

Code Documentation for Restaurant Chain Database System

Table of Contents

1	Environment setup.....	5
2	Create Database.....	8
3	Create Tables.....	10
4	Insert values to the database tables.....	45
5	Question and answers.....	56
6	Report and Analysis	63

List of Figures

Figure 1:	XAMPP Control Panel application	5
Figure 2:	Apache and MySQL starting position	6
Figure 3:	Initial start of apache and MySQL	6
Figure 4:	MYSQL admin button.....	7
Figure 5:	phpMyAdmin localhost interface	7
Figure 6:	shell button for Command Line Interface.....	8
Figure 7 :	new button for creating database	9
Figure 8 :	restaurant_chain_db database	9
Figure 9:	created the restaurant_chain_db database.....	10
Figure 10:	create table query of Menu_item	11
Figure 11:	structure of Menu_item table.....	12
Figure 12 :	select query for menu_item	12
Figure 13:	create table query of category.....	13
Figure 14:	structure of category table	14
Figure 15:	select query for category table.....	14
Figure 16:	create table query of customer.....	15

Figure 17: structure of customer table	16
Figure 18: select query for customer table.....	16
Figure 19: create table query of reservation.....	17
Figure 20: structure of reservation table	17
Figure 21: select query for reservation table.....	18
Figure 22: create table query of restaurant location.....	19
Figure 23: structure of restaurant location table	19
Figure 24: select query for restaurant location table.....	20
Figure 25: create table query of Sales	21
Figure 26: structure of sales table	21
Figure 27: select query for sales table.....	22
Figure 28: create table query of inventory	23
Figure 29: structure of inventory table.....	23
Figure 30: select query for inventory table	24
Figure 31: create table query of ingredient category	25
Figure 32: structure of ingredient category	25
Figure 33: select query for ingredient category table.....	26
Figure 34: create table query of menu ingredient	27
Figure 35: structure of menu ingredient.....	27
Figure 36: select query for menu ingredient table	28
Figure 37: create table query of employee.....	29
Figure 38: structure of menu ingredient table	29
Figure 39: select query for employee table	30
Figure 40: create table query of shift	31
Figure 41: structure of shift table	31
Figure 42: select query for shift table	32
Figure 43: create table query of supplier	33
Figure 44: structure of supplier table	33
Figure 45: select query for supplier table.....	34
Figure 46: create table query of supply order	35
Figure 47:structure of supply order table.....	35
Figure 48: select query for supply order table	36
Figure 49: create table query of payment.....	37
Figure 50: structure of payment table	37
Figure 51: select query for payment table.....	38
Figure 52: create table query of feedback.....	39
Figure 53: structure of feedback table.....	39
Figure 54: select query for feedback table	40
Figure 55: create table query of promotion.....	41
Figure 56: structure of promotion table	41
Figure 57: select query for promotion table.....	42
Figure 58: create table query of promotion menu item.....	43
Figure 59: structure of promotion menu item table	43
Figure 60: select query for junction table	44
Figure 61: insert value to the customer table	45
Figure 62: insert values to the category table	46
Figure 63: insert values to the menu item	46

Figure 64: insert values to the restaurant location	47
Figure 65: insert values to the reservation	48
Figure 66: insert values to the sales table	48
Figure 67: insert values to the ingredient category	49
Figure 68: insert values to the inventory table	50
Figure 69: insert values to the menu ingredient table	50
Figure 70: insert values to the employee table.....	51
Figure 71: insert values to the shift table	51
Figure 72: insert values to the supplier table	52
Figure 73: insert values to the supply order table	53
Figure 74: insert values to the payment table	53
Figure 75: insert values to the feedback table.....	54
Figure 76: insert values to the promotion table.....	55
Figure 77: insert values to the promotion menu item table.....	55
Figure 78: Total sales for each menu category	56
Figure 79: output of Total sales for each menu category	57
Figure 80: Top 10 most popular menu items	57
Figure 81: output of Top 10 most popular menu items.....	58
Figure 82: Customer reservation history.....	59
Figure 83: output of Customer reservation history	59
Figure 84: Current inventory levels by ingredient	60
Figure 85: output of Current inventory levels by ingredient.....	61
Figure 86: Revenue by restaurant location	62
Figure 87: output of Revenue by restaurant location	62
Figure 88: daily revenue report code	63
Figure 89: output of daily revenue code	64
Figure 90: itemized sales query code.....	65
Figure 91: outcome of itemized sales	66
Figure 92: inventory report query code.....	68
Figure 93: results of inventory margin.....	69

1 Environment setup

We need XAMPP Control Panel for our process. This implementation made on Apporto platform.

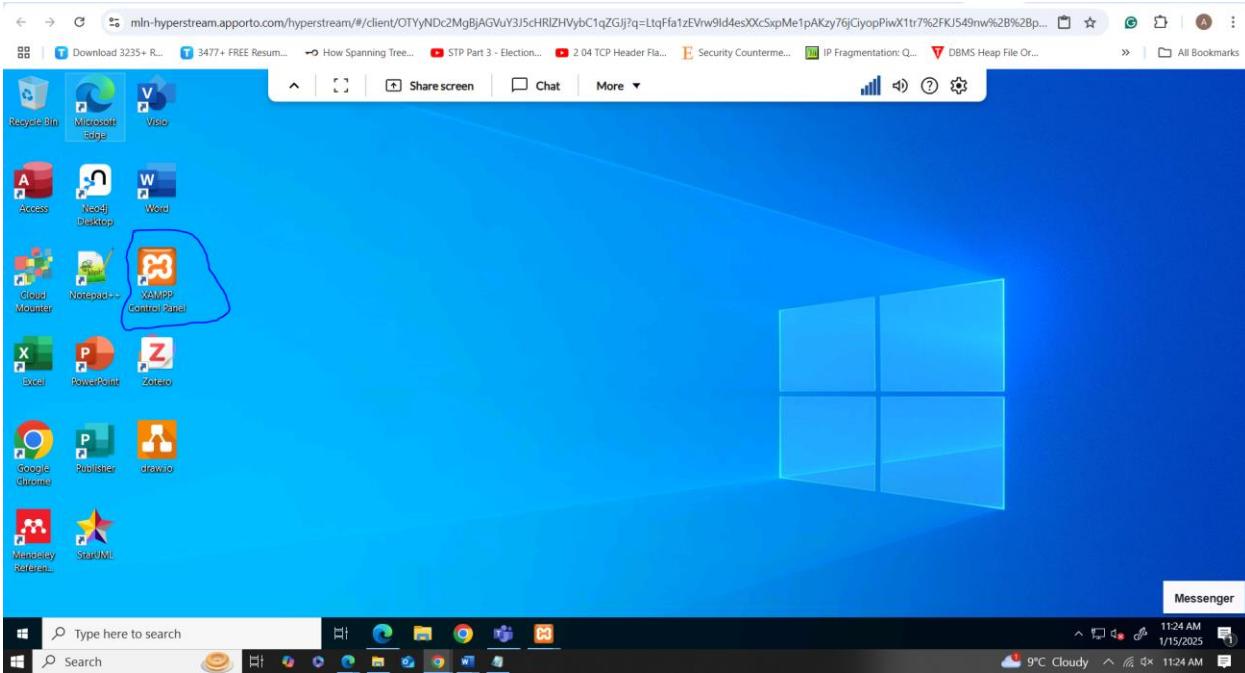


Figure 1: XAMPP Control Panel application

Inside that Apache and MySQL need to be start first.

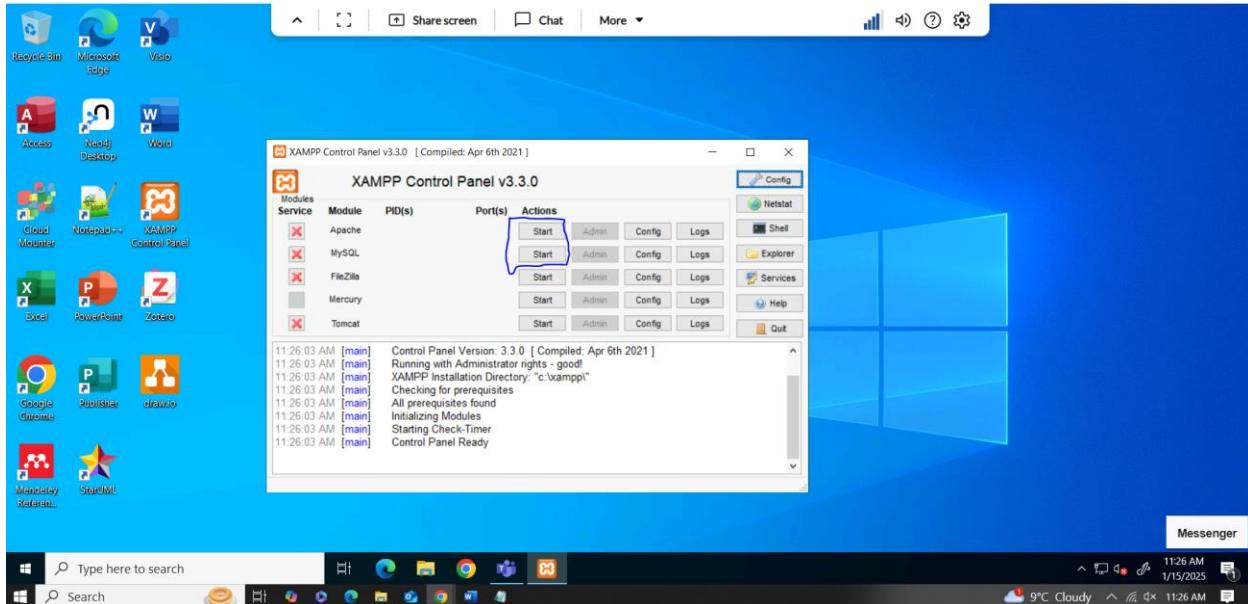


Figure 2: Apache and MySQL starting position

After the start, we can see Apache and MySQL were prompt green.

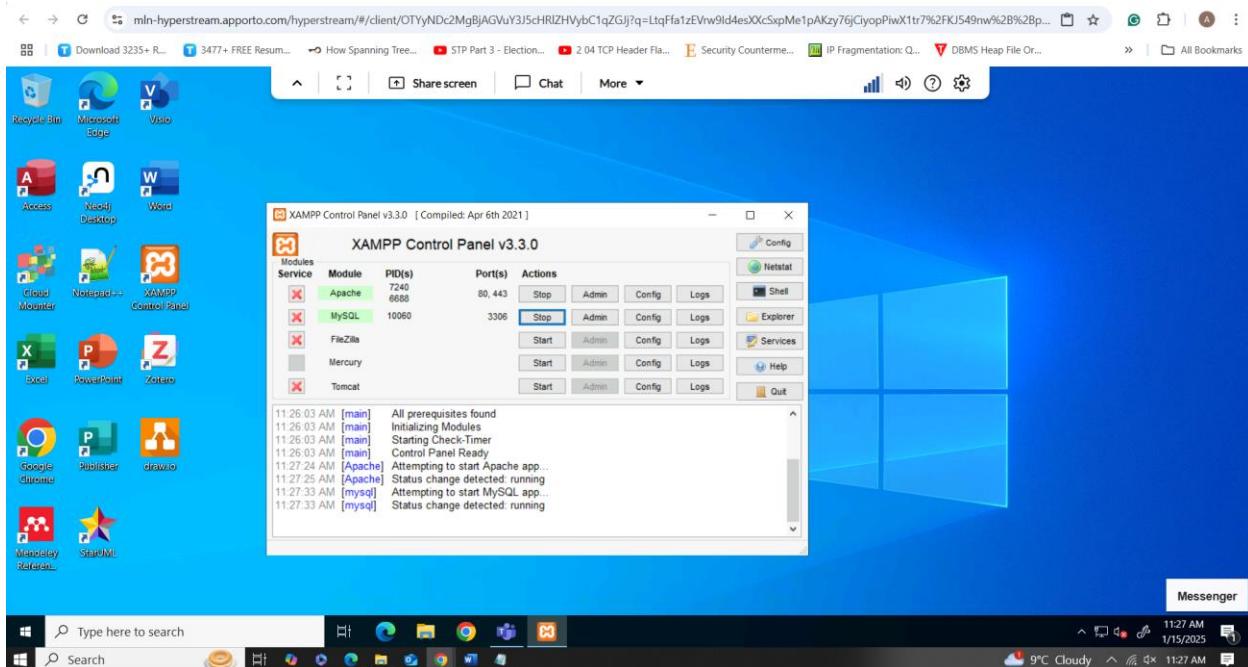


Figure 3: Initial start of apache and MySQL

After we need to click MySQL admin button to see the GUI of the phpMyAdmin.

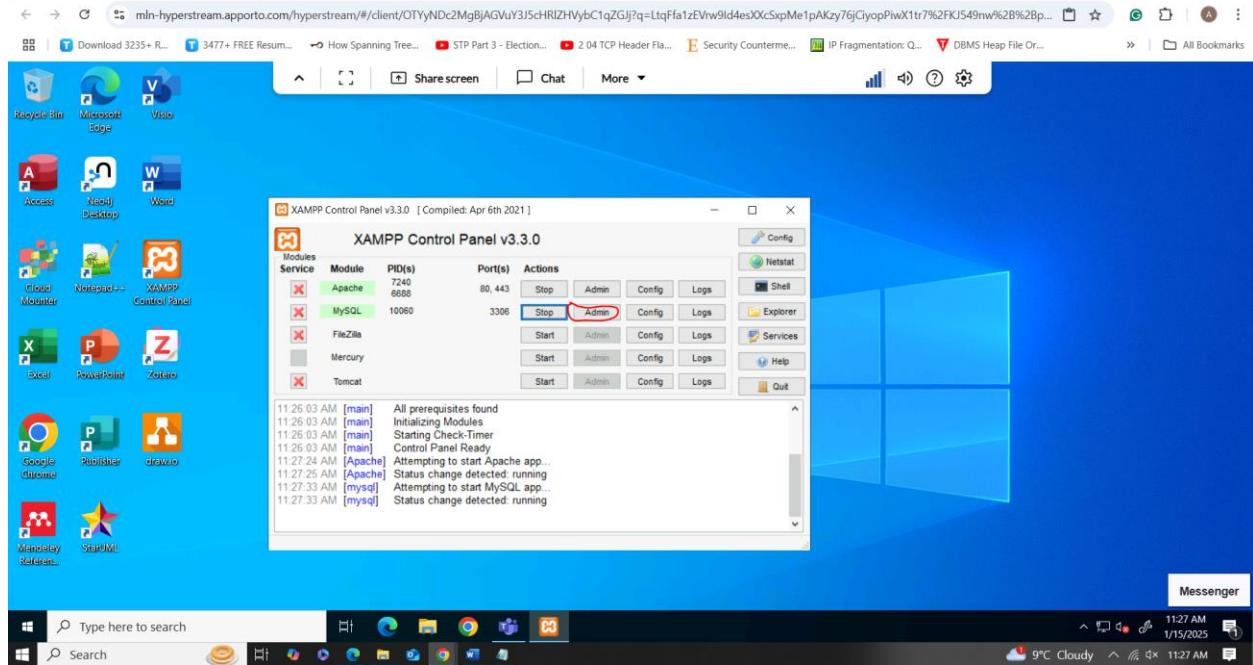


Figure 4: MySQL admin button

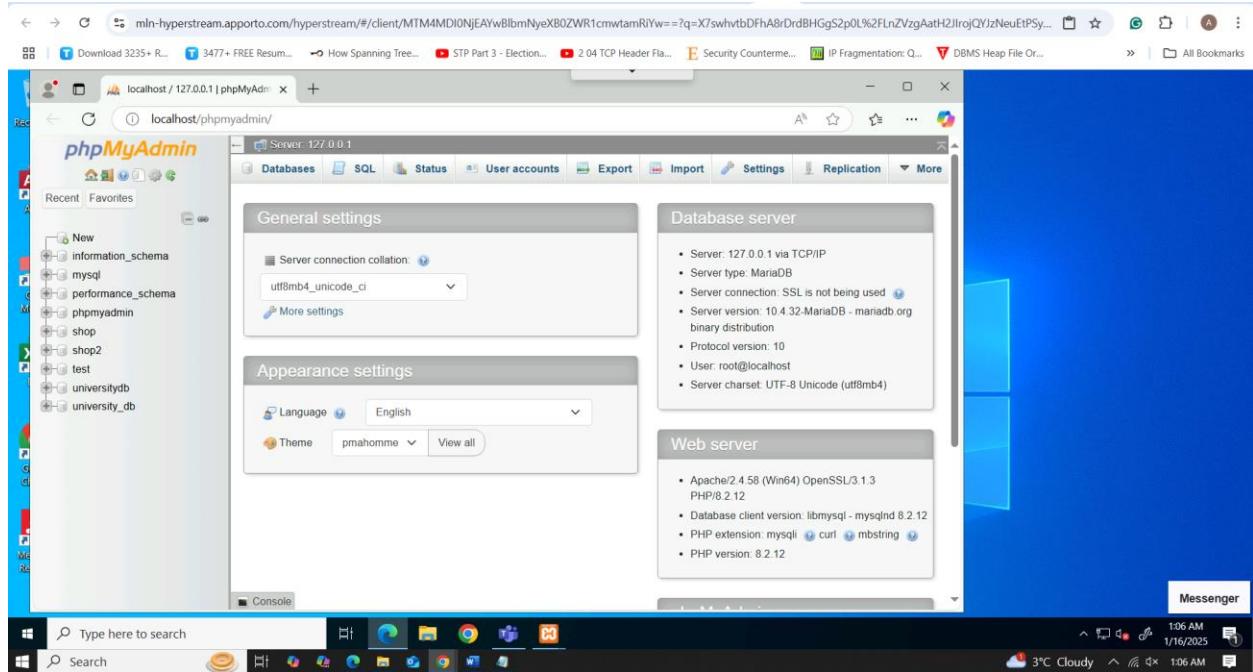


Figure 5: phpMyAdmin localhost interface

We can use shell also for entire process. For better view and understanding the queries, we used Admin interface (GUI).

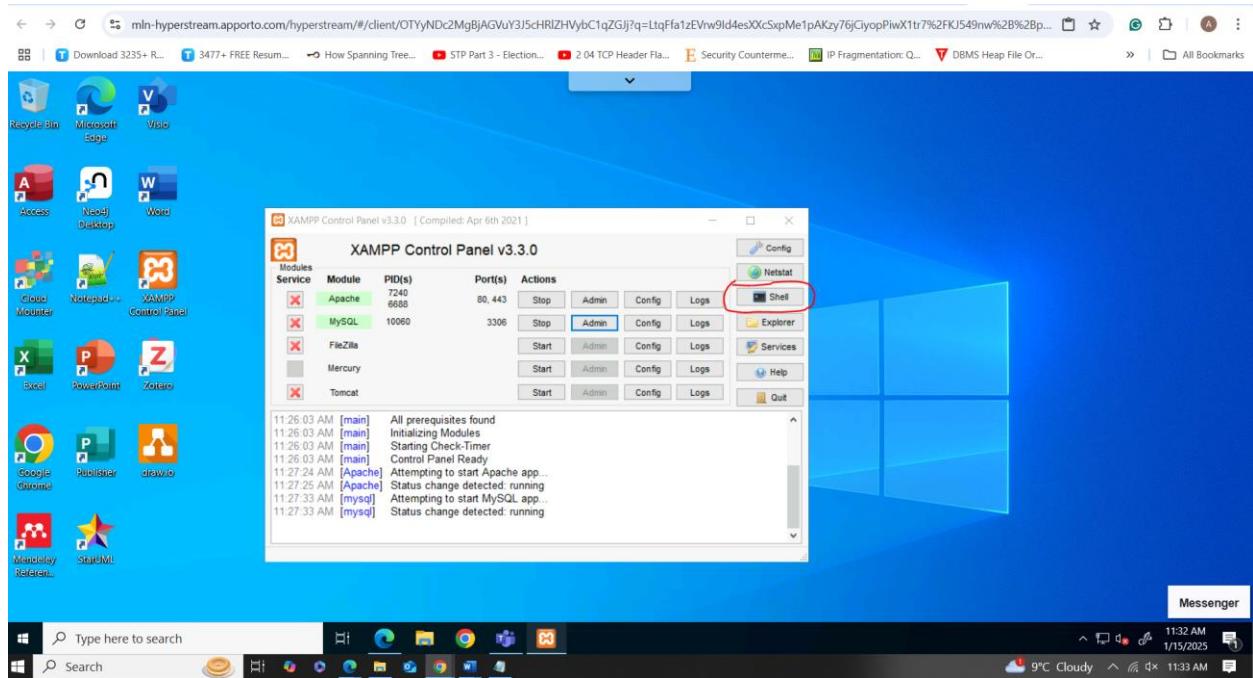


Figure 6: shell button for Command Line Interface

2 Create Database

First, we need to create database.

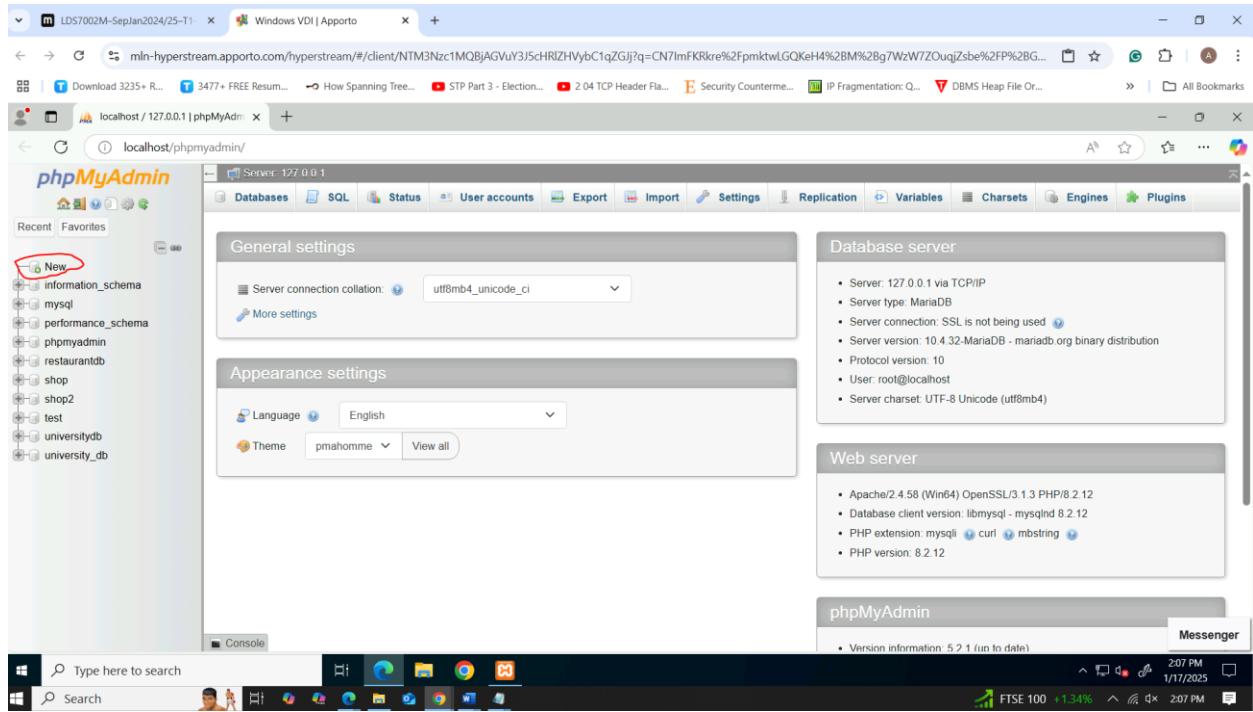


Figure 7 : new button for creating database

We named it Restaurant_Chain_DB.

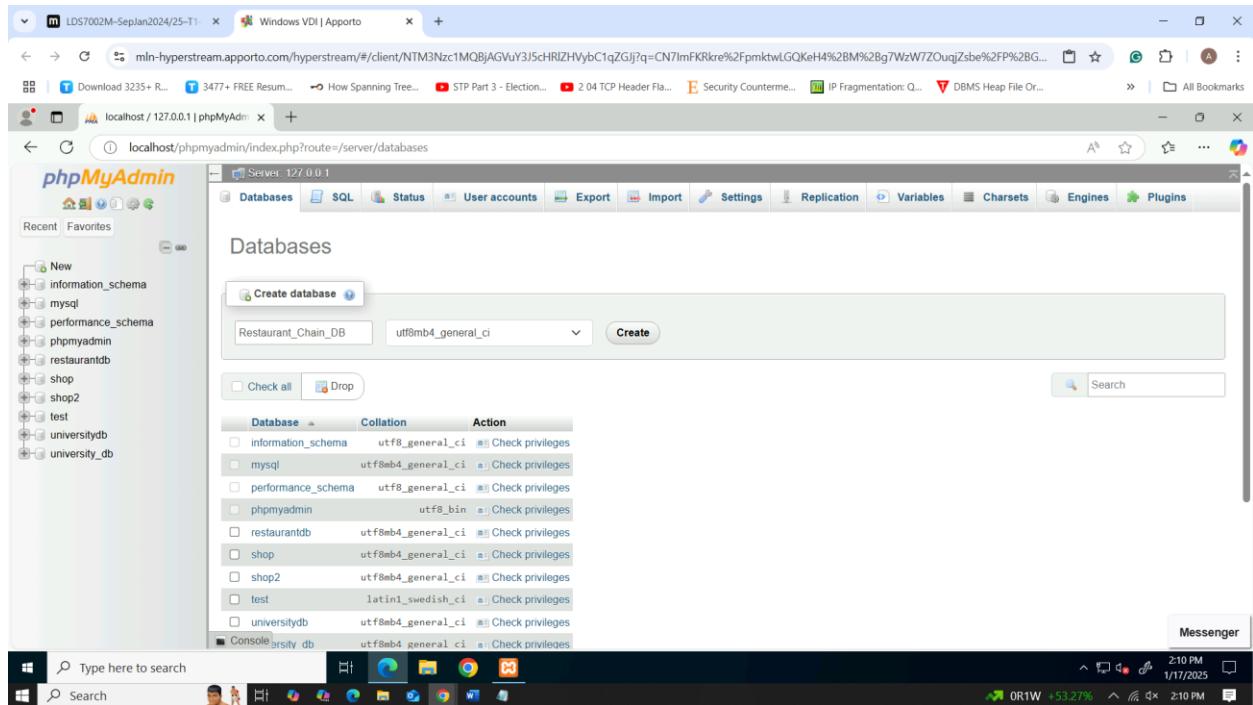


Figure 8 : restaurant_chain_db database

In left side we can see restaurant_chain_db database.

