ITSCM773_HW2

April 11, 2025

Step 1) Write create statements for each of the above tables, making sure to have appropriate data types and with all keys declared.

```
[45]: import mysql.connector
import os
from dotenv import load_dotenv

load_dotenv()
password = os.environ.get("MY_SQL_PASSWORD")

try:
    # Connect to MySQL
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password=password
    )

    cursor = conn.cursor()

    # Create database if doesn't exist
    cursor.execute("CREATE DATABASE IF NOT EXISTS restaurant_db")
    cursor.execute("USE restaurant_db")

    # Drop existing tables if they exist (in reverse order to respect foreign␣
 ↪keys)
    cursor.execute("DROP TABLE IF EXISTS Order_details")
    cursor.execute("DROP TABLE IF EXISTS Ticket")
    cursor.execute("DROP TABLE IF EXISTS Menu_item")

    # Create Menu_item table
    cursor.execute("""
    CREATE TABLE Menu_item (
        id INT PRIMARY KEY AUTO_INCREMENT,
        name VARCHAR(100) UNIQUE NOT NULL,
        description TEXT NOT NULL,
        price DECIMAL(6,2) NOT NULL,
        breakfast BOOLEAN NOT NULL,
        lunch BOOLEAN NOT NULL,
        dinner BOOLEAN NOT NULL
    );
    """)

    # Create Ticket table
    cursor.execute("""
    CREATE TABLE Ticket (
        id INT PRIMARY KEY AUTO_INCREMENT,
        server_name VARCHAR(50) NOT NULL,
        table_number INT NOT NULL,
        is_paid BOOLEAN NOT NULL
    );
    """)

    # Create Order_details table
    cursor.execute("""
    CREATE TABLE Order_details (
        menu_item_id INT NOT NULL,
        ticket_id INT NOT NULL,
        quantity INT NOT NULL,
        notes TEXT,
        PRIMARY KEY (menu_item_id, ticket_id),
        FOREIGN KEY (menu_item_id) REFERENCES Menu_item(id),
        FOREIGN KEY (ticket_id) REFERENCES Ticket(id)
    );
    """)

    # Commit the changes
    conn.commit()
    print("Tables created successfully!")

except mysql.connector.Error as err:
    print(f"Error: {err}")

finally:
    # Close the connection
    if 'cursor' in locals():
        cursor.close()
    if 'conn' in locals() and conn.is_connected():
        conn.close()
        print("MySQL connection closed")

conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password=password
```

```
)

cursor = conn.cursor()

# Create database if doesn't exist
cursor.execute("CREATE DATABASE IF NOT EXISTS restaurant_db")
cursor.execute("USE restaurant_db")

# Drop existing tables if they exist (in reverse order to respect foreign keys)
cursor.execute("DROP TABLE IF EXISTS Order_details")
cursor.execute("DROP TABLE IF EXISTS Ticket")
cursor.execute("DROP TABLE IF EXISTS Menu_item")

# Create Menu_item table
cursor.execute("""
CREATE TABLE Menu_item (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) UNIQUE NOT NULL,
    description TEXT NOT NULL,
    price DECIMAL(6,2) NOT NULL,
    breakfast BOOLEAN NOT NULL,
    lunch BOOLEAN NOT NULL,
    dinner BOOLEAN NOT NULL
);
""")

# Create Ticket table
cursor.execute("""
CREATE TABLE Ticket (
    id INT PRIMARY KEY AUTO_INCREMENT,
    server_name VARCHAR(50) NOT NULL,
    table_number INT NOT NULL,
    is_paid BOOLEAN NOT NULL
);
""")

# Create Order_details table
cursor.execute("""
CREATE TABLE Order_details (
    menu_item_id INT NOT NULL,
    ticket_id INT NOT NULL,
    quantity INT NOT NULL,
    notes TEXT,
    PRIMARY KEY (menu_item_id, ticket_id),
    FOREIGN KEY (menu_item_id) REFERENCES Menu_item(id),
    FOREIGN KEY (ticket_id) REFERENCES Ticket(id)
);
```

```
""")

# Commit the changes
conn.commit()
print("Tables created successfully!")

# Close the connection
cursor.close()
conn.close()
```

```
Tables created successfully!
MySQL connection closed
Tables created successfully!
```

Step 2) Create insert statements to populate your tables.

