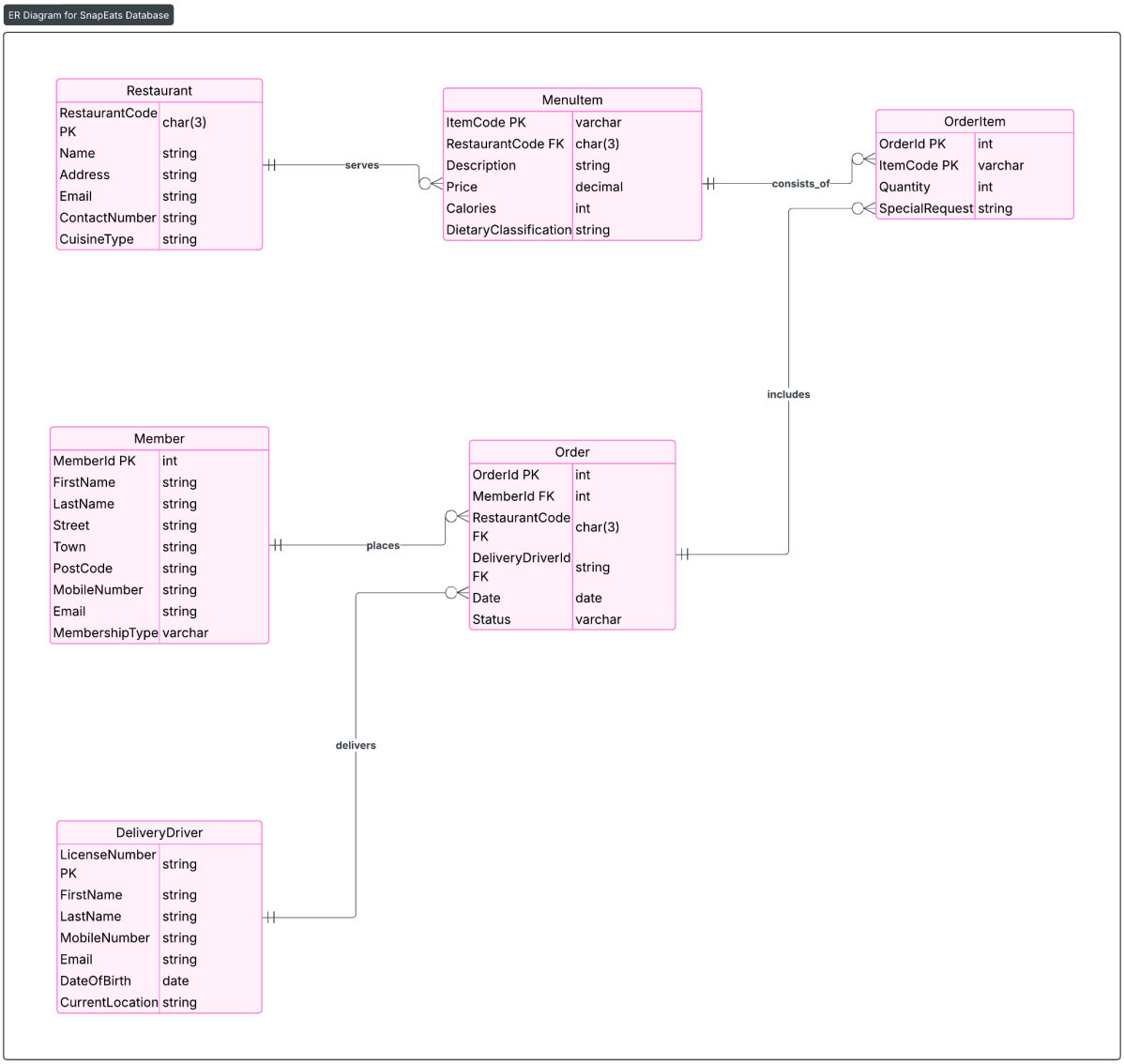
SnapEats Database Design – Applied Project (Assessment 3)

INFS1017 – Database Design and Management

Western Sydney University – 2025

Group 3 Members: Sean, Andrei, Akoon, Hudayfah

Task 1.1 - ER Diagram:



Task 1.2 - Relational Data Model:

Legend:

- [PK] = Primary Key
- [FK] = Foreign Key

1. Member

   Member(memberId [PK], firstName, lastName, street, town, postcode, mobileNumber, email, membershipType)

2. Restaurant

   Restaurant(restaurantCode [PK], name, address, email, contactNumber, cuisineType)

3. MenuItem

   MenuItem(itemCode [PK], restaurantCode [FK], description, price, calories, dietaryClassification)

4. DeliveryDriver

   DeliveryDriver(licenseNumber [PK], firstName, lastName, mobileNumber, email, dateOfBirth, currentLocation)

5. Order

   Order(orderId [PK], memberId [FK], restaurantCode [FK], deliveryDriverId [FK], date, status)

6. OrderItem

   OrderItem(orderId [PK, FK], itemCode [PK, FK], quantity, specialRequest)

Task 2 - Normalisation

1. Member

- Functional Dependencies:

  memberId → firstName, lastName, street, town, postcode, mobileNumber, email, membershipType

- Normal Form:
  3NF

- Justification:
  The primary key memberId uniquely determines all other attributes. All attributes are atomic, there are no partial or transitive dependencies, and the relation satisfies 1NF, 2NF, and 3NF.

