#install connector api using the command below.

!pip install mysql-connector-python

# Import the MySQL Connector/Python

import mysql.connector as connector

# Establish connection between Python and MySQL database via connector API

connection=connector.connect(

user="root", # use your own

password="", # use your own

)

print("Connection between MySQL and Python is established.\n")

# Create cursor object to communicate with entire MySQL database

cursor = connection.cursor()

print("Cursor is created to communicate with the MySQL using Python.\n")

# If exist, drop the database first, and create again

try:

cursor.execute("CREATE DATABASE little\_lemon")

except:

cursor.execute("drop database little\_lemon")

cursor.execute("CREATE DATABASE little\_lemon")

print("The database little\_lemon is created.\n")

# Set little\_lemon database for use

cursor.execute("USE little\_lemon")

print("The database little\_lemon is set for use.\n")

# The SQL query for MenuItems table is:

create\_menuitem\_table="""

CREATE TABLE MenuItems (

ItemID INT AUTO\_INCREMENT,

Name VARCHAR(200),

Type VARCHAR(100),

Price INT,

PRIMARY KEY (ItemID)

);"""

# Create MenuItems table

cursor.execute(create\_menuitem\_table)

print("MenuItmes table is created.\n")

# The SQL query for Menu table is:

create\_menu\_table="""

CREATE TABLE Menus (

MenuID INT,

ItemID INT,

Cuisine VARCHAR(100),

PRIMARY KEY (MenuID,ItemID)

);"""

# Create Menu table

cursor.execute(create\_menu\_table)

print("Menu table is created.\n")

# The SQL query for Bookings table is:

create\_booking\_table="""

CREATE TABLE Bookings (

BookingID INT AUTO\_INCREMENT,

TableNo INT,

GuestFirstName VARCHAR(100) NOT NULL,

GuestLastName VARCHAR(100) NOT NULL,

BookingSlot TIME NOT NULL,

EmployeeID INT,

PRIMARY KEY (BookingID)

);"""

# Create Bookings table

cursor.execute(create\_booking\_table)

print("Bookings table is created.\n")

# The SQL query for Bookings table is:

create\_orders\_table="""

CREATE TABLE Orders (

OrderID INT, TableNo INT, MenuID INT,

BookingID INT, BillAmount INT,

Quantity INT,

PRIMARY KEY (OrderID,TableNo)

);"""

# Create Orders table

cursor.execute(create\_orders\_table)

print("Orders table is created.\n")

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# Insert query to populate "MenuItems" table is:

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

insert\_menuitmes="""

INSERT INTO MenuItems (ItemID, Name, Type, Price)

VALUES

(1,'Olives','Starters',5),

(2,'Flatbread','Starters', 5),

(3, 'Minestrone', 'Starters', 8),

(4, 'Tomato bread','Starters', 8),

(5, 'Falafel', 'Starters', 7),

(6, 'Hummus', 'Starters', 5),

(7, 'Greek salad', 'Main Courses', 15),

(8, 'Bean soup', 'Main Courses', 12),

(9, 'Pizza', 'Main Courses', 15),

(10,'Greek yoghurt','Desserts', 7),

(11, 'Ice cream', 'Desserts', 6),

(12, 'Cheesecake', 'Desserts', 4),

(13, 'Athens White wine', 'Drinks', 25),

(14, 'Corfu Red Wine', 'Drinks', 30),

(15, 'Turkish Coffee', 'Drinks', 10),

(16, 'Turkish Coffee', 'Drinks', 10),

(17, 'Kabasa', 'Main Courses', 17);"""

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# Insert query to populate "Menu" table is:

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

insert\_menu="""

INSERT INTO Menus (MenuID,ItemID,Cuisine)

VALUES

(1, 1, 'Greek'),

(1, 7, 'Greek'),

(1, 10, 'Greek'),

(1, 13, 'Greek'),

(2, 3, 'Italian'),

(2, 9, 'Italian'),

(2, 12, 'Italian'),

(2, 15, 'Italian'),

(3, 5, 'Turkish'),

(3, 17, 'Turkish'),

(3, 11, 'Turkish'),

(3, 16, 'Turkish');"""

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# Insert query to populate "Bookings" table is:

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

insert\_bookings="""

INSERT INTO Bookings (BookingID, TableNo, GuestFirstName,

GuestLastName, BookingSlot, EmployeeID)

VALUES

(1,12,'Anna','Iversen','19:00:00',1),

(2, 12, 'Joakim', 'Iversen', '19:00:00', 1),

(3, 19, 'Vanessa', 'McCarthy', '15:00:00', 3),

(4, 15, 'Marcos', 'Romero', '17:30:00', 4),

(5, 5, 'Hiroki', 'Yamane', '18:30:00', 2),

(6, 8, 'Diana', 'Pinto', '20:00:00', 5);"""

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# Insert query to populate "Orders" table is:

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

insert\_orders="""

INSERT INTO Orders (OrderID, TableNo, MenuID, BookingID, Quantity, BillAmount)

VALUES

(1, 12, 1, 1, 2, 86),

(2, 19, 2, 2, 1, 37),

(3, 15, 2, 3, 1, 37),

(4, 5, 3, 4, 1, 40),

(5, 8, 1, 5, 1, 43);"""

print("Inserting data in MenuItems table.")