[46]:
```
import mysql.connector
import os
from dotenv import load_dotenv

load_dotenv()
password = os.environ.get("MY_SQL_PASSWORD")

try:
    # Connect to MySQL
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password=password
    )

    cursor = conn.cursor()

    # Create database if doesn't exist
    cursor.execute("CREATE DATABASE IF NOT EXISTS restaurant_db")
    cursor.execute("USE restaurant_db")

    # Drop existing tables if they exist (in reverse order to respect foreign␣
 ↪keys)
    cursor.execute("DROP TABLE IF EXISTS Order_details")
    cursor.execute("DROP TABLE IF EXISTS Ticket")
    cursor.execute("DROP TABLE IF EXISTS Menu_item")

    # Create Menu_item table
    cursor.execute("""
    CREATE TABLE Menu_item (
        id INT PRIMARY KEY AUTO_INCREMENT,
        name VARCHAR(100) UNIQUE NOT NULL,
        description TEXT NOT NULL,
        price DECIMAL(6,2) NOT NULL,
        breakfast BOOLEAN NOT NULL,
        lunch BOOLEAN NOT NULL,
        dinner BOOLEAN NOT NULL
    );
    """)

    # Create Ticket table
    cursor.execute("""
    CREATE TABLE Ticket (
        id INT PRIMARY KEY AUTO_INCREMENT,
        server_name VARCHAR(50) NOT NULL,
        table_number INT NOT NULL,
        is_paid BOOLEAN NOT NULL
    );
    """)

    # Create Order_details table
    cursor.execute("""
    CREATE TABLE Order_details (
        menu_item_id INT NOT NULL,
        ticket_id INT NOT NULL,
        quantity INT NOT NULL,
        notes TEXT,
        PRIMARY KEY (menu_item_id, ticket_id),
        FOREIGN KEY (menu_item_id) REFERENCES Menu_item(id),
        FOREIGN KEY (ticket_id) REFERENCES Ticket(id)
    );
    """)

    # Commit the changes
    conn.commit()
    print("Tables created successfully!")

except mysql.connector.Error as err:
    print(f"Error: {err}")

finally:
    # Close the connection
    if 'cursor' in locals():
        cursor.close()
    if 'conn' in locals() and conn.is_connected():
        conn.close()
        print("MySQL connection closed")

conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password=password
)

cursor = conn.cursor()

# Create database if doesn't exist
cursor.execute("CREATE DATABASE IF NOT EXISTS restaurant_db")
cursor.execute("USE restaurant_db")

# Drop existing tables if they exist (in reverse order to respect foreign keys)
cursor.execute("DROP TABLE IF EXISTS Order_details")
cursor.execute("DROP TABLE IF EXISTS Ticket")
cursor.execute("DROP TABLE IF EXISTS Menu_item")

# Create Menu_item table
cursor.execute("""
CREATE TABLE Menu_item (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) UNIQUE NOT NULL,
    description TEXT NOT NULL,
    price DECIMAL(6,2) NOT NULL,
    breakfast BOOLEAN NOT NULL,
    lunch BOOLEAN NOT NULL,
    dinner BOOLEAN NOT NULL
);
""")

# Create Ticket table
cursor.execute("""
CREATE TABLE Ticket (
    id INT PRIMARY KEY AUTO_INCREMENT,
    server_name VARCHAR(50) NOT NULL,
    table_number INT NOT NULL,
    is_paid BOOLEAN NOT NULL
);
""")

# Create Order_details table
cursor.execute("""
CREATE TABLE Order_details (
    menu_item_id INT NOT NULL,
    ticket_id INT NOT NULL,
    quantity INT NOT NULL,
    notes TEXT,
    PRIMARY KEY (menu_item_id, ticket_id),
    FOREIGN KEY (menu_item_id) REFERENCES Menu_item(id),
    FOREIGN KEY (ticket_id) REFERENCES Ticket(id)
);
""")

# Commit the changes
conn.commit()
print("Tables created successfully!")

# Close the connection
cursor.close()
conn.close()

try:
    # Connect to MySQL
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password=password,
        database="restaurant_db"
    )

    cursor = conn.cursor()

    # Insert data into Menu_item
    cursor.execute("""
    INSERT INTO Menu_item (name, description, price, breakfast, lunch, dinner)␣
 ↪VALUES
    ('Classic Pancakes', 'Fluffy pancakes served with maple syrup and butter',␣
 ↪8.99, TRUE, FALSE, FALSE),
    ('Eggs Benedict', 'Poached eggs on English muffin with hollandaise sauce',␣
 ↪12.99, TRUE, FALSE, FALSE),
    ('Chicken Caesar Salad', 'Romaine lettuce with grilled chicken, croutons,␣
 ↪and Caesar dressing', 14.99, FALSE, TRUE, TRUE),
    ('Cheeseburger', 'Angus beef patty with cheddar cheese, lettuce, tomato,␣
 ↪and special sauce', 13.99, FALSE, TRUE, TRUE),
    ('Spaghetti Bolognese', 'Pasta with rich meat sauce and parmesan cheese',␣
 ↪16.99, FALSE, FALSE, TRUE),
    ('Grilled Salmon', 'Fresh salmon fillet with seasonal vegetables and lemon␣
 ↪butter sauce', 22.99, FALSE, FALSE, TRUE),
    ('Ribeye Steak', 'Prime cut ribeye steak with garlic butter', 24.99, FALSE,␣
 ↪FALSE, TRUE)
    """)

    # Insert data into Ticket
    cursor.execute("""
    INSERT INTO Ticket (server_name, table_number, is_paid) VALUES
    ('John', 5, TRUE),
    ('Sarah', 10, FALSE),
    ('Michael', 3, FALSE),
    ('Juliette', 8, FALSE)
    """)

    # Insert data into Order_details
    cursor.execute("""
    INSERT INTO Order_details (menu_item_id, ticket_id, quantity, notes) VALUES
    (1, 1, 2, 'Extra syrup please'),
    (2, 1, 1, NULL),
    (3, 2, 1, 'No croutons'),
    (4, 2, 2, 'One medium rare, one well done'),
    (5, 2, 1, NULL),
    (6, 3, 2, 'Cook well done'),
    (7, 3, 3, 'Extra powdered sugar'),
    (1, 3, 1, NULL),
    (4, 1, 1, 'No onions'),
    (5, 3, 1, 'Extra cheese'),
    (3, 1, 1, 'Dressing on the side'),
    (2, 3, 2, 'Sauce on the side'),
    (6, 1, 1, 'Medium rare'),
    (7, 2, 2, NULL),
    (4, 3, 1, 'Add bacon')
    """)

    conn.commit()
    print("Data inserted successfully!")

except mysql.connector.Error as err:
    print(f"Error: {err}")

finally:
    if 'conn' in locals() and conn.is_connected():
        cursor.close()
        conn.close()
        print("MySQL connection closed")
```

```
Tables created successfully!
MySQL connection closed
Tables created successfully!
Data inserted successfully!
MySQL connection closed
```