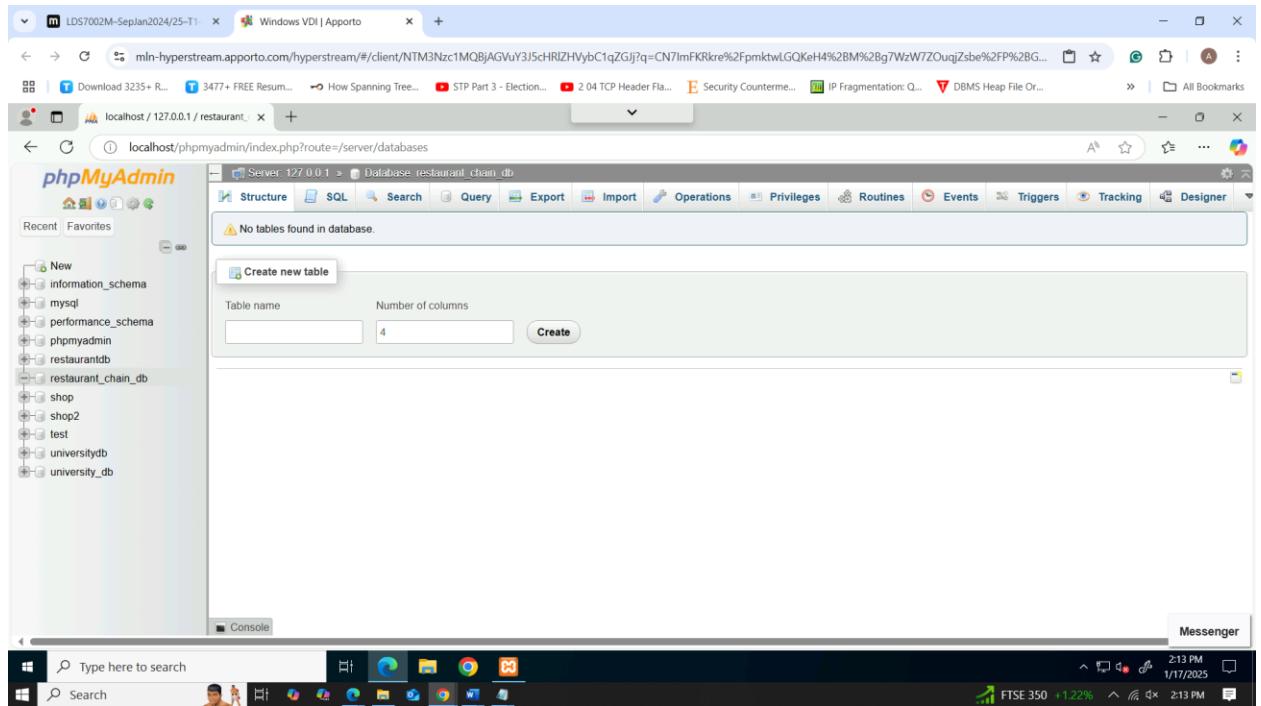


Figure 9: created the `restaurant_chain_db` database

3 Create Tables

➤ `Menu_item`

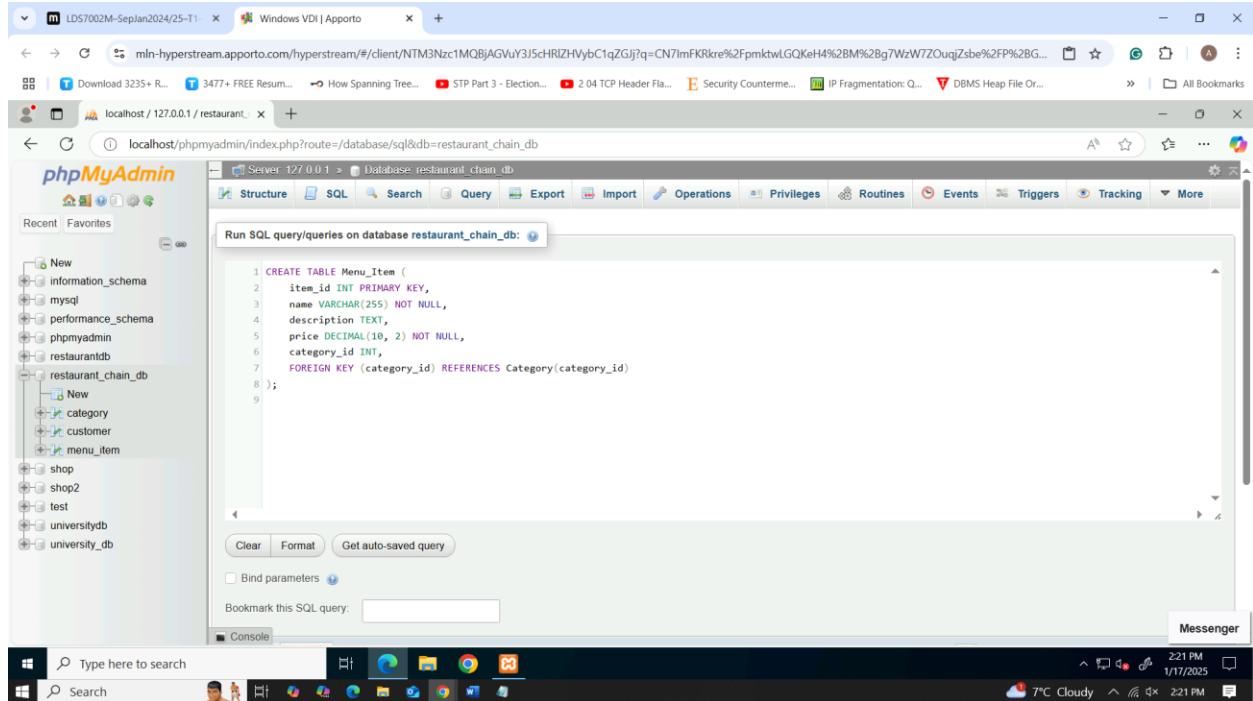


Figure 10: create table query of Menu_item

This SQL statement creates a `Menu_Item` table with columns: `item_id` (auto-incremental integer unique key), `name` (string that cannot be empty and has the length of 255 characters maximum), `description` (text intended for the display of detailed descriptions), `price` (non-null decimal of the 10S scale, 2D precision), and `category_id` (link to a category). The `FOREIGN KEY (category_id) REFERENCES Category(category_id)` guarantees that the `category_id` in the `Menu_Item` table matches the existing one in the `Category` table. Make certain that a table named `Category` already exists and it creates a column known as `category_id` upon which the key relationship is based.

The screenshot shows the phpMyAdmin interface for the `menu_item` table in the `restaurant_chain_db` database. The table structure is as follows:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	<code>item_id</code>	int(11)	utf8mb4_general_ci	No	None				Change Drop More
2	<code>name</code>	varchar(255)	utf8mb4_general_ci	No	None				Change Drop More
3	<code>description</code>	text	utf8mb4_general_ci	Yes	NULL				Change Drop More
4	<code>price</code>	decimal(10,2)		No	None				Change Drop More
5	<code>category_id</code>	int(11)		Yes	NULL				Change Drop More

Indexes:

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Rename Drop	PRIMARY	BTREE	Yes	No	<code>item_id</code>	0	A	No	
Edit Rename Drop	category_id	BTREE	No	No	<code>category_id</code>	0	A	Yes	

Figure 11: structure of Menu_item table

The screenshot shows the results of a `SELECT * FROM menu_item;` query. The results are as follows:

	<code>item_id</code>	<code>name</code>	<code>description</code>	<code>price</code>	<code>category_id</code>
1	1	Margherita Pizza	Classic cheese pizza	8.99	1
2	2	Pepperoni Pizza	Pepperoni and cheese	9.99	1
3	3	Veggie Burger	Grilled veggie patty	7.99	2
4	4	Beef Burger	Classic beef burger	8.49	2
5	5	Caesar Salad	Fresh romaine with Caesar dressing	6.99	3
6	6	Greek Salad	Feta, olives, and vegetables	7.49	3
7	7	Spaghetti Bolognese	Pasta with meat sauce	10.99	4
8	8	Fettuccine Alfredo	Creamy Alfredo sauce	11.49	4
9	9	Chicken Wings	Spicy buffalo wings	8.99	5
10	10	Fish Tacos	Grilled fish with toppings	9.49	5

Figure 12 : select query for menu_item

➤ Category

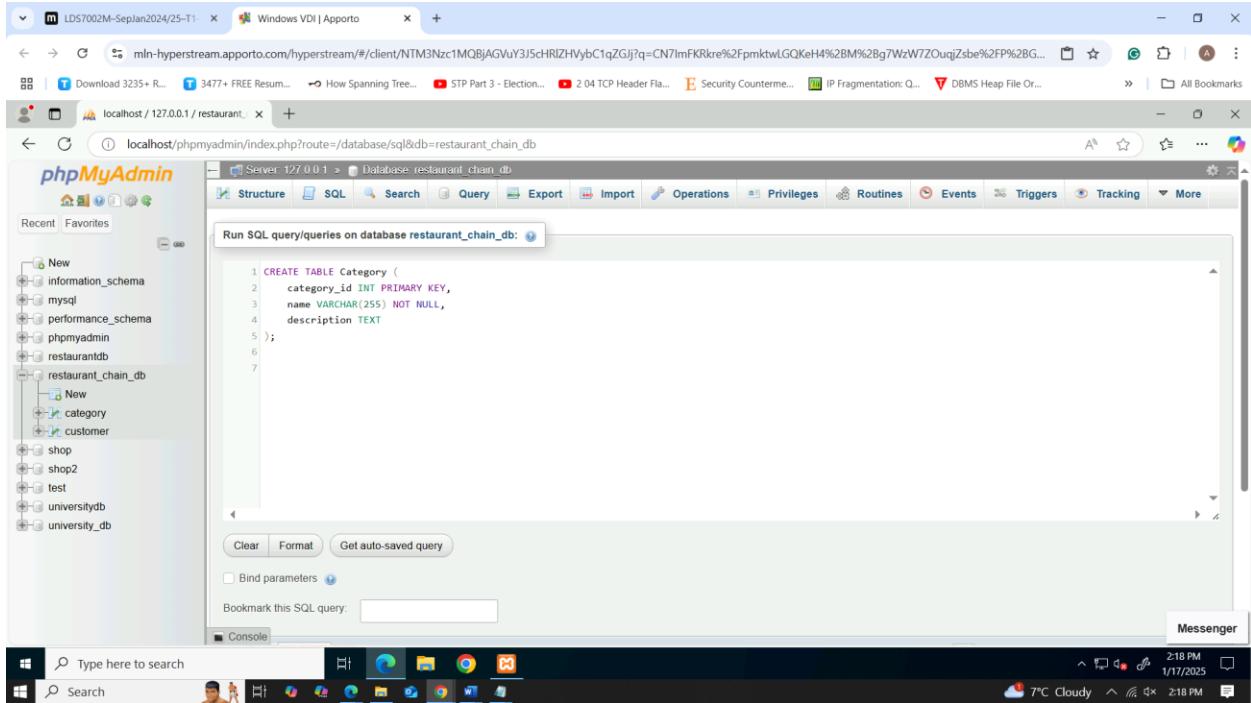


Figure 13: create table query of category

The Category table organizes menu items into distinct groups with three columns: category_id (an integer field that is Primary key and auto incremented, used to uniquely identify each category), name (string field for category name which can contain up to 255 characters) and description (used to contain additional information, actually a text field). This table makes the items appear consistent and it can easily be categorized on the menu. The Menu_Item table use the category_id in it which is a foreign key that helps in relating the items to the appropriate categories to avoid data inconsistency as well as to enhance easy sorting of the menu categories.

The screenshot shows the phpMyAdmin interface for the `category` table in the `restaurant_chain_db` database. The table structure is as follows:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	<code>category_id</code>	int(11)	utf8mb4_general_ci		No	None			Change Drop More
2	<code>name</code>	varchar(255)	utf8mb4_general_ci		No	None			Change Drop More
3	<code>description</code>	text	utf8mb4_general_ci		Yes	NULL			Change Drop More

Below the table structure, there is a section for indexes:

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Rename Drop	PRIMARY	BTREE	Yes	No	<code>category_id</code>	0	A	No	

At the bottom, there is a button to "Create an index on 1 columns".

Figure 14: structure of category table

The screenshot shows the results of a `SELECT * FROM category;` query. The results are as follows:

	<code>category_id</code>	<code>name</code>	<code>description</code>
1	1	Pizza	Various types of pizzas
2	2	Burgers	Delicious grilled burgers
3	3	Salads	Fresh and healthy salads
4	4	Pasta	Classic Italian pasta dishes
5	5	Appetizers	Starters to begin your meal
6	6	Desserts	Sweet treats to end your meal
7	7	Drinks	Cold beverages
8	8	Coffee	Hot coffee drinks
9	9	Entrees	Main course dishes
10	10	Sides	Accompaniments to your meal
11	11	Seafood	Fresh seafood dishes

Figure 15: select query for category table

➤ Customer

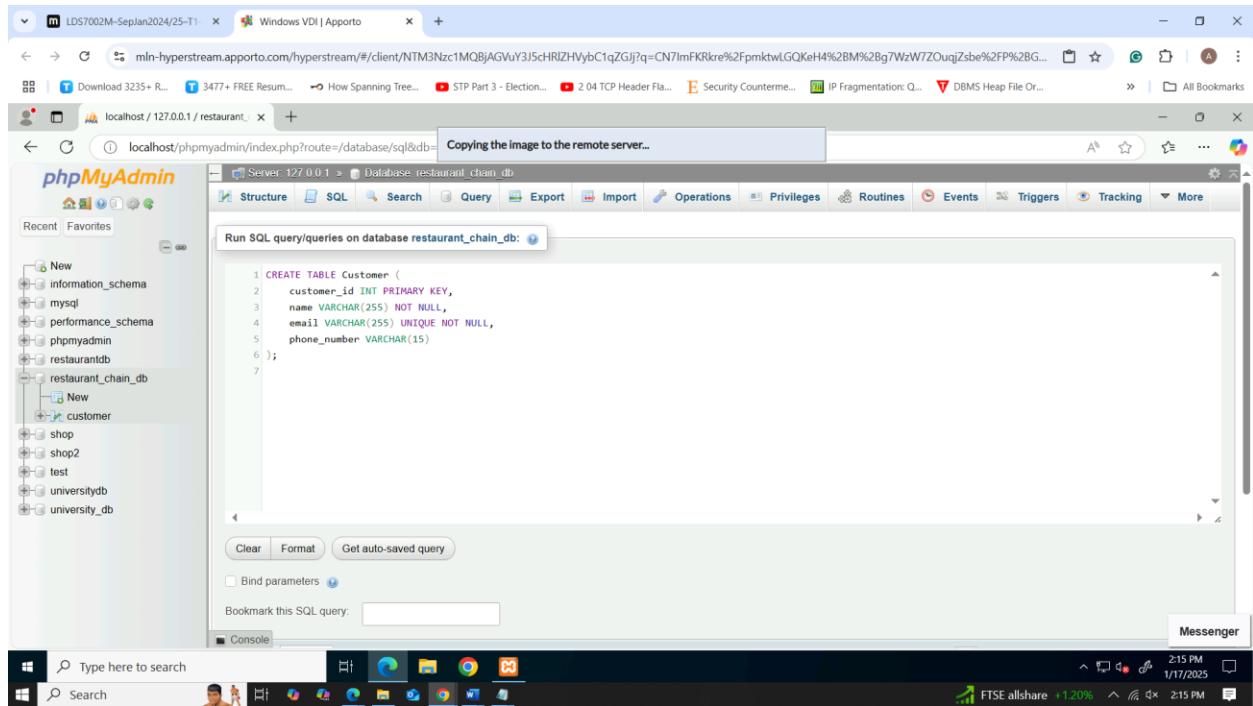


Figure 16: create table query of customer

The SQL statement generates the structure of the Customer's table consisting of five columns: customer_id, which is defined as the primary key and cannot accept null values, represents a unique integer; name, which is a mandatory string of up to 255 symbols; email, which is also a mandatory string of up to 255 symbols, cannot contain duplicate values; phone_number which is an optional string of up to 15 symbols. This structure also reduces the possibility of having duplicate complications and Governor putting restrictions on fields such as the email field.

The screenshot shows the phpMyAdmin interface for the `customer` table in the `restaurant_chain_db` database. The table structure is as follows:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	<code>customer_id</code>	<code>int(11)</code>	<code>utf8mb4_general_ci</code>		No	<code>None</code>			<code>Change</code> <code>Drop</code> <code>More</code>
2	<code>name</code>	<code>varchar(255)</code>	<code>utf8mb4_general_ci</code>		No	<code>None</code>			<code>Change</code> <code>Drop</code> <code>More</code>
3	<code>email</code>	<code>varchar(255)</code>	<code>utf8mb4_general_ci</code>		No	<code>None</code>			<code>Change</code> <code>Drop</code> <code>More</code>
4	<code>phone_number</code>	<code>varchar(15)</code>	<code>utf8mb4_general_ci</code>		Yes	<code>NULL</code>			<code>Change</code> <code>Drop</code> <code>More</code>

Indexes section:

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
<code>Edit</code> <code>Rename</code> <code>Drop</code>	<code>PRIMARY</code>	<code>BTREE</code>	Yes	No	<code>customer_id</code>	0	A	No	
<code>Edit</code> <code>Rename</code> <code>Drop</code>	<code>email</code>	<code>BTREE</code>	Yes	No	<code>email</code>	0	A	No	

Figure 17: structure of customer table

The screenshot shows the results of a `SELECT * FROM customer;` query. The results are as follows:

	<code>customer_id</code>	<code>name</code>	<code>email</code>	<code>phone_number</code>
1	1	John Doe	john.doe@example.com	1234567890
2	2	Jane Smith	jane.smith@example.com	9876543210
3	3	Alice Brown	alice.brown@example.com	5551234567
4	4	Bob Johnson	bob.johnson@example.com	5559876543
5	5	Charlie Davis	charlie.davis@example.com	4445556666
6	6	Diana Evans	diana.evans@example.com	3334445555
7	7	Ethan Foster	ethan.foster@example.com	2223334444
8	8	Fiona Green	fiona.green@example.com	1112223333
9	9	George Harris	george.harris@example.com	9998887777
10	10	Hannah Clark	hannah.clark@example.com	8889990000
11	11	Ian Martin	ian.martin@example.com	7776665555

Figure 18: select query for customer table

➤ Reservation

The screenshot shows a Windows desktop environment. In the top taskbar, there are several pinned icons and a search bar. Below the taskbar, the system tray displays the date (1/17/2025), time (2:24 PM), and weather (7°C Cloudy). The main focus is a browser window titled "localhost / 127.0.0.1 / restaurant". Inside the browser, the URL is "localhost/phpmyadmin/index.php?route=/database/sql&db=restaurant_chain_db". The page content is the phpMyAdmin interface for the "reservation" table. On the left sidebar, under the "Structure" tab, the table structure is displayed:

```

CREATE TABLE Reservation (
    reservation_id INT PRIMARY KEY,
    customer_id INT,
    location_id INT,
    reservation_date DATETIME NOT NULL,
    number_of_people INT NOT NULL,
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),
    FOREIGN KEY (location_id) REFERENCES Restaurant_Location(location_id)
);

```

The "Operations" tab is also visible at the top of the interface.

Figure 19: create table query of reservation

The screenshot shows a Windows desktop environment. The system tray indicates the date (1/17/2025), time (2:26 PM), and weather (7°C Cloudy). The main focus is a browser window titled "localhost / 127.0.0.1 / restaurant". Inside the browser, the URL is "localhost/phpmyadmin/index.php?route=/table/structure&db=restaurant_chain_db&table=reservation". The page content is the phpMyAdmin interface for the "reservation" table structure. On the left sidebar, under the "Structure" tab, the table structure is displayed:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	reservation_id	int(11)			No	None			Change Drop More
2	customer_id	int(11)			Yes	NULL			Change Drop More
3	location_id	int(11)			Yes	NULL			Change Drop More
4	reservation_date	datetime			No	None			Change Drop More
5	number_of_people	int(11)			No	None			Change Drop More

Below the table structure, there are sections for "Indexes" and "Indexes". The "Indexes" section lists three indexes:

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Rename Drop	PRIMARY	BTREE	Yes	No	reservation_id	0	A	No	
Edit Rename Drop	customer_id	BTREE	No	No	customer_id	0	A	Yes	
Edit Rename Drop	location_id	BTREE	No	No	location_id	0	A	Yes	

Figure 20: structure of reservation table

The Reservation table manages booking details with six columns: reservation_id(primary key-integer,unique identifier for each reservation), customer_id (used to link the two tables with foreign key pointing to the Customer table, customer_id used here is required), location_id (used to connect the two tables by using foreign key pointing to the Restaurant_Location table, location_id used here is also required), reservation_date DATETIME, not flexible-needed; number_of_people-int, not flexible-needed. The relationships that this table defines are simple but effective: defining how customers are related to restaurants, restaurants are related to reservations and how all of them can be related to customers to maintain data integrity as well as track all restaurants bookings effectively.

The screenshot shows a Windows desktop environment with multiple windows open. In the center is the phpMyAdmin interface, version 4.9.2, connected to a MySQL database named 'restaurant_chain_db'. The 'reservation' table is selected. A green status bar at the top indicates 'Showing rows 0 - 24 (50 total, Query took 0.0002 seconds)'. Below the status bar is a SQL query editor containing the command 'SELECT * from reservation;'. To the right of the query is a results grid displaying 24 rows of data. The columns are labeled 'reservation_id', 'customer_id', 'location_id', 'reservation_date', and 'number_of_people'. The data shows various reservation entries with IDs ranging from 1 to 10, dates between January 1st and 10th, and group sizes from 2 to 6. The phpMyAdmin interface includes a sidebar with database and table navigation, and a top menu bar with various tabs like Browse, Structure, SQL, and Export.

reservation_id	customer_id	location_id	reservation_date	number_of_people
1	1	1	2025-01-01 18:30:00	4
2	2	2	2025-01-02 19:00:00	2
3	3	1	2025-01-03 20:00:00	6
4	4	3	2025-01-04 18:00:00	3
5	5	2	2025-01-05 19:30:00	5
6	6	3	2025-01-06 18:45:00	2
7	7	1	2025-01-07 20:15:00	4
8	8	2	2025-01-08 19:45:00	3
9	9	3	2025-01-09 18:15:00	6
10	10	1	2025-01-10 19:00:00	2

Figure 21: select query for reservation table

➤ Restaurant location

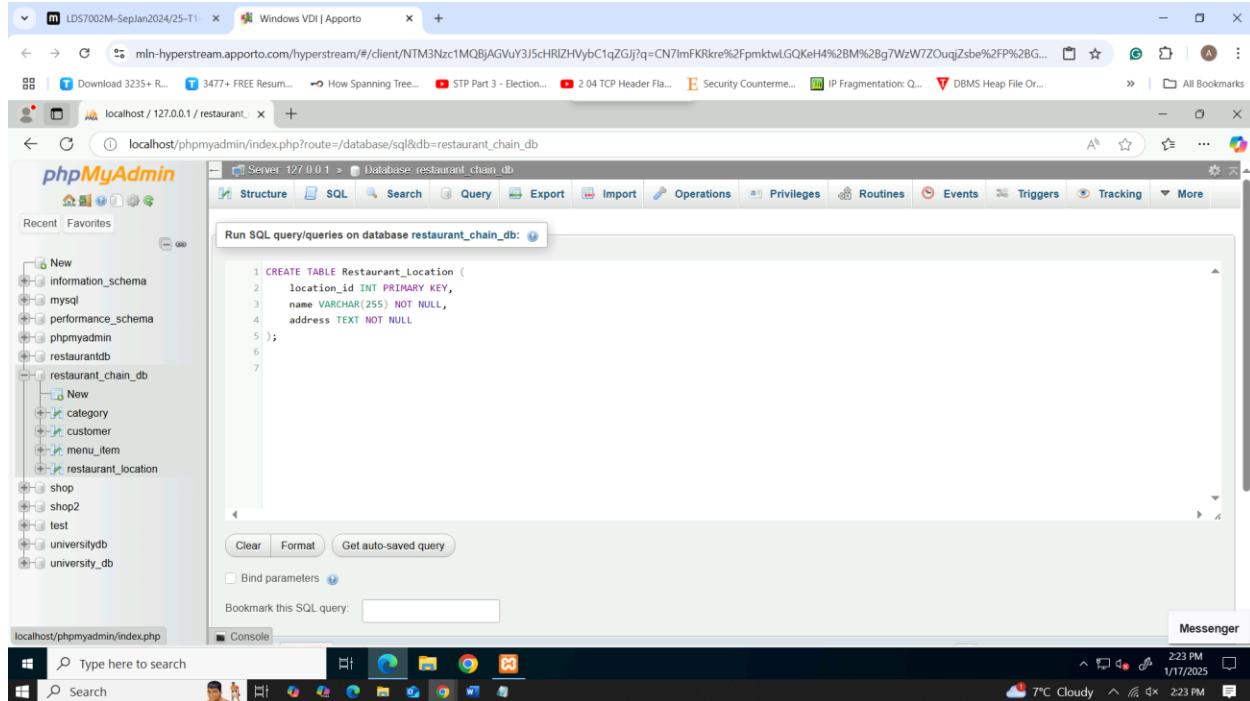


Figure 22: create table query of restaurant location

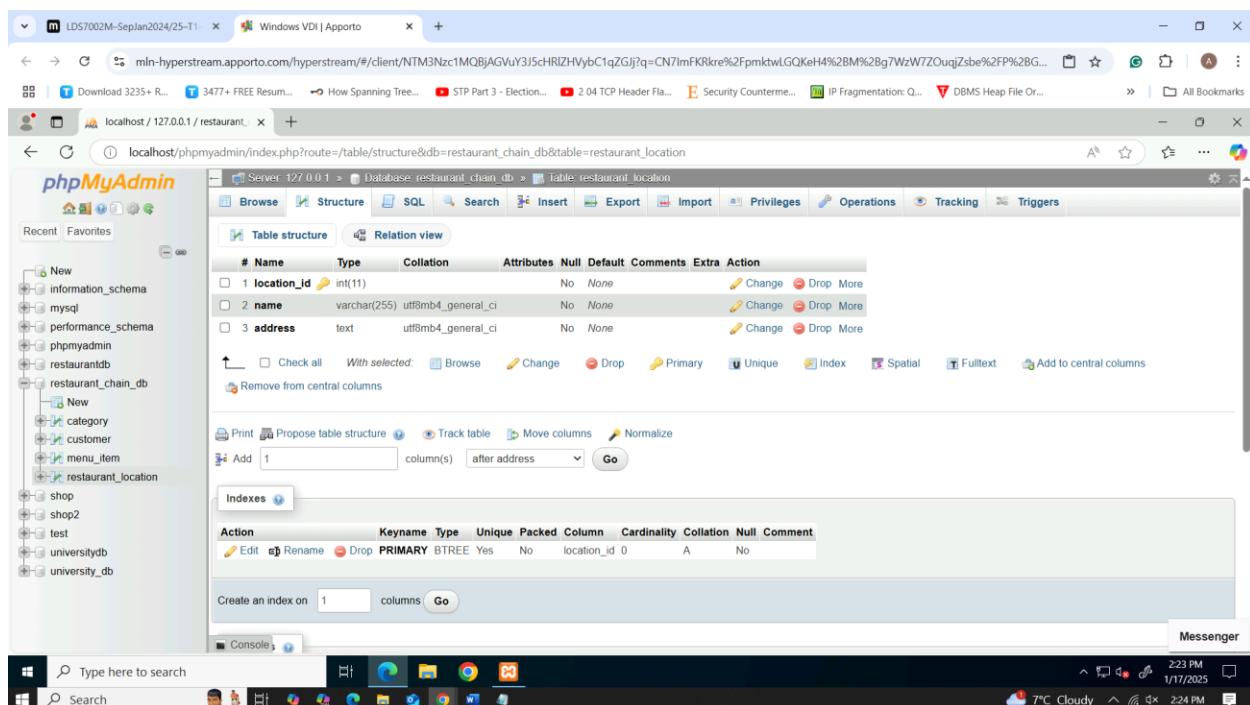


Figure 23: structure of restaurant location table

The Restaurant_Location table stores information about restaurant locations with three columns: location_id (an integer auto incremented unique field), name (a field that contains the name of the location, with a maximum length of 255 characters) and address (a required field for the full address of the location). It is also important to provide all the necessary information concerning the restaurant location and which acts as a key to make reference with other tables e.g. Reservation table where location_id is a foreign key that links all the reservations with particular restaurants. His design is coherent and makes organizing location-related data less problematic.