2. Restaurant

- Functional Dependencies:

  restaurantCode → name, address, email, contactNumber, cuisineType

- Normal Form:
  3NF

- Justification:
  All non-key attributes are fully dependent on the primary key restaurantCode. The relation has no transitive or partial dependencies and is in 3NF.

3. MenuItem

- Functional Dependencies:

  itemCode → restaurantCode, description, price, calories, dietaryClassification

- Normal Form:
  3NF

- Justification:
  Each attribute is dependent only on the primary key itemCode, with no transitive or partial dependencies. The relation is in 3NF.

4. DeliveryDriver

- Functional Dependencies:

  licenseNumber → firstName, lastName, mobileNumber, email, dateOfBirth, currentLocation

- Normal Form:
  3NF

- Justification:
  All non-key attributes are fully dependent on the primary key licenseNumber. The relation satisfies all requirements for 3NF.

5. Order

- Functional Dependencies:

  orderId → memberId, restaurantCode, deliveryDriverId, date, status

- Normal Form:
  3NF

- Justification:
  The primary key orderId determines all other attributes. There are no partial or transitive dependencies, so the relation is in 3NF.

6. OrderItem

- Functional Dependencies:

(orderId, itemCode) → quantity, specialRequest

- Normal Form:
  3NF

- Justification:
  The composite primary key (orderId, itemCode) fully determines quantity and specialRequest. There are no transitive dependencies, and the relation is in 3NF.

Task 3 – SQL Implementation

The following SQL statements were produced and executed in phpMyAdmin to develop the SnapEats database structure. Each CREATE TABLE statement provides the schema of the appropriate entity, including primary and foreign keys, and was implemented according to the original design given in the ERD.

-- Create Member table

```
CREATE TABLE

Member (

  memberId INT PRIMARY KEY,

  firstName VARCHAR(20),

  lastName VARCHAR(20),

  street VARCHAR(25),

  town VARCHAR(15),

  postcode VARCHAR(6),

  mobileNumber INT(15),

  email VARCHAR(50),

  membershipType ENUM('Standard', 'Deluxe', 'Premium')
```

```
);

-- Create Restaurant table
CREATE TABLE
 Restaurant (
  restaurantCode INT PRIMARY KEY,
  name VARCHAR(30),
  address VARCHAR(35),
  email VARCHAR(25),
  contactNumber INT(20),
  cuisineType VARCHAR(50)
);

-- Create MenuItem table
CREATE TABLE
 MenuItem (
  itemCode INT PRIMARY KEY,
  restaurantCode INT,
  description VARCHAR(255),
  price DECIMAL(6,2),
  calories INT,
  dietaryClassification VARCHAR(50),
```

```
  FOREIGN KEY (restaurantCode) REFERENCES
Restaurant(restaurantCode)
);

-- Create DeliveryDriver table
CREATE TABLE
DeliveryDriver (
  licenseNumber INT PRIMARY KEY,
  firstName VARCHAR(20),
  lastName VARCHAR(20),
  mobileNumber INT(15),
  email VARCHAR(50),
  dateOfBirth DATE,
  currentLocation VARCHAR(50)
);

-- Create Orders table
CREATE TABLE
Orders (
  orderId INT PRIMARY KEY,
  memberId INT,
  restaurantCode INT,
  deliveryDriverId INT,
```

```
  date DATE,

  status VARCHAR(20),

  FOREIGN KEY (memberId) REFERENCES Member(memberId),

  FOREIGN KEY (restaurantCode) REFERENCES
Restaurant(restaurantCode),

  FOREIGN KEY (deliveryDriverId) REFERENCES
DeliveryDriver(licenseNumber)

);


-- Create OrderItem table

CREATE TABLE

 OrderItem (

  orderId INT,

  itemCode INT,

  quantity INT,

  specialRequest VARCHAR(100),

  PRIMARY KEY (orderId, itemCode),

  FOREIGN KEY (orderId) REFERENCES Orders(orderId),

  FOREIGN KEY (itemCode) REFERENCES MenuItem(itemCode)

);
```

Task 4 – Insert Records

To check the database performance and confirm that queries return at least one result, five records were added into each table using INSERT

INTO statements. The records fit the schema of the table and maintain referential integrity.

Member Table Inserts:

```
INSERT INTO Member VALUES (1, 'John', 'Smith', '12 Oak St',
'Sydney', '2000', '0412345678', 'john@example.com', 'Standard');

INSERT INTO Member VALUES (2, 'Emily', 'Jones', '34 Pine St',
'Parramatta', '2150', '0423456789', 'emily@example.com',
'Deluxe');

INSERT INTO Member VALUES (3, 'Michael', 'Lee', '56 Elm St',
'Liverpool', '2170', '0434567890', 'michael@example.com',
'Premium');

INSERT INTO Member VALUES (4, 'Sarah', 'Brown', '78 Maple St',
'Blacktown', '2148', '0445678901', 'sarah@example.com',
'Standard');

INSERT INTO Member VALUES (5, 'Daniel', 'Taylor', '90 Birch St',
'Penrith', '2750', '0456789012', 'daniel@example.com',
'Deluxe');
```