[35]:
```
import mysql.connector
import pandas as pd
```

```python
import os
from dotenv import load_dotenv

load_dotenv()
password = os.environ.get("MY_SQL_PASSWORD")

# Connect to MySQL
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password=password,
    database="restaurant_db"
)

# Get all menu items
query = "SELECT * FROM Menu_item"
df_menu = pd.read_sql(query, conn)
print("Menu Items:")
display(df_menu)

# Get all tickets
query = "SELECT * FROM Ticket"
df_tickets = pd.read_sql(query, conn)
print("\nTickets:")
display(df_tickets)

# Get all order details with menu item names
query = """
SELECT od.ticket_id, m.name as menu_item, od.quantity, od.notes
FROM Order_details od
JOIN Menu_item m ON od.menu_item_id = m.id
ORDER BY od.ticket_id
"""
df_orders = pd.read_sql(query, conn)
print("\nOrder Details:")
display(df_orders)

# Close the connection
conn.close()
```

```
Menu Items:

C:\Users\Katarina\AppData\Local\Temp\ipykernel_4712\3519187817.py:19:
UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not
tested. Please consider using SQLAlchemy.
  df_menu = pd.read_sql(query, conn)

   id                  name  \
0   1      Classic Pancakes
1   2         Eggs Benedict
2   3  Chicken Caesar Salad
3   4           Cheeseburger
4   5   Spaghetti Bolognese
5   6        Grilled Salmon
6   7          Ribeye Steak

                                      description  price  breakfast  lunch  \
0  Fluffy pancakes served with maple syrup and bu…   8.99          1      0
1  Poached eggs on English muffin with hollandais…  12.99          1      0
2  Romaine lettuce with grilled chicken, croutons…  14.99          0      1
3  Angus beef patty with cheddar cheese, lettuce,…  13.99          0      1
4     Pasta with rich meat sauce and parmesan cheese  16.99          0      0
5  Fresh salmon fillet with seasonal vegetables a…  22.99          0      0
6          Prime cut ribeye steak with garlic butter  24.99          0      0

   dinner
0       0
1       0
2       1
3       1
4       1
5       1
6       1

Tickets:
```

```
C:\Users\Katarina\AppData\Local\Temp\ipykernel_4712\3519187817.py:25:
UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not
tested. Please consider using SQLAlchemy.
  df_tickets = pd.read_sql(query, conn)
```

```
   id server_name  table_number  is_paid
0   1        John             5        1
1   2       Sarah            10        0
2   3     Michael             3        0
3   4    Juliette             8        0


Order Details:
```

```
C:\Users\Katarina\AppData\Local\Temp\ipykernel_4712\3519187817.py:36:
UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not
tested. Please consider using SQLAlchemy.
  df_orders = pd.read_sql(query, conn)
```

```
    ticket_id             menu_item  quantity                           notes
```

```
0           1          Cheeseburger         1                          No onions
1           1  Chicken Caesar Salad         1              Dressing on the side
2           1      Classic Pancakes         2                Extra syrup please
3           1         Eggs Benedict         1                              None
4           1        Grilled Salmon         1                       Medium rare
5           2          Cheeseburger         2  One medium rare, one well done
6           2  Chicken Caesar Salad         1                       No croutons
7           2          Ribeye Steak         2                              None
8           2   Spaghetti Bolognese         1                              None
9           3          Cheeseburger         1                         Add bacon
10          3      Classic Pancakes         1                              None
11          3         Eggs Benedict         2                 Sauce on the side
12          3        Grilled Salmon         2                    Cook well done
13          3          Ribeye Steak         3             Extra powdered sugar
14          3   Spaghetti Bolognese         1                      Extra cheese
```