The screenshot shows the phpMyAdmin interface running in a browser window. The left sidebar lists databases and tables, with 'restaurant_chain_db' selected. The main area shows the results of a SELECT query on the 'restaurant_location' table. The results are displayed in a table with columns: location_id, name, and address. The data consists of 12 rows, each representing a restaurant location with its ID, name, and address.

	location_id	name	address
1	1	Downtown Diner	123 Main St, Cityville
2	2	Uptown Eatery	456 High St, Cityville
3	3	Suburban Grill	789 Oak Ave, Suburbia
4	4	Central Cafe	101 Center Blvd, Cityville
5	5	Seaside Bistro	202 Ocean Dr, Seaside
6	6	Mountain Retreat	303 Alpine Rd, Mountainview
7	7	Valley View	404 Valley Rd, Valleyville
8	8	City Plaza	505 Plaza St, Cityville
9	9	Parkside Pub	606 Park Ln, Cityville
10	10	Riverside Restaurant	707 River Rd, Rivertown
11	11	Lakeside Lounge	808 Lake St, Lakeview
12	12	Canyon Tacos	909 Canyon Ave, Cannaburg

Figure 24: select query for restaurant location table

➤ Sales

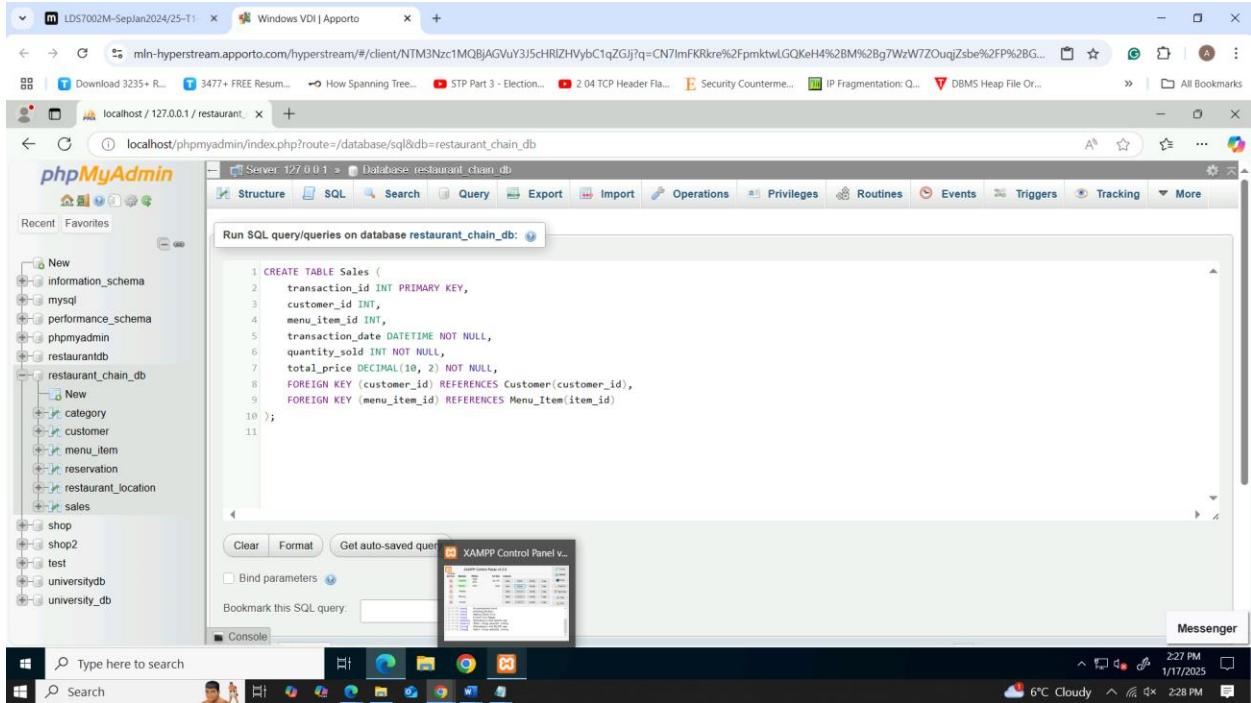


Figure 25: create table query of Sales

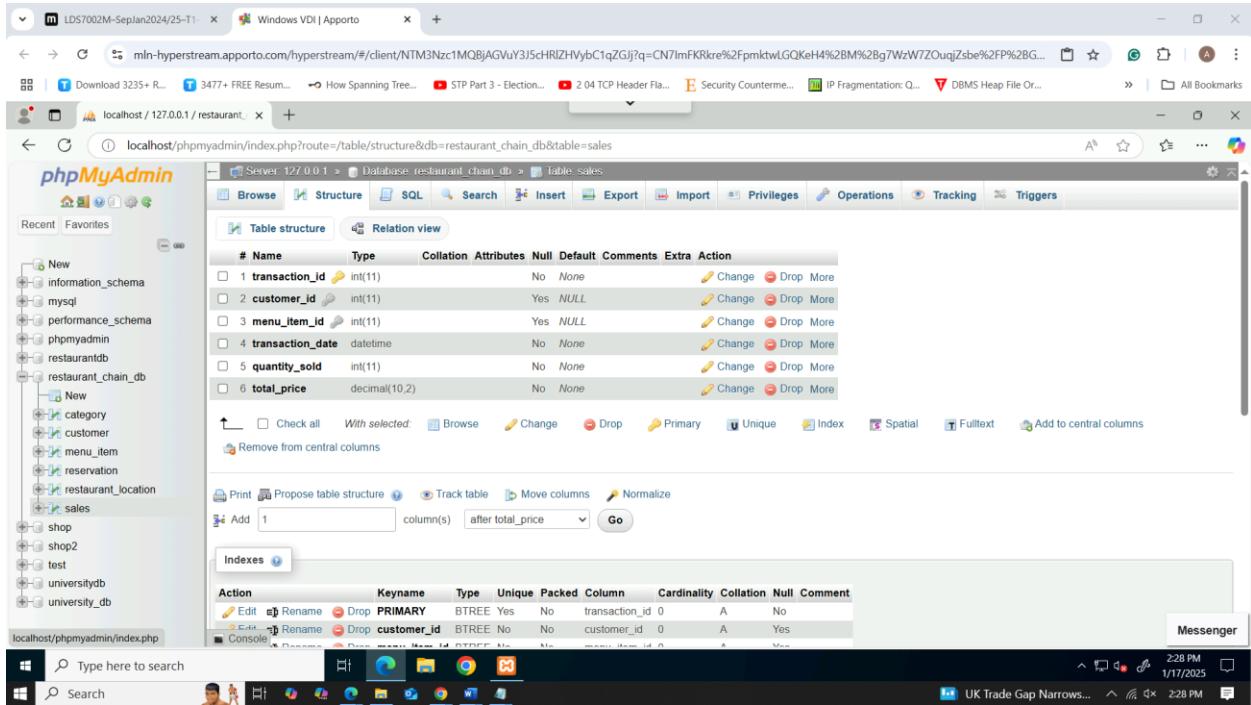


Figure 26: structure of sales table

The Sales table tracks sales transactions with six columns: transaction_id (an integer field serving as the primary key for the transactions table), customer_id (an integer used as a foreign key in reference to the

Customer.customer_id to connect the transaction to the original customer), menu_item_id (a foreign key referencing the Menu_Item.item_id used to determine the sold menu item), transaction_date (a required DATETIME field specifying the date and time of the transaction), quantity_sold (an integer indicating how many units of This table is also used to monitor and control sales information through the relation between customers and food options offered.

The screenshot shows the phpMyAdmin interface running on a Windows VDI session. The left sidebar lists various database tables under the 'restaurant_chain_db' schema. The main area displays a query result for the 'sales' table. The query is:

```
SELECT * from sales;
```

The results show 24 rows of data:

	transaction_id	customer_id	menu_item_id	transaction_date	quantity_sold	total_price
1	1	1	1	2025-01-01 19:00:00	2	17.98
2	2	2	3	2025-01-02 19:30:00	1	7.99
3	3	3	5	2025-01-03 20:00:00	3	20.97
4	4	4	7	2025-01-04 20:15:00	1	10.99
5	5	5	9	2025-01-05 18:45:00	2	17.98
6	6	6	11	2025-01-06 19:00:00	1	4.99
7	7	7	13	2025-01-07 19:30:00	4	11.96
8	8	8	15	2025-01-08 20:00:00	2	7.98
9	9	9	17	2025-01-09 18:30:00	1	14.99

Figure 27: select query for sales table

➤ Inventory

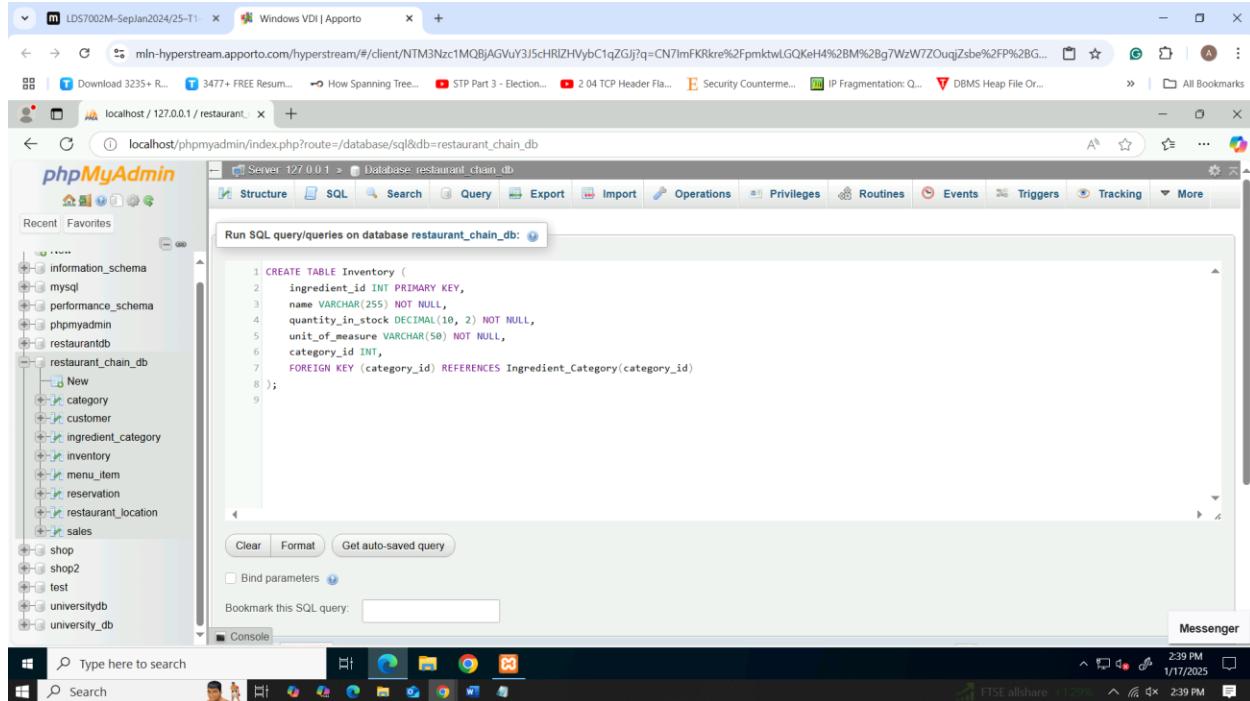


Figure 28: create table query of inventory

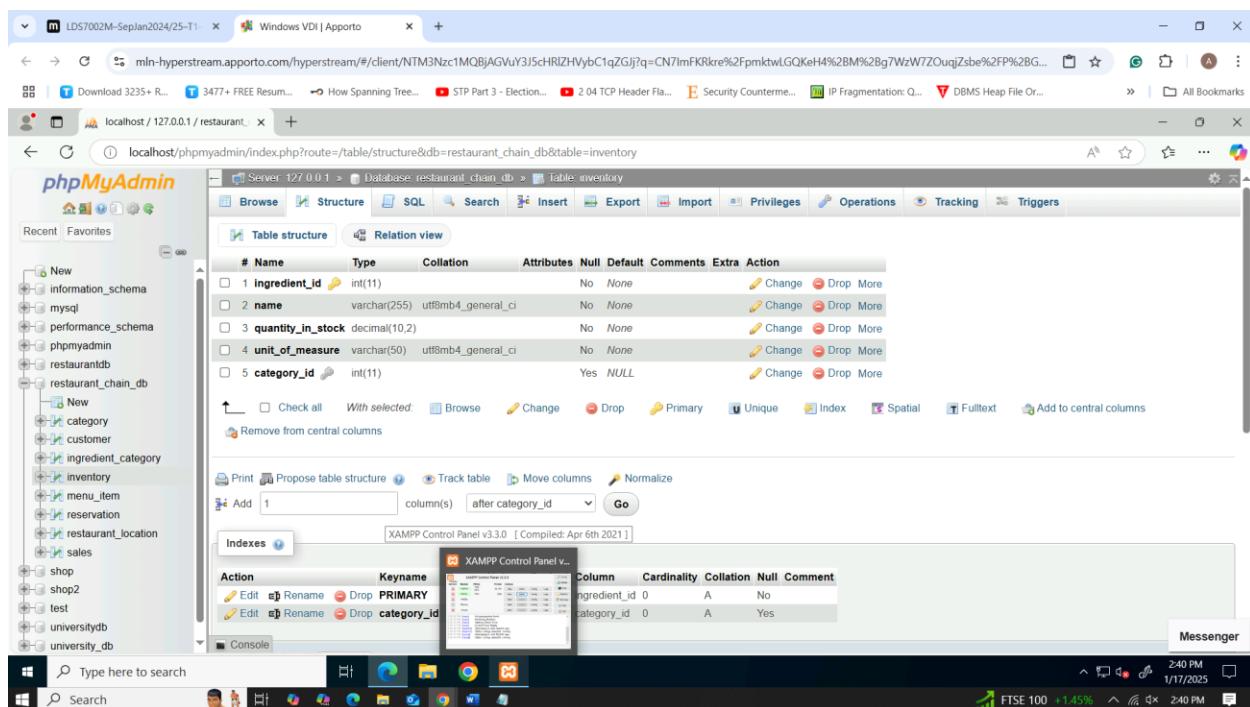


Figure 29: structure of inventory table

The Inventory table stores information about ingredients used in the restaurant, with five columns: ingredient_id (an integer.primary key to identify each ingredient uniquely), name (string required 255 character for the ingredient name), quantity_in_stock (required field, DECIMAL 10, 2 to store the quantity of the ingredient in stock), unit_of_measure (string required 50 character to have the unit of measure which can be kg/liter), and category_id (foreign key referencing the category of the ingredient using Ingredient_Category's category_id). It is useful to use this table for managing of ingredients and keep the check through the category reference table.

The screenshot shows the phpMyAdmin interface on a Windows desktop. The left sidebar lists databases and tables under 'restaurant_chain_db'. The 'inventory' table is selected. The main area displays a SELECT query and its results. The results table has columns: ingredient_id, name, quantity_in_stock, unit_of_measure, and category_id. The data is as follows:

ingredient_id	name	quantity_in_stock	unit_of_measure	category_id
1	Tomato Sauce	100.00	liters	1
2	Mozzarella Cheese	50.00	kg	1
3	Beef Patty	200.00	pieces	2
4	Lettuce	30.00	kg	3
5	Spaghetti	80.00	kg	4
6	Chicken Wings	150.00	pieces	5
7	Chocolate	40.00	kg	6
8	Coffee Beans	25.00	kg	8
9	Salmon	20.00	kg	9
10	Garlic	10.00	kg	10
		0.00	kg	9

Figure 30: select query for inventory table

➤ Ingredient Category

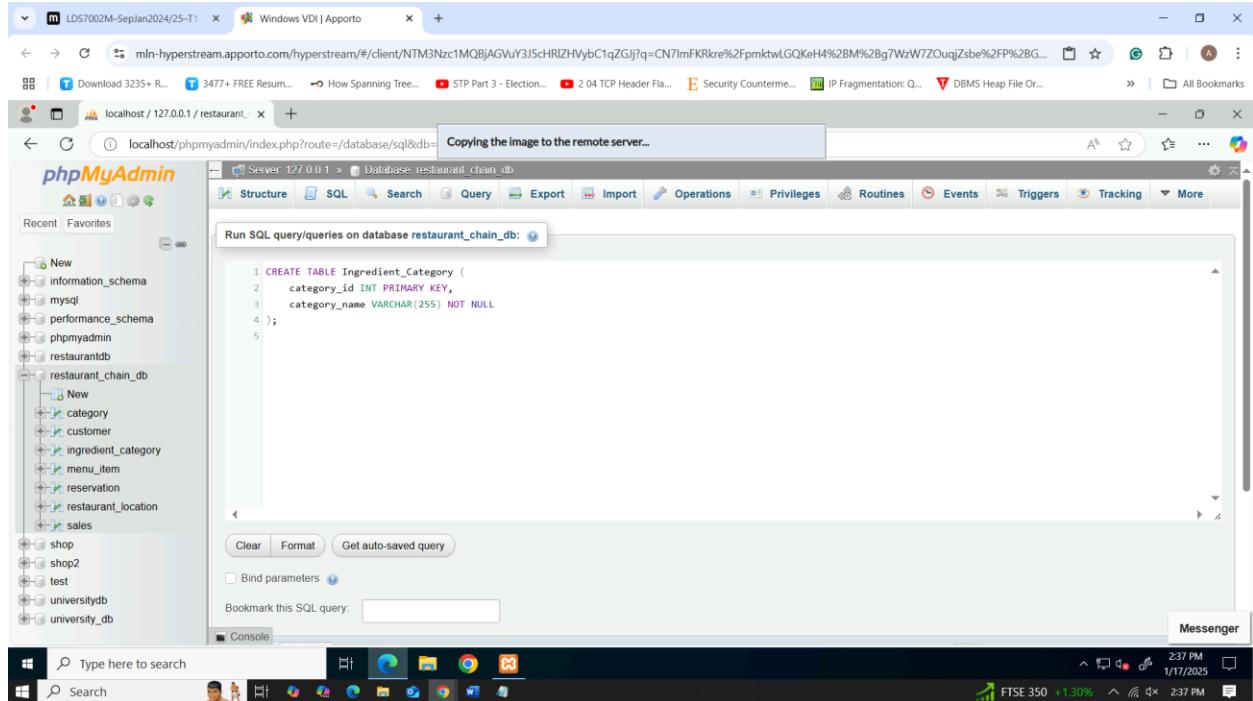


Figure 31: create table query of ingredient category

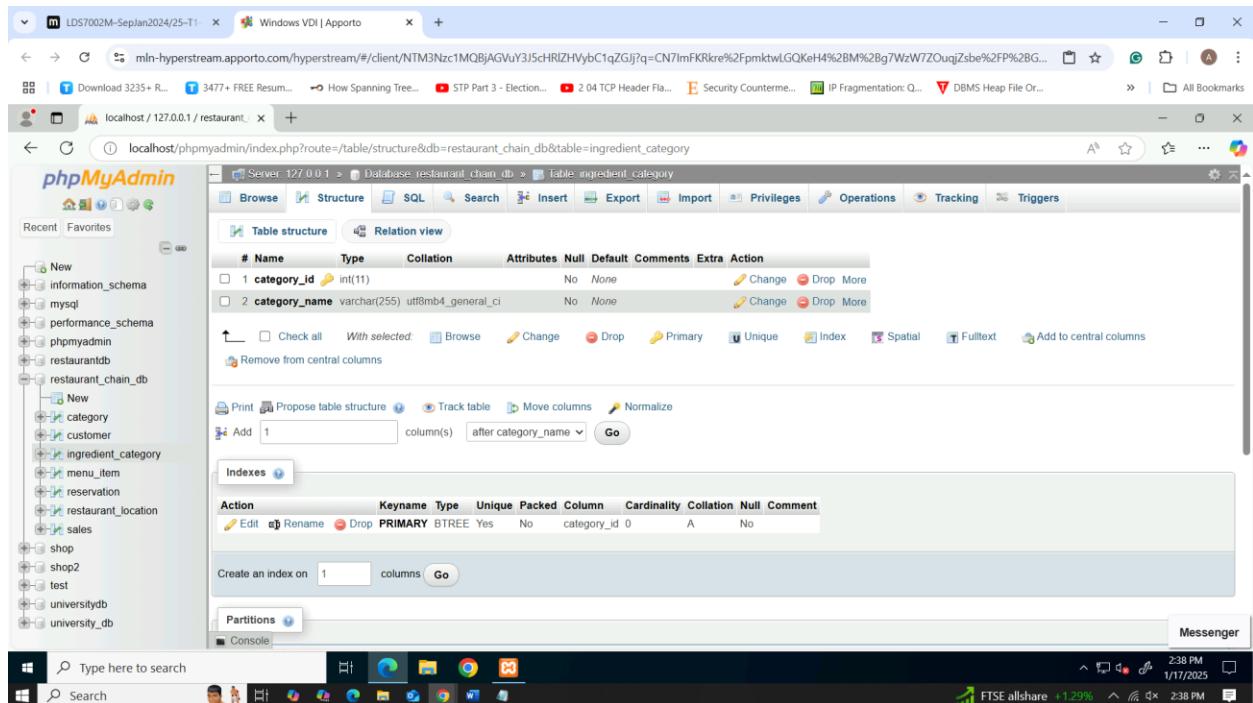


Figure 32: structure of ingredient category

The Ingredient_Category table organizes ingredients into specific categories, with two columns: The relevant fields are category_id (an integer with no less than 0, but no more than 2147483647, being the primary key of the table and uniquely identifying each category) and category_name (a mandatory string of no more than 255 characters containing the name of the category). This table enables.shtml the organisation of ingredients in a meaner that will facilitate their tracking and inventory management. This category is linked to the Inventory table by the category_id foreign key so that every ingredient can be linked with a valid category. This structure also enhances a better way of storage and identification of these ingredients in the restaurant's stock management system.

	category_id	category_name
<input type="checkbox"/>	1	Pizza Ingredients
<input type="checkbox"/>	2	Burger Ingredients
<input type="checkbox"/>	3	Salad Ingredients
<input type="checkbox"/>	4	Pasta Ingredients
<input type="checkbox"/>	5	Appetizer Ingredients
<input type="checkbox"/>	6	Dessert Ingredients
<input type="checkbox"/>	7	Drink Ingredients
<input type="checkbox"/>	8	Coffee Ingredients
<input type="checkbox"/>	9	Entree Ingredients
<input type="checkbox"/>	10	Side Ingredients

Figure 33: select query for ingredient category table

➤ Menu Ingredient

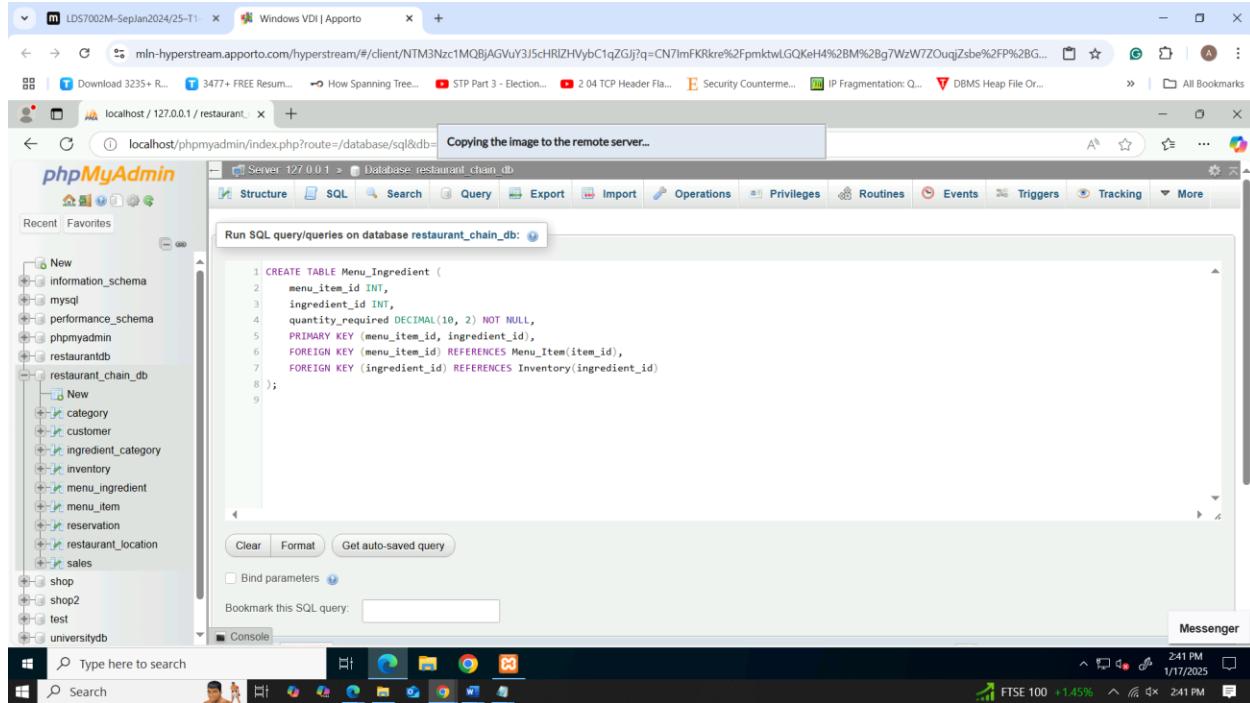


Figure 34: create table query of menu ingredient

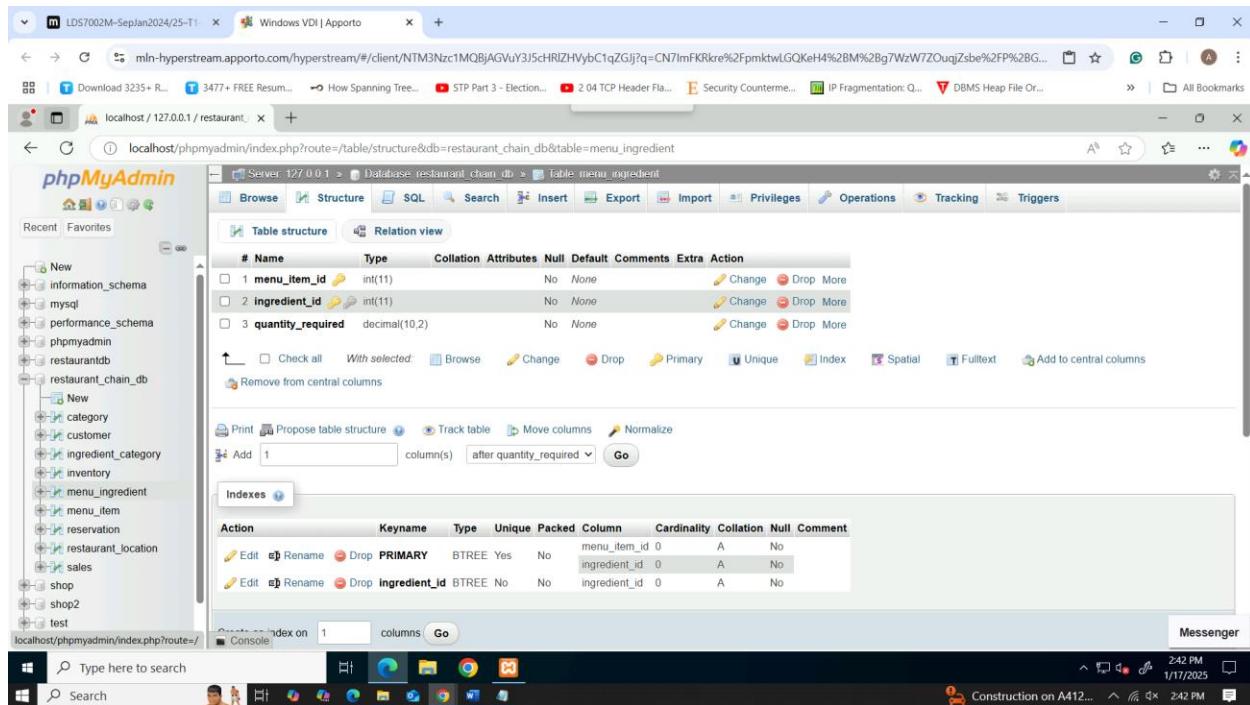


Figure 35: structure of menu ingredient

The Menu_Ingredient table links menu items to the ingredients required to prepare them, with three columns: Menu_Item ID (a reference link key to the Menu_Item table to identify which ingredient belongs to which menu item), Ingredient ID (a reference link key to the Inventory.item ID to state which ingredient is used), Quantity required (data type DECIMAL(10, 2) which is mandatory to define how much of the ingredient is required to prepare the menu item). The primary key is chosen as menu_item_id and ingredient_id as each of these make up a unique entity. This table supports the identification of the ingredients served with each dish and provides control and consistency between dishes and stocks.

The screenshot shows a Windows desktop environment with a browser window and a phpMyAdmin interface.

