercise-9: Database Client Add query functions (Task-1,2,3)

Task-1 Database Client connect to database ---> DatabaseClient\_connect\_to\_db.png

Task-2 Database Client show tables ---> DatabaseClient\_show\_tables.png

Task-3 Database Client query with table JOIN ---> DatabaseClient\_query\_with\_JOIN.png

---> configuring\_mysql\_connector.ipynb

**Exercise-1: Committing the project (Task-1,2,3)**

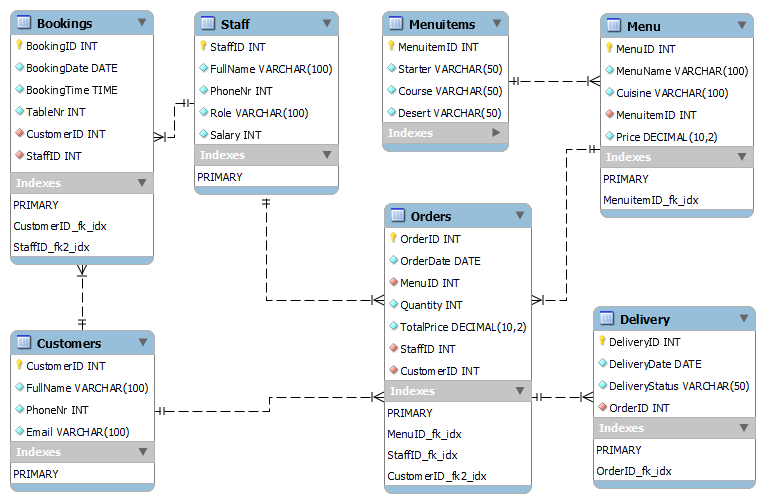
**Task-1 Create a normalized ER diagram ---> LittleLemonDM.png**

First the conceptual model, then the logical model, finally the physical model was created.

The data type and length was considered for all columns. Primary and foreign keys were designed.

The normalization process steps were performed. The final version is in 3rd normal form.

**Evidence(s):**

****

**Task-2 Implement the Little Lemon data model ---> LittleLemonDB.sql**

The data model was implemented using the forward engineer feature in MySQL Workbench.

**Evidence(s):**



**Task-3 Write ‘show databases’ SQL code ---> show\_databases\_result.png**

Create SQL statement to show the databases.

The SQL statement and the result was also captured.

The created database **littlelemon** is highlighted in the picture.

**Evidence(s):**

A screenshot of a computer

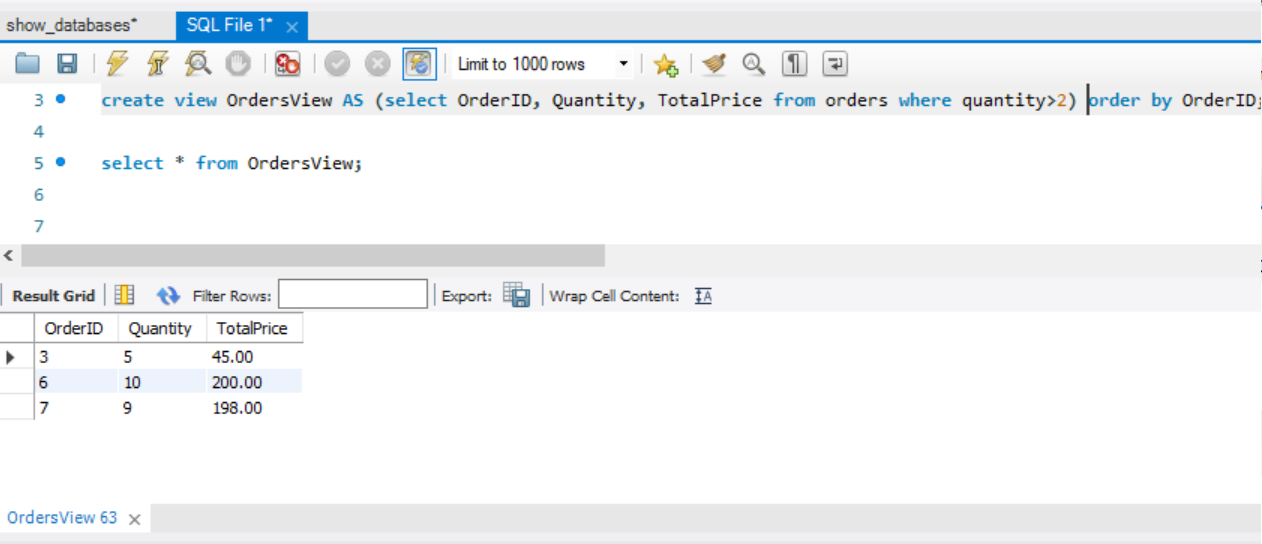
Description automatically generated

**Exercise-2: Create a virtual table to summarize data (Task-1,2,3)**

**Task-1 Create a virtual table ---> OrdersView.png**

Create a virtual table called OrdersView that focuses on OrderID, Quantity and Cost columns within the Orders table for all orders with a quantity greater than 2

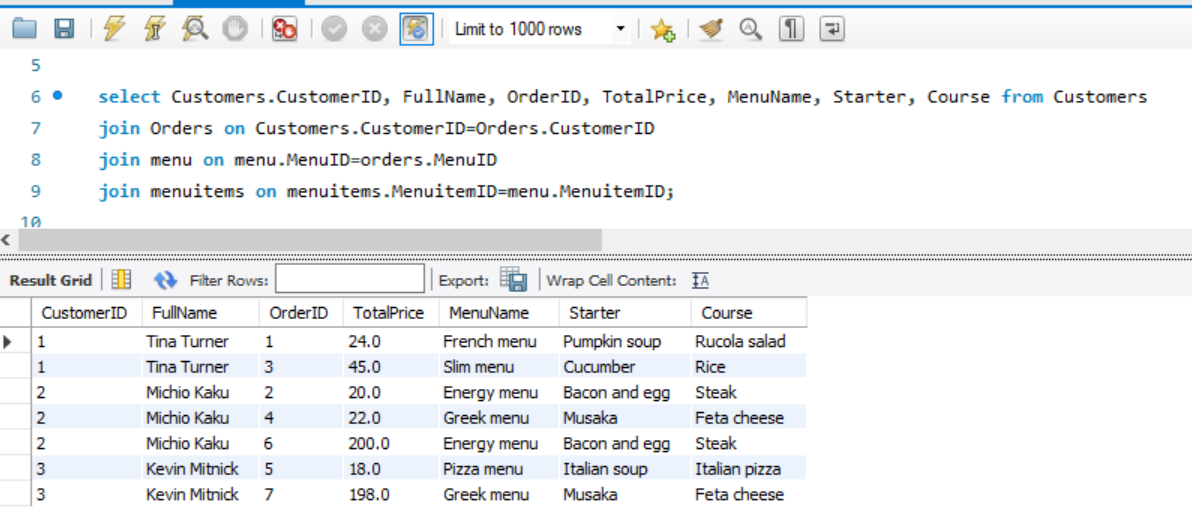
**Evidence(s):**



**Task-2 Create a JOIN ---> join.png**

Query the information from four tables on all customers with orders that cost more than $150. Extract the information by using the relevant JOIN clause.

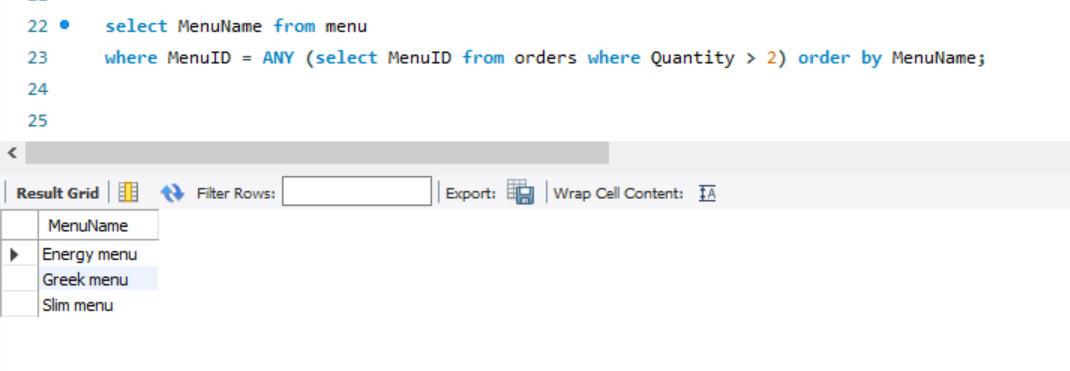
**Evidence(s):**



**Task-3 Use the ANY operator in a subquery ---> any.png**

Prepare a query in order to find all menu items for which more than 2 orders have been placed.

**Evidence(s):**

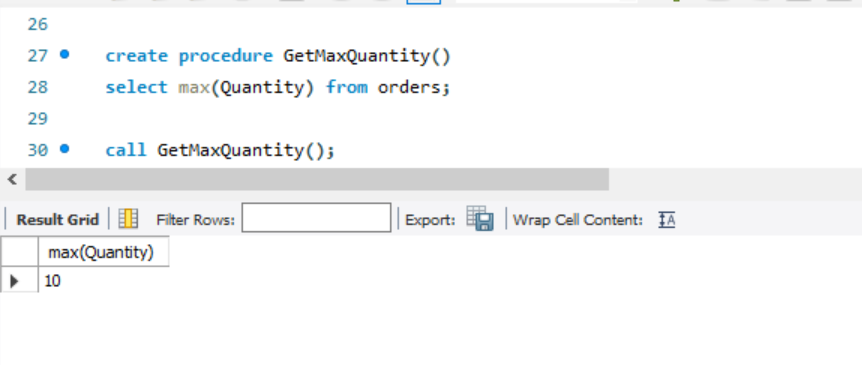


**Exercise-3: Create optimized queries (Task-1,2,3)**

**Task-1 Create stored procedure GetMaxQuantity ---> stored\_procedure\_GetMaxQuantity.png**

Create a procedure that displays the maximum ordered quantity in the Orders table. The name of the procedure should be **GetMaxQuantity**.

**Evidence(s):**



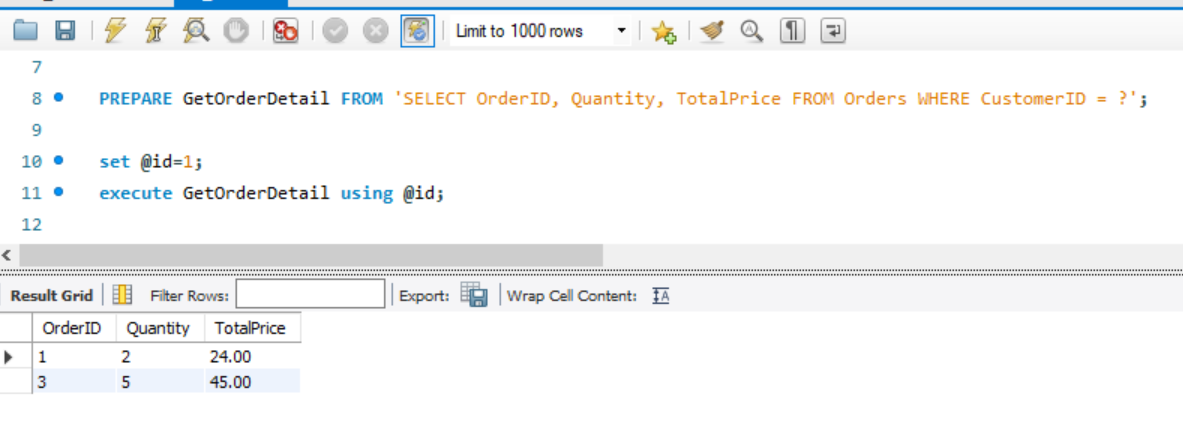
**Task-2 Create prepared statement GetOrderDetail ---> prepared\_statement\_GetOrderDetail.png**

Create a procedure that displays order information for a specified CustomerID. The name of the prepared statement should be **GetOrderDetail**.

