Implement and query stored procedures

In [1]:
```
import mysql.connector
from mysql.connector import Error
import pandas as pd
```

Conecting to MySQL Server

In [2]:
```
def make_server_connection(host_name, user_name, user_password):
    connection = None
    try:
        connection = mysql.connector.connect(
            host=host_name,
            user=user_name,
            passwd=user_password
        )
        print("MySQL Server connection successful")
    except Error as err:
        print(f"Error: '{err}'")

    return connection

host_name = "localhost"
user_name = "waleed"
user_password = "Waleed@18574"


connection = make_server_connection(host_name, user_name, user_password)
```

```
MySQL Server connection successful
```

Dropping Databases

In [ ]:
```
def drop_database(connection, drop_database_query):
    cursor = connection.cursor()
    try:
        cursor.execute(drop_database_query)
        print("Database dropped successfully")
    except Error as err:
        print(f"Error: '{err}'")

drop_database_query = "DROP DATABASE IF EXISTS little_lemon_db "
drop_database(connection, drop_database_query)
```

Add Database

In [ ]:
```
def make_database(connection, make_database_query):
    cursor = connection.cursor()
    try:
        cursor.execute(make_database_query)
        print("Database made successfully")
    except Error as err:
        print(f"Error: '{err}'")

make_database_query = "CREATE DATABASE little_lemon_db"
make_database(connection, make_database_query)
```

```
Database made successfully
```

Use Database

In [ ]:
```
cursor = connection.cursor()
cursor.execute("USE little_lemon_db")
```

Add Tables

In [ ]:
```
#MenuItems table
create_menuitem_table = """CREATE TABLE MenuItems (
ItemID INT AUTO_INCREMENT,
Name VARCHAR(200),
Type VARCHAR(100),
Price INT,
PRIMARY KEY (ItemID)
);"""

create_menu_table = """CREATE TABLE Menus (
MenuID INT,
ItemID INT,
Cuisine VARCHAR(100),
PRIMARY KEY (MenuID,ItemID)
);"""

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_orders_table = """CREATE TABLE Orders (
OrderID INT,
TableNo INT,
MenuID INT,
BookingID INT,
BillAmount INT,
Quantity INT,
```

```
PRIMARY KEY (OrderID,TableNo)
);"""

create_employees_table = """CREATE TABLE Employees (
EmployeeID INT AUTO_INCREMENT PRIMARY KEY,
Name VARCHAR (255),
Role VARCHAR (100),
Address VARCHAR (255),
Contact_Number INT,
Email VARCHAR (255),
Annual_Salary VARCHAR (100)
);"""

# Create MenuItems table
cursor.execute(create_menuitem_table)

# Create Menu table
cursor.execute(create_menu_table)

# Create Bookings table
cursor.execute(Create_booking_table)

# Create Orders table
cursor.execute(create_orders_table)

# Create Employees table
cursor.execute(create_employees_table)
```

Populate Tables

In [ ]:

```
#*********************************************************#
# Insert query to populate "MenuItems" table:
#*********************************************************#
insert_menuitems = """
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:
#*********************************************************#
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:
#*********************************************************#
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:
#*********************************************************#
insert_orders = """
INSERT INTO Orders (OrderID, TableNo, MenuID, BookingID, Quantity, BillAmour
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);"""

#*********************************************************#
# Insert query to populate "Employees" table:
#*********************************************************#
insert_employees = """
INSERT INTO Employees (EmployeeID, Name, Role, Address, Contact_Number, Emai
VALUES
(1,'Mario Gollini','Manager','724, Parsley Lane, Old Town, Chicago, IL',3512
(2,'Adrian Gollini','Assistant Manager','334, Dill Square, Lincoln Park, Chi
(3,'Giorgos Dioudis','Head Chef','879 Sage Street, West Loop, Chicago, IL',3
(4,'Fatma Kaya','Assistant Chef','132  Bay Lane, Chicago, IL',351963569,'Fat
(5,'Elena Salvai','Head Waiter','989 Thyme Square, EdgeWater, Chicago, IL',3
```

```
(6,'John Millar','Receptionist','245 Dill Square, Lincoln Park, Chicago, IL'
"""



# Populate MenuItems table
cursor.execute(insert_menuitems)
connection.commit()

# Populate MenuItems table
cursor.execute(insert_menu)
connection.commit()

# Populate Bookings table
cursor.execute(insert_bookings)
connection.commit()

# Populate Orders table
cursor.execute(insert_orders)
connection.commit()

# Populate Employees table
cursor.execute(insert_employees)
connection.commit()
```