[36]:
```
import mysql.connector
import os
from dotenv import load_dotenv

load_dotenv()
password = os.environ.get("MY_SQL_PASSWORD")
try:
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password=password,
        database="restaurant_db"
    )

    cursor = conn.cursor()

    # Add a new menu item
    cursor.execute("""
    INSERT INTO Menu_item (name, description, price, breakfast, lunch, dinner)
    VALUES ('French Toast', 'Thick slices of bread dipped in egg batter and␣
 ↪grilled', 9.99, TRUE, FALSE, FALSE)
    """)

    # Add a new ticket
    cursor.execute("""
    INSERT INTO Ticket (server_name, table_number, is_paid)
    VALUES ('Emma', 12, FALSE)
    """)

    # Get the new menu item ID and ticket ID
    cursor.execute("SELECT id FROM Menu_item WHERE name = 'French Toast'")
    menu_item_id = cursor.fetchone()[0]

    cursor.execute("SELECT id FROM Ticket WHERE server_name = 'Emma'")
    ticket_id = cursor.fetchone()[0]

    # Add order details for the new ticket
    cursor.execute(f"""
    INSERT INTO Order_details (menu_item_id, ticket_id, quantity, notes)
    VALUES ({menu_item_id}, {ticket_id}, 3, 'Extra maple syrup')
    """)

    conn.commit()
    print("New data added successfully!")

except mysql.connector.Error as err:
    print(f"Error: {err}")

finally:
    if 'conn' in locals() and conn.is_connected():
        cursor.close()
        conn.close()
        print("MySQL connection closed")
```

```
New data added successfully!
MySQL connection closed
```

[37]:
```
import mysql.connector
import os
from dotenv import load_dotenv

load_dotenv()
password = os.environ.get("MY_SQL_PASSWORD")
try:
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password=password,
        database="restaurant_db"
    )

    cursor = conn.cursor()

    # Update a ticket to mark it as paid
    cursor.execute("""
    UPDATE Ticket
    SET is_paid = TRUE
    WHERE id = 2
    """)

    conn.commit()
    print("Ticket updated successfully!")

except mysql.connector.Error as err:
    print(f"Error: {err}")

finally:
    if 'conn' in locals() and conn.is_connected():
        cursor.close()
        conn.close()
        print("MySQL connection closed")
```

```
Ticket updated successfully!
MySQL connection closed
```

Step 3) Now create SQL statements to answer the following questions. Each must be answered in a single query. IMPORTANT: your query must work correctly even if the data in your tables is changed! Hint: If a query asks for you to test on a specific value that does not exist in your dataset, you may either (A) add values to your dataset so that the query will work as written or (B) show both the correct query solution without it having any impact/retrieving values, and then swap out the values to something that matches your data and show its execution. 1. List all the information stored in the menu_item table. 2. List the id, server_name, and table_number for all unpaid tickets. 3. List all the menu_item information for all items available for both lunch and dinner. (To be retrieved the item must fulfill both conditions.) 4. Show all breakfast items that cost less than $14.99. 5. Show each item that was ordered at table 10. (Note: its okay if this retrieves multiple orders worth of items.) 6. Update the ticket table so ticket 9 is now paid. (Hint: You will need to use an update statement) 7. Update the menu_item tables so that all items with the word "steak" in their title are now $2 more expensive. (Hint: You will need to use an update statement and remember that you can do arithmetic in an update statement.) 8. For each ticket_id, show the number of items sold. 9. Show the list of menu items names sold by the server Juliette who's notes field is not null.