# Populate MenuItems table

cursor.execute(insert\_menuitmes)

print("Total number of rows in MenuItem table: {}\n".format(cursor.rowcount))

# Once the query is executed, you commit the change into the database

connection.commit()

print("Inserting data in Menus table.")

# Populate MenuItems table

cursor.execute(insert\_menu)

print("Total number of rows in Menu table: {}\n".format(cursor.rowcount))

connection.commit()

print("Inserting data in Bookings table.")

# Populate Bookings table

cursor.execute(insert\_bookings)

print("Total number of rows in Bookings table: {}\n".format(cursor.rowcount))

connection.commit()

print("Inserting data in Orders table.")

# Populate Orders table

cursor.execute(insert\_orders)

print("Total number of rows in Orders table: {}\n".format(cursor.rowcount))

connection.commit()

print("""The database "little\_lemon" is ready for use.""")

## **Task 1**

Little Lemon is running a marketing campaign this month. They need to issue a discount coupon to the top spender on daily bases.

Create a stored procedure TopSpender that can retrieve the following:

* the booking ID,
* the guest’s full name,
* and the bill amount of the top spender at closing.

Call the procedure and print the results.

cursor.execute("DROP PROCEDURE IF EXISTS TopSpender;")

# Stored procedure name >> TopSpender

# Our stored procedure query is

stored\_procedure\_query="""

CREATE PROCEDURE TopSpender()

BEGIN

SELECT bookings.BookingID,

CONCAT(

bookings.GuestFirstname, ' ', bookings.GuestLastname) AS CustomerName,

Orders.BillAmount FROM Bookings INNER JOIN

Orders ON bookings.BookingID=Orders.BookingID

ORDER BY BillAmount DESC LIMIT 1;

END

"""

# Execute the query

cursor.execute(stored\_procedure\_query)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# Call the stored procedure with its name

cursor.callproc("TopSpender")

# Retrieve recrods in "dataset"

results = next( cursor.stored\_results() )

dataset = results.fetchall()

# Retrieve column names using list comprehension in a 'for' loop

for column\_id in cursor.stored\_results():

columns = [ column[0] for column in column\_id.description ]

# Print column names

print(columns)

# Print data

for data in dataset:

print(data)

Task 2

Help little lemon to retrieve all those bookings where the guests did not appear today. How will you implement this task using a stored procedure?

Use NoArrival as a name for your stored procedure.

cursor.execute("DROP PROCEDURE IF EXISTS NoArrival;")

# Stored procedure name >> NoArrival

# Our stored procedure query is

stored\_procedure\_query="""

CREATE PROCEDURE NoArrival()

BEGIN

SELECT bookings.BookingID,

Orders.BillAmount

FROM Bookings

LEFT JOIN

Orders ON Bookings.BookingID=Orders.BookingID

WHERE BillAmount IS NULL;

END

"""

# Execute the query

cursor.execute(stored\_procedure\_query)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# Call the stored procedure with its name

cursor.callproc("NoArrival")

# Retrieve recrods in "dataset"

results = next( cursor.stored\_results() )

dataset = results.fetchall()

# Retrieve column names using list comprehension in a 'for' loop

for column\_id in cursor.stored\_results():

columns = [ column[0] for column in column\_id.description ]

# Print column names

print(columns)

# Print data

for data in dataset:

print(data)

cursor.stored\_results()

cursor.callproc("NoArrival")

results = next( cursor.stored\_results() ) → output <mysql.connector.cursor.MySQLCursorBuffered at 0x10dc39eb0>

results#.fetchall()

type(results) → mysql.connector.cursor.MySQLCursorBuffered

cursor.stored\_results() → <list\_iterator at 0x10e63b9d0>

## **Task 3**

Little Lemon need to keep track of the status of each order and display it on the screen to keep their guests informed.

They categorize the orders as follows:

* If not assigned to any employee, the status is In Queue.
* If assigned to the employees with IDs 4 and 5, the status is Preparing Order.
* If assigned to the employees with IDs 1, 2, and 3, the status is Order Served.

Create a stored procedure named OrderStatus for the above task and call it to check if everything is working.

# Stored procedure name >> OrderStatus

# Our stored procedure query is

stored\_procedure\_query="""

CREATE PROCEDURE OrderStatus()

BEGIN

SELECT bookingID,

CASE

WHEN employeeID IN (1,2,3) THEN "Order Served"

WHEN employeeID IN (4,5) THEN "Preparing Order"

ELSE "In Queue"

END AS Status

FROM Bookings;

END

"""

# Execute the query

cursor.execute(stored\_procedure\_query)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# Call the stored procedure with its name

cursor.callproc("OrderStatus")

# Retrieve recrods in "dataset"

results = next( cursor.stored\_results() )

dataset = results.fetchall()

# Retrieve column names using list comprehension in a 'for' loop

for column\_id in cursor.stored\_results():

columns = [ column[0] for column in column\_id.description ]

# Print column names

print(columns)

# Print data

for data in dataset:

print(data)

# Let's close the cursor and the connection

if connection.is\_connected():

cursor.close()

print("The cursor is closed.")

connection.close()

print("MySQL connection is closed.")

else:

print("Connection is already closed")