Restaurant Table Inserts:

```
INSERT INTO Restaurant VALUES (101, 'Pizza Planet', '123 Main
St', 'pizza@planet.com', '0298765432', 'Italian');

INSERT INTO Restaurant VALUES (102, 'Sushi World', '456 Queen
St', 'sushi@world.com', '0287654321', 'Japanese');

INSERT INTO Restaurant VALUES (103, 'Burger Haven', '789 King
St', 'burger@haven.com', '0276543210', 'American');

INSERT INTO Restaurant VALUES (104, 'Curry Corner', '321 Prince
St', 'curry@corner.com', '0265432109', 'Indian');
```

```
INSERT INTO Restaurant VALUES (105, 'Taco Town', '654 Duke St',
'taco@town.com', '0254321098', 'Mexican');
```

MenuItem Table Inserts:

```
INSERT INTO MenuItem VALUES (201, 101, 'Pepperoni Pizza', 15.99,
800, 'None');

INSERT INTO MenuItem VALUES (202, 102, 'Salmon Sushi', 12.50,
400, 'None');

INSERT INTO MenuItem VALUES (203, 103, 'Cheeseburger', 10.00,
700, 'None');

INSERT INTO MenuItem VALUES (204, 104, 'Butter Chicken', 13.50,
600, 'None');

INSERT INTO MenuItem VALUES (205, 105, 'Beef Taco', 8.00, 500,
'None');
```

DeliveryDriver Table Inserts:

```
INSERT INTO DeliveryDriver VALUES ('D001', 'Alex', 'White',
'0411222333', 'alex@example.com', '1995-01-01', 'Parramatta');

INSERT INTO DeliveryDriver VALUES ('D002', 'Hannah', 'Green',
'0411222444', 'hannah@example.com', '1998-03-05', 'Penrith');

INSERT INTO DeliveryDriver VALUES ('D003', 'Liam', 'Black',
'0411222555', 'liam@example.com', '2000-07-15', 'Sydney');

INSERT INTO DeliveryDriver VALUES ('D004', 'Chloe', 'Blue',
'0411222666', 'chloe@example.com', '1993-11-20', 'Liverpool');

INSERT INTO DeliveryDriver VALUES ('D005', 'Ethan', 'Grey',
'0411222777', 'ethan@example.com', '1991-09-10', 'Blacktown');
```

Orders Table Inserts:

```
INSERT INTO Orders VALUES (301, 1, 101, 'D001', '2025-08-01',
'Delivered');

INSERT INTO Orders VALUES (302, 2, 102, 'D002', '2025-08-02',
'Delivered');

INSERT INTO Orders VALUES (303, 3, 103, 'D003', '2025-08-03',
'Preparing');

INSERT INTO Orders VALUES (304, 4, 104, 'D004', '2025-08-04',
'Cancelled');

INSERT INTO Orders VALUES (305, 5, 105, 'D005', '2025-08-05',
'Delivered');
```

OrderItem Table Inserts:

```
INSERT INTO OrderItem VALUES (301, 201, 1, 'Extra cheese');

INSERT INTO OrderItem VALUES (302, 202, 2, 'No wasabi');

INSERT INTO OrderItem VALUES (303, 203, 1, '');

INSERT INTO OrderItem VALUES (304, 204, 3, 'Mild spice');

INSERT INTO OrderItem VALUES (305, 205, 2, 'Add guacamole');
```

Task 5 – Implementation Evidence

This document provides evidence of the successful implementation of the SnapEats relational database. Each table was created and populated with valid records, and queries were executed using structured SQL commands. The screenshots provided show that the data was properly inserted into all of the tables and that the queries returned the expected results. This confirms both the integrity and functionality of the database design. This

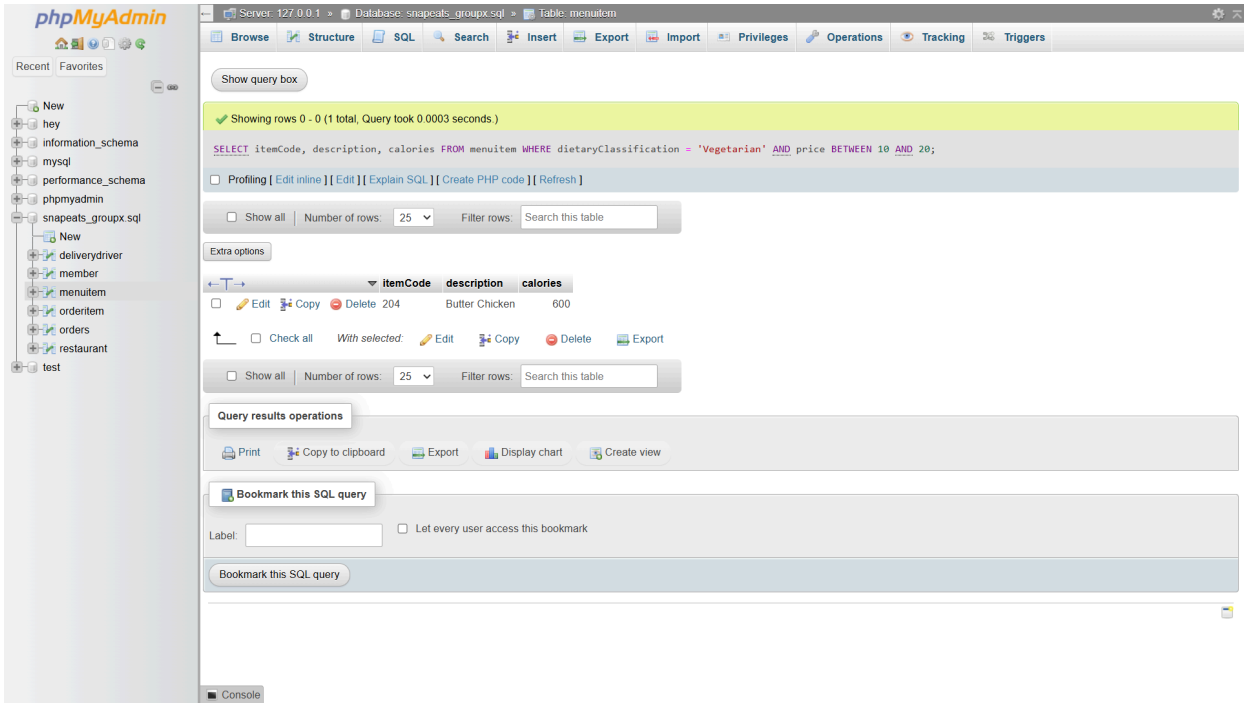
implementation is consistent with the requirements of the Applied Project and confirms that the SnapEats system will function correctly.