[38]:
```
import mysql.connector
import pandas as pd
import os
from dotenv import load_dotenv

load_dotenv()
password = os.environ.get("MY_SQL_PASSWORD")

try:
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password=password,
        database="restaurant_db"
    )

    #3.1) List all the information stored in the menu_item table.
    query1 = "SELECT * FROM Menu_item"
    df_query1 = pd.read_sql(query1, conn)
    print("Query 1: All information from menu_item table")
    display(df_query1)

    #3.2) List the id, server_name, and table_number for all unpaid tickets.
    query2 = "SELECT id, server_name, table_number FROM Ticket WHERE is_paid =␣
    ↪FALSE"
    df_query2 = pd.read_sql(query2, conn)
    print("\nQuery 2: Unpaid tickets")
    display(df_query2)

    #3.3) List all menu_item information for items available for both lunch and␣
    ↪dinner.
    query3 = "SELECT * FROM Menu_item WHERE lunch = TRUE AND dinner = TRUE"
    df_query3 = pd.read_sql(query3, conn)
    print("\nQuery 3: Items available for both lunch and dinner")
    display(df_query3)

    #3.4) Show all breakfast items that cost less than $14.99.
    query4 = "SELECT * FROM Menu_item WHERE breakfast = TRUE AND price < 14.99"
    df_query4 = pd.read_sql(query4, conn)
    print("\nQuery 4: Breakfast items under $14.99")
    display(df_query4)

    #3.5) Show each item that was ordered at table 10
    query5 = """
    SELECT m.*
    FROM Menu_item m
    JOIN Order_details od ON m.id = od.menu_item_id
    JOIN Ticket t ON od.ticket_id = t.id
    WHERE t.table_number = 10
    """
    df_query5 = pd.read_sql(query5, conn)
    print("\nQuery 5: Items ordered at table 10")
    display(df_query5)

    #3.8) For each ticket_id, show the number of items sold
    query8 = """
    SELECT ticket_id, SUM(quantity) as total_items_sold
    FROM Order_details
    GROUP BY ticket_id
    ORDER BY ticket_id
    """
```

```
    df_query8 = pd.read_sql(query8, conn)
    print("\nQuery 8: Number of items sold per ticket")
    display(df_query8)

    #3.9) Show menu items with notes sold by Juliette
    query9 = """
    SELECT m.name as menu_item_name
    FROM Menu_item m
    JOIN Order_details od ON m.id = od.menu_item_id
    JOIN Ticket t ON od.ticket_id = t.id
    WHERE t.server_name = 'Juliette' AND od.notes IS NOT NULL
    """
    df_query9 = pd.read_sql(query9, conn)
    print("\nQuery 9: Menu items with notes sold by Juliette")
    display(df_query9)

except mysql.connector.Error as err:
    print(f"Error: {err}")

finally:
    if 'conn' in locals() and conn.is_connected():
        conn.close()
        print("MySQL connection closed")
```

```
Query 1: All information from menu_item table

C:\Users\Katarina\AppData\Local\Temp\ipykernel_4712\1299111868.py:19:
UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not
tested. Please consider using SQLAlchemy.
  df_query1 = pd.read_sql(query1, conn)

   id                 name  \
0   1      Classic Pancakes
1   2         Eggs Benedict
2   3  Chicken Caesar Salad
3   4          Cheeseburger
4   5   Spaghetti Bolognese
5   6        Grilled Salmon
6   7          Ribeye Steak
7   8          French Toast

                                         description  price  breakfast  lunch  \
0  Fluffy pancakes served with maple syrup and bu…   8.99          1      0
1  Poached eggs on English muffin with hollandais…  12.99          1      0
2  Romaine lettuce with grilled chicken, croutons…  14.99          0      1
3  Angus beef patty with cheddar cheese, lettuce,…  13.99          0      1
4     Pasta with rich meat sauce and parmesan cheese  16.99          0      0
5  Fresh salmon fillet with seasonal vegetables a…  22.99          0      0
```

```
6         Prime cut ribeye steak with garlic butter  24.99          0      0
7  Thick slices of bread dipped in egg batter and…   9.99          1      0

   dinner
0       0
1       0
2       1
3       1
4       1
5       1
6       1
7       0
```