Browser Window:

- Title bar: LDS7002M-SepJan2024/25-T1 | Windows VDI | Apporto
- Address bar: mln-hyperstream.apporto.com/hyperstream/#/client/MTk5ODAxMDMAwBlbmNyeXB0ZWR1cmwtamRjYw==?q=xwxsLOnnCuNI7YeXFajUZ079lBDvQdUJaMjwyiZk2owAznfrn...
- Tab bar: Download 3235+ R... 3477+ FREE Resum... How Spanning Tree... STP Part 3 - Election... 2 04 TCP Header Fla... Security Counterme... IP Fragmentation: Q... DBMS Heap File Or...
- Bottom status bar: Rain on Thursday 5:10 PM 1/20/2025 5:10 PM

phpMyAdmin Interface:

- Left sidebar: Database structure for restaurant_chain_db, including tables: category, customer, employee, feedback, ingredient_category, inventory, menu_ingredient, menu_item, payment, promotion, promotion_menu_item, reservation, restaurant_location, sales, shift, supplier, supply_order, shop.
- Current database: restaurant_chain_db
- Current table: menu_ingredient
- Table structure: menu_item_id, ingredient_id, quantity_required
- Table data (selected rows):

	menu_item_id	ingredient_id	quantity_required
<input type="checkbox"/>	1	1	0.20
<input type="checkbox"/>	1	2	0.10
<input type="checkbox"/>	2	1	0.20
<input type="checkbox"/>	2	12	0.15
<input type="checkbox"/>	3	3	1.00
<input type="checkbox"/>	3	4	0.05
<input type="checkbox"/>	4	3	1.00
<input type="checkbox"/>	4	4	0.05
<input type="checkbox"/>	5	4	0.10
<input type="checkbox"/>	5	11	0.02
<input type="checkbox"/>	6	13	0.10

Figure 36: select query for menu ingredient table

➤ Employee

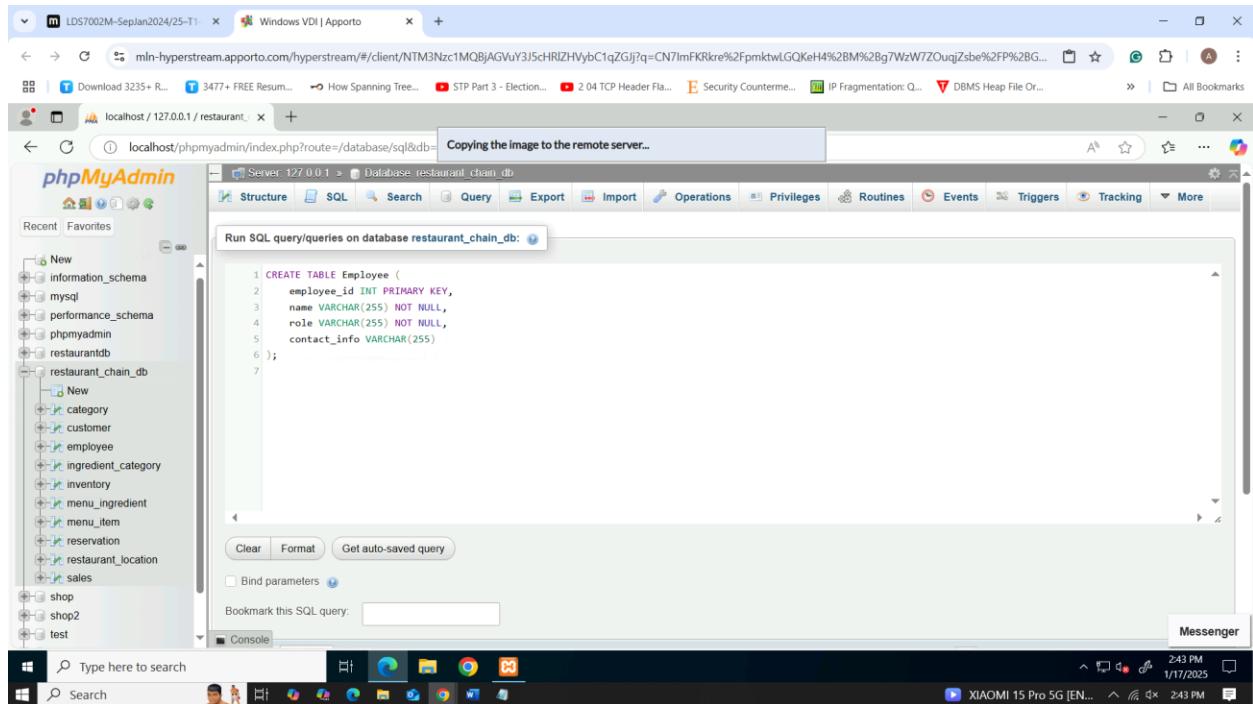


Figure 37: create table query of employee

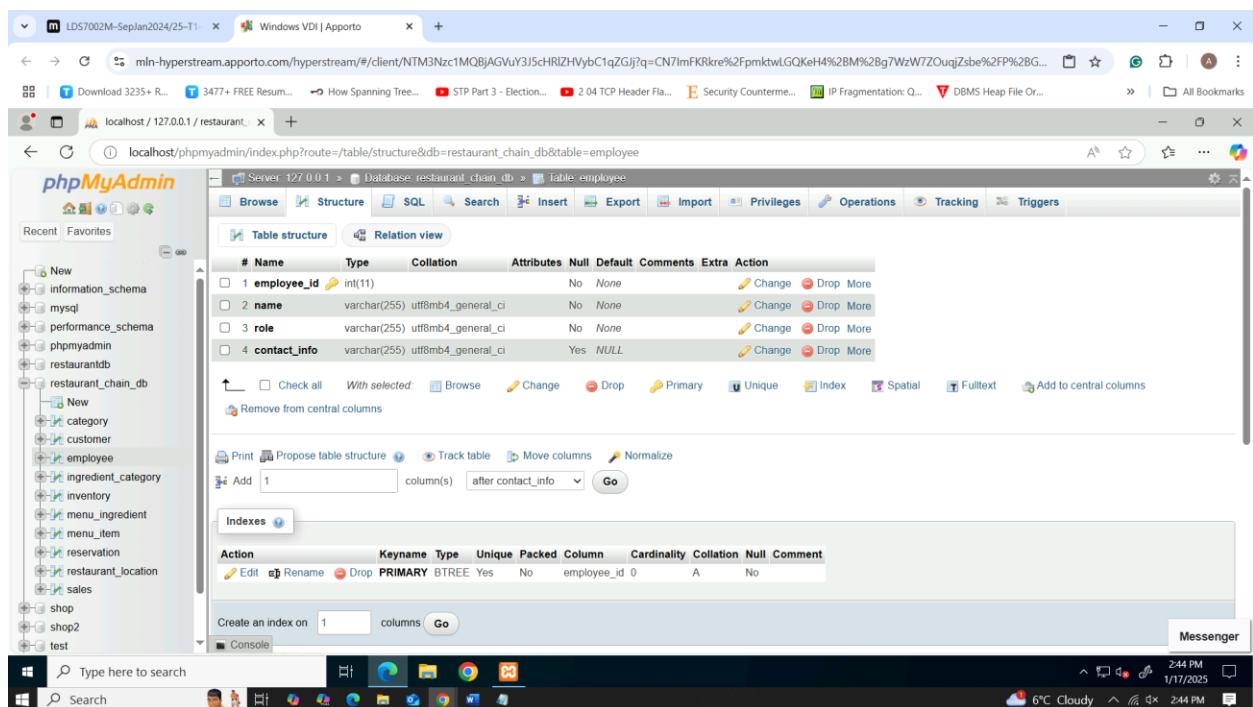


Figure 38: structure of menu ingredient table

The Employee table stores information about employees with four columns: employee_id, an integer unique key for each employee, name, which has a constraint to be mandatory string and must contain less than 256 characters to hold the name of the employee, role, which should also be a mandatory string with less than 256 characters to contain the position/department/role of the employee and contact_info, which is an option string containing less than 256 characters to include the phone number and/or email of the employee. This table assists in organization of the employees, tracking of those working within the company and the position they hold.

The screenshot shows the phpMyAdmin interface on a Windows desktop. The left sidebar lists databases and tables, with 'employee' selected under 'restaurant_chain_db'. The main area displays the results of a SELECT query:

```
SELECT * from employee;
```

Showing rows 0 - 24 (70 total, Query took 0.0002 seconds.)

	employee_id	name	role	contact_info	
<input type="checkbox"/>	1	Alice Manager	Manager	alice.manager@example.com	
<td><input type="checkbox"/></td> <td>2</td> <td>Bob Cook</td> <td>Cook</td> <td>bob.cook@example.com</td>	<input type="checkbox"/>	2	Bob Cook	Cook	bob.cook@example.com
<td><input type="checkbox"/></td> <td>3</td> <td>Charlie Waiter</td> <td>Waiter</td> <td>charlie.waiter@example.com</td>	<input type="checkbox"/>	3	Charlie Waiter	Waiter	charlie.waiter@example.com
<td><input type="checkbox"/></td> <td>4</td> <td>Diana Host</td> <td>Host</td> <td>diana.host@example.com</td>	<input type="checkbox"/>	4	Diana Host	Host	diana.host@example.com
<td><input type="checkbox"/></td> <td>5</td> <td>Ethan Cleaner</td> <td>Cleaner</td> <td>ethan.cleaner@example.com</td>	<input type="checkbox"/>	5	Ethan Cleaner	Cleaner	ethan.cleaner@example.com
<td><input type="checkbox"/></td> <td>6</td> <td>Fiona Chef</td> <td>Chef</td> <td>fiona.chef@example.com</td>	<input type="checkbox"/>	6	Fiona Chef	Chef	fiona.chef@example.com
<td><input type="checkbox"/></td> <td>7</td> <td>George Bartender</td> <td>Bartender</td> <td>george.bartender@example.com</td>	<input type="checkbox"/>	7	George Bartender	Bartender	george.bartender@example.com
<td><input type="checkbox"/></td> <td>8</td> <td>Hannah Manager</td> <td>Manager</td> <td>hannah.manager@example.com</td>	<input type="checkbox"/>	8	Hannah Manager	Manager	hannah.manager@example.com
<td><input type="checkbox"/></td> <td>9</td> <td>Ian Waiter</td> <td>Waiter</td> <td>ian.waiter@example.com</td>	<input type="checkbox"/>	9	Ian Waiter	Waiter	ian.waiter@example.com
<td><input type="checkbox"/></td> <td>10</td> <td>Jessica Cook</td> <td>Cook</td> <td>jessica.cook@example.com</td>	<input type="checkbox"/>	10	Jessica Cook	Cook	jessica.cook@example.com
<td><input type="checkbox"/></td> <td>11</td> <td>Kyle Host</td> <td>Host</td> <td>kyle.host@example.com</td>	<input type="checkbox"/>	11	Kyle Host	Host	kyle.host@example.com

Figure 39: select query for employee table

➤ Shift

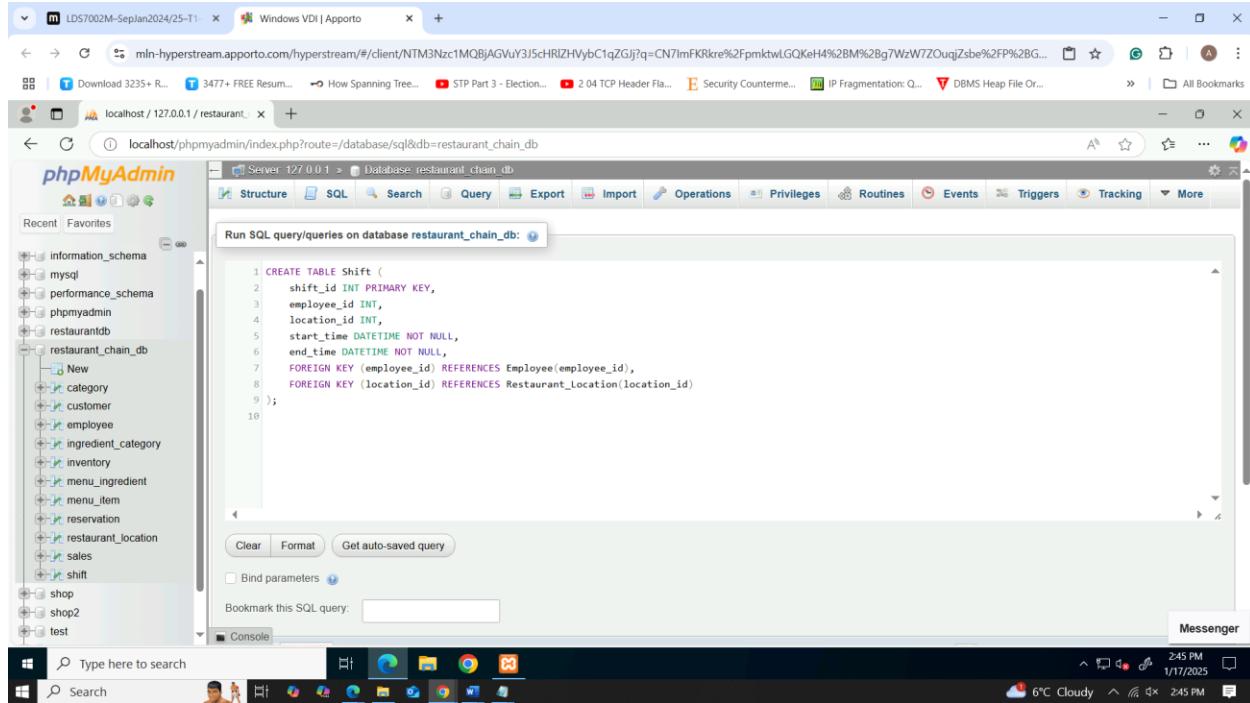


Figure 40: create table query of shift

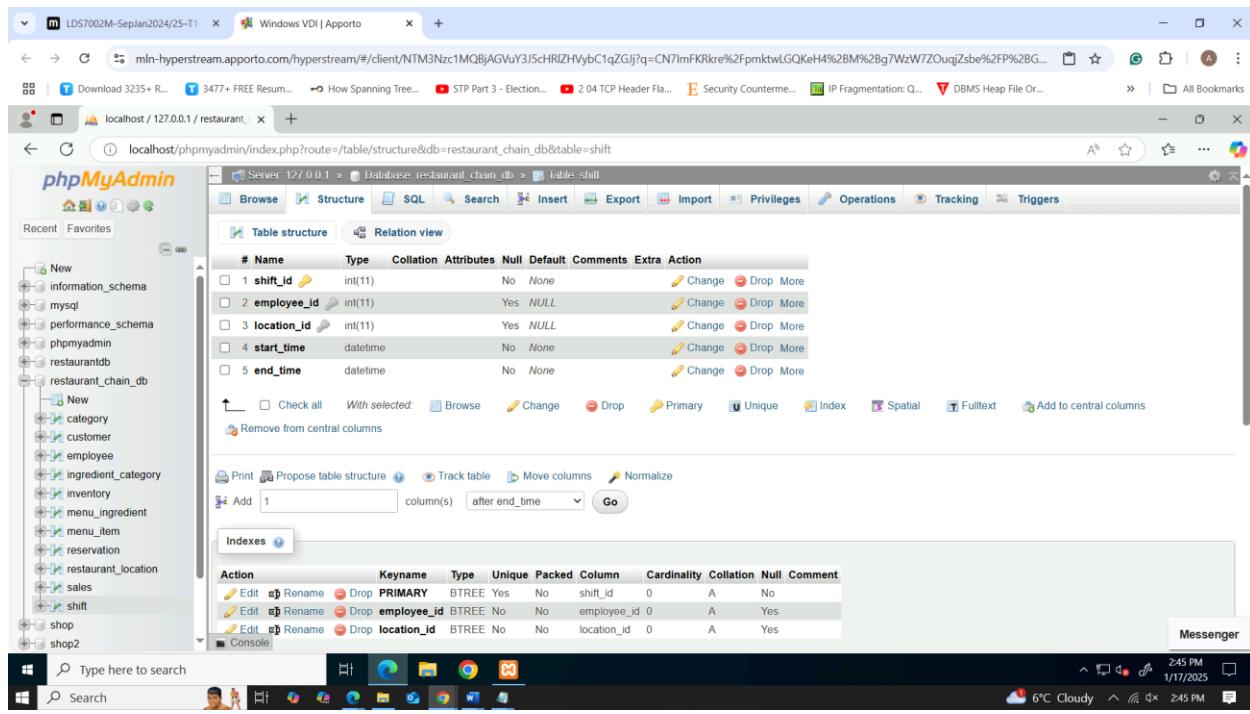


Figure 41: structure of shift table

The Shift table tracks employee work shifts with five columns: shift_id – integer unique identifier of the shift, employee_id – a foreign key referencing Employee Table to link the shift to the employee, location_id – a foreign key linking the shift to the restaurant location, start_time – a required DATETIME field indicating the time the shift started, end_time – a required DATETIME field indicating the time the shift ended. This table is useful in the employee scheduling process as it links the schedules to employees and restaurant locations for easy running.

The screenshot shows the phpMyAdmin interface for a database named 'restaurant_chain_db'. The left sidebar lists various tables: restaurantdb, New, category, customer, employee, feedback, ingredient_category, inventory, menu_ingredient, menu_item, payment, promotion, promotion_menu_item, reservation, restaurant_location, sales, shift, supplier, supply_order, and shop. The 'shift' table is selected in the main area. The table has the following structure:

	shift_id	employee_id	location_id	start_time	end_time
<input type="checkbox"/>	1	1	1	2025-01-01 09:00:00	2025-01-01 17:00:00
<input type="checkbox"/>	2	2	2	2025-01-02 10:00:00	2025-01-02 18:00:00
<input type="checkbox"/>	3	3	1	2025-01-03 11:00:00	2025-01-03 19:00:00
<input type="checkbox"/>	4	4	3	2025-01-04 12:00:00	2025-01-04 20:00:00
<input type="checkbox"/>	5	5	2	2025-01-05 13:00:00	2025-01-05 21:00:00
<input type="checkbox"/>	6	6	1	2025-01-06 14:00:00	2025-01-06 22:00:00
<input type="checkbox"/>	7	7	3	2025-01-07 15:00:00	2025-01-07 23:00:00
<input type="checkbox"/>	8	8	2	2025-01-08 16:00:00	2025-01-08 00:00:00
<input type="checkbox"/>	9	9	1	2025-01-09 17:00:00	2025-01-09 01:00:00
<input type="checkbox"/>	10	10	3	2025-01-10 18:00:00	2025-01-10 02:00:00
<input type="checkbox"/>	11	1	2	2025-01-11 09:00:00	2025-01-11 17:00:00

Figure 42: select query for shift table

➤ Supplier

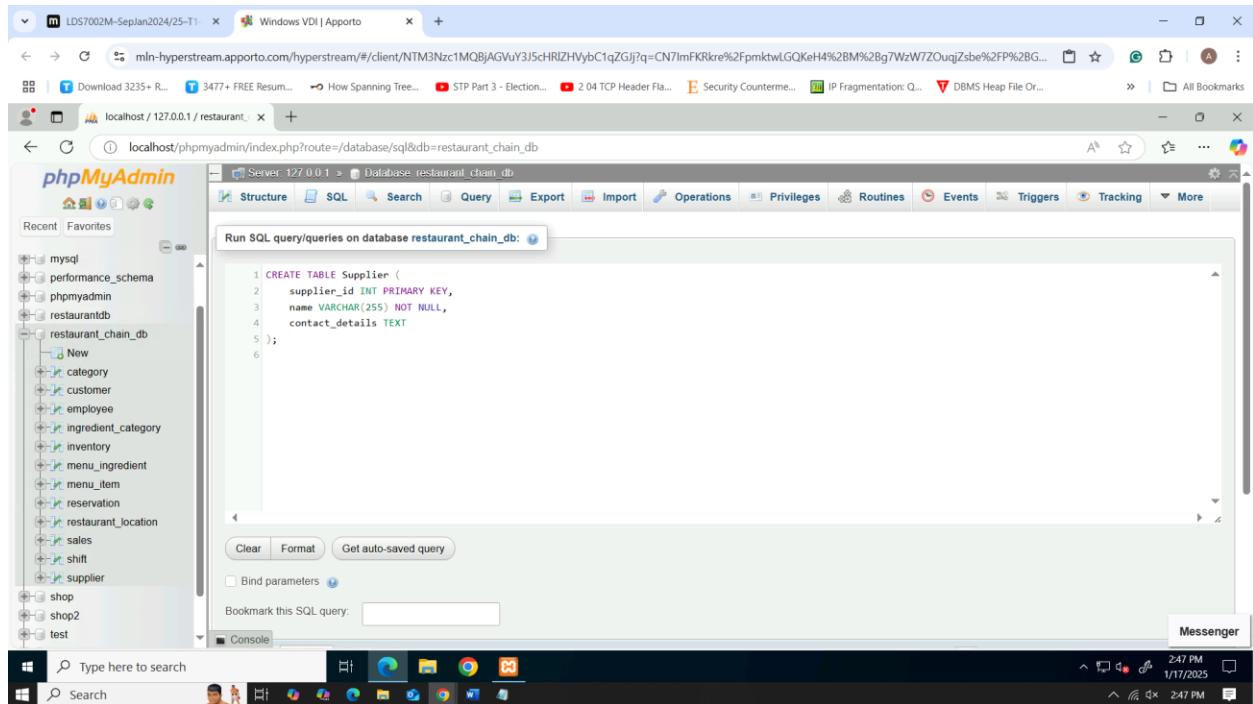


Figure 43: create table query of supplier

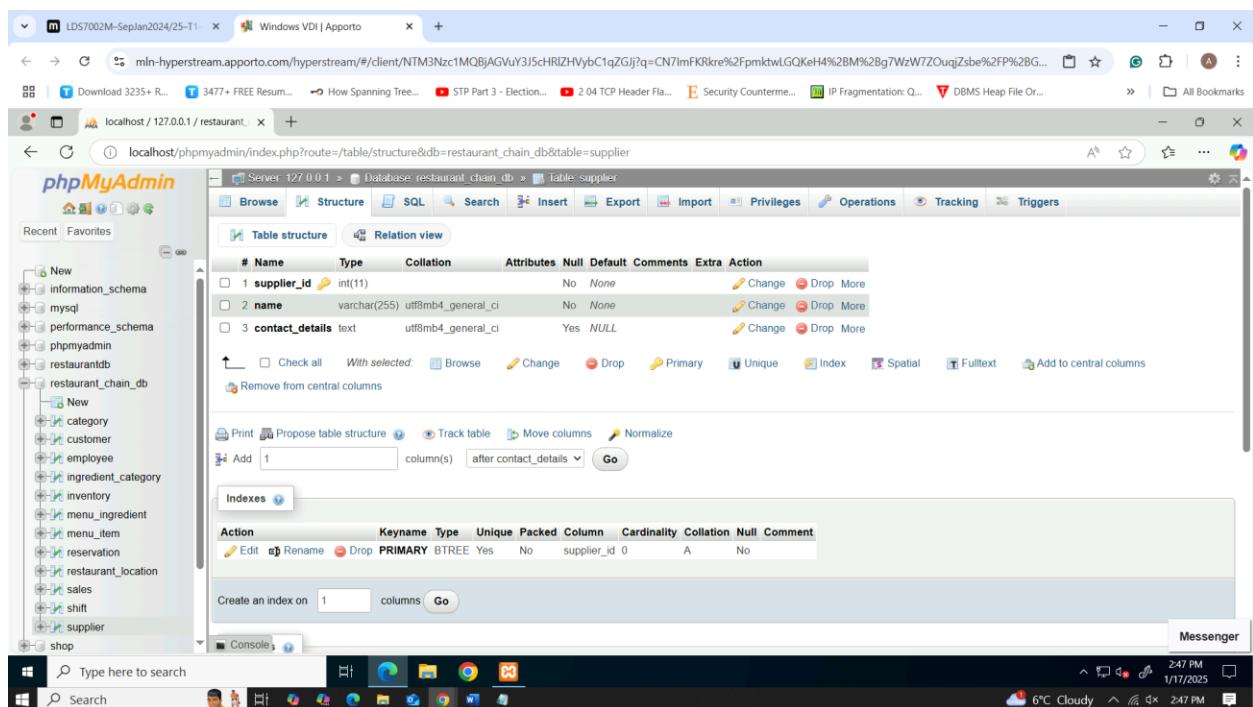


Figure 44: structure of supplier table

The Supplier table stores information about suppliers with three columns: name (an essential string field of 255 characters for the supplier name), and contact_details (a text type field to store all the supplier's contact information such as phone numbers, emails, and/or physical address, and supplier ID, which is an auto-generated integer to uniquely identify each supplier). This table assists in the relationship management of suppliers to guarantee that every supplier is recognized individually and his or her information is readily accessible for purchase and supply chain purposes.

The screenshot shows the phpMyAdmin interface on a Windows desktop. The left sidebar lists databases and tables, with 'supplier' selected under 'restaurant_chain_db'. The main area displays the 'supplier' table data:

	supplier_id	name	contact_details
<input type="checkbox"/>	1	Fresh Foods Co.	freshfoods@example.com, +1 234 567 890
<input type="checkbox"/>	2	Gourmet Ingredients Ltd	gourmetingredients@example.com, +1 345 678 901
<input type="checkbox"/>	3	Green Farms	greenfarms@example.com, +1 456 789 012
<input type="checkbox"/>	4	Dairy Delights	dairydelights@example.com, +1 567 890 123
<input type="checkbox"/>	5	Beverage Supply Co.	beveragesupply@example.com, +1 678 901 234
<input type="checkbox"/>	6	Spice World	spiceworld@example.com, +1 789 012 345
<input type="checkbox"/>	7	Meat Masters	meatmasters@example.com, +1 890 123 456
<input type="checkbox"/>	8	Oceanic Foods	oceanicfoods@example.com, +1 901 234 567
<input type="checkbox"/>	9	Bread Bakers	breadbakers@example.com, +1 123 345 678
<input type="checkbox"/>	10	Fruit & Veggies Ltd	fruitandveggies@example.com, +1 234 456 789
<input type="checkbox"/>	11	Tasty Treats Bakery	tastytreats@example.com, +1 234 567 890

Figure 45: select query for supplier table

➤ Supply Order

The screenshot shows the phpMyAdmin interface for the database `restaurant_chain_db`. In the left sidebar, the table `Supply_Order` is selected under the `restaurant_chain_db` schema. The main area displays the SQL query used to create the table:

```

1 CREATE TABLE Supply_Order (
2     order_id INT PRIMARY KEY,
3     supplier_id INT,
4     ingredient_id INT,
5     quantity_ordered DECIMAL(10, 2) NOT NULL,
6     order_date DATETIME NOT NULL,
7     delivery_date DATETIME,
8     FOREIGN KEY (supplier_id) REFERENCES Supplier(supplier_id),
9     FOREIGN KEY (ingredient_id) REFERENCES Inventory(ingredient_id)
10 );
11

```

Below the query, there are buttons for `Clear`, `Format`, and `Get auto-saved query`. A checkbox for `Bind parameters` is also present.

Figure 46: create table query of supply order

The screenshot shows the phpMyAdmin interface for the `Supply_Order` table within the `restaurant_chain_db` database. The left sidebar shows the table structure. The main area displays the table structure with columns:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	<code>order_id</code>	<code>int(11)</code>			No	<code>None</code>			<code>Change</code> <code>Drop</code> <code>More</code>
2	<code>supplier_id</code>	<code>int(11)</code>			Yes	<code>NULL</code>			<code>Change</code> <code>Drop</code> <code>More</code>
3	<code>ingredient_id</code>	<code>int(11)</code>			Yes	<code>NULL</code>			<code>Change</code> <code>Drop</code> <code>More</code>
4	<code>quantity_ordered</code>	<code>decimal(10,2)</code>			No	<code>None</code>			<code>Change</code> <code>Drop</code> <code>More</code>
5	<code>order_date</code>	<code>datetime</code>			No	<code>None</code>			<code>Change</code> <code>Drop</code> <code>More</code>
6	<code>delivery_date</code>	<code>datetime</code>			Yes	<code>NULL</code>			<code>Change</code> <code>Drop</code> <code>More</code>

Below the table structure, there are buttons for `Add`, `Browse`, `Change`, `Drop`, `Primary`, `Unique`, `Index`, `Spatial`, `Fulltext`, and `Add to central columns`. There is also a `Print` button and a `Propose table structure` button. The bottom section shows the `Indexes` table:

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
<code>Edit</code> <code>Rename</code> <code>Drop</code>	<code>PRIMARY</code>	<code>BTREE</code>	Yes	No	<code>order_id</code>	0	A	No	
<code>Edit</code> <code>Rename</code> <code>Drop</code>	<code>supplier_id</code>	<code>BTREE</code>	No	No	<code>supplier_id</code>	0	A	Yes	
<code>Edit</code> <code>Rename</code> <code>Drop</code>	<code>ingredient_id</code>	<code>BTREE</code>	No	No	<code>ingredient_id</code>	0	A	Yes	

Figure 47:structure of supply order table

The Supply_Order table tracks supplier orders for ingredients with six columns: The fields in the order table are Order_ID a unique key identifying each order, Supplier_ID a key to the supplier in the supplier table, Ingredient_ID key to a specific ingredient in the inventory table and quantity ordered which is a decimal that has to be required to store the quantity of the ingredient that was ordered, Order_date which is a required datetime for the date the order was placed, and delivery date being an optional datetime that would store the expected or This design help in tracking the orders relating to supplies, relation between ingredients and suppliers and order details.

The screenshot shows the phpMyAdmin interface for the 'supply_order' table in the 'restaurant_chain_db' database. The table has six columns: order_id, supplier_id, ingredient_id, quantity_ordered, order_date, and delivery_date. The data consists of 11 rows, each representing an order with a unique order_id (1-11), a supplier_id (1-10), an ingredient_id (1-12), and various order dates ranging from 2025-01-01 to 2025-01-13. The delivery_date column is mostly empty, with one entry for order_id 11.

	order_id	supplier_id	ingredient_id	quantity_ordered	order_date	delivery_date
1	1	1	1	50.00	2025-01-01 00:00:00	2025-01-03 00:00:00
2	2	2	2	20.00	2025-01-02 00:00:00	2025-01-04 00:00:00
3	3	3	4	30.00	2025-01-03 00:00:00	2025-01-05 00:00:00
4	4	4	5	40.00	2025-01-04 00:00:00	2025-01-06 00:00:00
5	5	5	6	100.00	2025-01-05 00:00:00	2025-01-07 00:00:00
6	6	6	7	25.00	2025-01-06 00:00:00	2025-01-08 00:00:00
7	7	7	8	60.00	2025-01-07 00:00:00	2025-01-09 00:00:00
8	8	8	9	20.00	2025-01-08 00:00:00	2025-01-10 00:00:00
9	9	9	10	15.00	2025-01-09 00:00:00	2025-01-11 00:00:00
10	10	10	11	50.00	2025-01-10 00:00:00	2025-01-12 00:00:00
11	1	1	12	30.00	2025-01-11 00:00:00	2025-01-13 00:00:00

Figure 48: select query for supply order table

➤ Payment

The screenshot shows the phpMyAdmin interface for the database `restaurant_chain_db`. In the left sidebar, under the `payment` table, there is a SQL query editor window titled "Run SQL query/queries on database `restaurant_chain_db`". The query is:

```

1 CREATE TABLE Payment (
2     payment_id INT PRIMARY KEY,
3     transaction_id INT,
4     method VARCHAR(50) NOT NULL,
5     status VARCHAR(50) NOT NULL,
6     FOREIGN KEY (transaction_id) REFERENCES Sales(transaction_id)
7 );
8

```

Below the query editor, there are buttons for "Clear", "Format", and "Get auto-saved query". There is also a checkbox for "Bind parameters" and a text input for "Bookmark this SQL query:". At the bottom of the window, there is a "Console" tab.