The prepared statement should accept one input argument, the **CustomerID** value, from a variable.

The statement should return the order id, the quantity and the order cost from the Orders table.

**Evidence(s):**



**Task-3 Create stored procedure CancelOrder ---> stored\_procedure\_CancelOrder.png**

Create a stored procedure to delete an order record based on the user input of the order id.

The name of the stored procedure should be **CancelOrder**.

I created a delete trigger on the orders table, and an audit table which logs the delete trigger results.

The procedure deletes the given row, and the trigger writes the event into the audit log. Then the audit log is shown in the second step.

--------------------------------------------------------------------

CREATE TABLE Audit (

Confirmation VARCHAR(100) NOT NULL,

Date DATE NOT NULL);

CREATE TRIGGER AfterDeleteOrder AFTER DELETE ON Orders FOR EACH ROW

INSERT INTO Audit VALUES (CONCAT('Order ',OLD.OrderID,' is cancelled'),CURRENT\_DATE());

delimiter //

CREATE PROCEDURE CancelOrder(OrderIDinput INT)

begin

delete from orders where OrderID=OrderIDinput;

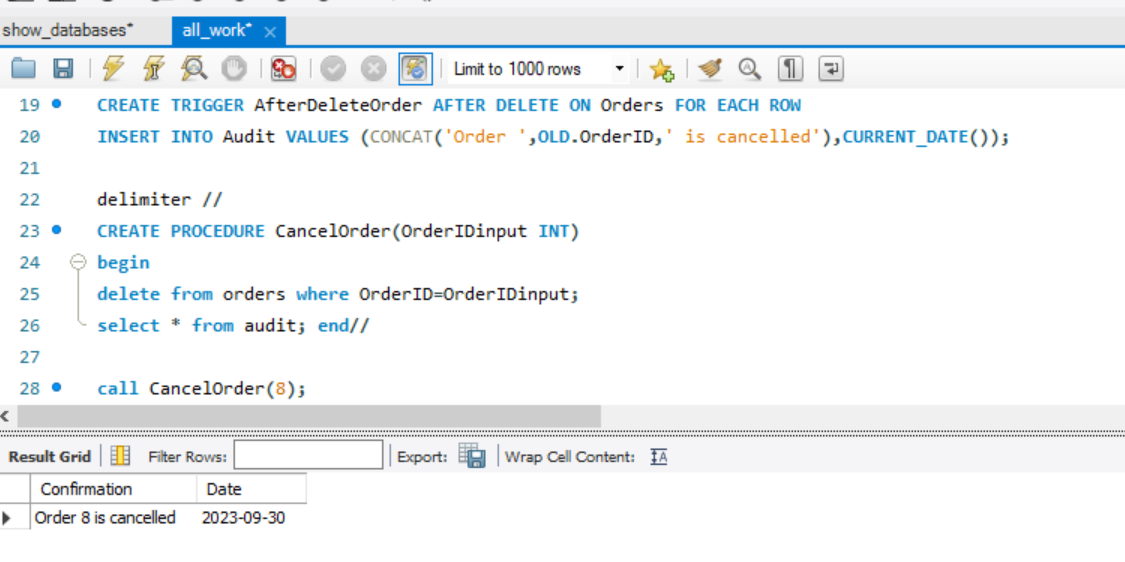
select \* from audit;

end//

call CancelOrder(8);

--------------------------------------------------------------------

**Evidence(s):**



**Exercise-4: Create SQL queries to check available bookings (Task-1,2,3)**

**Task-1 Insert into Bookings ---> insert\_into\_Bookings.png**

Create an Insert statement to add rows into the Bookings table.

--------------------------------------------------------------------

insert into bookings (BookingID, BookingDate, BookingTime, TableNr, CustomerID, StaffID) values

(0,'2022-10-10','18:00',5,1,2),

(0,'2022-11-12','19:00',3,3,2),

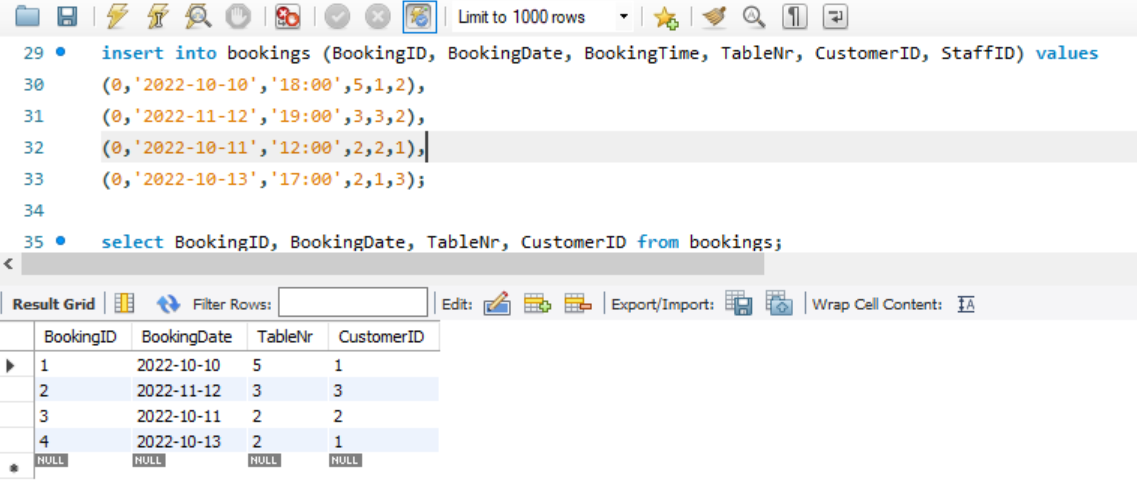
(0,'2022-10-11','12:00',2,2,1),

(0,'2022-10-13','17:00',2,1,3);

select BookingID, BookingDate, TableNr, CustomerID from bookings;

--------------------------------------------------------------------

**Evidence(s):**



**Task-2 Create stored procedure CheckBooking ---> stored\_procedure\_CheckBooking.png**

Create a stored procedure called CheckBooking to check whether a table in the restaurant is already booked.

The procedure should have two input parameters in the form of booking date and table number. You can also create a variable in the procedure to check the status of each table.

--------------------------------------------------------------------

DELIMITER //

CREATE Procedure CheckBooking(BookingDate\_input DATE, TableNr\_input INT)

BEGIN

DECLARE Booking\_status VARCHAR(80);

DECLARE TableCheck INT;

select count(\*) into TableCheck from Bookings where TableNr=TableNr\_input and BookingDate=BookingDate\_input;

IF TableCheck=1 THEN SET Booking\_status=concat('Table ',TableNr\_input,' is already booked') ;

ELSE SET Booking\_status=concat('Table ',TableNr\_input,' is free') ;

END IF;

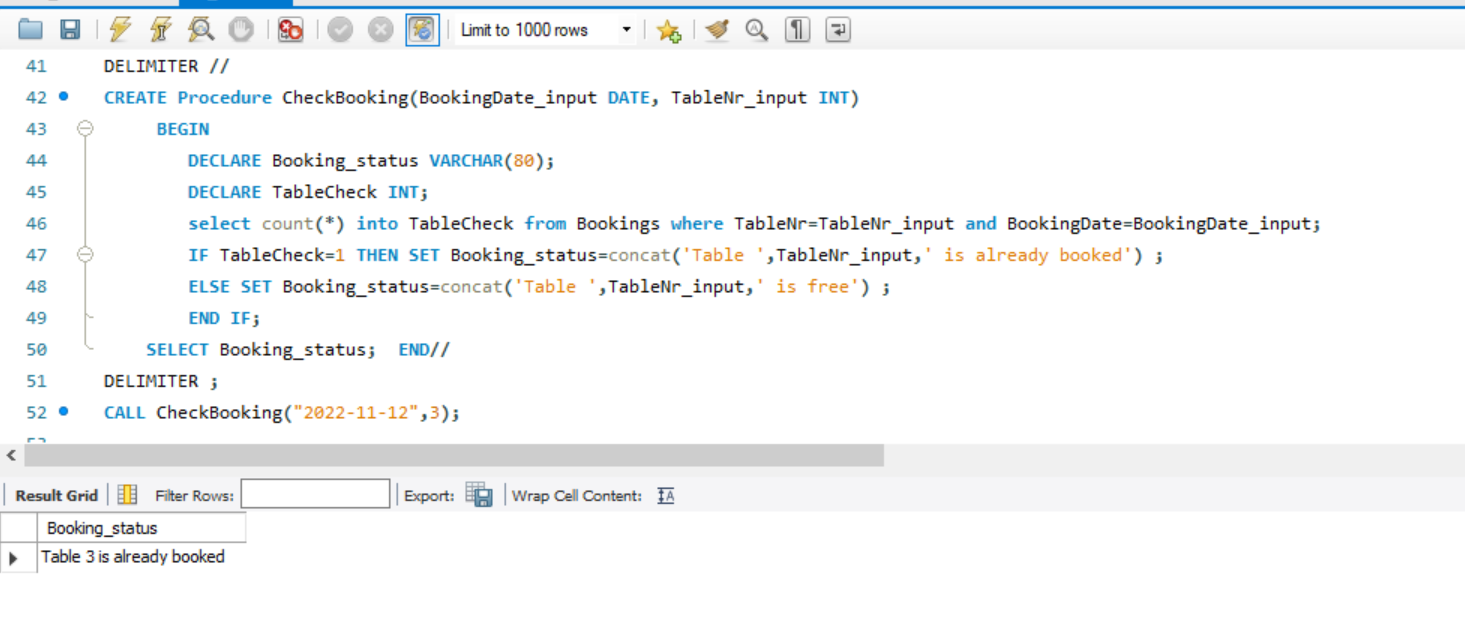
SELECT Booking\_status; END//

CALL CheckBooking("2022-11-12",3);

DELIMITER ;

--------------------------------------------------------------------

**Evidence(s):**



**Task-3 Create stored procedure AddValidBooking ---> stored\_procedure\_AddValidBooking.png**

Create a stored procedure which will need to verify a booking, and decline any reservations for tables that are already booked under another name. To implement these steps, you need to create a new procedure called **AddValidBooking**. This procedure must use a transaction statement to perform a rollback if a customer reserves a table that’s already booked under another name.