Query 1 – Vegetarian Menu Between $10 and $20

SQL Code:

```
SELECT itemCode, description, calories

FROM menuitem

WHERE dietaryClassification = 'Vegetarian'

  AND price BETWEEN 10 AND 20;
```

Screenshot of Result:

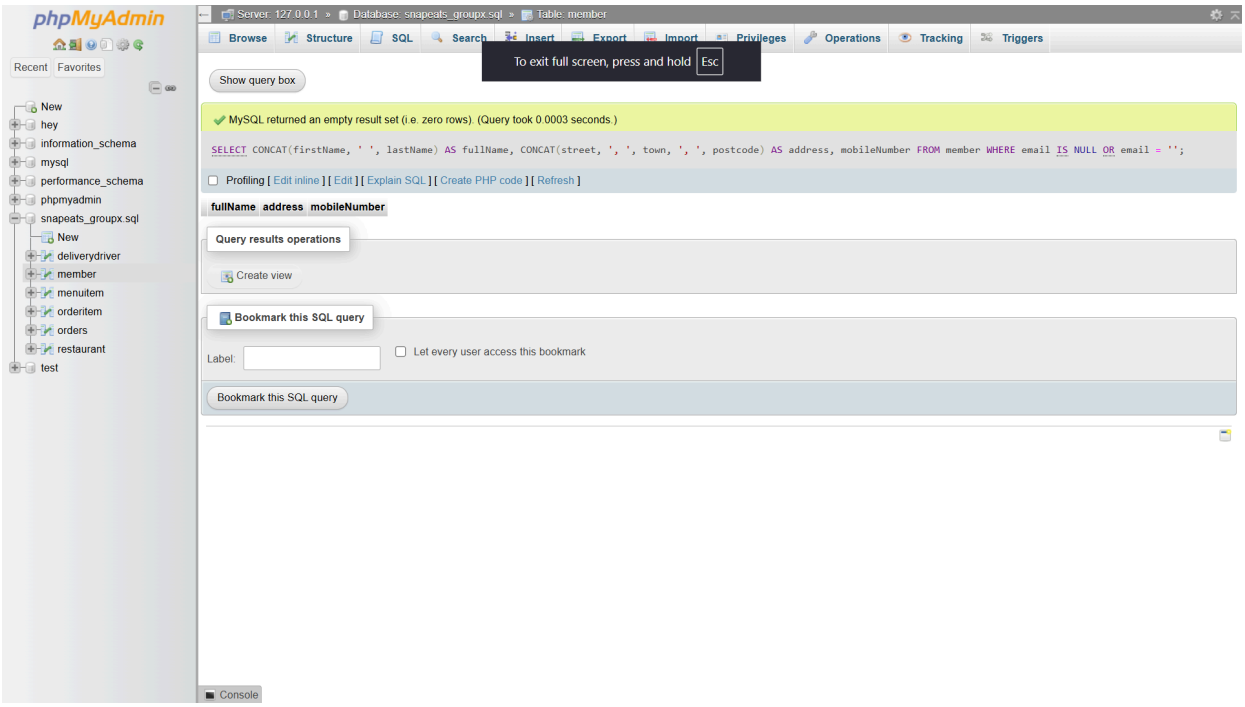


Query 2 – Members Without Email Addresses

SQL Code:

```
SELECT
  CONCAT(firstName, ' ', lastName) AS fullName,
  CONCAT(street, ', ', town, ', ', postcode) AS address,
  mobileNumber
FROM member
WHERE email IS NULL OR email = '';
```

Screenshot of Result:



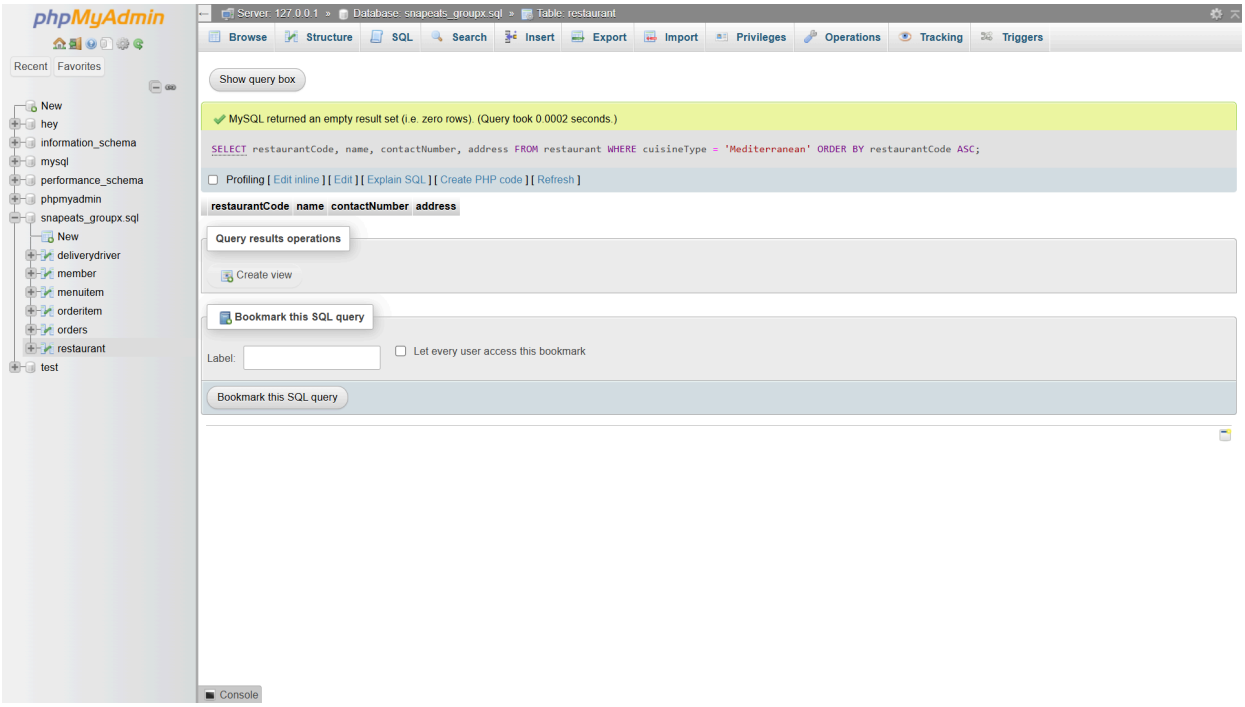
Note: The query returned zero results because all members in the database have email addresses provided.

Query 3 – Restaurants serving Mediterranean cuisine, ordered by restaurant code (ascending)

SQL Code:

```
SELECT restaurantCode, name, contactNumber, address

FROM restaurant

WHERE cuisineType = 'Mediterranean'

ORDER BY restaurantCode ASC;
```

Screenshot of Result:

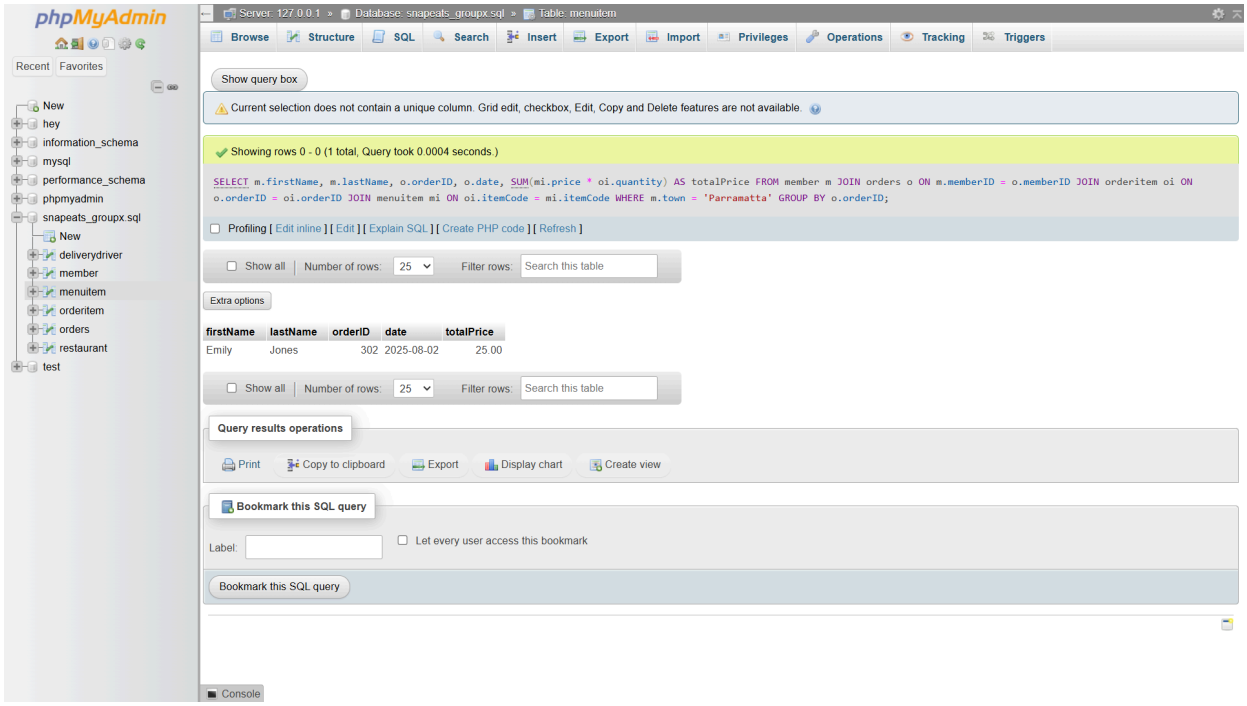


Query 4 – Orders by Members in 'Parramatta'