Task 1: Establish a connection

In [ ]:
```
from mysql.connector.pooling import MySQLConnectionPool
from mysql.connector import Error

dbconfig = {
    "database": "little_lemon_db",
    "user": "waleed",
    "password": "Waleed@18574"
}

try:
    pool_a = MySQLConnectionPool(pool_name="pool_a", pool_size=2, **dbconfig
    print("Connection pool 'pool_a' created successfully with 2 connections.
except Error as err:
    print(f"Error: {err}")
```

```
Connection pool 'pool_a' created successfully with 2 connections.
```

In [ ]:
```
# Get a connection from the pool and assign it to connection_1
connection_1 = pool_a.get_connection()
print("connection_1 acquired from pool_a.")
```

```
connection_1 acquired from pool_a.
```

Task 2: Implement a stored procedure called PeakHours

In [ ]:
```
# Create the PeakHours stored procedure
peak_hours_proc = '''
CREATE PROCEDURE PeakHours()
BEGIN
    SELECT HOUR(BookingSlot) AS booking_hour, COUNT(*) AS bookings_count
    FROM Bookings
    GROUP BY booking_hour
    ORDER BY bookings_count DESC;
END
'''

cursor = connection_1.cursor()

# Drop the procedure if it exists
cursor.execute("DROP PROCEDURE IF EXISTS PeakHours")

# Create the procedure (execute the full statement at once)
cursor.execute(peak_hours_proc)

# Call the stored procedure
cursor.callproc('PeakHours')

# Fetch results from the stored procedure using the recommended property
for result in cursor.stored_results():
    dataset = result.fetchall()
    columns = [desc[0] for desc in result.description]
    # Print column names
    print(columns)
    # Print data
    for data in dataset:
        print(data)
```

```
['booking_hour', 'bookings_count']
(19, 2)
(15, 1)
(17, 1)
(18, 1)
(20, 1)
```

```
/tmp/ipykernel_25666/2905342655.py:24: DeprecationWarning: Call to deprecate
d function stored_results. Reason: The property counterpart 'stored_results'
will be added in a future release, and this method will be removed.
  for result in cursor.stored_results():
```

Alternative Method

In [ ]:
```
# # Create the PeakHours stored procedure
# peak_hours_proc = '''
# CREATE PROCEDURE PeakHours()
# BEGIN
#     SELECT HOUR(BookingSlot) AS booking_hour, COUNT(*) AS bookings_count
#     FROM Bookings
#     GROUP BY booking_hour
#     ORDER BY bookings_count DESC;
# END
```

```
# '''

# cursor = connection_1.cursor()

# # Drop the procedure if it exists
# cursor.execute("DROP PROCEDURE IF EXISTS PeakHours")

# # Create the procedure (execute the full statement at once)
# cursor.execute(peak_hours_proc)

# # Call the stored procedure
# cursor.callproc('PeakHours')


# # 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 3: Implement a stored procedure GuestStatus

In [ ]:
```
# Get a connection from the pool and assign it to connection_1
connection_2 = pool_a.get_connection()
print("connection_2 acquired from pool_a.")
```

```
connection_2 acquired from pool_a.
```

In [ ]:
```
# Create the GuestStatus stored procedure
guest_status_proc = '''
CREATE PROCEDURE GuestStatus()
BEGIN
    SELECT
        CONCAT(b.GuestFirstName, ' ', b.GuestLastName) AS guest_name,
        e.Role,
        CASE
            WHEN e.Role = 'Manager' OR e.Role = 'Assistant Manager' THEN 'Re
            WHEN e.Role = 'Head Chef' THEN 'Ready to serve'
            WHEN e.Role = 'Assistant Chef' THEN 'Preparing Order'
            WHEN e.Role = 'Head Waiter' THEN 'Order served'
            ELSE 'Unknown Status'
        END AS guest_status
    FROM Bookings b
    LEFT JOIN Employees e ON b.EmployeeID = e.EmployeeID;
```