--------------------------------------------------------------------

DELIMITER //

CREATE Procedure AddValidBooking(BookingDate\_input DATE, TableNr\_input INT)

BEGIN

DECLARE Booking\_status VARCHAR(80);

DECLARE TableCheck INT;

select count(\*) into TableCheck from Bookings where TableNr=TableNr\_input and BookingDate=BookingDate\_input;

IF TableCheck=0 THEN SET Booking\_status=concat('Table ',TableNr\_input,' is successfully booked') ;

insert into bookings (BookingID, BookingDate, BookingTime, TableNr, CustomerID, StaffID)

values (0,BookingDate\_input,'18:00',TableNr\_input,1,2);

COMMIT;

ELSE SET Booking\_status=concat('Table ',TableNr\_input,' is already booked - booking cancelled') ;

ROLLBACK;

END IF;

SELECT Booking\_status;

END//

CALL AddValidBooking("2022-11-12",23);

DELIMITER ;

--------------------------------------------------------------------

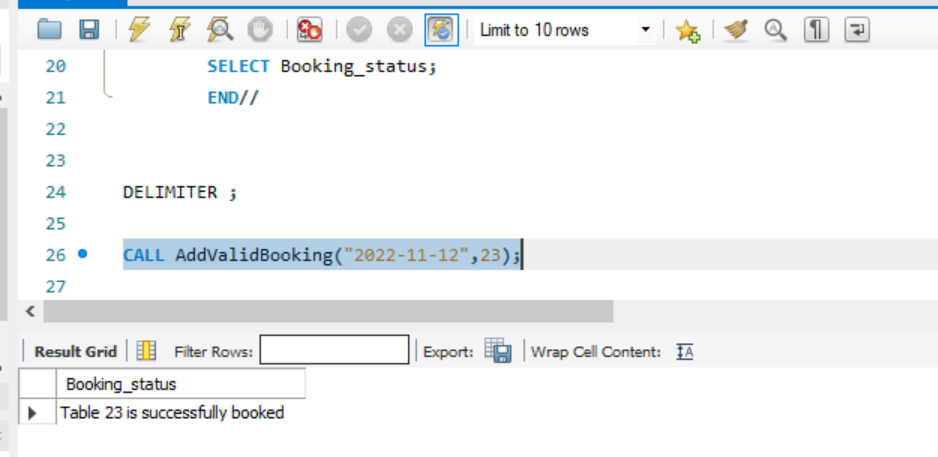
**Evidence(s):**

**Rollback:**

A screenshot of a computer

Description automatically generated

**Successful insert, the new row is committed:**



**Exercise-5: Create SQL queries to add and update bookings (Task-1,2,3)**

**Task-1 Create stored procedure AddBooking ---> stored\_procedure\_AddBooking.png**

Create a stored procedure create a new procedure called AddBooking to add a new table booking record.

--------------------------------------------------------------------

DELIMITER //

CREATE Procedure AddBooking(BookingID\_input INT,BookingDate\_input DATE, TableNr\_input INT, CustomerID\_input INT)

BEGIN

DECLARE Booking\_status VARCHAR(80);

DECLARE TableCheck INT;

DECLARE BookingID\_next INT;

select max(BookingID)+1 into BookingID\_next from Bookings;

select count(\*) into TableCheck from Bookings

where TableNr=TableNr\_input and BookingDate=BookingDate\_input and CustomerID=CustomerID\_input;

IF TableCheck=0 THEN SET Booking\_status=concat('New booking with BookingID ',BookingID\_next,' is added') ;

insert into bookings (BookingID, BookingDate, BookingTime, TableNr, CustomerID, StaffID)

values (0,BookingDate\_input,'18:00',TableNr\_input,CustomerID\_input,2);

COMMIT;

ELSE SET Booking\_status=concat('BookingID ',BookingID\_next,' is already exist - booking cancelled') ;

ROLLBACK;

END IF;

SELECT Booking\_status;

END//

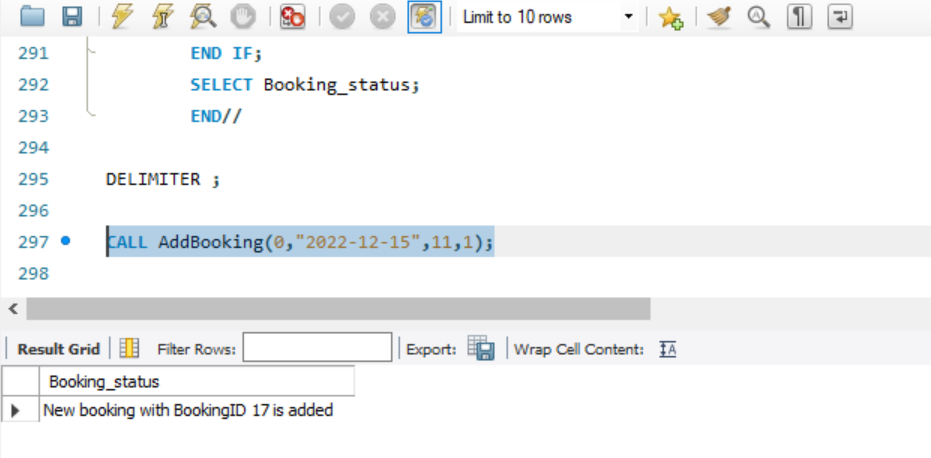
DELIMITER ;

CALL AddBooking(0,"2022-12-15",11,1);

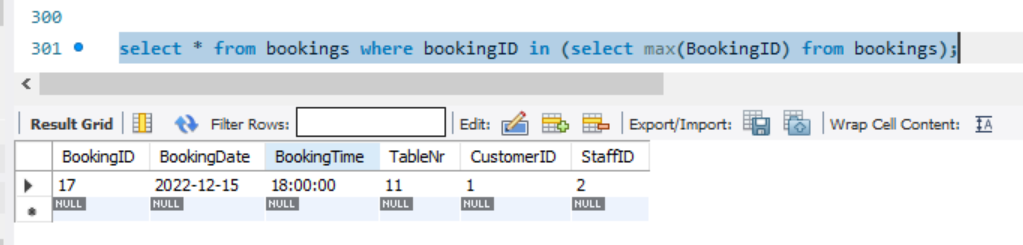
drop procedure AddBooking;

--------------------------------------------------------------------

**Evidence(s):**



**Test:**



**Task-2 Create stored procedure UpdateBooking ---> stored\_procedure\_UpdateBooking.png**

Create a new procedure called **UpdateBooking** that they can use to update existing bookings in the booking table. The procedure should have two input parameters in the form of booking id and booking date.

--------------------------------------------------------------------

DELIMITER //

CREATE Procedure UpdateBooking(BookingID\_input INT,BookingDate\_input DATE)

BEGIN

DECLARE Booking\_status VARCHAR(80);

DECLARE TableCheck INT;

update Bookings set BookingDate=BookingDate\_input where BookingID=BookingID\_input;

select count(\*) into TableCheck from Bookings

where BookingDate=BookingDate\_input and BookingID=BookingID\_input;

IF TableCheck=1 THEN SET Booking\_status=concat('BookingID ',BookingID\_input,' is updated') ;

COMMIT;

ELSE SET Booking\_status=concat('BookingID ',BookingID\_input,' update is failed') ;

ROLLBACK;

END IF;

SELECT Booking\_status;

END//

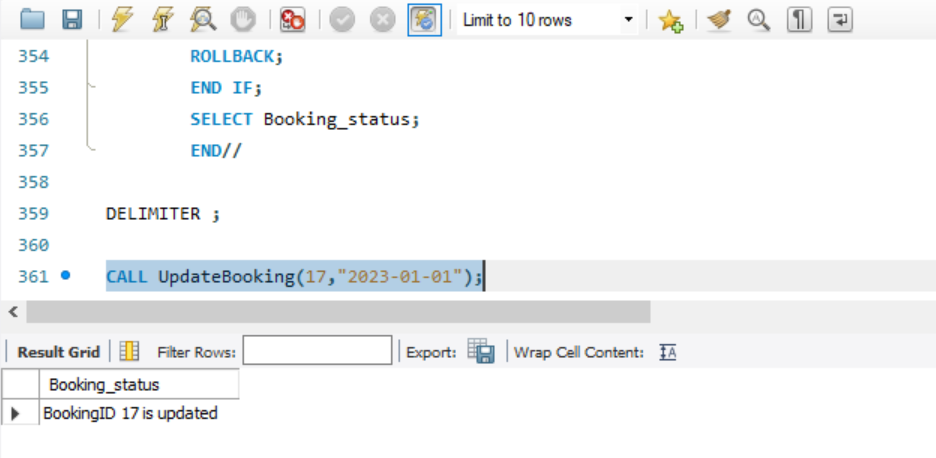
DELIMITER ;

CALL UpdateBooking(17,"2023-01-01");

drop procedure UpdateBooking;

--------------------------------------------------------------------

**Evidence(s):**

****

**Test:**

A screenshot of a computer

Description automatically generated

**Task-3 Create stored procedure CancelBooking ---> stored\_procedure\_CancelBooking.png**

Create a new procedure called **CancelBooking** that they can use to cancel or remove a booking.

The procedure should have one input parameter in the form of booking id. You must also write a DELETE statement inside the procedure.

--------------------------------------------------------------------

DELIMITER //

CREATE Procedure CancelBooking(BookingID\_input INT)

BEGIN

DECLARE Booking\_status VARCHAR(80);

DECLARE TableCheck INT;

delete from Bookings where BookingID=BookingID\_input;

select count(\*) into TableCheck from Bookings where BookingID=BookingID\_input;

IF TableCheck=0 THEN SET Booking\_status=concat('BookingID ',BookingID\_input,' cancelled') ;

COMMIT;

ELSE SET Booking\_status=concat('BookingID ',BookingID\_input,' cancellation failed') ;

ROLLBACK;

END IF;

SELECT Booking\_status;

END//

DELIMITER ;

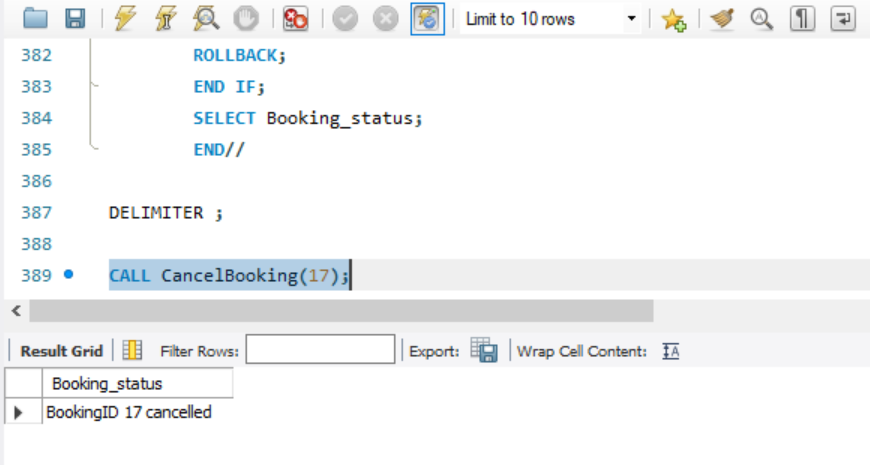
CALL CancelBooking(17);

drop procedure CancelBooking;

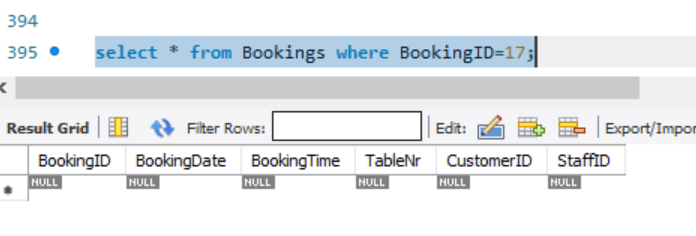
select \* from Bookings where BookingID=17;