Query 2: Unpaid tickets

```
C:\Users\Katarina\AppData\Local\Temp\ipykernel_4712\1299111868.py:25:
UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not
tested. Please consider using SQLAlchemy.
  df_query2 = pd.read_sql(query2, conn)
```

```
   id server_name  table_number
0   3     Michael             3
1   4    Juliette             8
2   5        Emma            12
```

Query 3: Items available for both lunch and dinner

```
C:\Users\Katarina\AppData\Local\Temp\ipykernel_4712\1299111868.py:31:
UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not
tested. Please consider using SQLAlchemy.
  df_query3 = pd.read_sql(query3, conn)
```

```
   id                 name  \
0   3  Chicken Caesar Salad
1   4          Cheeseburger

                                         description  price  breakfast  lunch  \
0  Romaine lettuce with grilled chicken, croutons…  14.99          0      1
1  Angus beef patty with cheddar cheese, lettuce,…  13.99          0      1

   dinner
0       1
1       1
```

Query 4: Breakfast items under $14.99

```
C:\Users\Katarina\AppData\Local\Temp\ipykernel_4712\1299111868.py:37:
UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not
tested. Please consider using SQLAlchemy.
  df_query4 = pd.read_sql(query4, conn)
```

```
   id               name                                        description  \
0   1  Classic Pancakes  Fluffy pancakes served with maple syrup and bu…
1   2     Eggs Benedict  Poached eggs on English muffin with hollandais…
2   8      French Toast  Thick slices of bread dipped in egg batter and…

   price  breakfast  lunch  dinner
0   8.99          1      0       0
1  12.99          1      0       0
2   9.99          1      0       0
```

```
Query 5: Items ordered at table 10
```

```
C:\Users\Katarina\AppData\Local\Temp\ipykernel_4712\1299111868.py:49:
UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not
tested. Please consider using SQLAlchemy.
  df_query5 = pd.read_sql(query5, conn)
```

```
   id                 name  \
0   3  Chicken Caesar Salad
1   4          Cheeseburger
2   5   Spaghetti Bolognese
3   7          Ribeye Steak

                                       description  price  breakfast  lunch  \
0  Romaine lettuce with grilled chicken, croutons…  14.99          0      1
1  Angus beef patty with cheddar cheese, lettuce,…  13.99          0      1
2     Pasta with rich meat sauce and parmesan cheese  16.99          0      0
3          Prime cut ribeye steak with garlic butter  24.99          0      0

   dinner
0       1
1       1
2       1
3       1
```

```
Query 8: Number of items sold per ticket
```

```
C:\Users\Katarina\AppData\Local\Temp\ipykernel_4712\1299111868.py:60:
UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not
tested. Please consider using SQLAlchemy.
  df_query8 = pd.read_sql(query8, conn)
```

```
   ticket_id  total_items_sold
0          1               6.0
1          2               6.0
2          3              10.0
3          5               3.0


Query 9: Menu items with notes sold by Juliette
```

```
C:\Users\Katarina\AppData\Local\Temp\ipykernel_4712\1299111868.py:72:
UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not
tested. Please consider using SQLAlchemy.
  df_query9 = pd.read_sql(query9, conn)
```

```
Empty DataFrame
Columns: [menu_item_name]
Index: []
```

```
MySQL connection closed
```

Query 3.1) There are 8 menu items including french toast, which was added later. All items have appropriate prices, descriptions, and meal availability flags. Query 3.2) There are 3 unpaid tickets: Michael (table 3), Juliette (table 8), and Emma (table 12). Furthermore, tickets 1, 2, and 9 are marked as paid. Query 3.3) Only 2 items are available for both lunch and dinner: chicken caesar salad and the cheeseburger. Query 3.4) There are 3 breakfast items under $14.99: Classic Pancakes ($8.99), Eggs Benedict ($12.99), and French Toast ($9.99). Query 3.5) Table 10 has ordered Chicken Caesar Salad, Cheeseburger, Spaghetti Bolognese, and Ribeye Steak. Query 3.8) For number of items sold per ticket, Ticket 1- 6 items, Ticket 2- 6 items, Ticket 3- 10 items, and Ticket 5- 3 items. Query 3.9) This query returned as an empty result, so that means Juliette hasn't taken any orders with notes or there might be a data issue. I'll add some data for Juliette with notes in the next code cell.