Figure 49: create table query of payment

The screenshot shows the phpMyAdmin interface for the `payment` table within the `restaurant_chain_db` database. The left sidebar shows the database structure. The main area displays the "Table structure" tab for the `payment` table. The table has four columns:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	<code>payment_id</code>	int(11)			No	None			Change Drop More
2	<code>transaction_id</code>	int(11)			Yes	NULL			Change Drop More
3	<code>method</code>	varchar(50)	utf8mb4_general_ci		No	None			Change Drop More
4	<code>status</code>	varchar(50)	utf8mb4_general_ci		No	None			Change Drop More

Below the table structure, there are buttons for "Print", "Propose table structure", "Track table", "Move columns", "Normalize", "Add", and "Indexes". The "Indexes" section shows one index:

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Rename Drop	PRIMARY	BTREE	Yes	No	payment_id	0	A	No	
Edit Rename Drop	transaction_id	BTREE	No	No	transaction_id	0	A	Yes	

Figure 50: structure of payment table

The Payment table records payment details for sales transactions with four columns: payment_id respectively; transaction_id respectively; method respectively, which must be a string of no more than 50 characters describing the type of payment, for example “credit card,” “cash,” or “online payment;” and status respectively, which also must be a string of no more than 50 characters describing the status of the payment, for instance “completed,” “pending,” or “failed.” This table facilitates adequate payment, relating it to transactions while displaying core information for payment processing and analysis.

The screenshot shows the phpMyAdmin interface on a Windows desktop. The left sidebar lists databases and tables, with 'payment' selected under 'restaurant_chain_db'. The main area displays a table of payment records with columns: payment_id, transaction_id, method, and status. A SQL query is visible above the table:

```
SELECT * from payment;
```

	payment_id	transaction_id	method	status
<input type="checkbox"/>	1	1	Credit Card	Completed
<input type="checkbox"/>	2	2	Cash	Completed
<input type="checkbox"/>	3	3	Debit Card	Pending
<input type="checkbox"/>	4	4	Credit Card	Completed
<input type="checkbox"/>	5	5	Cash	Completed
<input type="checkbox"/>	6	6	Debit Card	Failed
<input type="checkbox"/>	7	7	Credit Card	Completed
<input type="checkbox"/>	8	8	Cash	Completed
<input type="checkbox"/>	9	9	Debit Card	Completed
<input type="checkbox"/>	10	10	Credit Card	Completed
<input type="checkbox"/>	11	11	Cash	Pending

Figure 51: select query for payment table

➤ Feedback

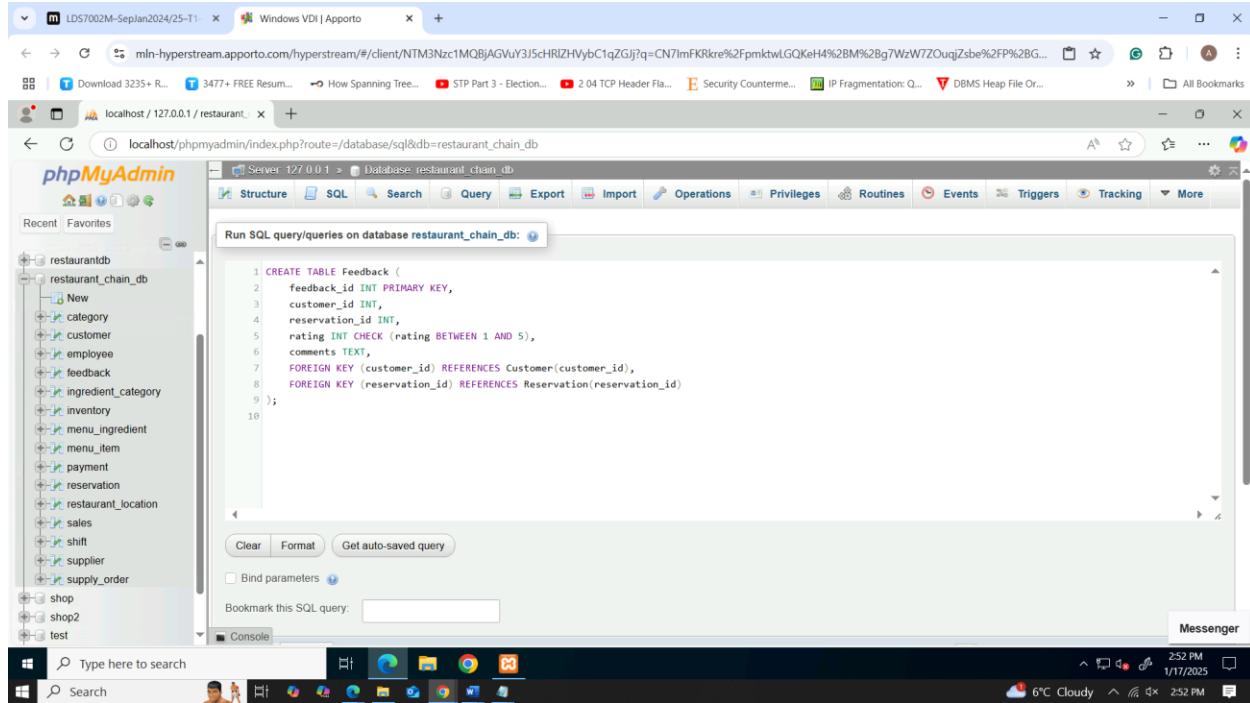


Figure 52: create table query of feedback

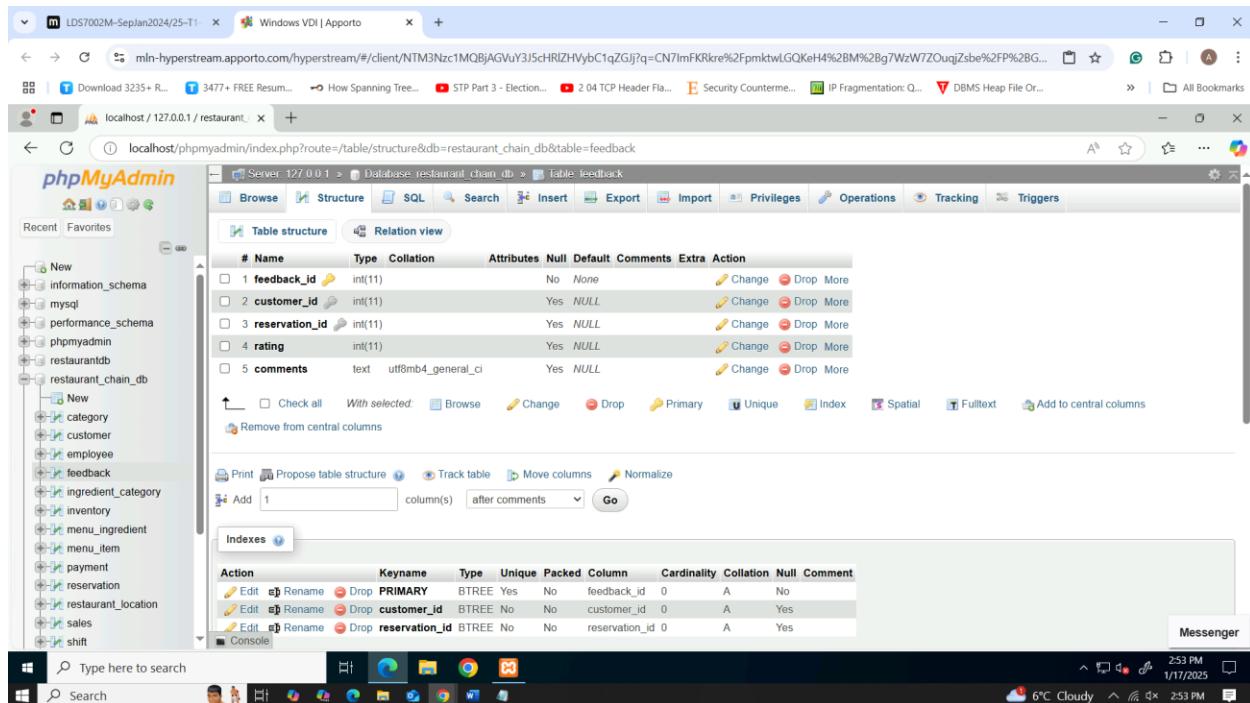
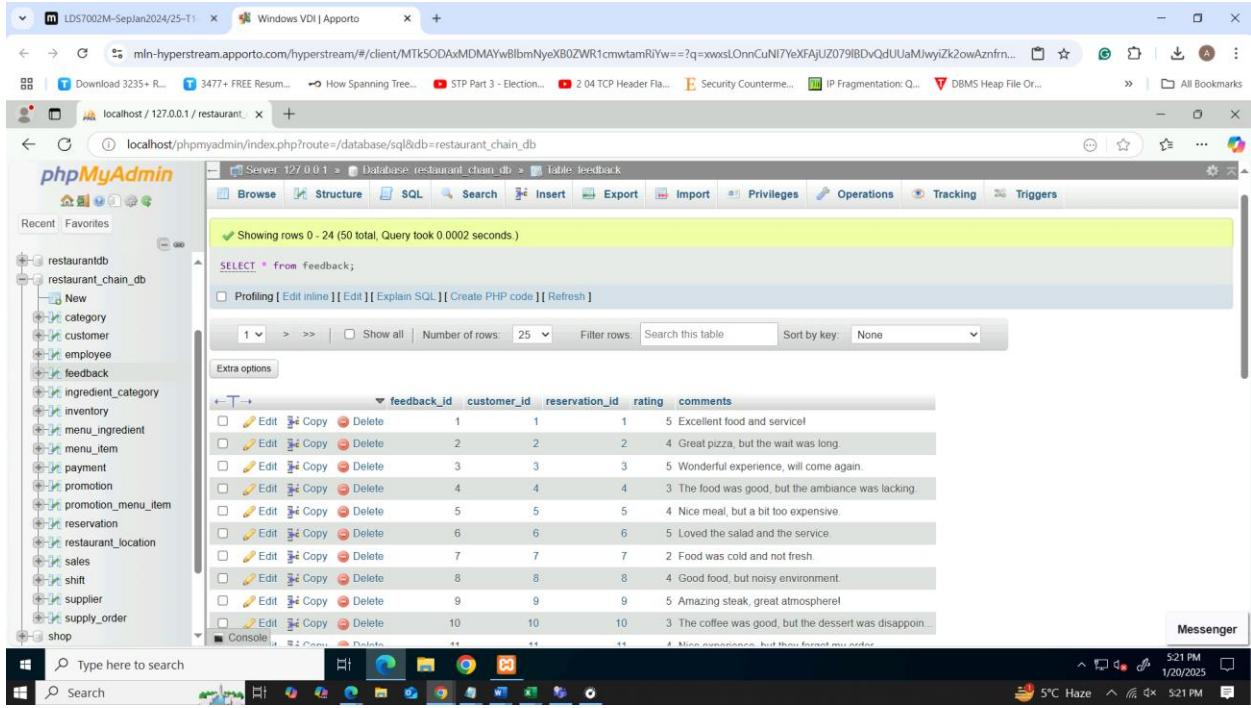


Figure 53: structure of feedback table

The Feedback table collects customer feedback on dining experiences with five columns: feedback_id (unique identifier for each feedback entry), customer_id (feedback connected with the client in table Customer), reservation_id (feedback connected with a specific reservation in table Reservation), rating (integer field with values from CHECK constraint between 1 and 5 – customer satisfaction), and comments (Text field for further notes or comments. With this table, collecting feedback regarding customers and reservations becomes easier to collect and analyze probable feedback for the evaluation of services and improvements of the dining section.



The screenshot shows the phpMyAdmin interface for the 'feedback' table in the 'restaurant_chain_db' database. The table has five columns: feedback_id, customer_id, reservation_id, rating, and comments. The data is as follows:

feedback_id	customer_id	reservation_id	rating	comments
1	1	1	1	5 Excellent food and service!
2	2	2	2	4 Great pizza, but the wait was long.
3	3	3	3	5 Wonderful experience, will come again.
4	4	4	4	3 The food was good, but the ambience was lacking.
5	5	5	5	4 Nice meal, but a bit too expensive.
6	6	6	6	5 Loved the salad and the service.
7	7	7	7	2 Food was cold and not fresh.
8	8	8	8	4 Good food, but noisy environment.
9	9	9	9	5 Amazing steak, great atmosphere!
10	10	10	3	4 The coffee was good, but the dessert was disappointing.
11	11	11	4	3 Nice ambiance, but they forgot my order.

Figure 54: select query for feedback table

➤ Promotion

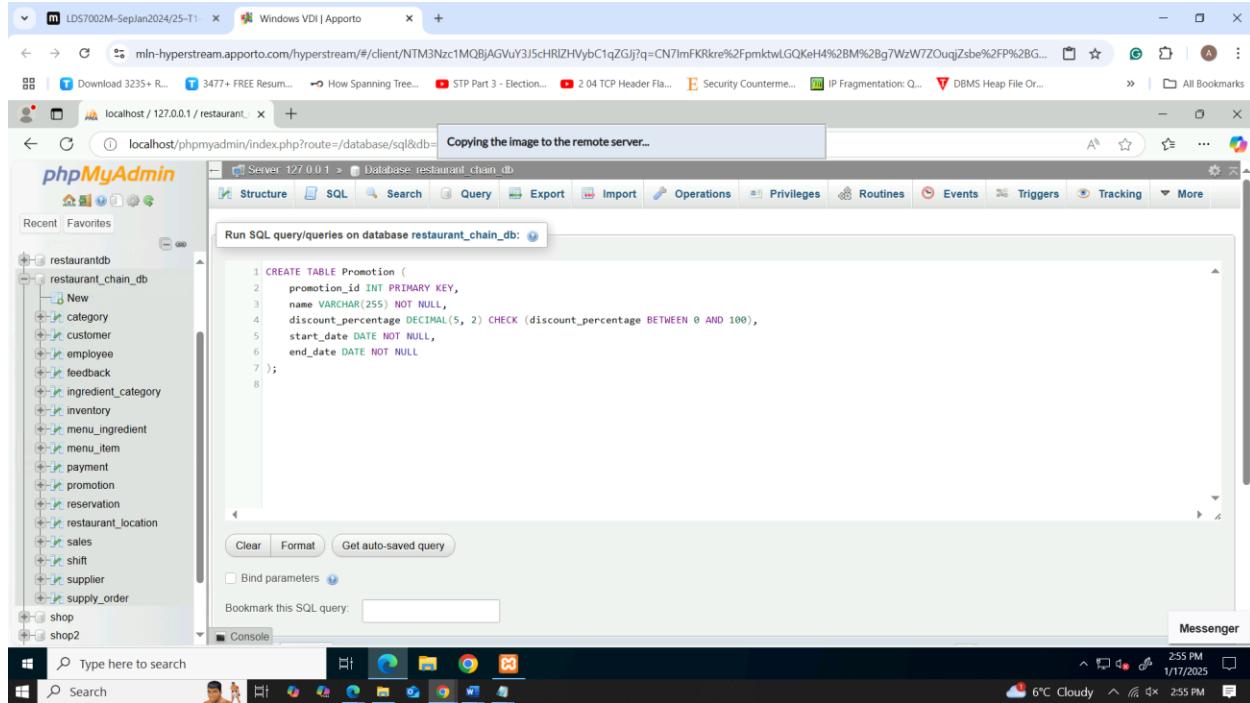


Figure 55: create table query of promotion

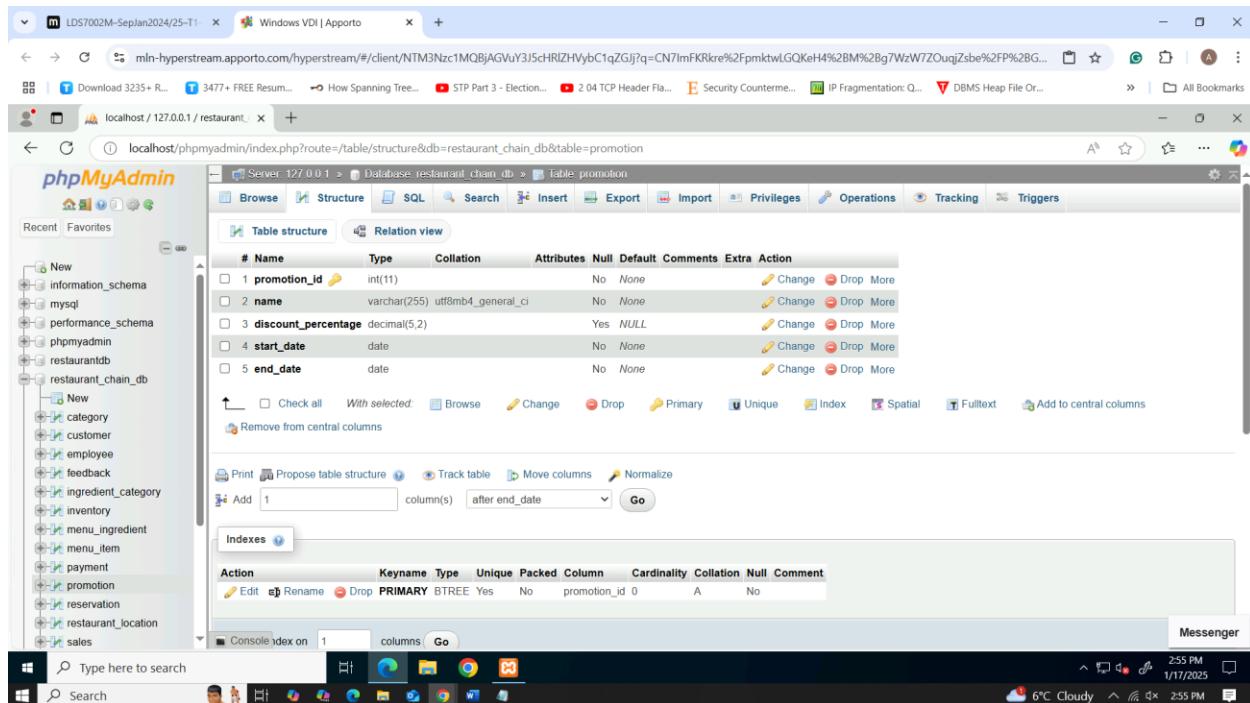


Figure 56: structure of promotion table

The Promotion table stores details about promotional offers with five columns: promotion_id (integer, primary key uniquely identifying each promotion), name (string of up to 255 characters, which required field used for storing the name or title of the promotion), discount_percentage (DECIMAL(5 2), which is the required field storing the percentage of discount for the products and it has a check constraint to ensure that the values entered are between 0 and 100), start_date (required date field that which denote the start This table helps in manning and monitoring promotions by verifying the proper periods of discounts, and in preserving crucial promotional data for usage.

	promotion_id	name	discount_percentage	start_date	end_date
<input type="checkbox"/>	1	New Year Special	20.00	2025-01-01	2025-01-15
<input type="checkbox"/>	2	Winter Sale	15.00	2025-01-05	2025-01-10
<input type="checkbox"/>	3	Happy Hour	25.00	2025-01-01	2025-01-31
<input type="checkbox"/>	4	Family Deal	30.00	2025-01-10	2025-01-20
<input type="checkbox"/>	5	Valentine's Day Promo	10.00	2025-02-01	2025-02-14
<input type="checkbox"/>	6	Weekend Discount	20.00	2025-01-01	2025-01-03
<input type="checkbox"/>	7	Lunch Special	15.00	2025-01-01	2025-01-31
<input type="checkbox"/>	8	Free Dessert	100.00	2025-01-05	2025-01-10
<input type="checkbox"/>	9	Early Bird	10.00	2025-01-01	2025-01-07
<input type="checkbox"/>	10	Super Saver	5.00	2025-01-01	2025-01-31
<input type="checkbox"/>	11	Winter Warmth	15.00	2025-01-15	2025-01-31

Figure 57: select query for promotion table

➤ Junction Table

A junction table is used to manage many-to-many (M:(N) relational databases structures because direct relationships go against the concept of normalization. The latter saves repetitional instances of the same relationship, thus making the database normalized. Also, it offers than-kadox for attributes related to the relationship only, such as periodic discounts in promotions. A junction table also has another advantage of being able to easily include or exclude relationships. For instance when considering the entities of promotions and menu items, one can create a junction table known as Promotion_Menu_Item which relates the two tables without a problem of atomicity or having the two entities repeat the same data through the existence of a foreign key or two.

Menu Item and Promotion junction table

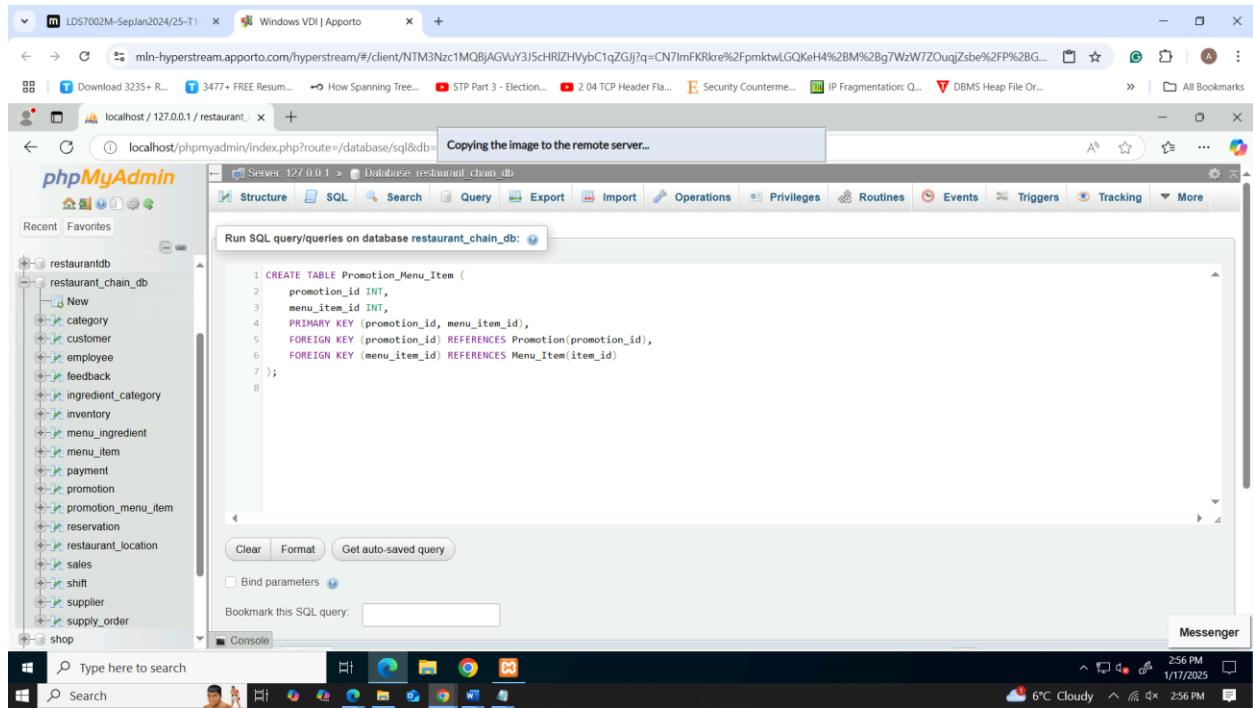


Figure 58: create table query of promotion menu item

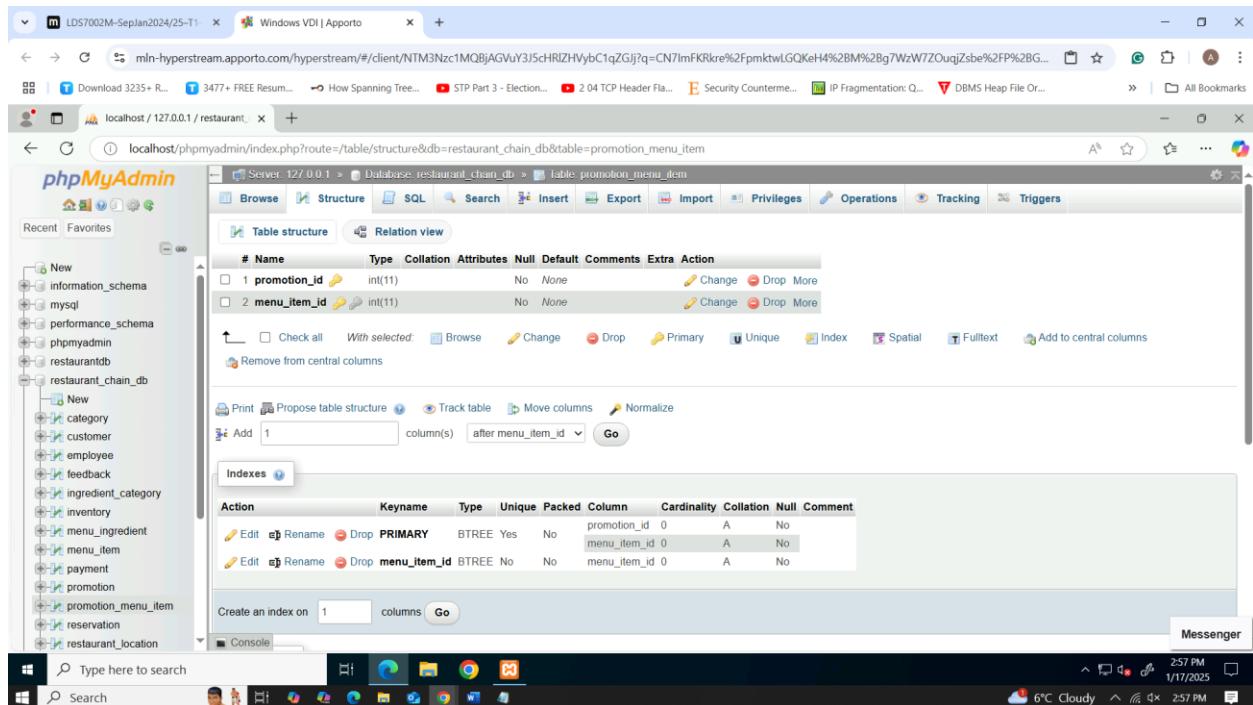


Figure 59: structure of promotion menu item table

Getting back to the description of the given shortage, let me note that Promotion_Menu_Item is a junction table that connects promotions with particular menu items. It has two foreign key columns: promotion_id in relation to the promotion_id in the Promotion table, and menu_item_id in relation with the item_id in the Menu_Item table. These foreign keys define relations between promotions and the products that belong to them, so, a promotion can be applied to definite plates or products.

Promotion ID and menu item ID are used as a composite key and therefore it is the primary key of the table. This helps in the uniqueness of the menu items so that in the promotion only one item is selected to be promoted. This structure makes it easy to map several items in the menu to a single promotion, help the restaurant group offer and keep accurate records of promotions and their relative menu items where necessary.

The screenshot shows a Windows desktop environment with a browser window open to a PHP port page and a MySQL database management tool (phpMyAdmin) running on a local host. The browser tab shows a URL related to hyperstream.apporto.com. The phpMyAdmin interface is focused on the 'promotion_menu_item' table within the 'restaurant_chain_db' database. A SELECT query is displayed in the SQL tab:

```
SELECT * from promotion_menu_item;
```

The results of the query are shown in a grid table:

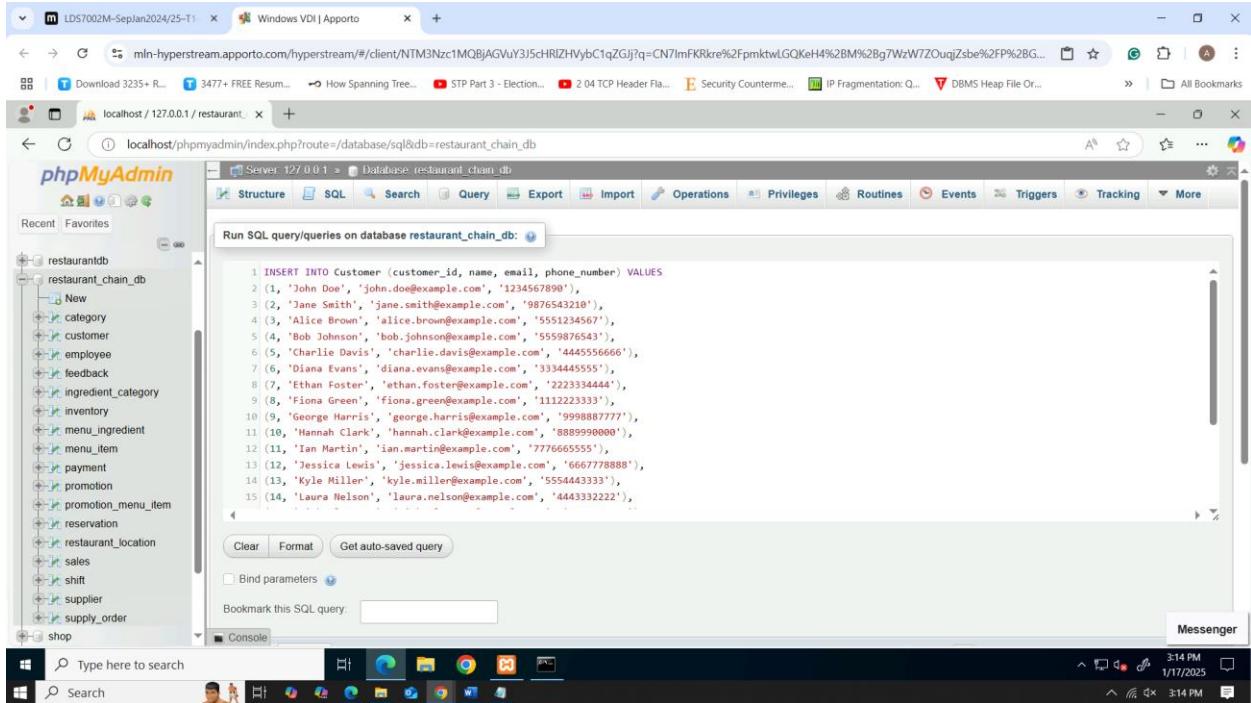
	promotion_id	menu_item_id
<input type="checkbox"/>	1	1
<input type="checkbox"/>	1	2
<input type="checkbox"/>	2	3
<input type="checkbox"/>	2	4
<input type="checkbox"/>	3	5
<input type="checkbox"/>	3	6
<input type="checkbox"/>	4	7
<input type="checkbox"/>	4	8
<input type="checkbox"/>	5	9
<input type="checkbox"/>	5	10
<input type="checkbox"/>	6	1

Below the table are standard MySQL edit, copy, and delete icons for each row. The phpMyAdmin interface includes tabs for Browse, Structure, SQL, Search, Insert, Export, Import, Privileges, Operations, Tracking, and Triggers. The left sidebar lists other tables in the database, such as category, customer, employee, feedback, ingredient_category, inventory, menu_ingredient, menu_item, payment, promotion, reservation, restaurant_location, sales, shift, supplier, supply_order, and shop.