--------------------------------------------------------------------

**Evidence(s):**



**Test:**

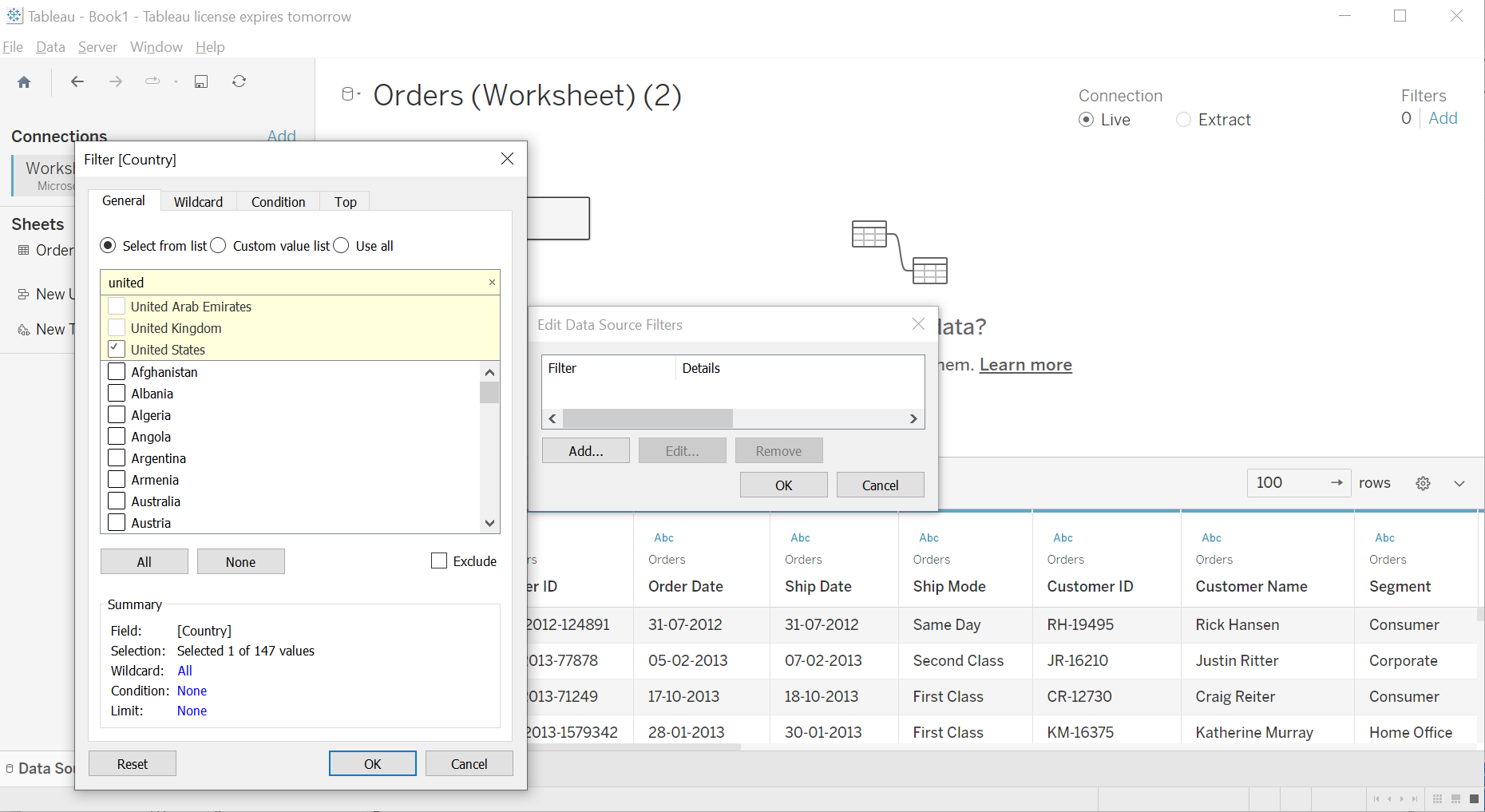


**Exercise-6: Set up the Tableau Workspace (Task-1,2,3)**

**Task-1 Tableau connect and filter the source ---> Tableau\_connect\_and\_filter.png**

Connect to Little Lemon data stored in the Excel Sheet called LittleLemonDB. Then filter data in the data source page and select the United States as the country

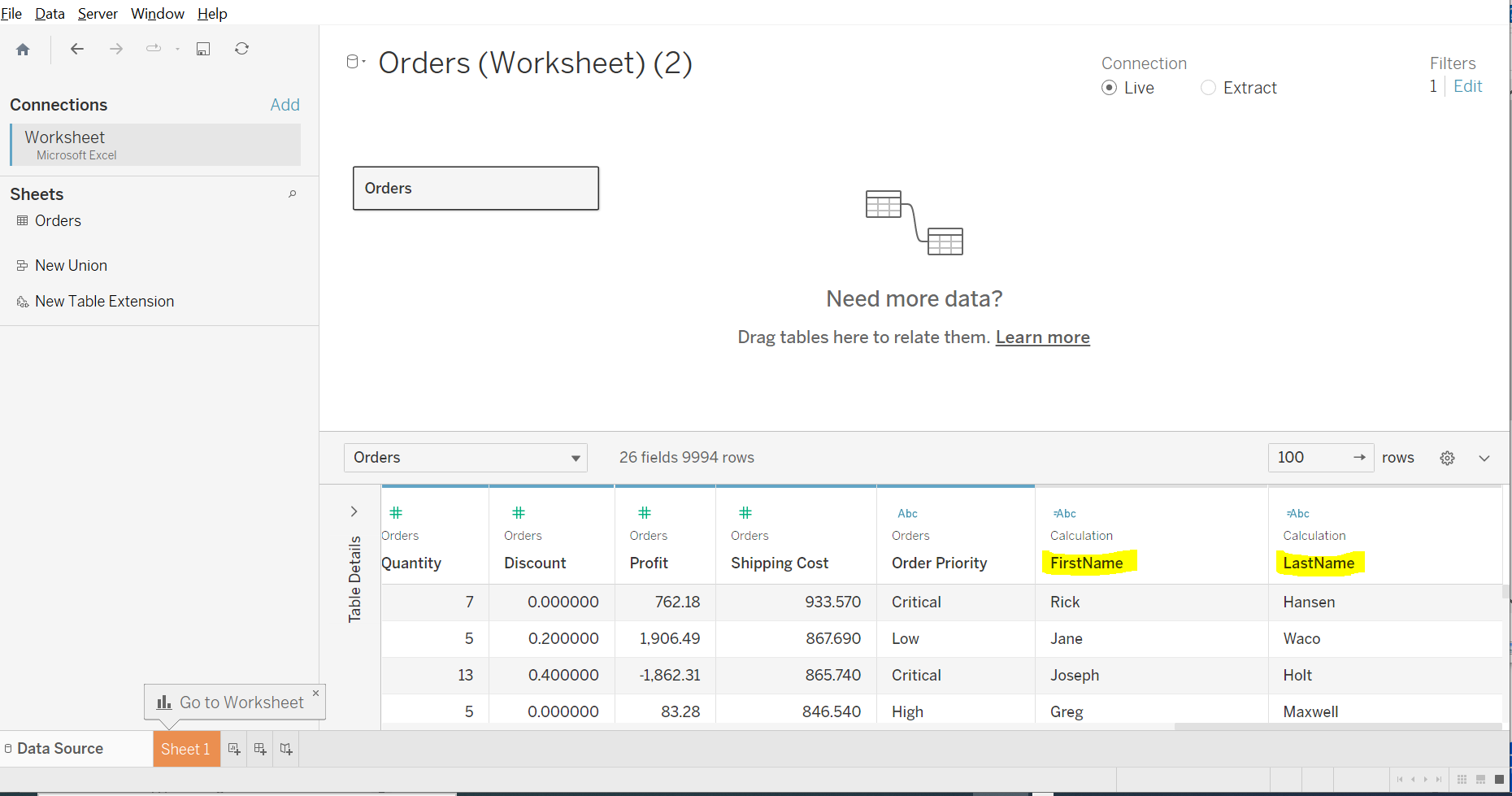
**Evidence(s):**

****

**Task-2 Tableau split columns ---> Tableau\_split\_columns.png**

Create two new data fields called First Name and Last Name. Related values should be extracted from the Full Name field.

**Evidence(s):**

****

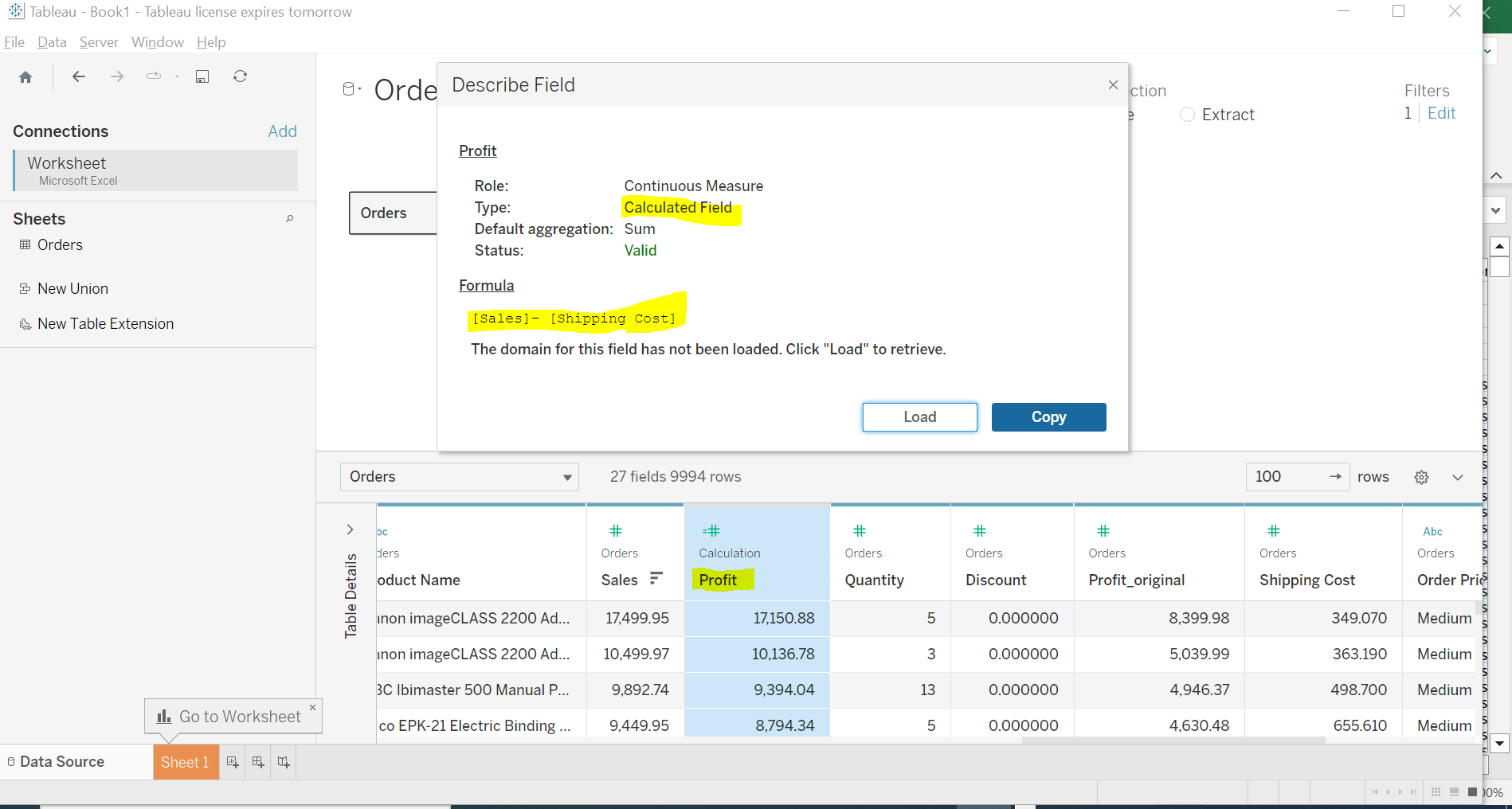
**Task-3 Tableau calculated column ---> Tableau\_calculated\_column.png**

Create a new data field that stores the profits for each sale, or order as shown in the screenshot below.