[39]:
```python
import mysql.connector
import os
from dotenv import load_dotenv

load_dotenv()
password = os.environ.get("MY_SQL_PASSWORD")
try:
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password=password,
        database="restaurant_db"
    )

    cursor = conn.cursor()

    # First, find Juliette's ticket ID
    cursor.execute("SELECT id FROM Ticket WHERE server_name = 'Juliette'")
    juliette_ticket = cursor.fetchone()

    if juliette_ticket:
        juliette_id = juliette_ticket[0]

        # Add an order with notes for Juliette
        cursor.execute(f"""
        INSERT INTO Order_details (menu_item_id, ticket_id, quantity, notes)
        VALUES (3, {juliette_id}, 1, 'Extra dressing on the side')
        ON DUPLICATE KEY UPDATE notes = 'Extra dressing on the side', quantity␣
 ↪= 1
        """)

        conn.commit()
        print(f"Added order with notes for Juliette's ticket (ID:␣
 ↪{juliette_id})")
    else:
        print("Juliette not found in the Ticket table")

except mysql.connector.Error as err:
    print(f"Error: {err}")

finally:
    if 'conn' in locals() and conn.is_connected():
        cursor.close()
        conn.close()
        print("MySQL connection closed")
```

```
Added order with notes for Juliette's ticket (ID: 4)
MySQL connection closed
```

[40]:

```
import mysql.connector
import pandas as pd
import os
from dotenv import load_dotenv

load_dotenv()
password = os.environ.get("MY_SQL_PASSWORD")
try:
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password=password,
        database="restaurant_db"
    )
    # Query 9: Show menu items with notes sold by Juliette
    query9 = """
    SELECT m.name as menu_item_name
    FROM Menu_item m
    JOIN Order_details od ON m.id = od.menu_item_id
    JOIN Ticket t ON od.ticket_id = t.id
    WHERE t.server_name = 'Juliette' AND od.notes IS NOT NULL
    """
    df_query9 = pd.read_sql(query9, conn)
    print("Query 9: Menu items with notes sold by Juliette")
    display(df_query9)

except mysql.connector.Error as err:
    print(f"Error: {err}")

finally:
    if 'conn' in locals() and conn.is_connected():
        conn.close()
        print("MySQL connection closed")
```

```
Query 9: Menu items with notes sold by Juliette
```

```
C:\Users\Katarina\AppData\Local\Temp\ipykernel_4712\484908961.py:24:
UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not
tested. Please consider using SQLAlchemy.
  df_query9 = pd.read_sql(query9, conn)
```

```
         menu_item_name
0   Chicken Caesar Salad
```

```
MySQL connection closed
```

Ok, now the query for 3.9 is working correctly. It shows Chicken Caesar Salad as a menu item with notes sold by Juliette. I will now do parts 3.6 and 3.7, which were the UPDATE queries that need to be run separately since they modify the database (changing values) and need to be committed to the database.

[41]:
```
import mysql.connector
import os
from dotenv import load_dotenv

load_dotenv()
password = os.environ.get("MY_SQL_PASSWORD")
try:
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password=password,
        database="restaurant_db"
    )

    cursor = conn.cursor()

    # First, check if ticket 9 exists
    cursor.execute("SELECT id FROM Ticket WHERE id = 9")
    ticket_exists = cursor.fetchone()

    if ticket_exists:
        # Update ticket 9 to be paid
        cursor.execute("""
        UPDATE Ticket
        SET is_paid = TRUE
        WHERE id = 9
        """)
        print("Query 3.6 Ticket 9 updated to paid status")
    else:
        # If ticket 9 doesn't exist, insert it first
        cursor.execute("""
        INSERT INTO Ticket (id, server_name, table_number, is_paid)
        VALUES (9, 'Alex', 15, FALSE)
        """)
        # Then update it
        cursor.execute("""
        UPDATE Ticket
        SET is_paid = TRUE
        WHERE id = 9
        """)
        print("Query 3.6 Ticket 9 created and updated to paid status")

    conn.commit()

    # Verify update
    cursor.execute("SELECT * FROM Ticket WHERE id = 9")
    result = cursor.fetchone()
    if result:
        print(f"Ticket 9 is now: {result}")

except mysql.connector.Error as err:
    print(f"Error: {err}")

finally:
    if 'conn' in locals() and conn.is_connected():
        cursor.close()
        conn.close()
        print("MySQL connection closed")
```

```
Query 3.6 Ticket 9 created and updated to paid status
Ticket 9 is now: (9, 'Alex', 15, 1)
MySQL connection closed
```