```
END
'''

cursor = connection_2.cursor()

# Drop the procedure if it exists
cursor.execute("DROP PROCEDURE IF EXISTS GuestStatus")

# Create the procedure
cursor.execute(guest_status_proc)

# Call the stored procedure
cursor.callproc('GuestStatus')

# Fetch results from the stored procedure using the recommended property
for result in cursor.stored_results():
    dataset = result.fetchall()
    columns = [desc[0] for desc in result.description]
    # Print column names
    print(columns)
    # Print data
    for data in dataset:
        print(data)
```

```
['guest_name', 'Role', 'guest_status']
('Anna Iversen', 'Manager', 'Ready to pay')
('Joakim Iversen', 'Manager', 'Ready to pay')
('Vanessa McCarthy', 'Head Chef', 'Ready to serve')
('Marcos Romero', 'Assistant Chef', 'Preparing Order')
('Hiroki Yamane', 'Assistant Manager', 'Ready to pay')
('Diana Pinto', 'Head Waiter', 'Order served')
/tmp/ipykernel_25666/3059535688.py:32: DeprecationWarning: Call to deprecate
d function stored_results. Reason: The property counterpart 'stored_results'
will be added in a future release, and this method will be removed.
  for result in cursor.stored_results():
```

Closing the connection

In [ ]:
```
# Let's close the cursor and the connection
if connection_1.is_connected():
    cursor.close()
    print("The cursor is closed.")
    connection.close()
    print("MySQL connection is closed.")
else:
    print("Connection_1 is already closed")
```

```
The cursor is closed.
MySQL connection is closed.
```

In [ ]:
```
# Let's close the cursor and the connection
if connection_2.is_connected():
    cursor.close()
    print("The cursor is closed.")
    connection.close()
```

```
    print("MySQL connection is closed.")
else:
    print("Connection_2 is already closed")
```

```
The cursor is closed.
MySQL connection is closed.
```

Little Lemon analysis and sales report

importing Required Libraries

In [1]:
```
import mysql.connector
from mysql.connector import Error
import pandas as pd
```

Conecting to MySQL Server

In [2]:
```
def make_server_connection(host_name, user_name, user_password):
    connection = None
    try:
        connection = mysql.connector.connect(
            host=host_name,
            user=user_name,
            passwd=user_password
        )
        print("MySQL Server connection successful")
    except Error as err:
        print(f"Error: '{err}'")

    return connection

host_name = "localhost"
user_name = "waleed"
user_password = "Waleed@18574"


connection = make_server_connection(host_name, user_name, user_password)
```

```
MySQL Server connection successful
```

Use Database

In [3]:
```
cursor = connection.cursor()
cursor.execute("USE little_lemon_db")
```

Task 1: Establish a connection

In [4]:
```
from mysql.connector.pooling import MySQLConnectionPool
from mysql.connector import Error
from mysql.connector import PoolError
```

In [5]:
```
# Define the database configuration as a dictionary for use in the connectic
dbconfig = {
```

```
    "database": "little_lemon_db",
    "user": "waleed",
    "password": "Waleed@18574"
}

# Try to create a connection pool named 'pool_b' with 2 connections using th
# settings.

try:
    pool_b = MySQLConnectionPool(pool_name="pool_b", pool_size=2, **dbconfig
    print("Connection pool 'pool_b' created successfully with 2 connections.
except Error as err:
    print(f"Error: {err}")
```

```
Connection pool 'pool_b' created successfully with 2 connections.
```

In [13]:
```
# # Get a connection from the pool and assign it to connection_1
# connection_1 = pool_b.get_connection()
# print("connection_1 acquired from pool_b.")
```

Task 2

In [6]:
```
# Guest booking data
guests = [
    (8, "Anees", "Java", "18:00:00", 6),
    (5, "Bald", "Vin", "19:00:00", 6),
    (12, "Jay", "Kon", "19:30:00", 6)
]

connections = []
cursors = []