Here’s some guidance for completing this task:

1. Select Sales field in the Data Pane, then select Create Calculated field.
2. Name the calculated field Profit.
3. Write a formula that deducts Cost from Sales.

**Evidence(s):**

****

**Exercise-7: Create interactive dashboard (Task-1,2,3,4,5)**

**Task-1 Tableau Customer Sales bar chart ---> Tableau\_CustomerSales.png**

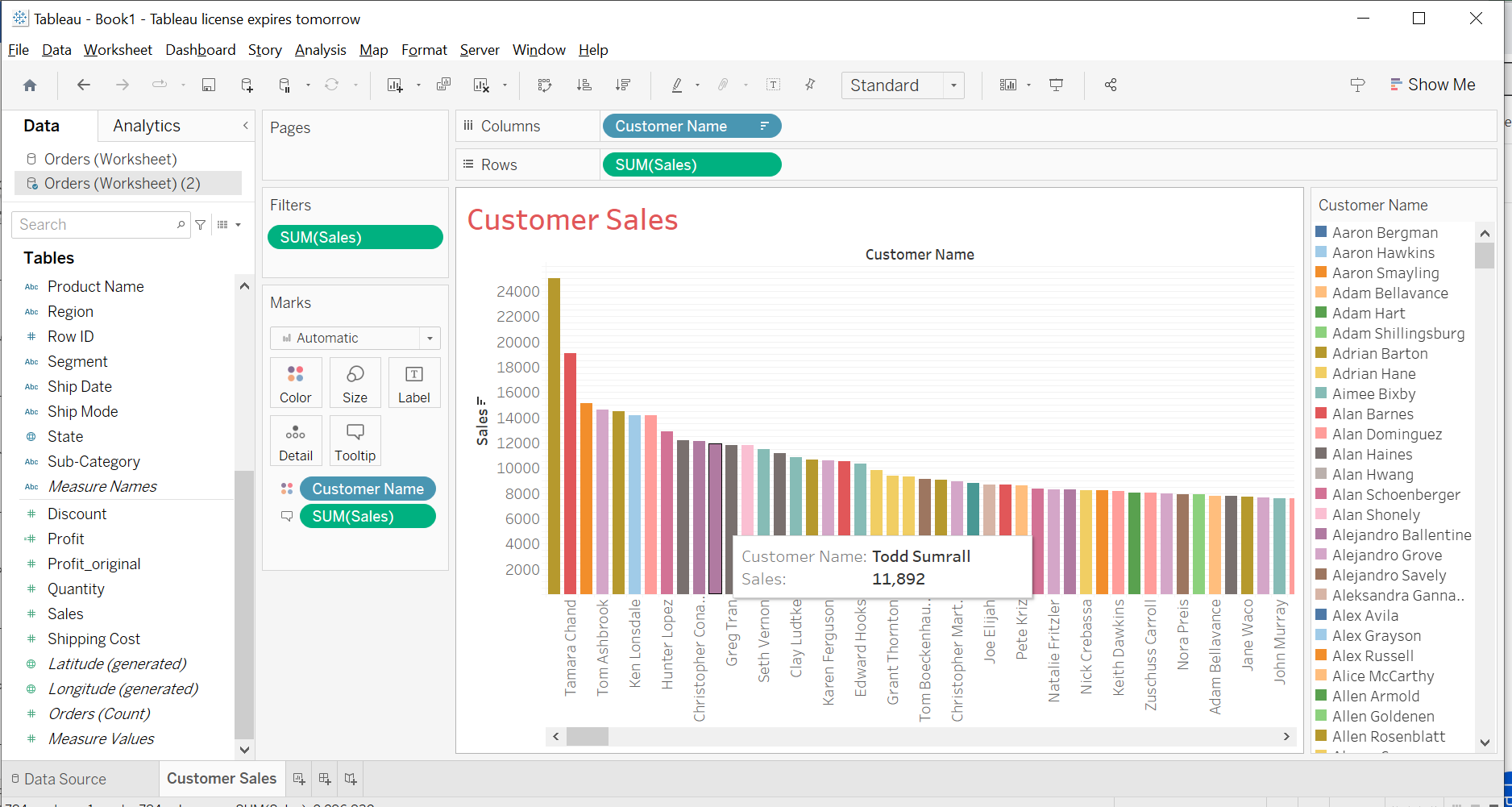
Create a bar chart that shows customers sales and filter data based on sales with at least $70.

Here’s some guidance for completing this task:

1. Drag and drop relevant fields from the data pane into the shelves section.
2. Use a suitable colour scheme.
3. Filter sales based on sales >= $70.
4. Name the chart Customers sales.

If you roll over a bar, the customer names and sale figures should be displayed.

**Evidence(s):**

****

**Task-2 Tableau Profit Chart line chart ---> Tableau\_ProfitChart.png**

Create a line chart to show the sales trend from 2019 to 2022.

Here’s some guidance for completing this task:

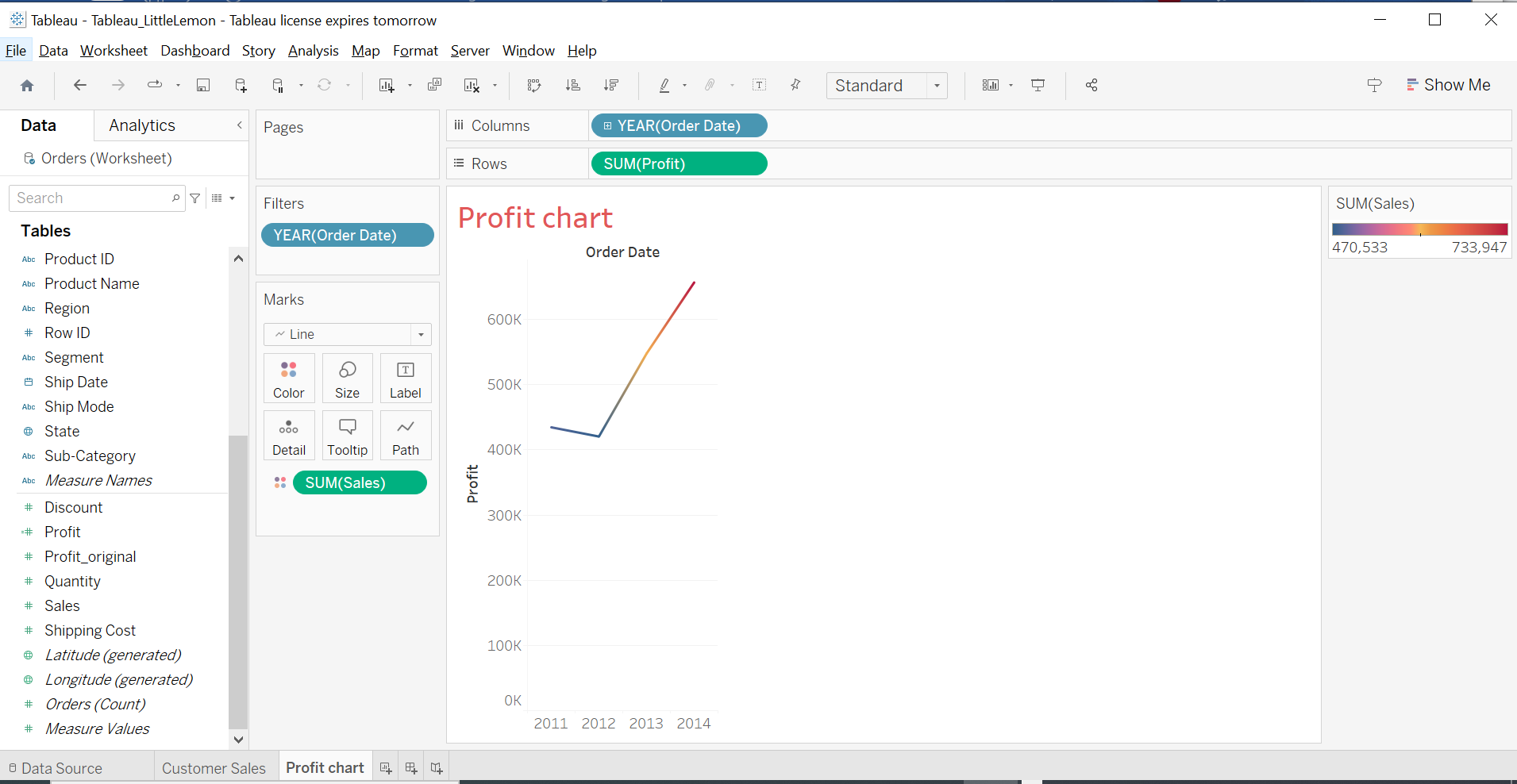
1. Drag and drop relevant fields from the data pane.
2. Use a suitable colour scheme.
3. Filter data to exclude 2023.
4. Name the chart Profit chart.

Your chart should show the trend of sales from 2019 to 2022 only.

My data is different, the source is the Worksheet.xlsx which I downloaded an earlier module within the course. It contains data between 2011 and 2014, and the Sales data is also different, this is why the chart does not look the same as in the example. I decided not to filter out did not filter out any years, because if I do so then there will be too few years in the chart. But I know how to filter out a year, here it is:

1. Click the small arrow on the year(OrderDate) in the Columns self
2. Click Edit filter, and unselect the year which you do not want to see.