[42]:
```python
import mysql.connector
import os
from dotenv import load_dotenv

load_dotenv()
password = os.environ.get("MY_SQL_PASSWORD")
try:
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password=password,
        database="restaurant_db"
    )

    cursor = conn.cursor()

    # Update prices for items with "steak" in the name
    cursor.execute("""
    UPDATE Menu_item
    SET price = price + 2
    WHERE name LIKE '%steak%'
    """)

    conn.commit()
    print("Query 3.7: Updated prices for items with 'steak' in the name")

    # Verify update
    cursor.execute("SELECT id, name, price FROM Menu_item WHERE name LIKE␣
 ↪'%steak%'")
    results = cursor.fetchall()
    for row in results:
        print(f"Item {row[0]}: {row[1]} - ${row[2]}")

except mysql.connector.Error as err:
    print(f"Error: {err}")

finally:
    if 'conn' in locals() and conn.is_connected():
        cursor.close()
        conn.close()
        print("MySQL connection closed")
```

```
Query 3.7: Updated prices for items with 'steak' in the name
Item 7: Ribeye Steak - $26.99
```

```
MySQL connection closed
```

Looking at 3.6 and 3.7 queries, for 3.6, Ticket 9 did not exist in the database before running the query so the code first created a new ticket with ID:9, Server name: ‘Alex’, Table number: 15, and is_paid: FALSE (initially). Then, the code updated the ticket to market it as paid (is_paid = TRUE). The verification query shows the final state of ticket 9: (9, ‘Alex’, 15, 1) where the values represent (id, server_name, table_number, is_paid). Also, the last value is 1, meaning TRUE for the is_paid field. This query essentially demonstrates how to check if a record exists before updating it; insert a new record if it doesn’t exist; update the record with new values; and verify the changes were made correctly. Since ticket 9 did not exist, the query handled the case by creating it first and then updating it.

For 3.7, the UPDATE statement successfully found all menu items with “steak” in their names and increased their prices by \$2. The verification query found one matching item, which is Item ID: 7, Name: Ribeye Steak, New Price: \$26.99. The price was increased \$2 from the original price of \$24.99. Therefore, the new price is \$26.99. 3.7 is a query that shows how to use the LIKE operator to find items containing a specific word; perform arithmetic in an UPDATE statement (price + 2); and verify that any changes were correctly made.

Step 4) Reflecting on the material in this chapter and your work on this assignment, what insight does this give you into how databases can power digital dashboards & reports?

The insight from this chapter’s material and the assignment is how databases can power digital dashboards and reports in several important ways: First of all, for real-time data access, the SQL queries created can retrieve up-to-date information instantly, allowing dashboards to display current business status like unpaid tickets or menu items that are popular. Secondly is data aggregation and summarisation: Queries like the one counting items sold per ticket demonstrates how databases can aggregate raw, transactional data into meaningful business metrics that can be displayed on dashboards. The ability to filter data like breakfast items under a certain price allows dashboard users to focus on specific subsets of data relevant to their needs. Moreover, by joining tables like finding items ordered at specific tables, databases enable dashboards to show connections between different business entities. Data manipulation like updating queries show how databases can not only display information but also modify it, allowing dashboards to be tools that are interactive for business management, and there is consistency and integrity–the relational structure ensures that reports draw from a single source of truth with enforced relationships between entities. Scalability is also important since for example here, as the restaurant grows and collects more and more data, the same queries will continue to work, which provides consistent reporting regardless of data volume. And business intelligence can be seen through this assignment with having these queries enable insights such as identifying items that are popular, tracking server importance, or analysing pricing strategies.

The restaurant database created and the queries written represent potential insight that could appear on a dashboard. For example, query 3.2 (unpaid tickets) could power a real-time dashboard element showing which tables still need to settle their bills, helping managers monitor cash flow; query 3.3 (items for both lunch and dinner) could inform inventory management dashboards, highlighting versatile menu items that need consistent stock levels throughout the day; and query 3.8 (items sold per ticket) could feed into dashboards for sales analytics, showing average order size and aiding identification for upselling opportunities. Last week, I was speaking to a data scientist who is working at UCLA, and he spoke of how SQL was like the elevated Excel hahaha. From this assignment, I do concur that SQL’s power lies in its ability to transform raw data into meaningful information through different query perspectives such as easily shifting from operational views (current unpaid tickets) to analytical views (sales patterns across meal times); the same database can simultaneously support different dashboard needs for various stakeholders (servers, kitchen staff, management, etc); and complex business questions can be answered through relatively simple queries as demonstrated in the joins across multiple tables. It always fascinates me to think of what is going on behind dashboards during a presentation filled with data visuals.