Figure 60: select query for junction table

4 Insert values to the database tables

➤ Customer



The screenshot shows a Windows desktop environment with a browser window open to a phpMyAdmin interface. The browser title bar reads "LDS7002M-SepJan2024/25-T1" and "mln-hyperstream.apporto.com/hyperstream/#/client/NTM3Nzc1MQ8jAGVuY3J5cHRIZHVybC1qZGji?q=CN7ImFKRkre%2FpmktwlGQKeH4%2BM%2Bq7WzW7ZOujzsbe%2FP%2BG...". The phpMyAdmin window is titled "localhost/127.0.0.1 / restaurant" and shows the "Structure" tab selected for the "restaurant_chain_db" database. The left sidebar lists various tables: restaurantdb, restaurant_chain_db, New, category, customer, employee, feedback, ingredient_category, inventory, menu_ingredient, menu_item, payment, promotion, promotion_menu_item, reservation, restaurant_location, sales, shift, supplier, supply_order, shop. The main area contains an SQL query editor with the following code:

```
1 INSERT INTO Customer (customer_id, name, email, phone_number) VALUES
2 (1, 'John Doe', 'john.doe@example.com', '1234567890'),
3 (2, 'Jane Smith', 'jane.smith@example.com', '9876543210'),
4 (3, 'Alice Brown', 'alice.brown@example.com', '5551234567'),
5 (4, 'Bob Johnson', 'bob.johnson@example.com', '5559876543'),
6 (5, 'Charlie Davis', 'charlie.davis@example.com', '4445566666'),
7 (6, 'Diana Evans', 'diana.evans@example.com', '3334445555'),
8 (7, 'Ethan Foster', 'ethan.foster@example.com', '2223334444'),
9 (8, 'Fiona Green', 'fiona.green@example.com', '1112223333'),
10 (9, 'George Harris', 'george.harris@example.com', '9998887777'),
11 (10, 'Hannah Clark', 'hannah.clark@example.com', '8889990000'),
12 (11, 'Ian Martin', 'ian.martin@example.com', '7776665555'),
13 (12, 'Jessica Lewis', 'jessica.lewis@example.com', '6667778888'),
14 (13, 'Kyle Miller', 'kyle.miller@example.com', '5554443333'),
15 (14, 'Laura Nelson', 'laura.nelson@example.com', '4443332222'),
```

Figure 61: insert value to the customer table

➤ Category

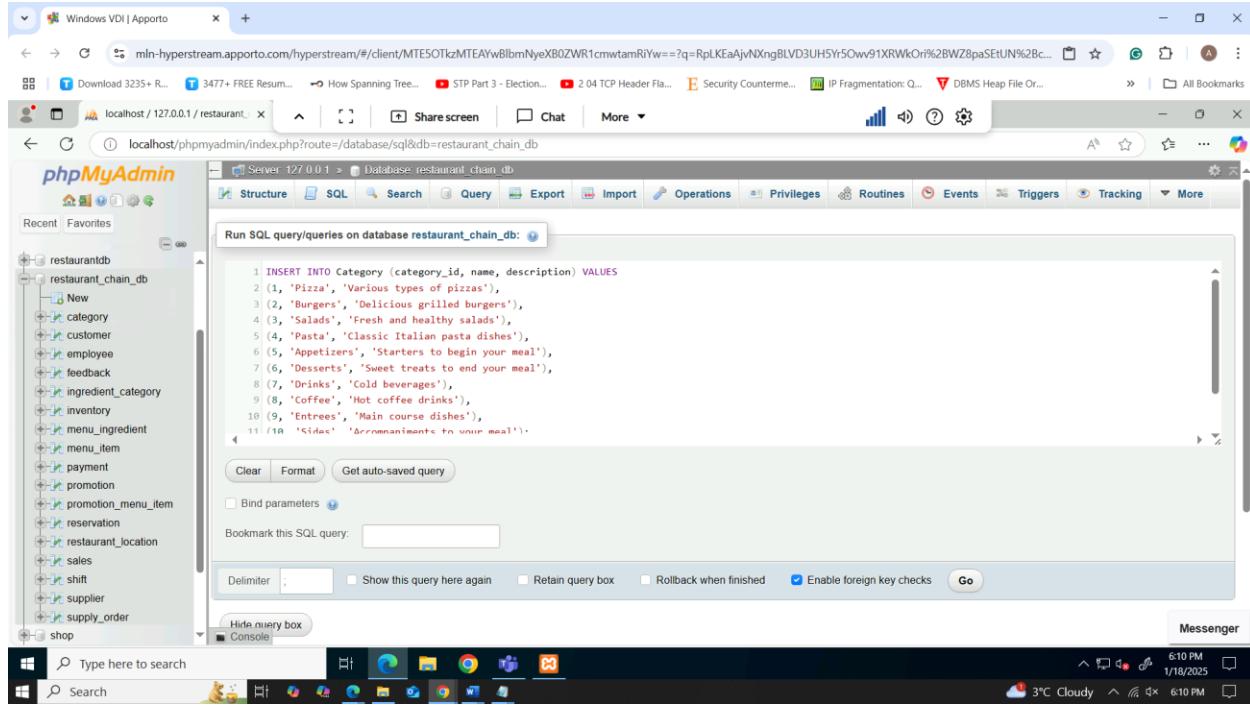


Figure 62: insert values to the category table

➤ Menu items

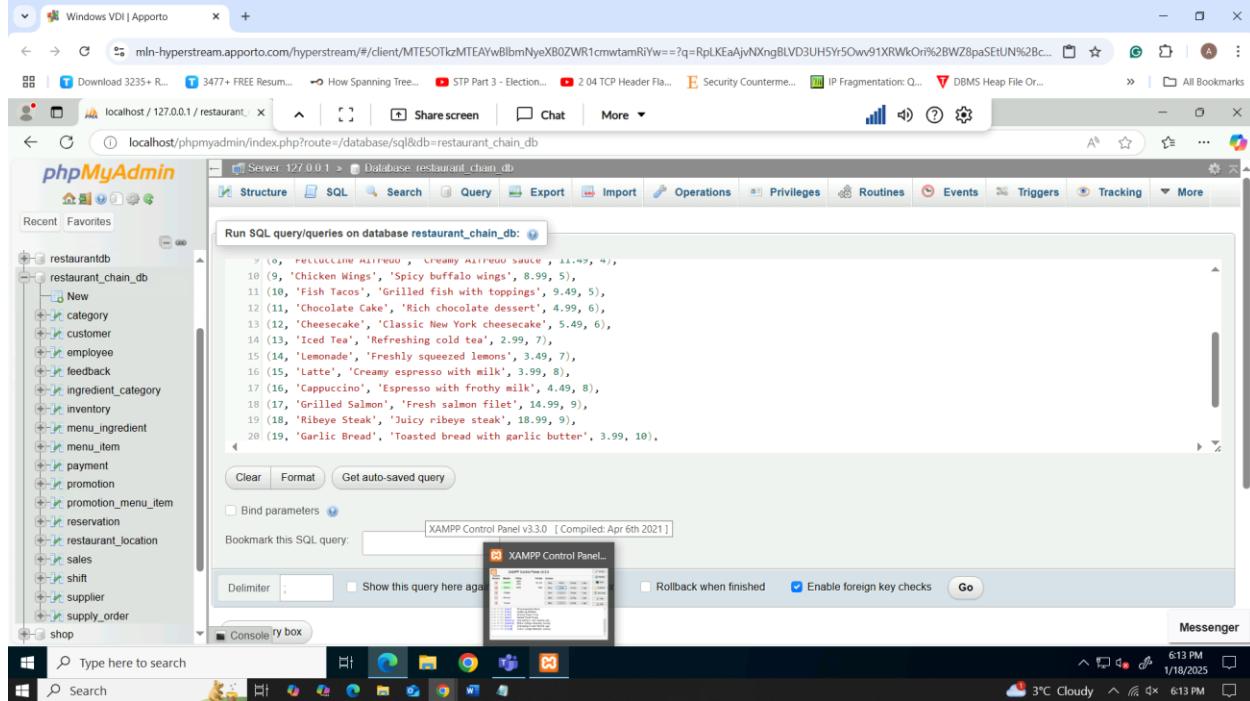


Figure 63: insert values to the menu item

➤ Restaurant location

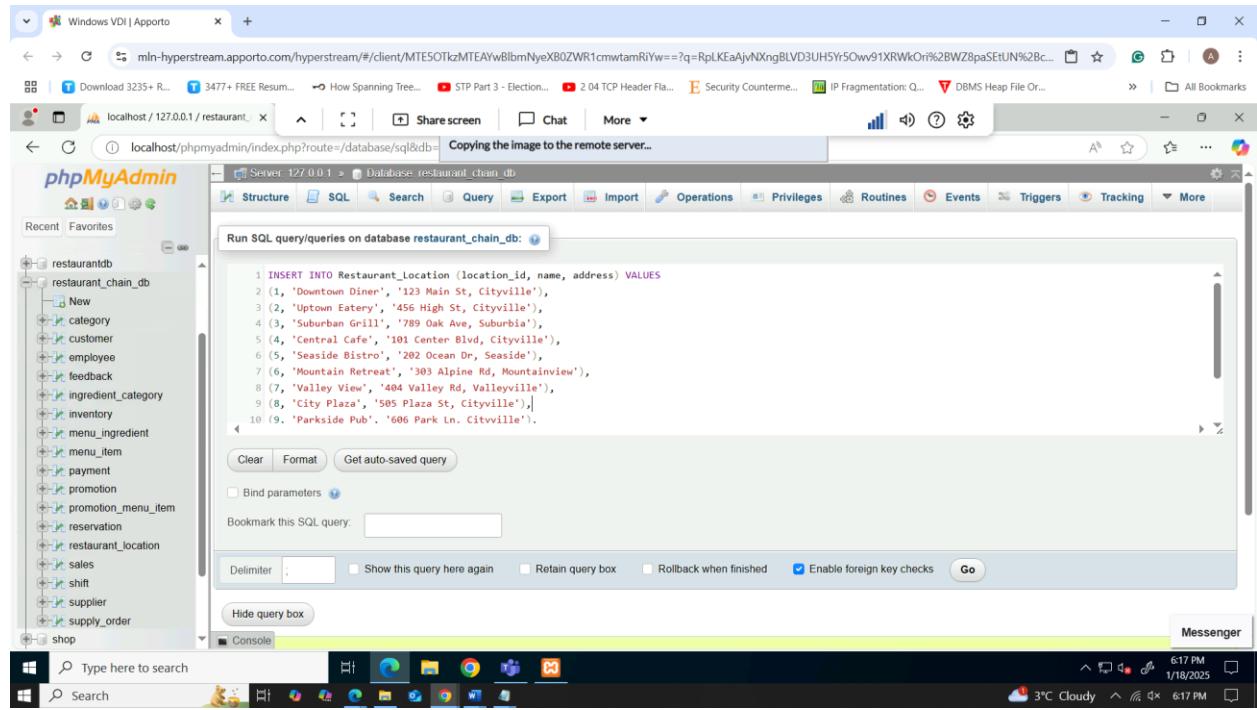


Figure 64: insert values to the restaurant location

➤ Reservation

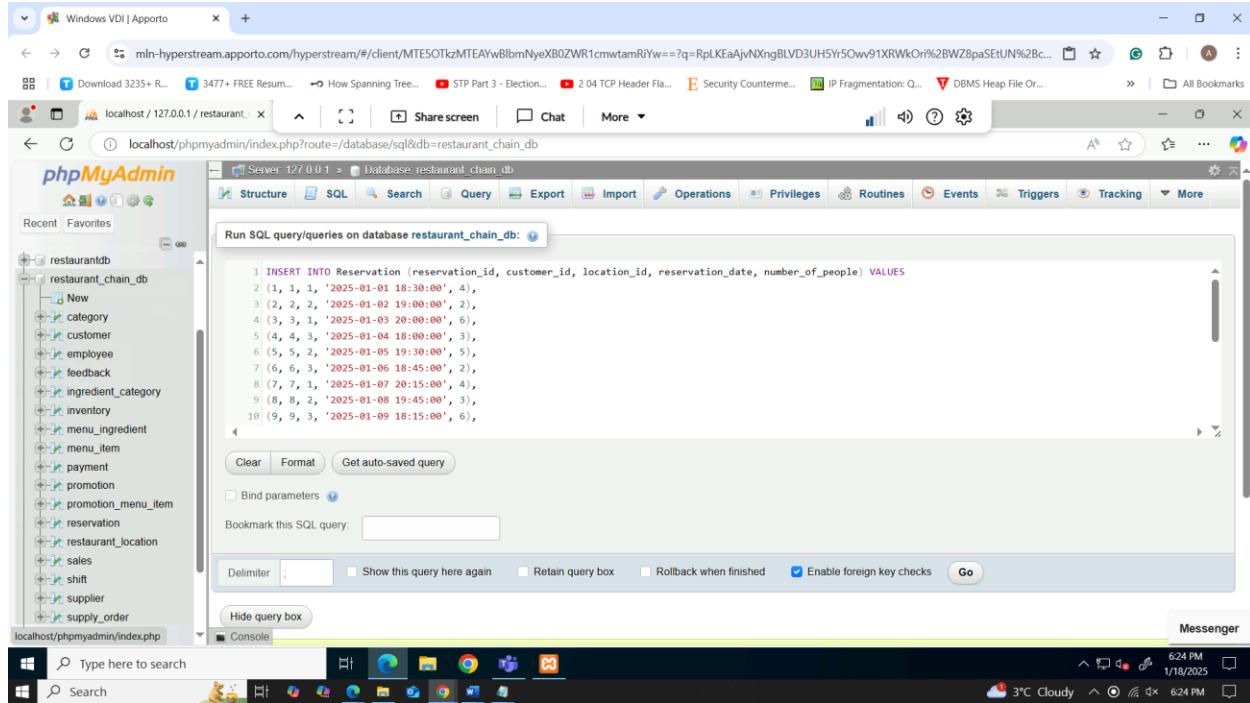


Figure 65: insert values to the reservation

➤ Sales

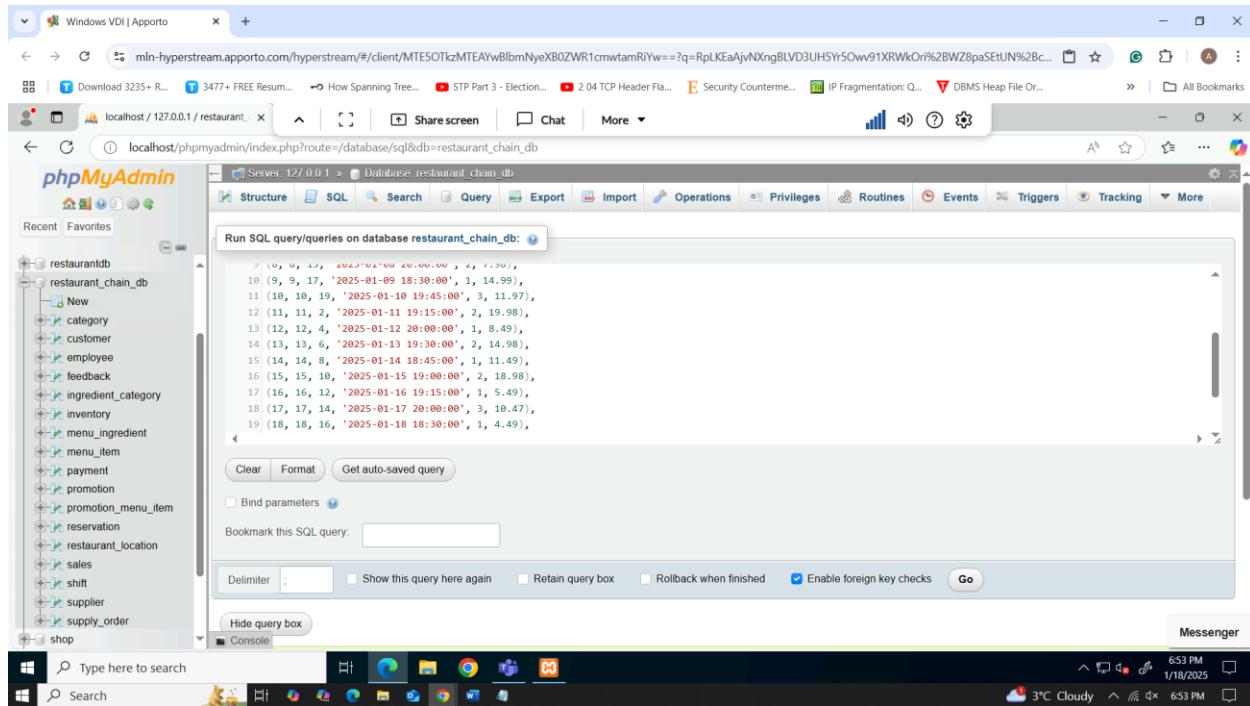
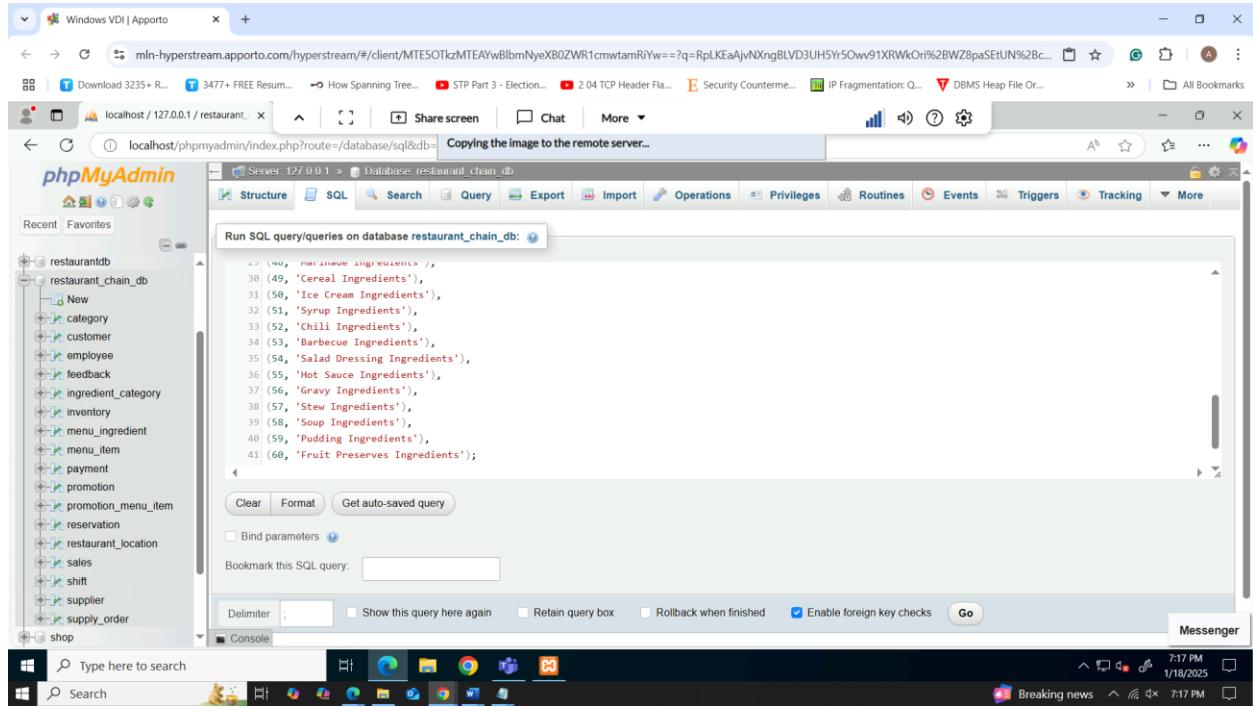


Figure 66: insert values to the sales table

➤ Ingredient category



The screenshot shows a Windows VDI session with a browser window displaying the phpMyAdmin interface. The database selected is 'restaurant_chain_db'. The left sidebar shows tables such as 'category', 'customer', 'employee', 'feedback', 'ingredient_category', 'inventory', 'menu_ingredient', 'menu_item', 'payment', 'promotion', 'promotion_menu_item', 'reservation', 'restaurant_location', 'sales', 'shift', 'supplier', and 'supply_order'. The main area contains an SQL query:

```
INSERT INTO `ingredient_category`(`id`, `name`) VALUES
(29, 'Breakfast Ingredients'),
(30, 'Cereal Ingredients'),
(31, 'Ice Cream Ingredients'),
(32, 'Syrup Ingredients'),
(33, 'Chili Ingredients'),
(34, 'Barbecue Ingredients'),
(35, 'Salad Dressing Ingredients'),
(36, 'Hot Sauce Ingredients'),
(37, 'Gravy Ingredients'),
(38, 'Stew Ingredients'),
(39, 'Soup Ingredients'),
(40, 'Pudding Ingredients'),
(41, 'Fruit Preserves Ingredients');
```

Below the query, there are buttons for 'Clear', 'Format', and 'Get auto-saved query'. There are also checkboxes for 'Bind parameters', 'Bookmark this SQL query', 'Show this query here again', 'Retain query box', 'Rollback when finished', and 'Enable foreign key checks'. A 'Go' button is at the bottom right.

Figure 67: insert values to the ingredient category

➤ Inventory

```

1 INSERT INTO Inventory (ingredient_id, name, quantity_in_stock, unit_of_measure, category_id) VALUES
2 (21, 'Olive Oil', 30, 'liters', 7),
3 (22, 'Cucumber', 50, 'kg', 3),
4 (23, 'Parmesan Cheese', 25, 'kg', 1),
5 (24, 'Tomatoes', 40, 'kg', 3),
6 (25, 'Pineapple', 15, 'kg', 9),
7 (26, 'Avocado', 20, 'kg', 3),
8 (27, 'Chicken Breast', 120, 'pieces', 5),
9 (28, 'Pork Sausage', 100, 'pieces', 2),
10 (29, 'Bell Peppers', 30, 'kg', 3),
11 (30, 'Onions', 50, 'kg', 3),
12 (31, 'Lemon', 20, 'kg', 9),
13 (32, 'Spinach', 25, 'kg', 3),
14 (33, 'Zucchini', 35, 'kg', 3),
15 (34, 'Ground Beef', 150, 'kg', 2),

```

Figure 68: insert values to the inventory table

➤ Menu_ingredient

```

1 INSERT INTO Menu_Ingredient (menu_item_id, ingredient_id, quantity_required)
VALUES
2 (1, 1, 0.2),
3 (1, 2, 0.1),
4 (2, 1, 0.2),
5 (2, 12, 0.15),
6 (3, 3, 1),
7 (3, 4, 0.05),
8 (4, 3, 1),
9 (4, 4, 0.05),
10 (5, 4, 0.1),
11 (5, 11, 0.02),
12 (6, 13, 0.1),
13 (6, 14, 0.05),
14 (7, 5, 0.2),
15 (7, 11, 0.02),

```

Figure 69: insert values to the menu ingredient table

➤ Employee

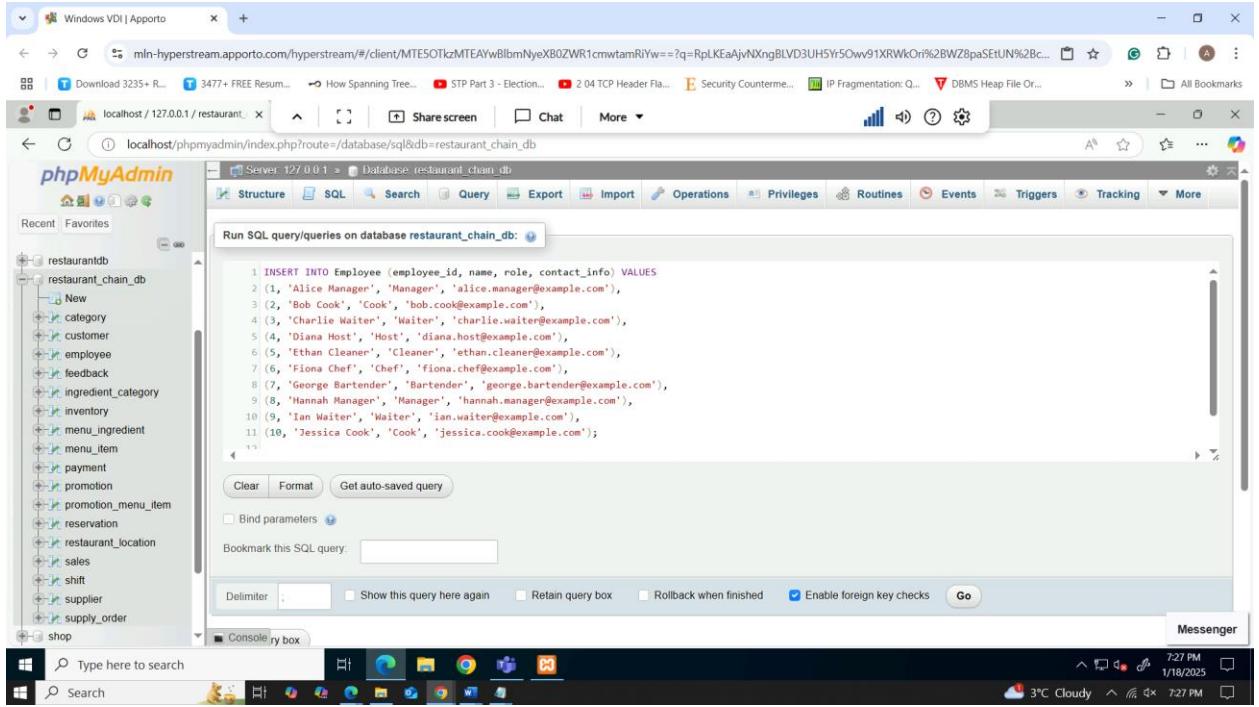


Figure 70: insert values to the employee table

➤ Shift

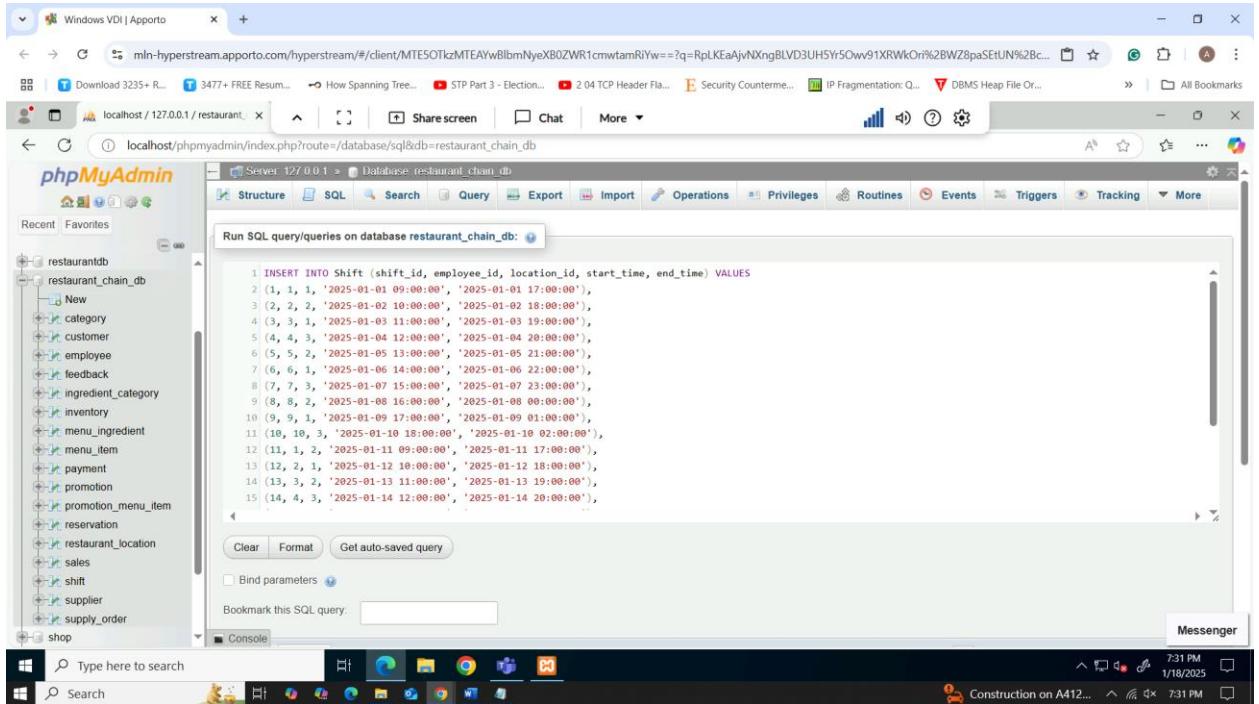
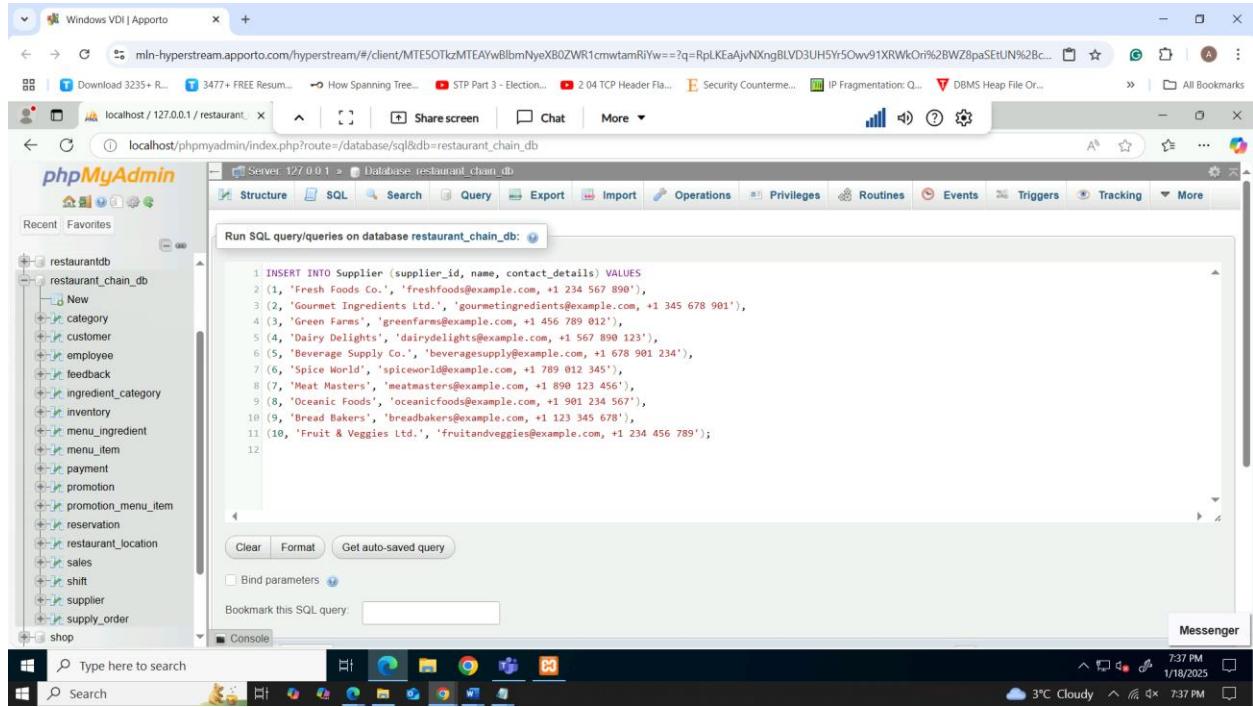