**Evidence(s):**

****

**Task-3 Tableau Sales Bubble Chart ---> Tableau\_SalesBubbleChart.png**

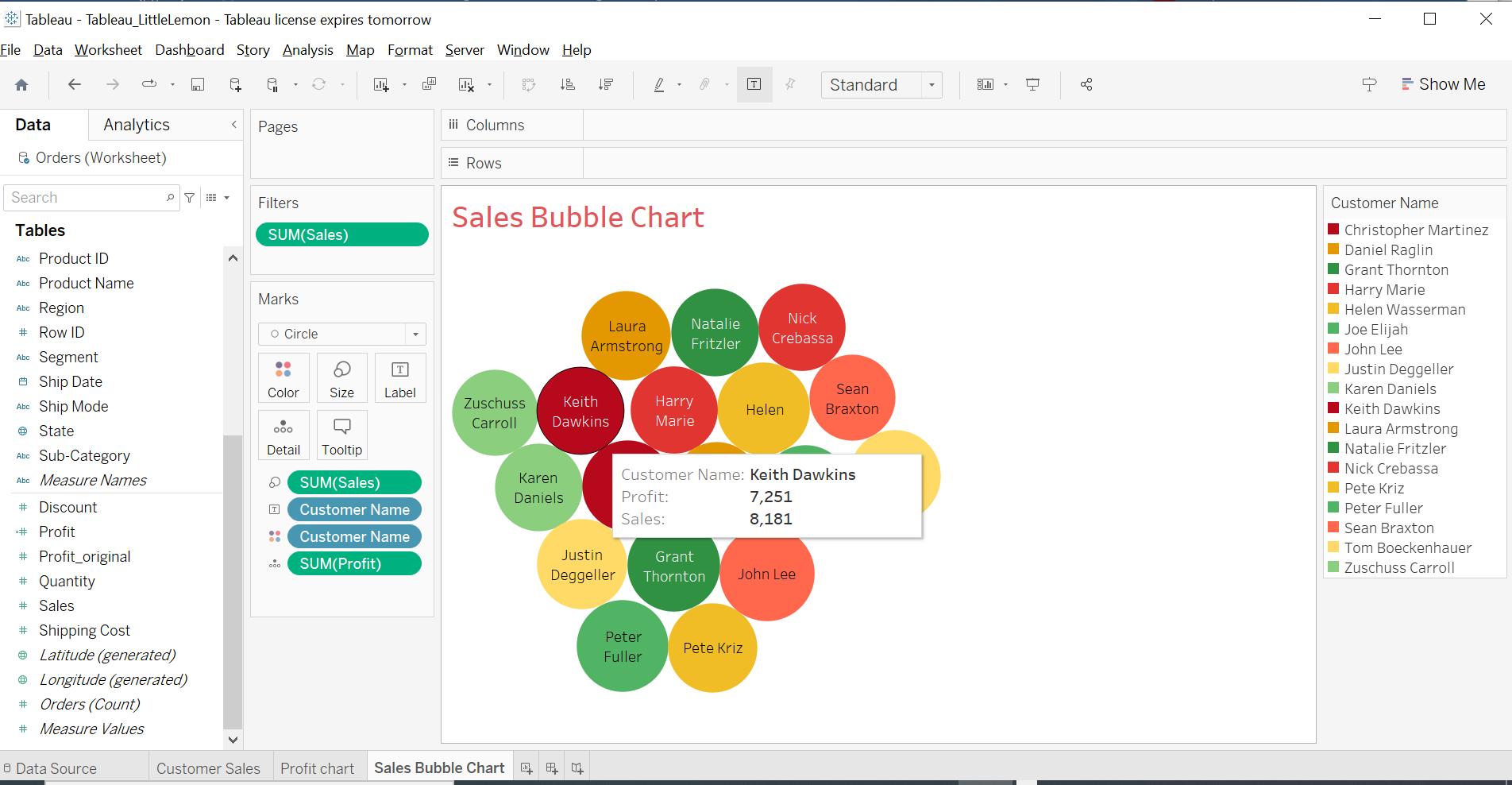
Create a Bubble chart of sales for all customers. The chart should show the names of all customers. Once you roll over a bubble, the chart should show the name, profit and sale.

Here’s some guidance for completing this task:

1. Drag and drop relevant fields from the data pane.
2. Use a suitable colour scheme.

Name the chart Sales Bubble Chart.

**Evidence(s):**

****

**Task-4 Tableau Sales and Profits ---> Tableau\_SalesAndProfits.png**

Compare the sales of the three different cuisines sold at Little Lemon. Create a Bar chart that shows the sales of the Turkish, Italian and Greek cuisines.

You need to display sales data for 2020, 2021, and 2022 only. Each bar should display the profit of each cuisine.

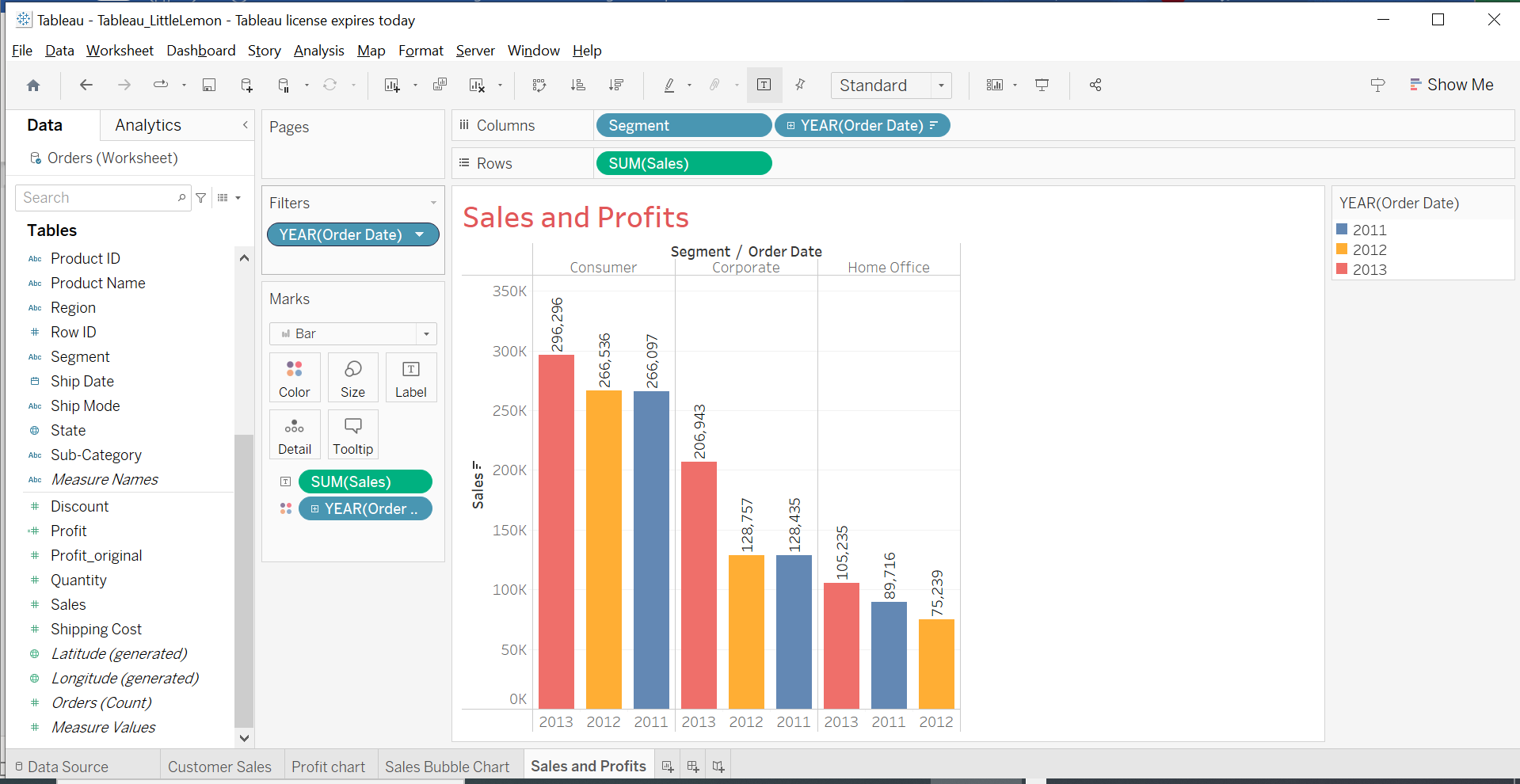
Here’s some guidance for completing this task:

1. Drag and drop relevant fields from the data pane.
2. Use a suitable color scheme.
3. Name the worksheet Cuisine Sales and Profits.
4. Sort data in descending order by the sum of the sale.

As the input is not for Little Lemon, but about another company, the task cannot be executed as required.

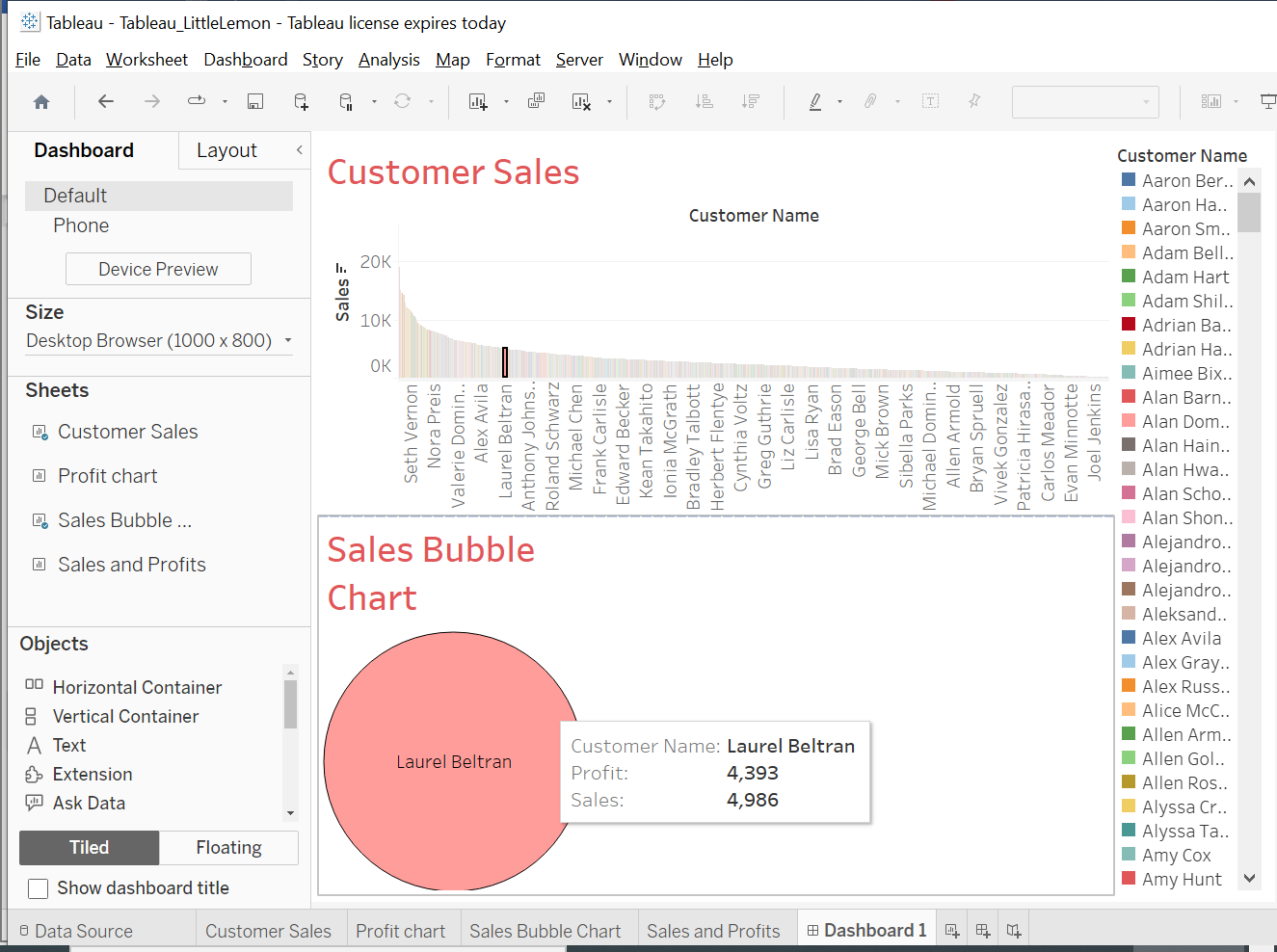
* There is no Cuisine in this dataset, so I used Segment instead.
* There is no 2020,2021 and 2022 in the dataset, so I used 2011,2021,2013.