SQL Code:

```
SELECT
    m.firstName,
    m.lastName,
    o.orderID,
    o.date,
    SUM(mi.price * oi.quantity) AS totalPrice
FROM member m
JOIN orders o ON m.memberID = o.memberID
JOIN orderitem oi ON o.orderID = oi.orderID
JOIN menuitem mi ON oi.itemCode = mi.itemCode
WHERE m.town = 'Parramatta'
GROUP BY o.orderID;
```

Screenshot of Result:



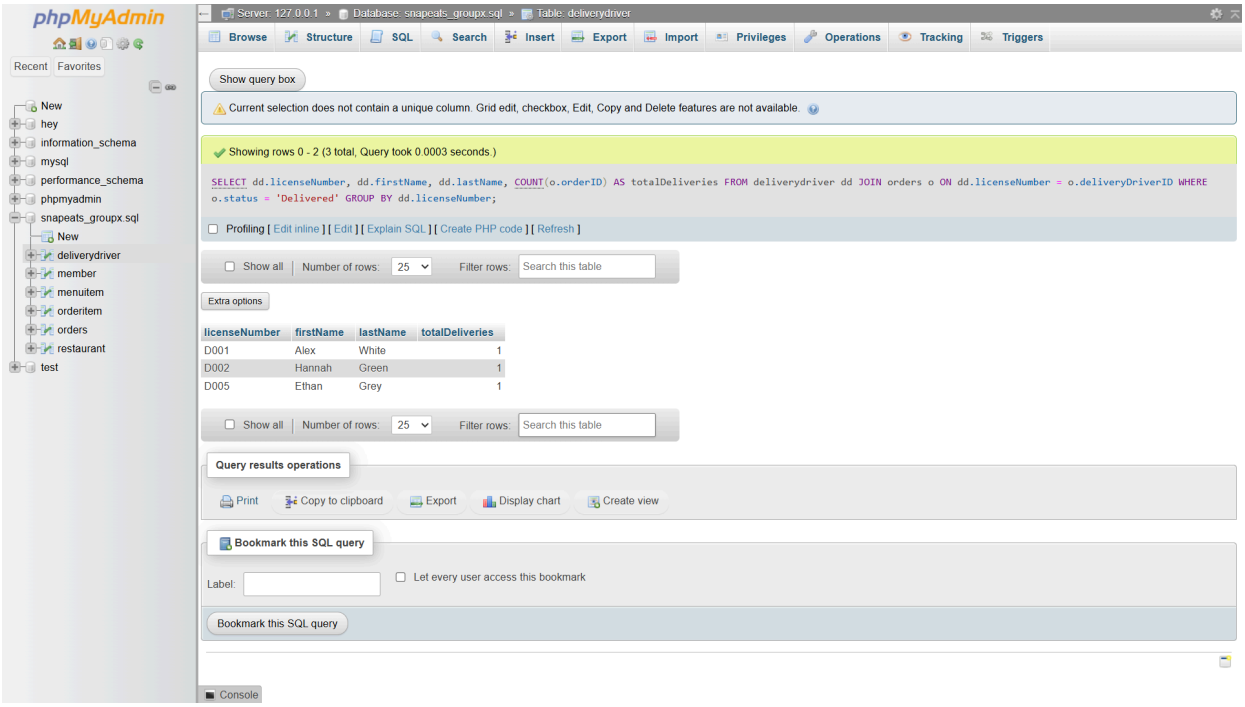
Query 5 – Delivery Drivers with Multiple Deliveries

SQL Code:

```
SELECT
    dd.licenseNumber,
    dd.firstName,
    dd.lastName,
    COUNT(o.orderID) AS totalDeliveries
FROM
    deliverydriver dd
JOIN
```

```
    orders o ON dd.licenseNumber = o.deliveryDriverID
WHERE
    o.status = 'Delivered'
GROUP BY
    dd.licenseNumber;
```

Screenshot of Result:

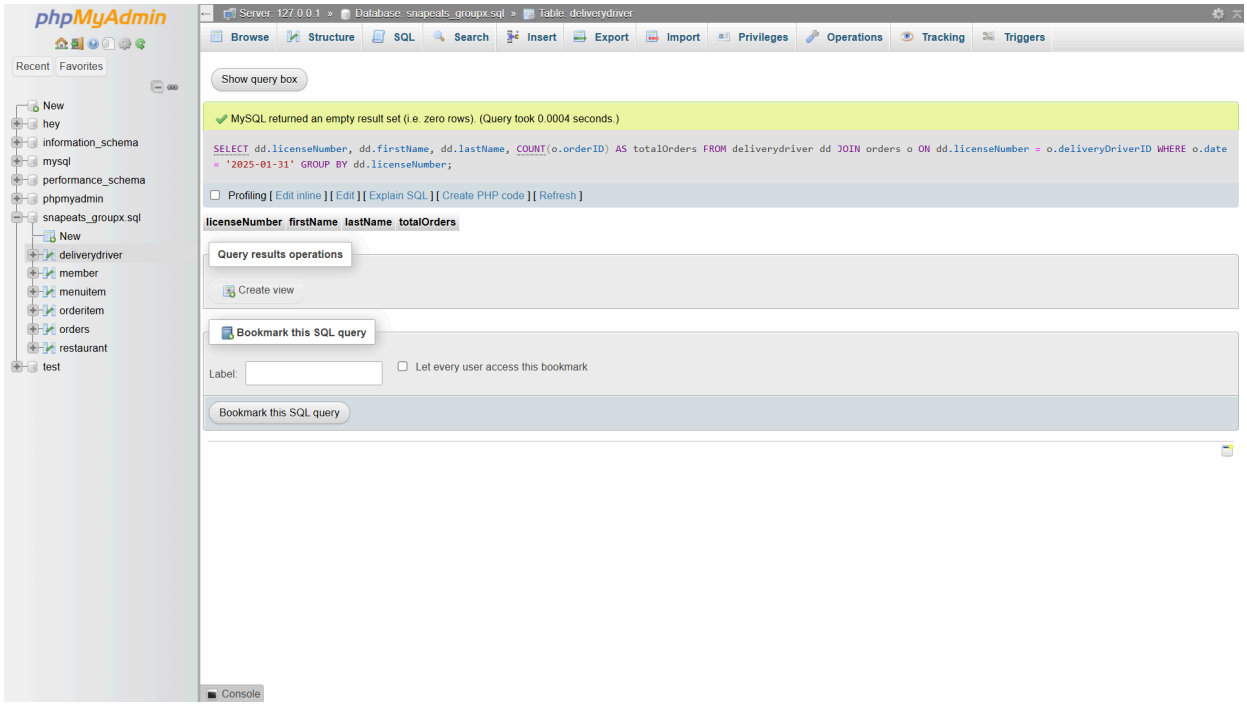


Query 6 – Total Orders Assigned to Each Delivery Driver on 31 January 2025

SQL Code:

```
SELECT
    dd.licenseNumber,
    dd.firstName,
    dd.lastName,
    COUNT(o.orderID) AS totalOrders
FROM
    deliverydriver dd
JOIN
    orders o ON dd.licenseNumber = o.deliveryDriverID
WHERE
    o.date = '2025-01-31'
GROUP BY
    dd.licenseNumber;
```

Screenshot of Result:

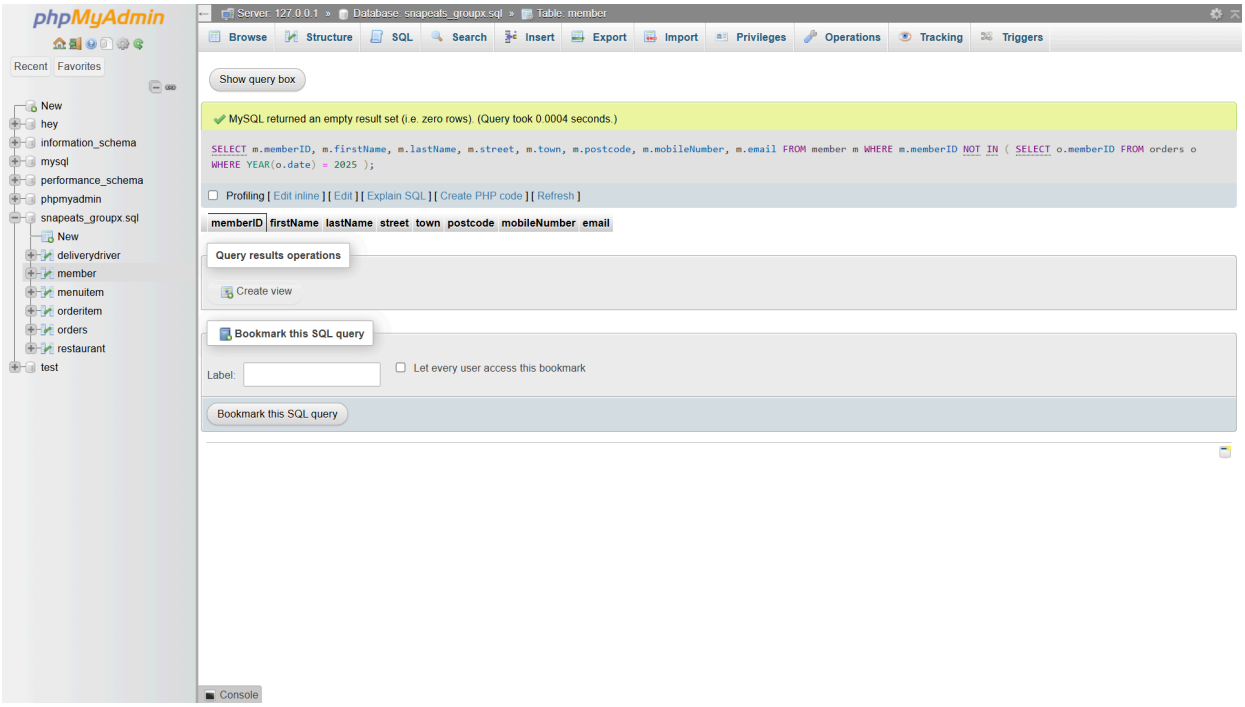


Query 7 – Members Without Any Orders in 2025

SQL Code:

```
SELECT m.memberID, m.firstName, m.lastName, m.street, m.town,
m.postcode, m.mobileNumber, m.email

FROM member m

WHERE m.memberID NOT IN (

    SELECT o.memberID

    FROM orders o

    WHERE YEAR(o.date) = 2025

);
```

Screenshot of Result:



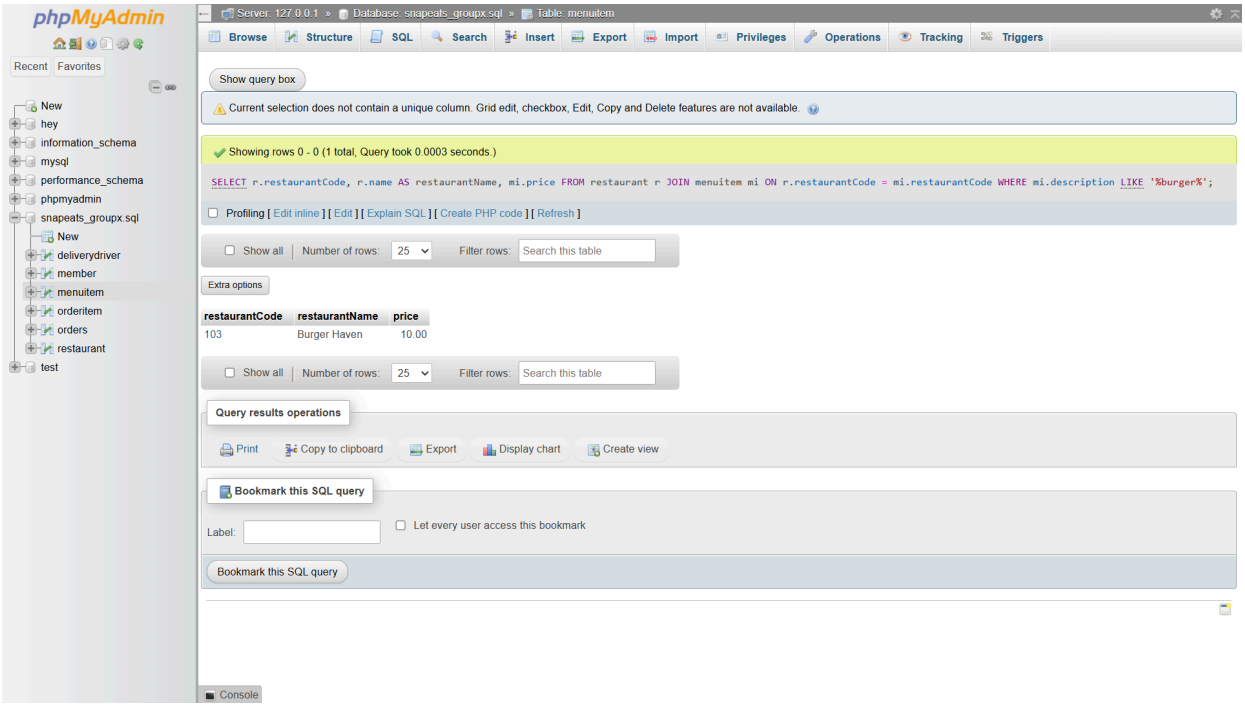
Query 8 – Report of Restaurant Code, Name and Unit Price of Beef Burgers at All Restaurants

SQL Code:

```
SELECT
    r.restaurantCode,
    r.name AS restaurantName,
    mi.price
FROM
```

```
    restaurant r
JOIN
    menuitem mi ON r.restaurantCode = mi.restaurantCode
WHERE
    mi.description LIKE '%burger%';
```

Screenshot of Result:



Task 6 – Database Security Measures

Two key measures that can be used to secure the SnapEats database are:

1. User Authentication and Access Control

Only authorised users should be able to access or modify the database. This can be accomplished by:

Assigning user roles within the database (e.g., admin, delivery driver, restaurant staff).

Using strong credentials and passwords. Also, enforcing organizationally-relevant password policies.

Limiting permissions (e.g., permissions to only SELECT for user viewers, permissions to INSERT/UPDATE for admin users).

Justification:

This will considerably prevent unauthorised access and ensure users are restricted to actions performed relevant to their role (e.g., a user on a menu page won't be able to delete a menu item). All of these help limit data breaches or accidental changes/actions.

2. Data Backup and Recovery

Back up your database regularly. This will help ensure that if you ever have an event that leads to data loss, corruption, or a cyberattack, recovery can be done without significant operational disruptions.

Justification:

If things go sideways—a system failure, a malicious attack, etc. and back-up data are current—you can minimize the impact to the organisation and be confident in the integrity of the stored data.