Figure 71: insert values to the shift table

➤ Supplier



The screenshot shows a Windows VDI session with multiple browser tabs open. The main focus is a phpMyAdmin interface connected to a database named 'restaurant_chain_db'. In the left sidebar, there's a tree view of tables under 'restaurantdb' and 'restaurant_chain_db', including 'New', 'category', 'customer', 'employee', 'feedback', 'ingredient_category', 'inventory', 'menu_ingredient', 'menu_item', 'payment', 'promotion', 'promotion_menu_item', 'reservation', 'restaurant_location', 'sales', 'shift', 'supplier', and 'supply_order'. The 'supplier' table is selected. The central area contains a SQL query editor with the following code:

```
1: INSERT INTO Supplier (supplier_id, name, contact_details) VALUES
2: (1, 'Fresh Foods Co.', 'freshfoods@example.com, +1 234 567 890'),
3: (2, 'Gourmet Ingredients Ltd.', 'gourmetingredients@example.com, +1 345 678 901'),
4: (3, 'Green Farms', 'greenfarms@example.com, +1 456 789 012'),
5: (4, 'Dairy Delights', 'dairydelights@example.com, +1 567 890 123'),
6: (5, 'Beverage Supply Co.', 'beveragesupply@example.com, +1 678 901 234'),
7: (6, 'Spice World', 'spiceworld@example.com, +1 789 012 345'),
8: (7, 'Meat Masters', 'meatmasters@example.com, +1 890 123 456'),
9: (8, 'Oceanic Foods', 'oceanicfoods@example.com, +1 981 234 567'),
10: (9, 'Bread Bakers', 'breadbakers@example.com, +1 123 345 678'),
11: (10, 'Fruit & Veggies Ltd.', 'fruitandveggies@example.com, +1 234 456 789');
```

Figure 72: insert values to the supplier table

➤ Supply_order

```

Run SQL query/queries on database restaurant_chain_db: ⚡

9 (6, 8, 9, 20, '2025-01-09', '2025-01-10'),
10 (9, 9, 10, 15, '2025-01-09', '2025-01-11'),
11 (10, 10, 11, 50, '2025-01-10', '2025-01-12'),
12 (11, 1, 12, 30, '2025-01-11', '2025-01-13'),
13 (12, 2, 13, 20, '2025-01-12', '2025-01-14'),
14 (13, 3, 14, 15, '2025-01-13', '2025-01-15'),
15 (14, 4, 15, 100, '2025-01-14', '2025-01-16'),
16 (15, 5, 16, 25, '2025-01-15', '2025-01-17'),
17 (16, 6, 17, 50, '2025-01-16', '2025-01-18'),
18 (17, 7, 18, 30, '2025-01-17', '2025-01-19'),
19 (18, 8, 19, 20, '2025-01-18', '2025-01-20'),
20 (19, 9, 20, 10, '2025-01-19', '2025-01-21'),
21 (20, 10, 1, 50, '2025-01-20', '2025-01-22');
22

```

Figure 73: insert values to the supply order table

➤ Payment

```

Run SQL query/queries on database restaurant_chain_db: ⚡

1 INSERT INTO Payment (payment_id, transaction_id, method, status) VALUES
2 (21, 21, 'Debit Card', 'Completed'),
3 (22, 22, 'Credit Card', 'Pending'),
4 (23, 23, 'Cash', 'Completed'),
5 (24, 24, 'Debit Card', 'Completed'),
6 (25, 25, 'Credit Card', 'Failed'),
7 (26, 26, 'Cash', 'Completed'),
8 (27, 27, 'Debit Card', 'Completed'),
9 (28, 28, 'Credit Card', 'Completed'),
10 (29, 29, 'Cash', 'Pending'),
11 (30, 30, 'Debit Card', 'Completed'),
12 (31, 31, 'Credit Card', 'Completed'),
13 (32, 32, 'Cash', 'Completed'),
14 (33, 33, 'Debit Card', 'Completed'),
15 (34, 34, 'Credit Card', 'Pending'),
...

```

Figure 74: insert values to the payment table

➤ Feedback

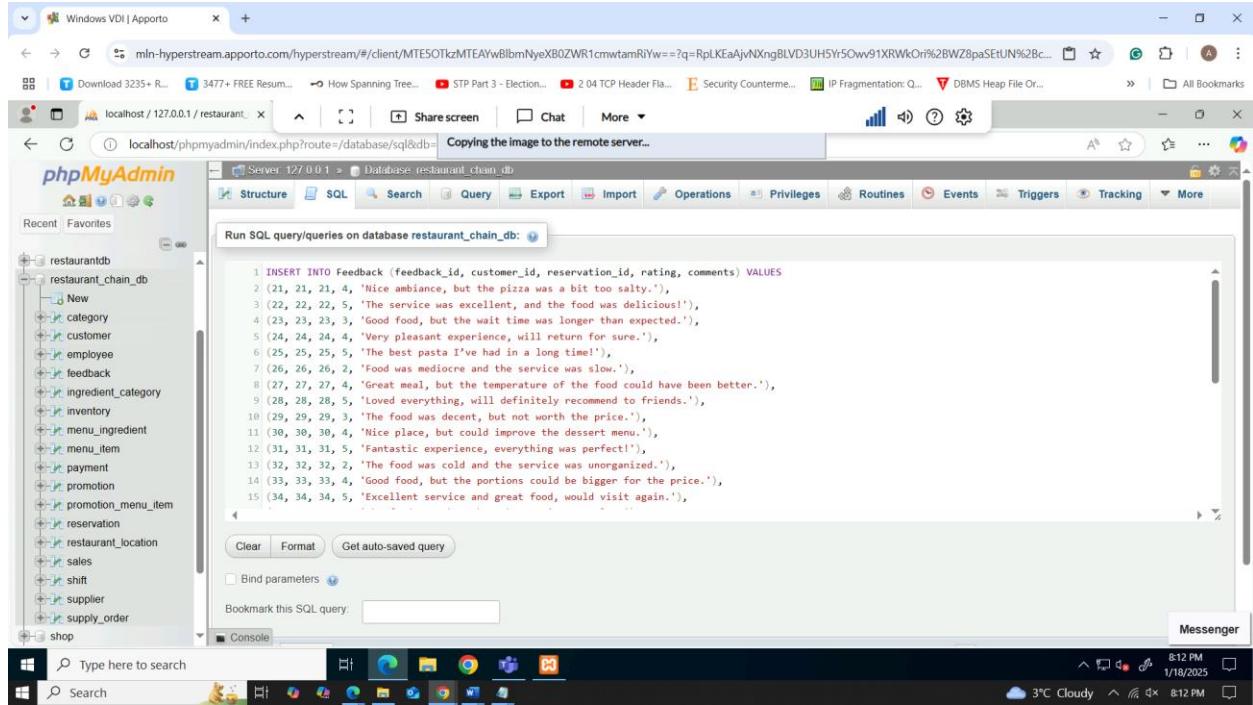


Figure 75: insert values to the feedback table

➤ Promotion

```

Run SQL query/queries on database restaurant_chain_db:

9 (6, 'Free Dessert', 200, '2023-01-07', '2023-01-10'),
10 (9, 'Early Bird', 10, '2025-01-01', '2025-01-07'),
11 (10, 'Super Saver', 5, '2025-01-01', '2025-01-31'),
12 (11, 'Winter Warmth', 15, '2025-01-15', '2025-01-31'),
13 (12, 'Buy 1 Get 1 Free', 50, '2025-01-10', '2025-01-20'),
14 (13, 'Student Discount', 20, '2025-01-01', '2025-01-31'),
15 (14, 'Holiday Special', 30, '2025-12-01', '2025-12-25'),
16 (15, 'Spring Break', 10, '2025-03-01', '2025-03-15'),
17 (16, 'Loyalty Reward', 5, '2025-01-01', '2025-12-31'),
18 (17, 'Anniversary Sale', 25, '2025-01-01', '2025-01-31'),
19 (18, 'Festive Offer', 15, '2025-12-01', '2025-12-25'),
20 (19, 'Buy 2 Get 1 Free', 33, '2025-01-01', '2025-01-15'),
21 (20, 'Chef's Special', 10, '2025-01-01', '2025-01-31);
22

```

Figure 76: insert values to the promotion table

➤ Promotion_menu_item (junction table)

```

Run SQL query/queries on database restaurant_chain_db:

1 INSERT INTO Promotion_Menu_Item (promotion_id, menu_item_id) VALUES
2 (21, 11),
3 (21, 12),
4 (22, 13),
5 (22, 14),
6 (23, 15),
7 (23, 16),
8 (24, 17),
9 (24, 18),
10 (25, 19),
11 (25, 20),
12 (26, 21),
13 (26, 22),
14 (27, 23),
15 (27, 24);


```

Figure 77: insert values to the promotion menu item table

5 Question and answers

- Total menu discipline for the different menu categories over a given date period

The screenshot shows a Windows desktop environment. In the center is a Microsoft Edge browser window displaying the phpMyAdmin interface for a database named 'restaurant_chain_db'. The left sidebar lists various tables: New, category, customer, employee, feedback, ingredient_category, inventory, menu_ingredient, menu_item, payment, promotion, promotion_menu_item, reservation, restaurant_location, sales, shift, supplier, and shop. The main area contains a SQL query editor with the following code:

```
1 SELECT
2     c.name AS category_name,
3     SUM(s.total_price) AS total_sales
4 FROM
5     Sales s
6 JOIN
7     Menu_Item m ON s.menu_item_id = m.item_id
8 JOIN
9     Category c ON m.category_id = c.category_id
10 WHERE
11     s.transaction_date BETWEEN '2025-01-01' AND '2025-01-31' -- specify the date range
12 GROUP BY
13     c.category_id
14 ORDER BY
15     total_sales DESC;
16
```

Below the query editor are three buttons: 'Clear', 'Format', and 'Get auto-saved query'. At the bottom of the browser window, there are tabs for 'Structure', 'SQL', 'Search', 'Query', 'Export', 'Import', 'Operations', 'Privileges', 'Routines', 'Events', 'Triggers', and 'Tracking'. The status bar at the bottom of the screen shows the date as 1/18/2025 and the time as 9:05 PM.

Figure 78: Total sales for each menu category

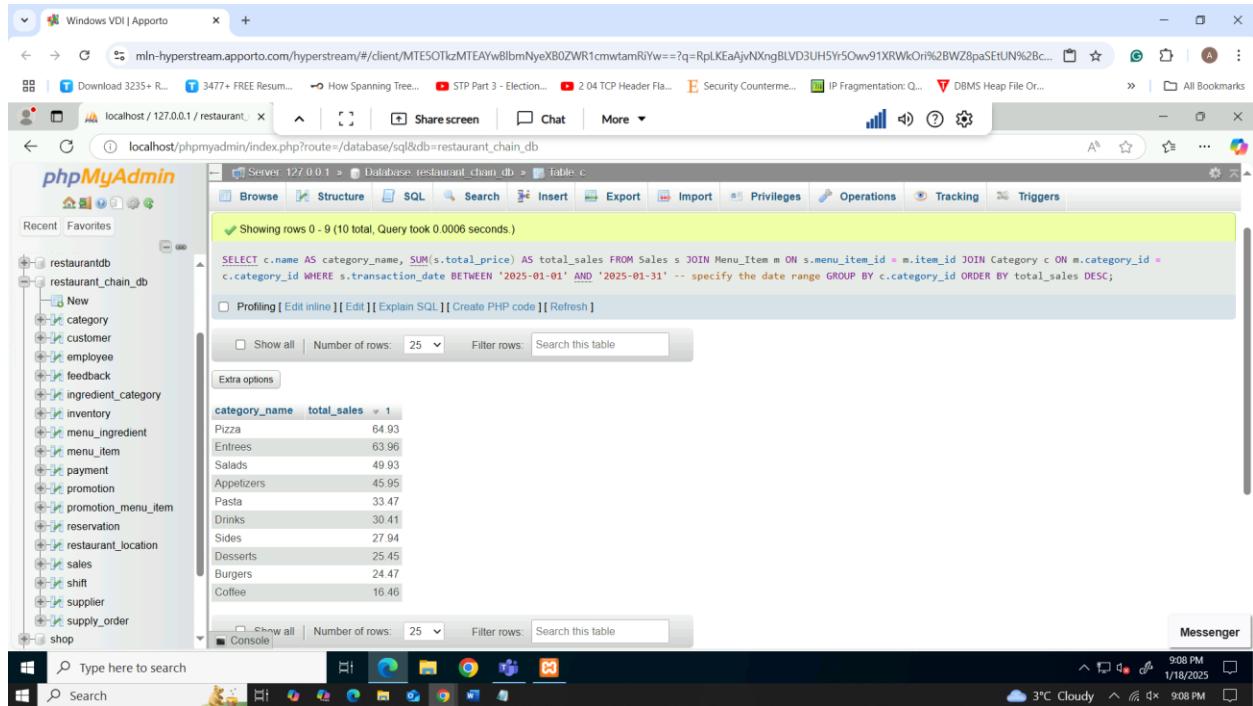


Figure 79: output of Total sales for each menu category

- Ranking of a total of 10 most ordered foods for specific days with breakdowns of name and type of food, total quantity sold

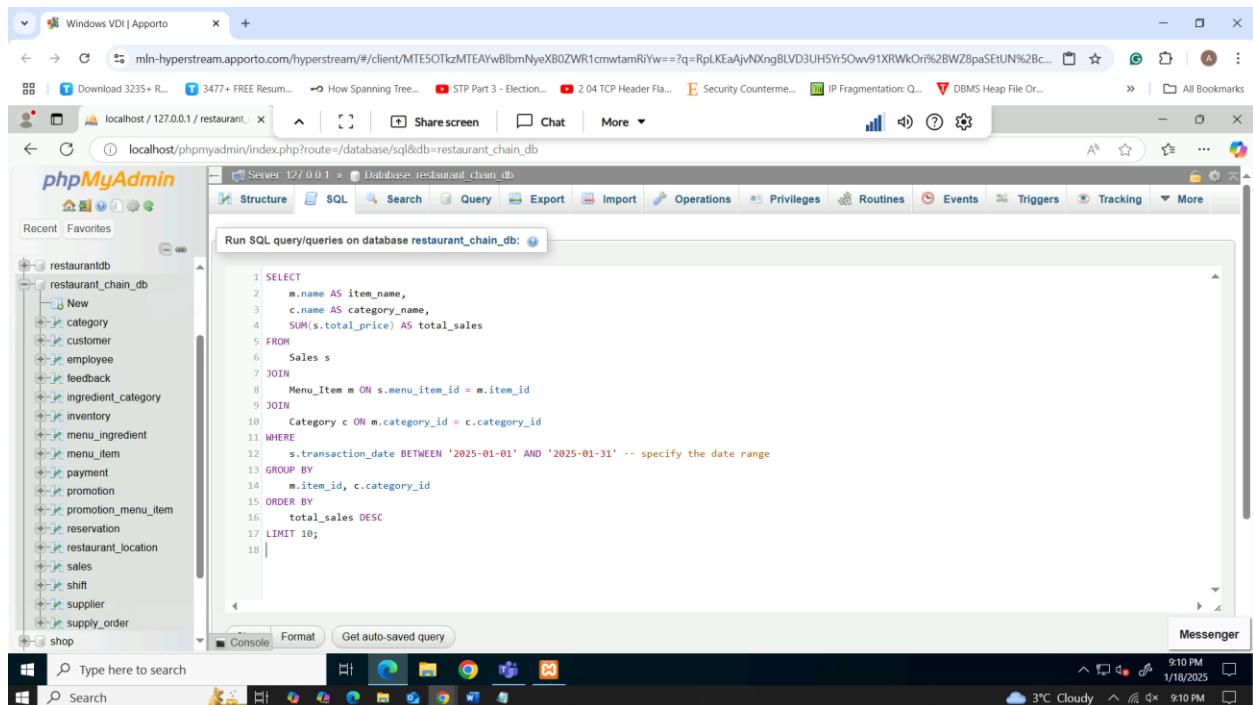


Figure 80: Top 10 most popular menu items

The screenshot shows a Windows VDI session with a browser window open to a MySQL query results page. The browser address bar shows a URL starting with 'mln-hyperstream.apporto.com/hyperstream/#/client/MTE5OTkzMTEAYwBibmNyeXB0ZWR1cmwtamRiYw==?q=RpLKeAjqNXngBLVD3UH5Yr5Ovv91XRWkOr%2BWZ8paSEtUN%2Bc...'. The main content of the browser is a phpMyAdmin interface for a database named 'restaurant_chain_db'. The 'Query' tab is selected, displaying a SQL query and its results. The results show the top 10 most popular menu items based on total sales from January 1, 2025, to January 31, 2025. The results table has columns: item_name, category_name, and total_sales. The data is as follows:

item_name	category_name	total_sales
Grilled Salmon	Entrees	44.97
Margherita Pizza	Pizza	44.95
Caesar Salad	Salads	34.95
Chicken Wings	Appetizers	26.97
Spaghetti Bolognese	Pasta	21.98
Pepperoni Pizza	Pizza	19.98
Chocolate Cake	Desserts	19.96
Iced Tea	Drinks	19.94
Ribeye Steak	Entrees	18.99
Fish Tacos	Appetizers	18.98

The browser toolbar includes links for 'Share screen', 'Chat', and 'More'. The taskbar at the bottom shows various pinned icons and the system clock indicating 9:10 PM on 1/18/2025.

Figure 81: output of Top 10 most popular menu items

- Order information with the customer's first name and last name, phone number, number of bookings made within the last 30 days

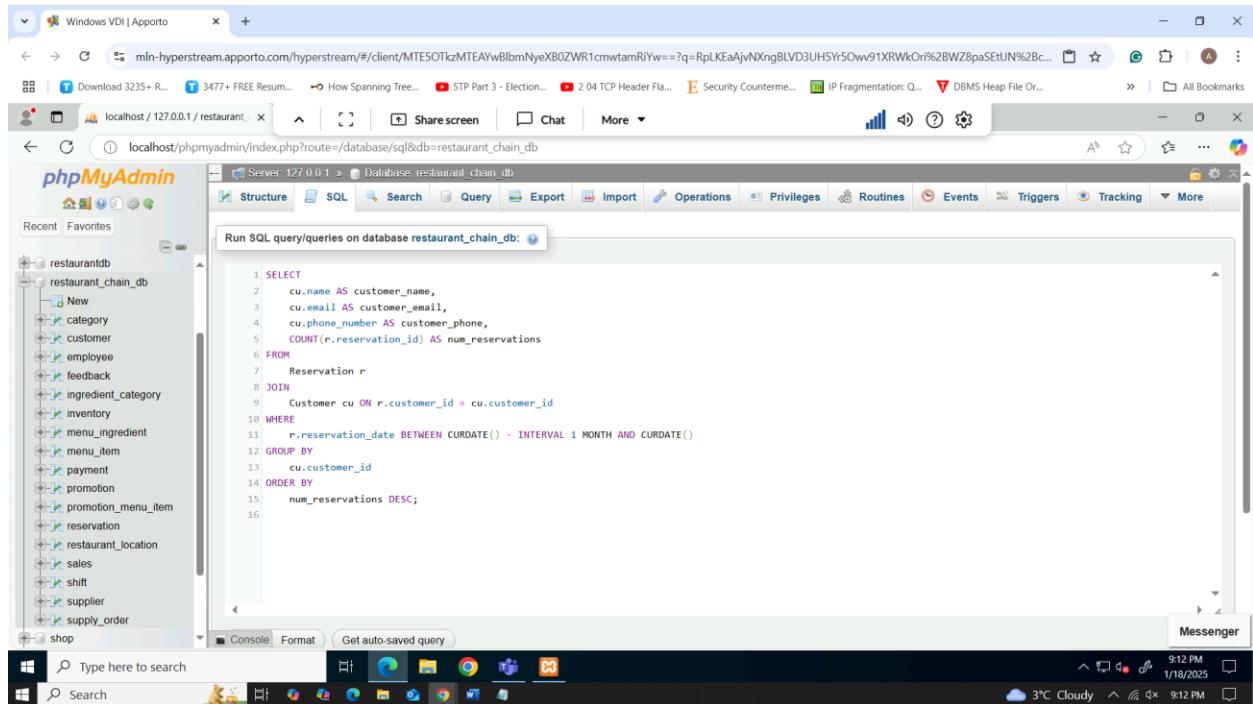
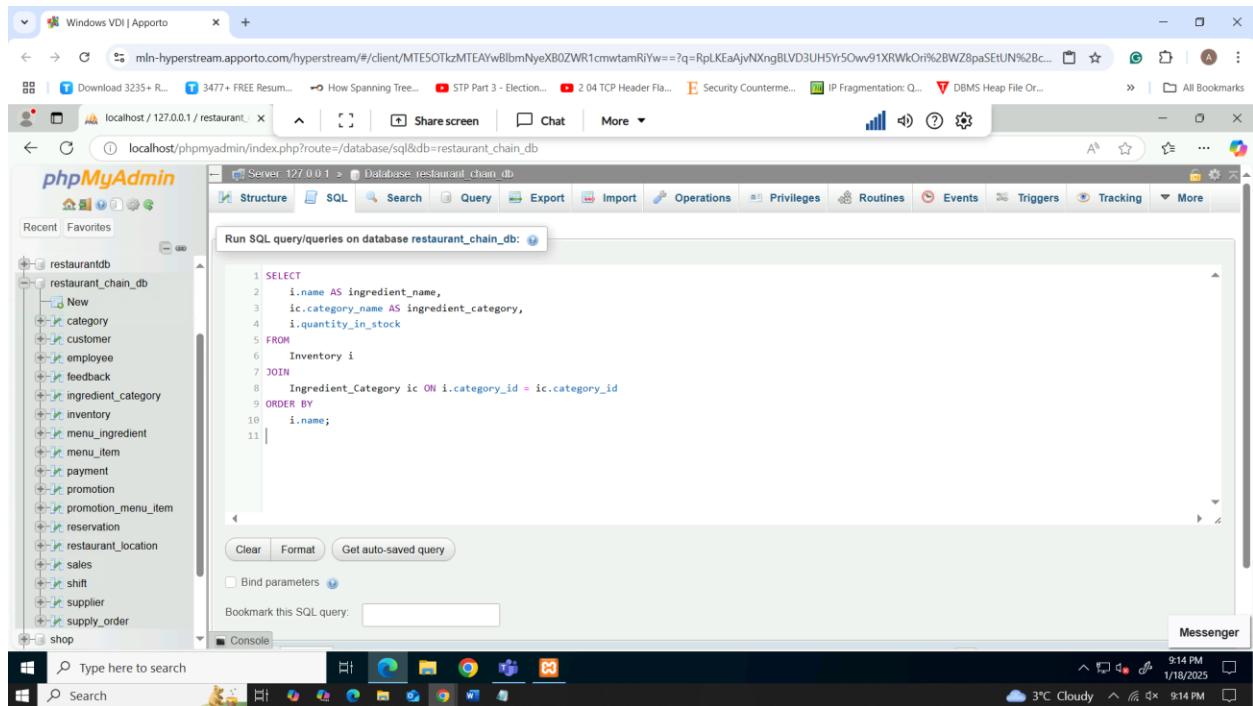


Figure 82: Customer reservation history

customer_name	customer_email	customer_phone	num_reservations
Diana Evans	diana.evans@example.com	333445555	1
Oliver Quinn	oliver.quinn@example.com	1110009999	1
Hannah Clark	hannah.clark@example.com	8889990000	1
Alice Brown	alice.brown@example.com	5551234567	1
Laura Nelson	laura.nelson@example.com	444332222	1
Ethan Foster	ethan.foster@example.com	222334444	1
Ian Martin	ian.martin@example.com	7776665555	1
Bob Johnson	bob.johnson@example.com	559876543	1
Michael Owen	michael.owens@example.com	3332221111	1
Fiona Green	fiona.green@example.com	1112223333	1
John Doe	john.doe@example.com	1234567890	1
Jessica Lewis	jessica.lewis@example.com	6667778888	1
Charlie Davis	charlie.davis@example.com	4445566666	1
Shelley Davis	shelley.davis@example.com	2224440000	1

Figure 83: output of Customer reservation history

- Basic stock position of each ingredient: The ingredient name, the type of ingredient, and the amount it is currently in store.



```

Run SQL query/queries on database restaurant_chain_db:
1 SELECT
2     i.name AS ingredient_name,
3     ic.category_name AS ingredient_category,
4     i.quantity_in_stock
5 FROM
6     Inventory i
7 JOIN
8     Ingredient_Category ic ON i.category_id = ic.category_id
9 ORDER BY
10    i.name;
11

```

Clear Format Get auto-saved query
 Bind parameters
Bookmark this SQL query:
Console

9:14 PM 1/18/2025 3°C Cloudy 9:14 PM

Figure 84: Current inventory levels by ingredient

The screenshot shows a Windows desktop environment with a browser window open to the phpMyAdmin interface. The browser's address bar shows the URL `localhost/phpmyadmin/index.php?route=/database/sql&db=restaurant_chain_db`. The phpMyAdmin interface displays a table titled "Showing rows 0 - 24 (60 total, Query took 0.0003 seconds.)" with the following data:

ingredient_name	ingredient_category	quantity_in_stock
Avocado	Salad Ingredients	20.00
Bacon	Appetizer Ingredients	80.00
Basil	Salad Ingredients	5.00
Beef Patty	Burger Ingredients	200.00
Bell Peppers	Salad Ingredients	30.00
Butter	Side Ingredients	
Carrots	Salad Ingredients	
Cheddar Cheese	Pizza Ingredients	
Chicken Breast	Appetizer Ingredients	
Chicken Wings	Appetizer Ingredients	
Chili Peppers	Salad Ingredients	
	Dessert Ingredients	

The browser's toolbar includes "Share screen", "Chat", and "More". Below the browser is a taskbar with several icons, including the XAMPP Control Panel, which is currently active. The taskbar also shows system information like the date (1/18/2025), time (9:16 PM), and weather (3°C Cloudy).