**Evidence(s):**

****

**Task-5 Tableau Interactive dashboard ---> Tableau\_interactive\_dashboard.png**

Create an interactive dashboard that combines the Bar chart called Customers sales and the Sales Bubble Chart. Once you click a bar, and roll over the related bubble, the name, sales and profit figures should be displayed in the Bubble chart.

**Evidence(s): **

**Exercise-8: Database Client Set up the client project (Task-1,2,3)**

**Task-1 Database Client install python ---> DatabaseClient\_install\_python.png**

Check python version

Navigate to your terminal and ensure that Python is installed and available on the command path. To complete this task, type the following syntax:

--------------------------------------------------------------------

python --version

--------------------------------------------------------------------

**Evidence(s):**

A black and white screen with green text

Description automatically generated

**Task-2 Database Client install jupyter ---> DatabaseClient\_install\_jupyter.png**

Install Jupyter. You can install Jupyter using the following code:

--------------------------------------------------------------------

python -m pip install jupyter

--------------------------------------------------------------------

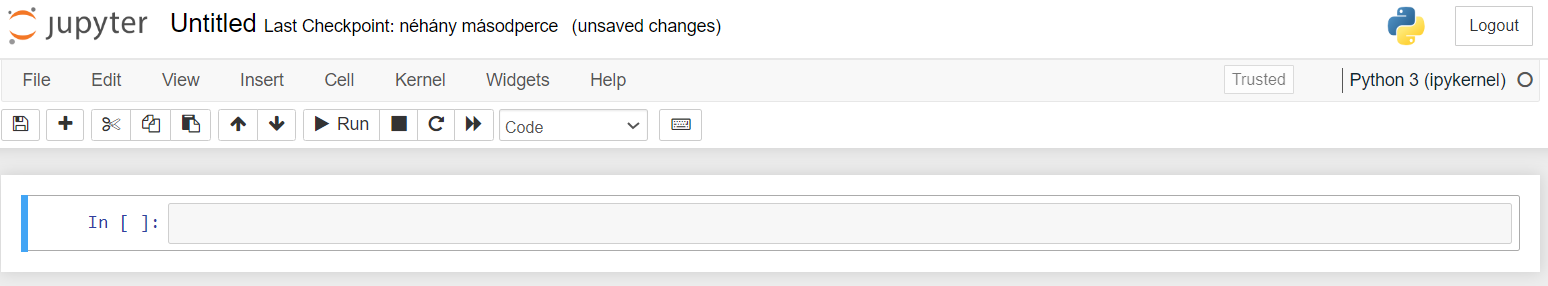
Once Jupyter is installed, you can open a notebook by typing the following command in the terminal:

--------------------------------------------------------------------

jupyter notebook

--------------------------------------------------------------------

**Evidence(s):**



**Task-3 Database Client install mysql-connector ---> DatabaseClient\_install\_mysql-conn.png**

Create a new jupyter notebook and name it **configuring\_mysql\_connector.**

Then install **msql-connector-python**.

--------------------------------------------------------------------

!pip install mysql-connector-python

--------------------------------------------------------------------

Import the connector under the alias connector:

--------------------------------------------------------------------

import mysql.connector as connector

--------------------------------------------------------------------

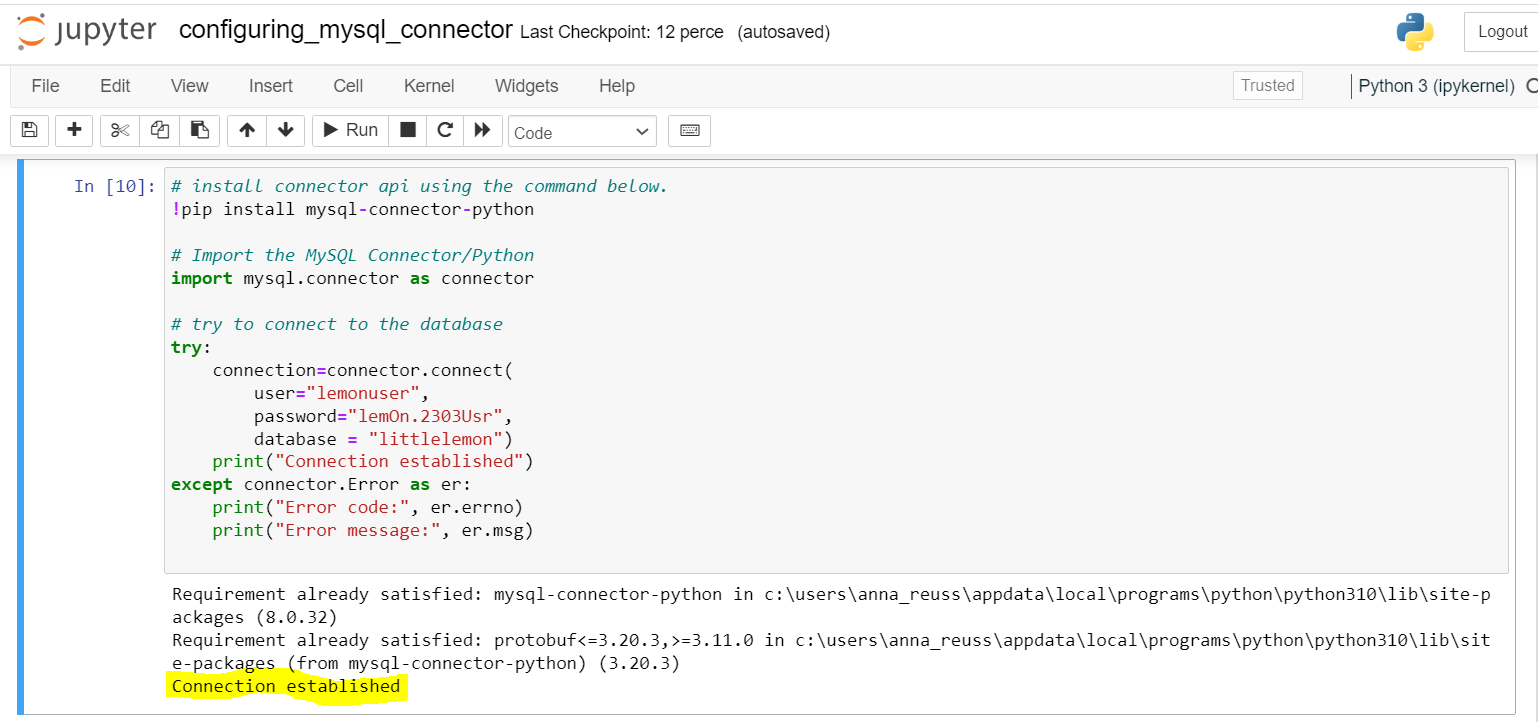
Check that your environment has been correctly configured:

--------------------------------------------------------------------

import mysql.connector as connector

--------------------------------------------------------------------

**Evidence(s):**



**Exercise-9: Database Client Add query functions (Task-1,2,3)**

**Task-1 Database Client connect to database ---> DatabaseClient\_connect\_to\_db.png**

To connect with your database, you can call the connect method of the connector class and pass in your details using. When instantiating the connection, you can pass the database name here in place of calling the USE command later.

--------------------------------------------------------------------

# try to connect to the database

try:

connection=connector.connect(

user="lemonuser",

password="lemOn.2303Usr",

database = "littlelemon")

print("Connection established")

except connector.Error as er:

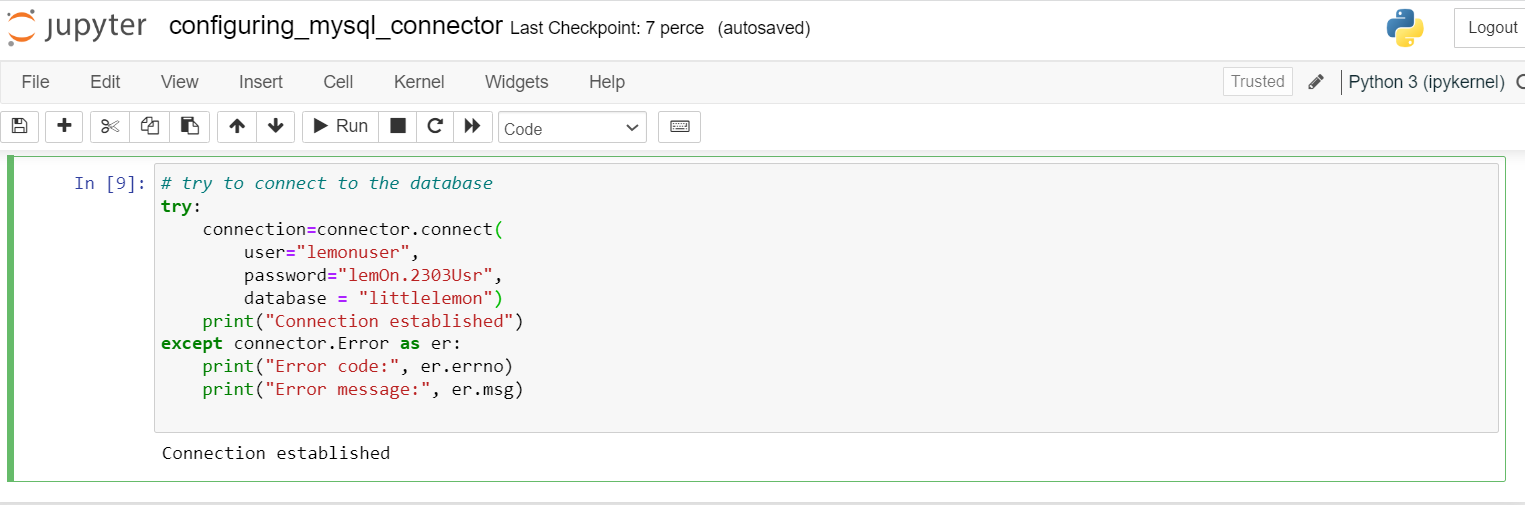
print("Error code:", er.errno)

print("Error message:", er.msg)

--------------------------------------------------------------------

Connection established

**Evidence(s):**

****

**Task-2 Database Client show tables ---> DatabaseClient\_show\_tables.png**

Query the database to show all tables within the database. Execute a generic test query to ensure that there are no issues. The test query should return a snapshot of the database tables.