# Try to get three connections from the pool and insert bookings
for i, guest in enumerate(guests):
    try:
        connection = pool_b.get_connection()
        connections.append(connection)
        cursor = connection.cursor()
        cursors.append(cursor)
        insert_query = """
            INSERT INTO Bookings (TableNo, GuestFirstName, GuestLastName, Bo
            VALUES (%s, %s, %s, %s, %s)
        """
        cursor.execute(insert_query, guest)
        connection.commit()
        print(f"Guest {i+1} booking inserted successfully.")
    except PoolError as pe:
        print(f"PoolError for Guest {i+1}: {pe}")
    except Error as err:
        print(f"Error for Guest {i+1}: {err}")

# Return connections to the pool (only two can be returned)
for i, connection in enumerate(connections):
    try:
        connection.close()
```

```
        print(f"Connection {i+1} returned to pool.")
    except PoolError as pe:
        print(f"PoolError when returning connection {i+1}: {pe}")
```

```
Guest 1 booking inserted successfully.
Guest 2 booking inserted successfully.
PoolError for Guest 3: Failed getting connection; pool exhausted
Connection 1 returned to pool.
Connection 2 returned to pool.
```

In [7]:
```
# Now that the previous two connections have been returned to the pool,
# get a new connection from pool_b and insert the third guest booking.

third_guest = (12, "Jay", "Kon", "19:30:00", 6)

try:
    connection = pool_b.get_connection()
    cursor = connection.cursor()
    insert_query = """
        INSERT INTO Bookings (TableNo, GuestFirstName, GuestLastName, Bookir
        VALUES (%s, %s, %s, %s, %s)
    """
    cursor.execute(insert_query, third_guest)
    connection.commit()
    print("Third guest booking inserted successfully.")
    connection.close()
    print("Connection returned to pool.")
except PoolError as pe:
    print(f"PoolError for third guest: {pe}")
except Error as err:
    print(f"Error for third guest: {err}")
```

```
Third guest booking inserted successfully.
Connection returned to pool.
```

In [ ]:
```
cursor.execute("SELECT * FROM Bookings")
bookings = cursor.fetchall()
bookings_df = pd.DataFrame(bookings, columns=[i[0] for i in cursor.descripti
bookings_df
```

Out[ ]:

	BookingID	TableNo	GuestFirstName	GuestLastName	BookingSlot	Emplo
0	1	12	Anna	Iversen	0 days 19:00:00	
1	2	12	Joakim	Iversen	0 days 19:00:00	
2	3	19	Vanessa	McCarthy	0 days 15:00:00	
3	4	15	Marcos	Romero	0 days 17:30:00	
4	5	5	Hiroki	Yamane	0 days 18:30:00	
5	6	8	Diana	Pinto	0 days 20:00:00	
6	7	8	Anees	Java	0 days 18:00:00	
7	8	5	Bald	Vin	0 days 19:00:00	
8	9	12	Jay	Kon	0 days 19:30:00	

Task 3

Name and EmployeeID of the Little Lemon manager

In [8]:
```
connection = pool_b.get_connection()
cursor = connection.cursor()

# Fetch all records from the Bookings table and display them as a DataFrame

query1 = "SELECT * FROM Employees WHERE Role = 'Manager'"
cursor.execute(query1)
results = cursor.fetchall()

print(cursor.column_names)
for row in results:
    print(row)
```

```
('EmployeeID', 'Name', 'Role', 'Address', 'Contact_Number', 'Email', 'Annual
_Salary')
(1, 'Mario Gollini', 'Manager', '724, Parsley Lane, Old Town, Chicago, IL',
351258074, 'Mario.g@littlelemon.com', '$70,000')
```

Name and Role of the Employee with the Highest Salary

In [9]:
```
# Query to get the name and role of the employee with the highest salary
query2 = """
SELECT Name, Role
```

```
FROM Employees
ORDER BY CAST(REPLACE(REPLACE(Annual_Salary, '$', ''), ',', '') AS UNSIGNED)
LIMIT 1;
"""

cursor.execute(query2)
result = cursor.fetchone()
# print("Name:", result[0])
# print("Role:", result[1])
print(cursor.column_names)
print(result)
```

```
('Name', 'Role')
('Mario Gollini', 'Manager')
```

Number of Guests Booked Between 18:00 and 20:00

In [10]:

```
# Query to get the number of guests booked between 18:00 and 20:00
query3 = """
SELECT COUNT(*) AS num_guests
FROM Bookings
WHERE BookingSlot BETWEEN '18:00:00' AND '20:00:00';
"""

cursor.execute(query3)
result = cursor.fetchone()
print("Number of guests booked between 18:00 and 20:00:", result[0], "guests
```

```
Number of guests booked between 18:00 and 20:00: 7 guests
```

Full Name and BookingID of all guests waiting to be Seated

In [11]:

```
# Query to get the full name and BookingID of all guests waiting to be seate
# sorted by their BookingSlot

query4 = """
SELECT
    CONCAT(GuestFirstName, ' ', GuestLastName) AS full_name,
    BookingID
FROM Bookings
WHERE EmployeeID = (
    SELECT EmployeeID FROM Employees WHERE Role = 'Receptionist'
)
ORDER BY BookingSlot;
"""

cursor.execute(query4)
results = cursor.fetchall()
# print("Full Name | BookingID")
# for row in results:
#     print(row[0], "|", row[1])
print(cursor.column_names)
```

```
for row in results:
    print(row)
```

```
('full_name', 'BookingID')
('Anees Java', 7)
('Bald Vin', 8)
('Jay Kon', 9)
```

Task 4

In [12]:

```
# Create the BasicSalesReport stored procedure
basic_sales_report_proc = """
CREATE PROCEDURE BasicSalesReport()
BEGIN
    SELECT
        SUM(BillAmount) AS total_sales,
        AVG(BillAmount) AS average_sale,
        MIN(BillAmount) AS minimum_bill_paid,
        MAX(BillAmount) AS maximum_bill_paid
    FROM Orders;
END
"""


# Drop the procedure if it exists
cursor.execute("DROP PROCEDURE IF EXISTS BasicSalesReport")

# Create the procedure
cursor.execute(basic_sales_report_proc)

# Call the stored procedure
cursor.callproc('BasicSalesReport')

# # Fetch and print results
# for result in cursor.stored_results():
#     dataset = result.fetchall()
#     columns = [desc[0] for desc in result.description]
#     print(columns)
#     for row in dataset:
#         print(row)


# Fetch and print results as a DataFrame
for result in cursor.stored_results():
    dataset = result.fetchall()
    columns = [desc[0] for desc in result.description]
    df = pd.DataFrame(dataset, columns=columns)
    print(df)
```

```
   total_sales average_sale  minimum_bill_paid  maximum_bill_paid
0          243      48.6000                 37                 86
```

```
/tmp/ipykernel_3137/4113673372.py:34: DeprecationWarning: Call to deprecated
function stored_results. Reason: The property counterpart 'stored_results' w
ill be added in a future release, and this method will be removed.
  for result in cursor.stored_results():
```

In [24]:
```
cursor.close()
connection.close()
```

Task 5

In [25]:
```
connection = pool_b.get_connection()

cursor = connection.cursor(buffered = True)
```

In [26]:
```
# Query to get the next three upcoming bookings with guest and assigned empl
query = """
SELECT
    b.BookingSlot,
    CONCAT(b.GuestFirstName, ' ', b.GuestLastName) AS Guest_name,
    CONCAT('Assigned to: ', e.Name, ' [', e.Role, ']') AS Assigned_to
FROM Bookings b
LEFT JOIN Employees e ON b.EmployeeID = e.EmployeeID
ORDER BY b.BookingSlot ASC
LIMIT 3;
"""

cursor.execute(query)
results = cursor.fetchall()
columns = [desc[0] for desc in cursor.description]
upcoming_df = pd.DataFrame(results, columns=columns)
print(upcoming_df)

# Return the connection to the pool
cursor.close()
connection.close()
```

```
     BookingSlot        Guest_name                                Assigned_t
o
0 0 days 15:00:00  Vanessa McCarthy  Assigned to: Giorgos Dioudis [Head Che
f]
1 0 days 17:30:00     Marcos Romero  Assigned to: Fatma Kaya [Assistant Che
f]
2 0 days 18:00:00        Anees Java   Assigned to: John Millar [Receptionis
t]
```