Figure 85: output of Current inventory levels by ingredient

- The total amount of money earned by each restaurant location in the past year along with the name locations

The screenshot shows a Windows VDI session with a browser window titled "Windows VDI | Apporto". The address bar contains a URL starting with "mln-hyperstream.apporto.com/hyperstream/#/client/MTE5OTkzMTEAYwBibmNyeXB0ZWR1cmwtamRiYw==?q=RpLKeAjvNXngBLVD3UH5Yr5Oww91XRWkOr%2BWZ8paSEtUN%2Bc...". Below the browser is a taskbar with various pinned icons.

The main content area displays the "localhost / 127.0.0.1 / restaurant" page of the phpMyAdmin interface. The left sidebar shows the database structure for "restaurant_chain_db", including tables like category, customer, employee, feedback, ingredient_category, inventory, menu_ingredient, menu_item, payment, promotion, promotion_menu_item, reservation, restaurant_location, sales, shift, supplier, supply_order, and shop.

The central panel shows a SQL query editor with the following code:

```

1 SELECT
2     rl.name AS location_name,
3     SUM(s.total_price) AS total_revenue
4     FROM
5     Sales s
6     JOIN
7         Reservation r ON s.customer_id = r.customer_id
8     JOIN
9         Restaurant_Location rl ON r.location_id = rl.location_id
10    WHERE
11        s.transaction_date BETWEEN '2025-01-01' AND '2025-12-31' -- specify the date range for last year
12    GROUP BY
13        rl.location_id
14    ORDER BY
15        total_revenue DESC;
16

```

Below the query editor are buttons for "Clear", "Format", and "Get auto-saved query".

Figure 86: Revenue by restaurant location

The screenshot shows the same Windows VDI session and browser setup as Figure 86. The phpMyAdmin interface now displays the results of the executed SQL query.

The message bar at the top indicates "Showing rows 0 - 2 (3 total). Query took 0.0007 seconds."

The SQL query shown in the editor is identical to the one in Figure 86.

The results table shows the following data:

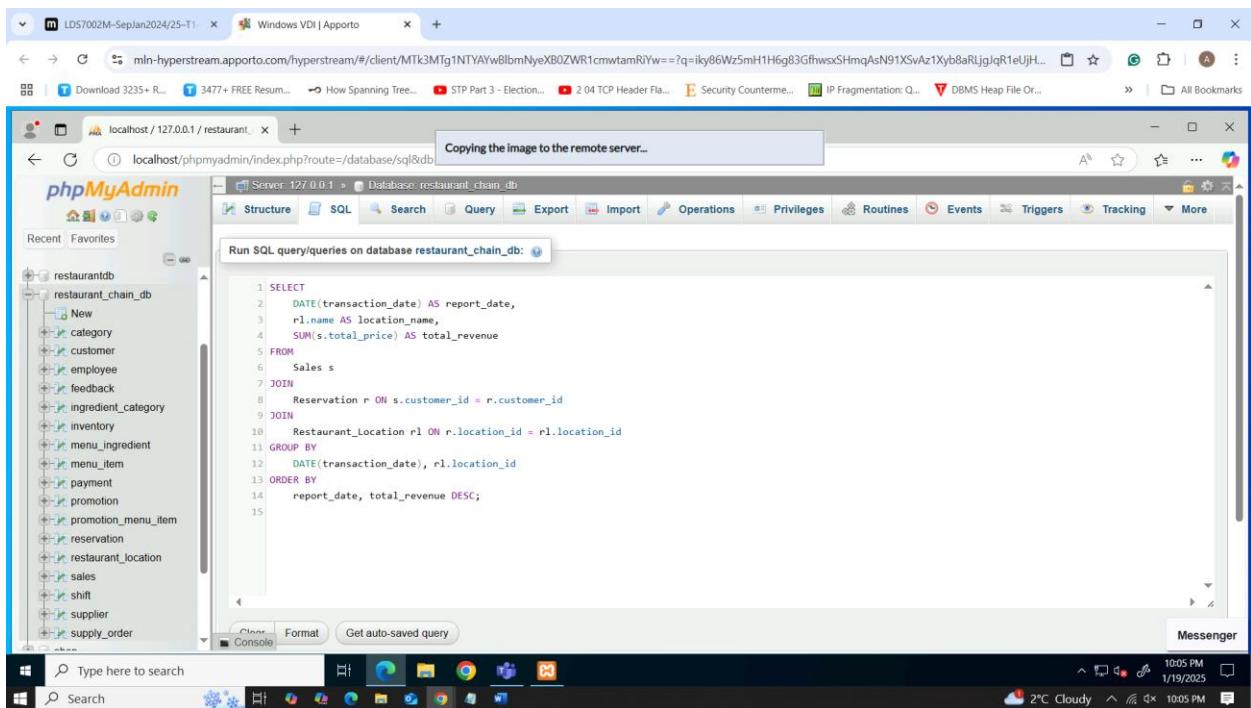
location_name	total_revenue
Uptown Eatery	248.69
Suburban Grill	235.22
Downtown Diner	208.20

Below the results table are buttons for "Print", "Copy to clipboard", "Export", "Display chart", and "Create view".

Figure 87: output of Revenue by restaurant location

6 Report and Analysis

- Daily Revenue report



The screenshot shows a Windows desktop environment with a browser window open to a phpMyAdmin interface. The browser's address bar shows a URL related to a hyperstream application. The phpMyAdmin interface is connected to a database named 'restaurant_chain_db'. On the left, a tree view lists various tables: restaurantdb, restaurant_chain_db, New, category, customer, employee, feedback, ingredient_category, inventory, menu_ingredient, menu_item, payment, promotion, promotion_menu_item, reservation, restaurant_location, sales, shift, supplier, and supply_order. The 'sales' table is selected. The main area contains a SQL query editor with the following code:

```
1 SELECT
2     DATE(transaction_date) AS report_date,
3     rl.name AS location_name,
4     SUM(s.total_price) AS total_revenue
5 FROM
6     Sales s
7     JOIN
8         Reservation r ON s.customer_id = r.customer_id
9     JOIN
10        Restaurant_Location rl ON r.location_id = rl.location_id
11 GROUP BY
12     DATE(transaction_date), rl.location_id
13 ORDER BY
14     report_date, total_revenue DESC;
```

Figure 88: daily revenue report code

The SQL statement developed computes daily total revenues by restaurant location. It opts for the transaction date (transaction_date), and groups the sales, and links them with the restaurant locations through the joins. The Sales table contains records of sales and the Reservation table relates customers through customer_id to their chosen reservations. The Restaurant_Location table specifies where each of the reservations is to be made. When linking these tables it connects the sales with certain places or outlets in the restaurants. The SUM(s.total_price) accumulates a total of revenues for each location in each day. GROUP BY clause makes sure data is dumped by date and location while ORDER BY sorts the result set by date in ascending order and by decreasing revenue. This query gives a detailed, ordered, report on the daily performance of the total sales for each restaurant branch.

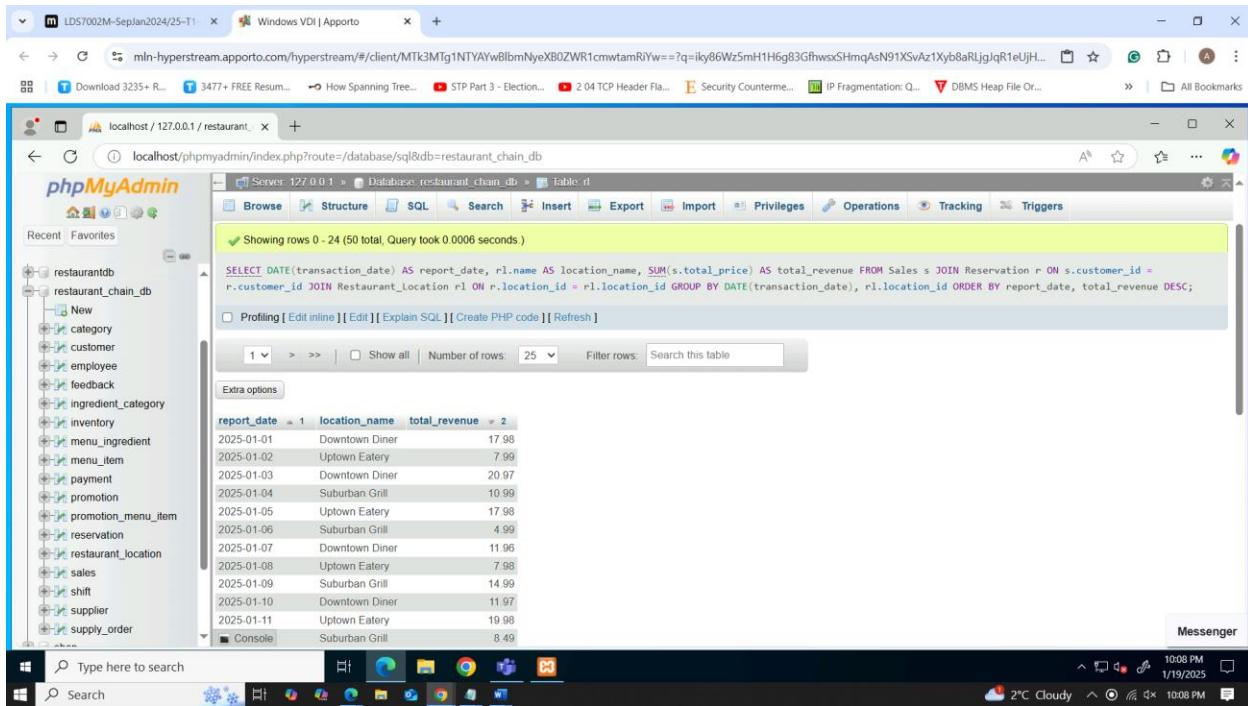
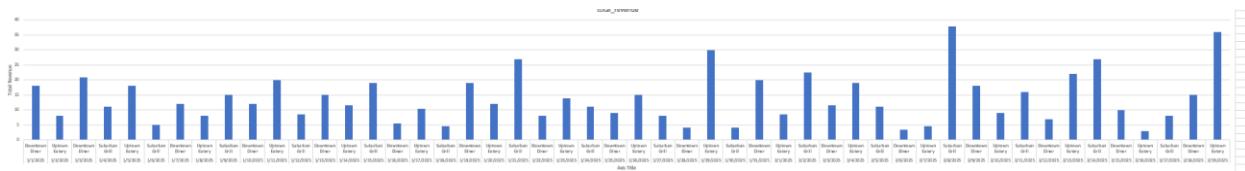


Figure 89: output of daily revenue code



The data shows daily revenue for three locations: Downtown Diner is a restaurant which implies some creation of it as a real centre of a town/square; Uptown Eatery looks like a creative and, let's say, progressive eating place in an upper-class area of a town; and finally, Suburban Grill which can be associated with a modern house in a suburban area or just a traditional diner. There are certain months when Revenues at the two restaurants vary, for instance, Suburban Grill, \$ 37.98 in the 8/2/2025 and Uptown Eatery \$ 35.97 in 19/2/2025. Downtown Diner steady but lower revenues are earned. It can be inferred that Uptown Eatery revenues rise towards the end of January, while the subtlety of Suburban Grill. This mean promotions could extend to Downtown Diner to encourage better performance, while at other areas, peak days could be optimally utilized. Comparing weekdays and weekends can also help to advance recognition of potential for the greatest total sales.

➤ Itemized Sales report by Category

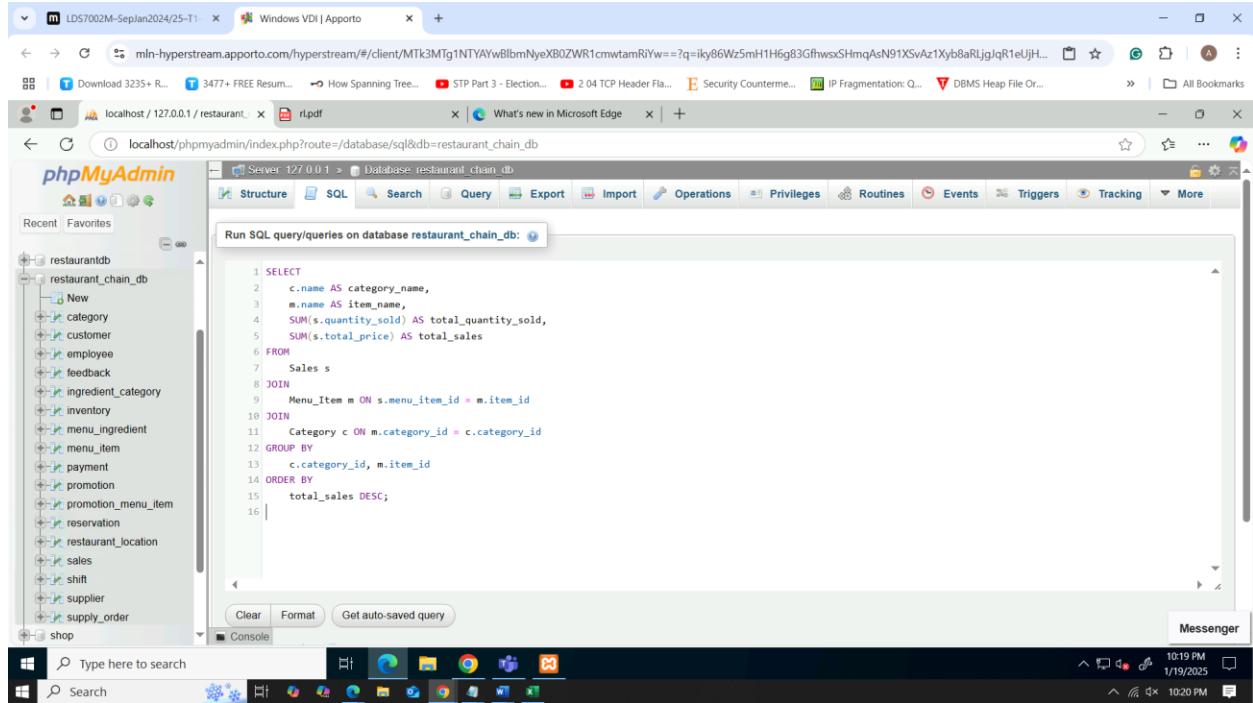


Figure 90: itemized sales query code

This SQL query creates a general report of the menu item and category sales with the emphasis on the number of items sold or total quantity and the total amount of money made from these items. The query selects the category name (c.name), item name (m.name), total quantity sold from table of status (SUM(s.quantity_sold)), and the total sales revenue from the same table (SUM(s.total_price)). It achieves this by joining three tables: Sales, that records some sales transactions, Menu_Item which associates the menu item with the sales records, and Category, to categorize the menu items. The GROUP BY clause filters data by category and particular menu item, to calculate totals, the operation has to be done for every item within the category. Plans and goods are sorted by total sales, which gives the list of items arranged in descending order and pointing at the items with the highest sales rates. This query is useful in determining market movers, category performance and planning for inventory, price changes and adding or eliminating products in the menu.

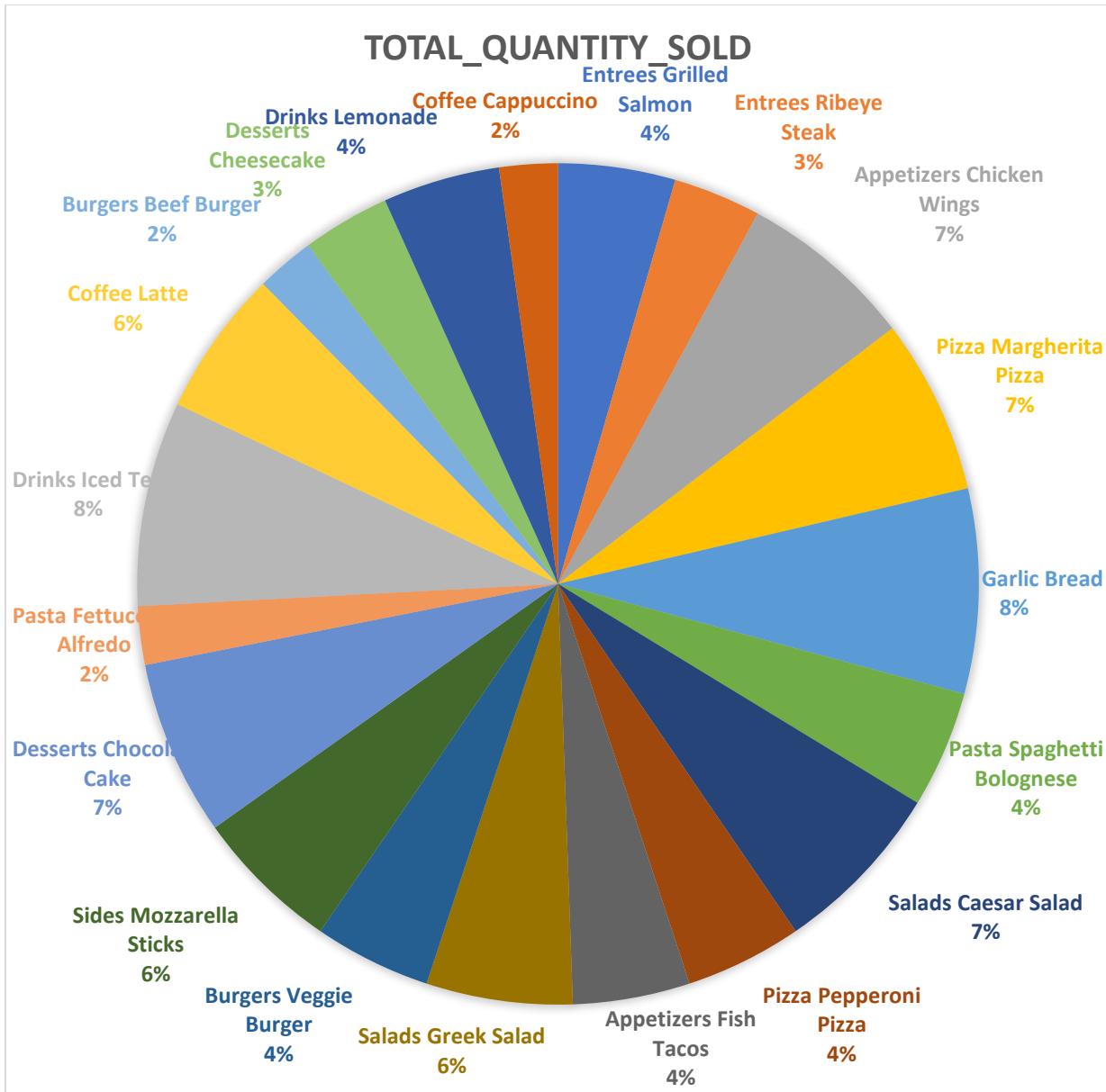
The screenshot shows a Windows desktop environment with a browser window open to a phpMyAdmin interface. The browser's address bar shows the URL `localhost/phpmyadmin/index.php?route=/database/sql&db=restaurant_chain_db`. The phpMyAdmin interface displays a query results page for the database `restaurant_chain_db`. The query is:

```
SELECT c.name AS category_name, m.name AS item_name, SUM(s.quantity_sold) AS total_quantity_sold, SUM(s.total_price) AS total_sales FROM Sales s JOIN Menu_Item m ON s.menu_item_id = m.item_id JOIN Category c ON m.category_id = c.category_id GROUP BY c.category_id, m.item_id ORDER BY total_sales DESC;
```

The results table shows the following data:

category_name	item_name	total_quantity_sold	total_sales
Entrees	Grilled Salmon	4	59.96
Entrees	Ribeye Steak	3	56.97
Appetizers	Chicken Wings	6	53.94
Pizza	Margherita Pizza	6	53.94
Sides	Garlic Bread	7	51.93
Pasta	Spaghetti Bolognese	4	43.96
Salads	Caesar Salad	6	41.94
Pizza	Pepperoni Pizza	4	39.96
Appetizers	Fish Tacos	4	37.96
Salads	Greek Salad	5	37.45
Burgers	Veggie Burger	4	31.96
Sides	Mozzarella Sticks	5	29.95
	Chocolate Cake	6	29.94

Figure 91: outcome of itemized sales



Using an intersection of sales summary by category in a pie chart displays the total contribution of each category to overall sales and show customers' favorite food items and those that are most popular. This perspective is relevant to striking the best balance of inventories in a restaurant and changing the menu with some items prioritized. Therefore, if appetizers or beverages tend to sell either higher than entrees, there could be promotions or new additions in that category boost revenues. As with poor quality, variable cost drivers may be altered: some products might be redesigned or rejected to increase gross margin. The calculation of the performance of the distinct categories enables restaurants to detect whether inventory requirements match consumption levels to offer customers preferred products, implement seasonal or promotional menus, and optimize total sales and cost.

➤ Inventory report

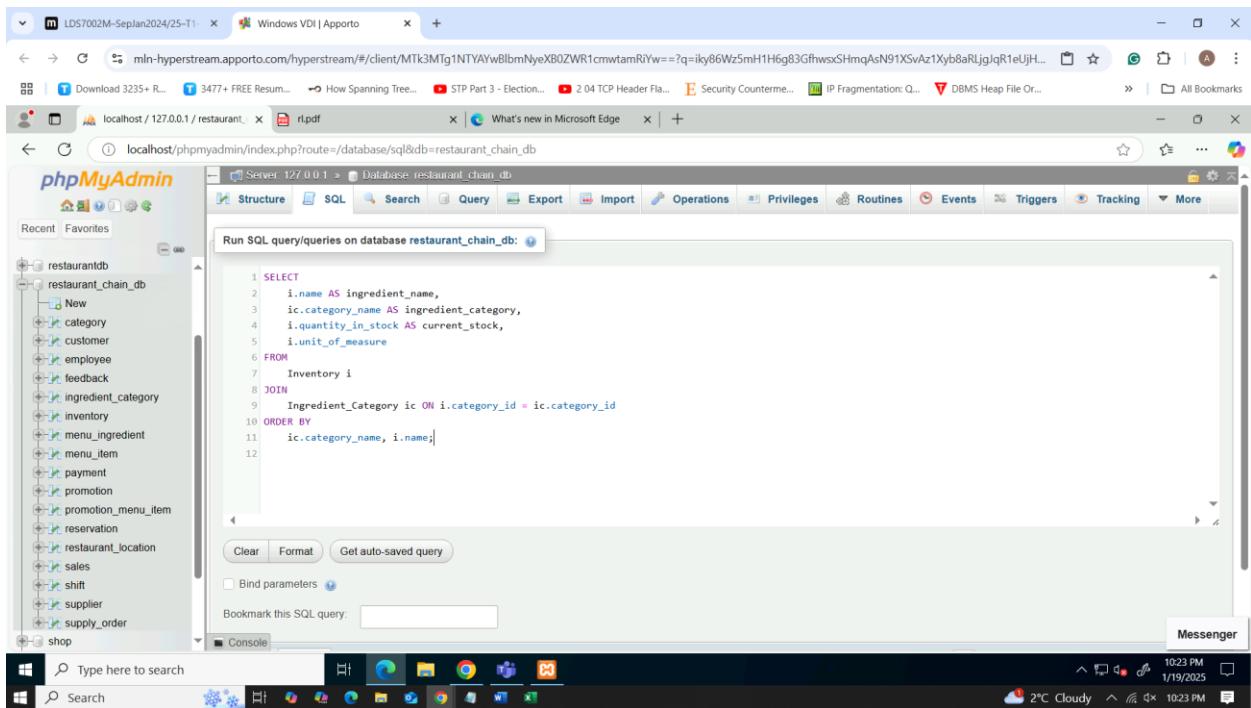
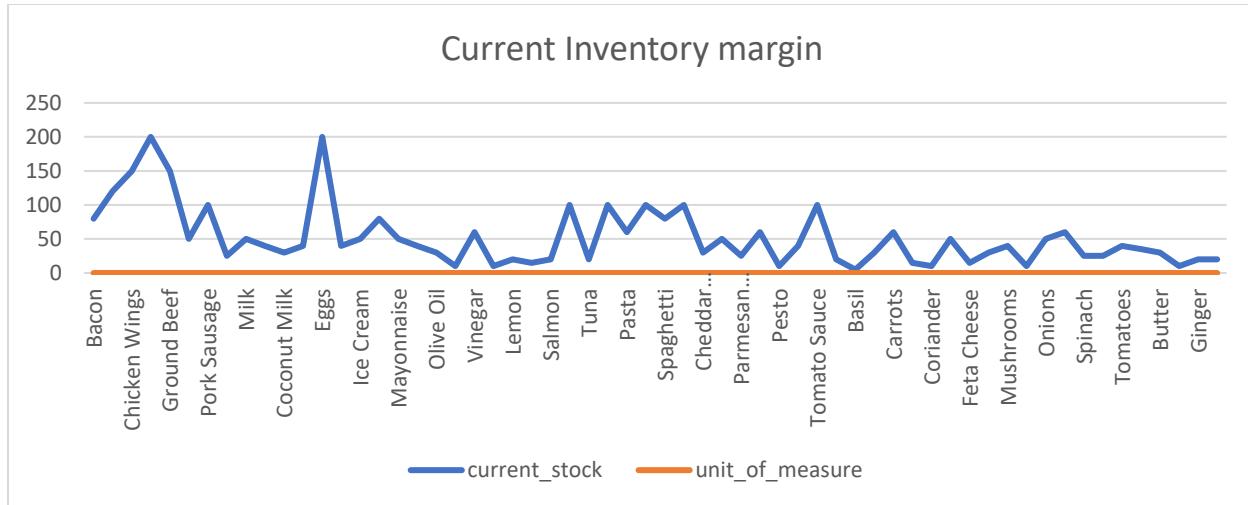


Figure 92: inventory report query code

The below SQL query gives the detailed inventory report and includes all the ingredients with their respective categories, quantity available and units. It brings back the name of the ingredient (i.name, called i for ingredient), the category to which the ingredient belongs (ic.category_name, called ic for ingredient category), the quantity of the ingredient in stock (i.quantity_in_stock), and the unit of measure of that quantity (i.unit_of_measure). The query links, through joining, between the table Inventory, which concern ingredient detail, and Ingredient_Category table which categorize the ingredient, based on the common identification number of category_id. Finally the last line orders the result set first by the name of the category in ascending alphabetical order then the ingredients inside each category. Such an arrangement makes the checking of inventories easier as far as categories are concerned and identification of certain ingredients easy. The query enables the organization to manage its inventory as it is precise and covers various aspects of the stock in detail to ensure that the businesses get an overview of the existing food ingredients that they need to order in order to restock.3.

The screenshot shows a Windows desktop environment. In the background, there are several browser tabs open, including one for 'mln-hyperstream.apporto.com' and another for 'localhost / 127.0.0.1 / restaurant'. In the foreground, a Microsoft Edge window is active, displaying the phpMyAdmin interface for a database named 'restaurant_chain_db'. The left sidebar lists various tables such as 'category', 'customer', 'employee', 'feedback', 'ingredient_category', 'inventory', 'menu_ingredient', 'menu_item', 'payment', 'promotion', 'reservation', 'restaurant_location', 'sales', 'shift', 'supplier', and 'supply_order'. The main query results page shows a table with columns 'ingredient_name', 'ingredient_category', 'current_stock', and 'unit_of_measure'. The table data includes items like Bacon, Chicken Breast, Chicken Wings, Beef Patty, Ground Beef, Milk, Eggs, Coconut Milk, Ice Cream, Mayonnaise, Olive Oil, Vinegar, Lemon, Salmon, Tuna, Pasta, Spaghetti, Cheddar..., Parmesan..., Pesto, Tomato Sauce, Basil, Carrots, Coriander, Feta Cheese, Mushrooms, Onions, Spinach, Tomatoes, Butter, and Ginger. The current stock values range from 25 to 200, while the unit of measure is consistently 'kg' or 'pieces'.

Figure 93: results of inventory margin



The inventory data provides current stock data for Appetizers, Burgers, Desserts, Drinks, and Salads, in terms of Current Stock. Products such as Chicken wings (150) and beef patty (200) show ample stock, on the other hand, sesame oil (10 litres) and worcestershire sauce (10 litres) may be stocked up loosely. This is because the degree of consumption is high in such categories as Burger & Salad Ingredients; hence require vigilance in order not to stock out. Stock levels can be presented in the form of a bar chart in order to easily pinpoint which items are critical. Subsequent tracking overtime can improve the purchasing decision since purchases will be made basing on the stock available rather than making assumptions and

end up with a surplus. The major idea of this strategy is that it helps to organize restaurants' work, controlling the constantly used products and giving proper attention to the products that are low in stock.