--------------------------------------------------------------------

# prepare a variable with the SQL statement

show\_tables\_query = "SHOW tables"

# execute the cursor with the variable

cursor.execute(show\_tables\_query)

# confirm that the statement has been successfully executed

for tables in cursor:

print(tables)

--------------------------------------------------------------------

('audit',)

('bookings',)

('bookings\_audit',)

('customers',)

('delivery',)

('menu',)

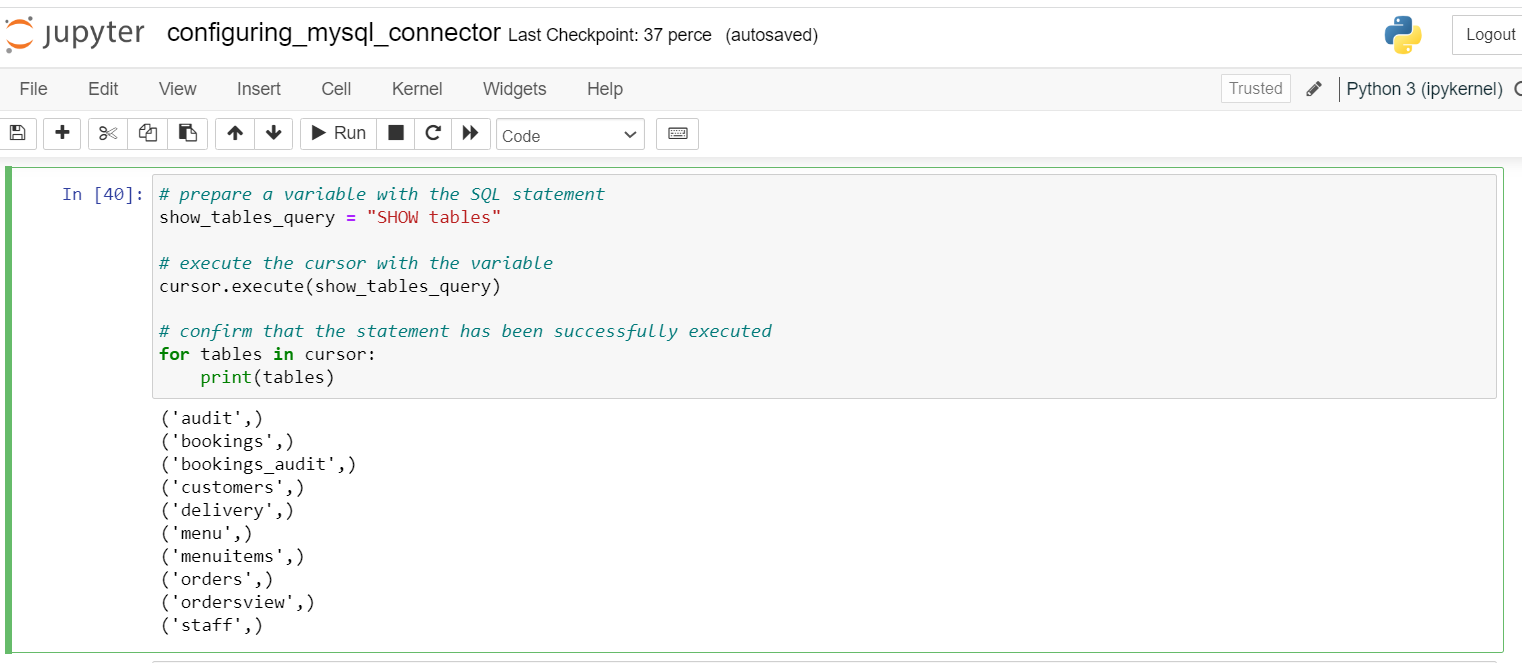
('menuitems',)

('orders',)

('ordersview',)

('staff',)

**Evidence(s):**

****

**Task-3 Database Client query with table JOIN ---> DatabaseClient\_query\_with\_JOIN.png**

**---> configuring\_mysql\_connector.ipynb**

Create a query with table JOIN. Return specific details from your database. They require the full name and contact details for every customer that has placed an order greater than $60 for a promotional campaign.

--------------------------------------------------------------------

# old\_result=cursor.fetchall()

select\_statement="""select Customers.FullName, Customers.PhoneNr, Customers.Email, Orders.OrderID, Orders.TotalPrice

from Orders join Customers

on Orders.CustomerID=Customers.CustomerID

where Orders.TotalPrice>=60

order by Orders.TotalPrice asc """

cursor.execute(select\_statement)

print(cursor.column\_names)

for row in cursor:

print(row)

--------------------------------------------------------------------

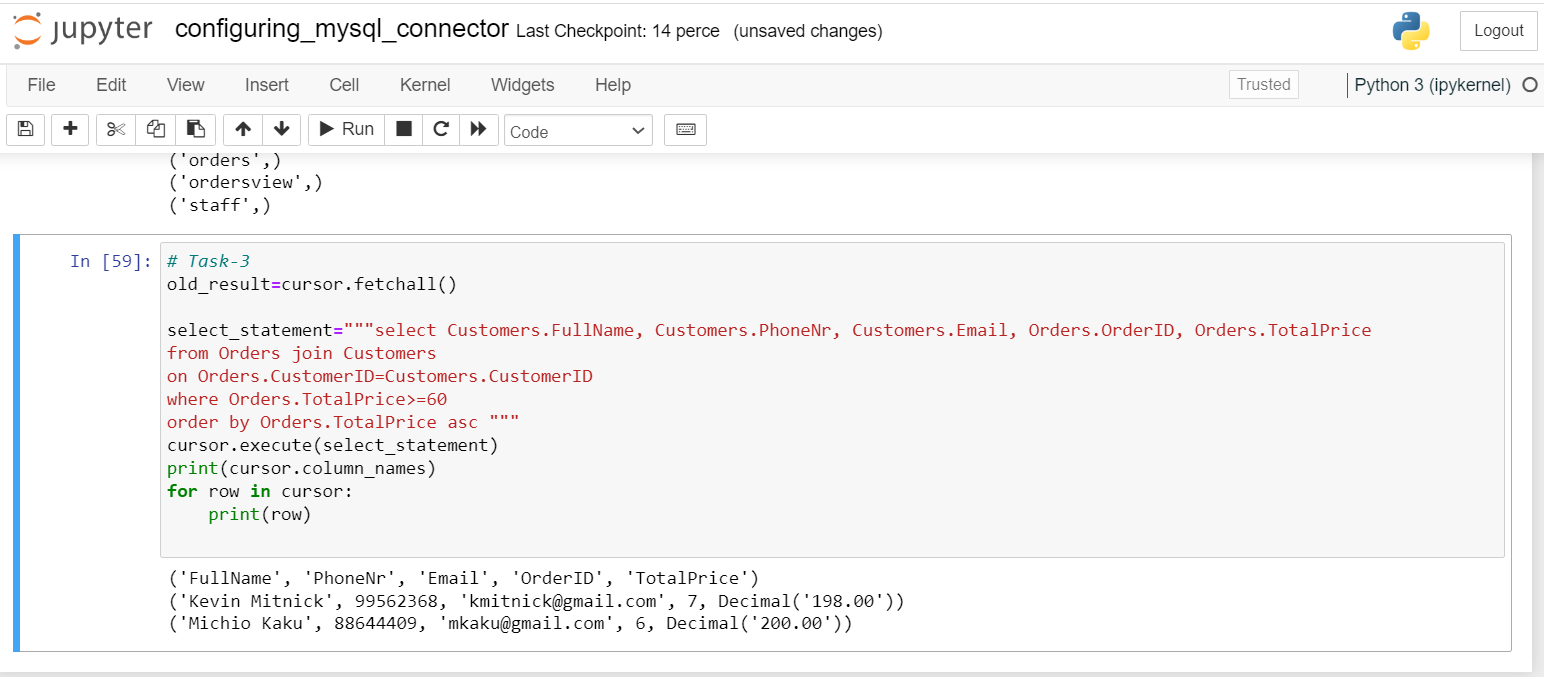
('FullName', 'PhoneNr', 'Email', 'OrderID', 'TotalPrice')

('Kevin Mitnick', 99562368, 'kmitnick@gmail.com', 7, Decimal('198.00'))

('Michio Kaku', 88644409, 'mkaku@gmail.com', 6, Decimal('200.00'))

**Evidence(s):**

****

****

# Appendix

The Workheet.xlsx file, which was used in the Tableau exercises, can be found in this link:

<https://www.coursera.org/learn/advanced-data-modeling/supplement/nGK9L/course-requirements>

**SQL statements used for the solutions:**

**insert data**

(run after the tables are created. Then re-run after the tables are dropped and re-created)

--------------------------------------------------------------------

use littlelemon;

insert into staff (StaffID, FullName, PhoneNr, Role, Salary) values

(0,'Bill Gates','01234567','waiter',1200),

(0,'Diana Ross','15644409','singer',5000),

(0,'George Hamilton','91562368','accountant',2500);

select \* from staff;

insert into menuitems (MenuitemID, Starter, Course, Desert) values

(0,'Pumpkin soup','Rucola salad','Fruit yogurt'),

(0,'Bacon and egg','Steak','Apple pie'),

(0,'Cucumber','Rice','Icecream'),

(0,'Musaka','Feta cheese','Vanile pudding'),

(0,'Italian soup','Italian pizza','Italian Cake');

select \* from menuitems;

INSERT INTO menu (MenuName, Cuisine, MenuitemID, Price) VALUES

('French menu', 'French', '1', '12.0'),

('Energy menu', 'American', '2', '20.0'),

('Slim menu', 'International', '3', '15.0'),

('Greek menu', 'Greek', '4', '11.0'),

('Pizza menu', 'Italian', '5', '18.0');

insert into customers values (0,'Tina Turner','221245671','tinaturner@gmail.com');

insert into customers values (0,'Michio Kaku','88644409','mkaku@gmail.com');

insert into customers values (0,'Kevin Mitnick','99562368','kmitnick@gmail.com');

select \* from customers;

insert into orders (OrderID, OrderDate, MenuID, Quantity, TotalPrice, StaffID, CustomerID) values

(0,'2023-09-01',1,2,24.0,1,1),

(0,'2023-09-02',2,1,20.0,2,2),

(0,'2023-09-03',3,5,45.0,3,1),

(0,'2023-09-05',4,2,22.0,1,2),

(0,'2023-09-08',5,1,18.0,2,3),

(0,'2023-09-19',2,10,200.0,1,2),

(0,'2023-09-19',4,9,198.0,2,3);

select \* from orders;

insert into delivery(DeliveryID, DeliveryDate, DeliveryStatus, OrderID) values

(0,'2023-09-02','delivered',1),

(0,'2023-09-02','in\_progress',2),

(0,'2023-09-03','delivered',3),

(0,'2023-09-05','in\_progress',4),

(0,'2023-09-08','delivered',5);

select \* from delivery;

insert into bookings (BookingID, BookingDate, BookingTime, TableNr, CustomerID, StaffID) values

(0,'2023-09-05','18:00',3,1,2),

(0,'2023-09-05','19:00',5,2,2),

(0,'2023-09-06','12:00',3,3,1),

(0,'2023-09-08','17:00',9,1,3),

(0,'2023-09-15','16:00',3,2,2);

select \* from bookings;

**verify the insert**

--------------------------------------------------------------------

select \* from staff;

select \* from menuitems;

select \* from menu;

select \* from customers;

select \* from orders;

select \* from delivery;

select \* from bookings;

--------------------------------------------------------------------

**drop tables**

(run in case the ER diagram or the database has to be re-created)

--------------------------------------------------------------------

use littlelemon;

drop table bookings;

drop table delivery;

drop table orders;

drop table menu;

drop table menuitems;

drop table staff;

drop table customers;

--------------------------------------------------------------------