CAFE DATABASE MANAGEMENT SYSTEM

CS 5318

FINAL PHASE DOCUMENT

Professor: Shengli Yuan

By:
Arkaprava Hajra
Kiran Feroz
Shikha Mehta

Project title: Cafe Database Management System

Team name: Data Brewers

Team members:

FULL NAME
Arkaprava Hajra
Kiran Feroz
Shikha Mehta

Contents

1. Abstract

The **Cafe Database Management System** is a software solution designed to support the operations of a cafe. The system offers a range of features to help cafes manage their day-to-day activities, from taking orders and managing inventory to generating sales reports and analyzing customer feedback. The system includes multiple user views that support the needs of various stakeholders, including customers, employees, managers, owners, and financial accountants.

The system will be designed to be user-friendly and intuitive, making it easy for cafe staff to use the system to manage their daily tasks. The project will involve designing and implementing a database schema, developing user interfaces, and integrating the system with other software solutions as needed.

The Cafe Database Management System has the potential to improve the efficiency and effectiveness of cafe operations. By providing real-time access to key business metrics, the system can help cafes optimize their operations, reduce waste, and increase profits. The system can also enhance the customer experience by enabling customers to easily place orders and provide feedback, making it more likely that they will return to the cafe in the future. Overall, the Cafe Database Management System is an innovative and practical solution that can support the growth and success of cafes.

Based on the application being modeled, the Cafe Database Management System, the following constraints need to be imposed:

1. **Security constraints**: The system must ensure that all user data, including customer information, employee information, and financial information, is secure and protected from unauthorized access. This can be achieved through the use of authentication and authorization mechanisms, encryption, and other security measures.
2. **Data consistency constraints**: The system must maintain data consistency across all users and all views. This means that any changes made to the system must be reflected across all relevant views and data stores. For example, if a customer places an order, that order must be reflected in the inventory view, the financial view, and the employee view.
3. **Performance constraints**: The system must be able to handle a large volume of requests and transactions in real-time, without experiencing any significant performance degradation. This means that the system must be designed to be scalable, with the ability to handle increasing volumes of data and users as the cafe grows.
4. **Availability constraints**: The system must be available 24/7 to support the needs of customers and employees. This means that the system must be designed with redundancy and fault tolerance in mind, to ensure that the system can continue to operate in the event of hardware or software failures.
5. **Regulatory constraints**: The system must comply with any applicable regulations or standards related to the collection, storage, and processing of customer and employee data. This may include regulations related to data privacy, financial reporting, and food safety.
6. **Backup and Recovery**: The system should include regular backups to protect against data loss due to system failure or other unforeseen events. The backup and recovery process should be tested regularly to ensure that it is working effectively.

These constraints are critical to the success of the Cafe Database Management System, and they must be carefully considered during the design, development, and deployment of the system. By meeting these constraints, the system can ensure that it is secure, reliable, and scalable, and that it can support the needs of cafes and their stakeholders.

2. Mission Statement

This system will manage data related to the customer, staff, menu, inventory, delivery order and reservations. Our system will empower cafe owners, managers, staff, and customers with the tools they need to easily access and analyze critical data. Our system would strive to make data management effortless and accessible, freeing up time and resources for cafe owners and managers to focus on delivering an exceptional customer experience.

3. Mission Objectives

- To maintain (enter, update and delete) data on staff.
- To maintain (enter, update and delete) data on customer.
- To maintain (enter, update and delete) data on menu.
- To maintain (enter, update and delete) data on inventory.
- To maintain (enter, update and delete) data on orders.
- To maintain (enter, update and delete) data on order items.
- To maintain (enter, update and delete) data on suppliers.
- To maintain (enter, update and delete) data on online deliveries.
- To maintain (enter, update and delete) data on sales.
- To maintain (enter, update and delete) data on bills.

- To perform searches on staff.
- To perform searches on customer.
- To perform searches on bills.
- To perform searches on orders and the items related to it.
- To perform searches on menu.
- TO perform searches on sales.
- To perform searches on deliveries.
- To perform searches on suppliers and current inventory.

- To track status of current order.
- To track status of to-go order.
- To track status of pickup order.
- To track status of delivery order.

- To track status of supplies.
- To track status of sales made.

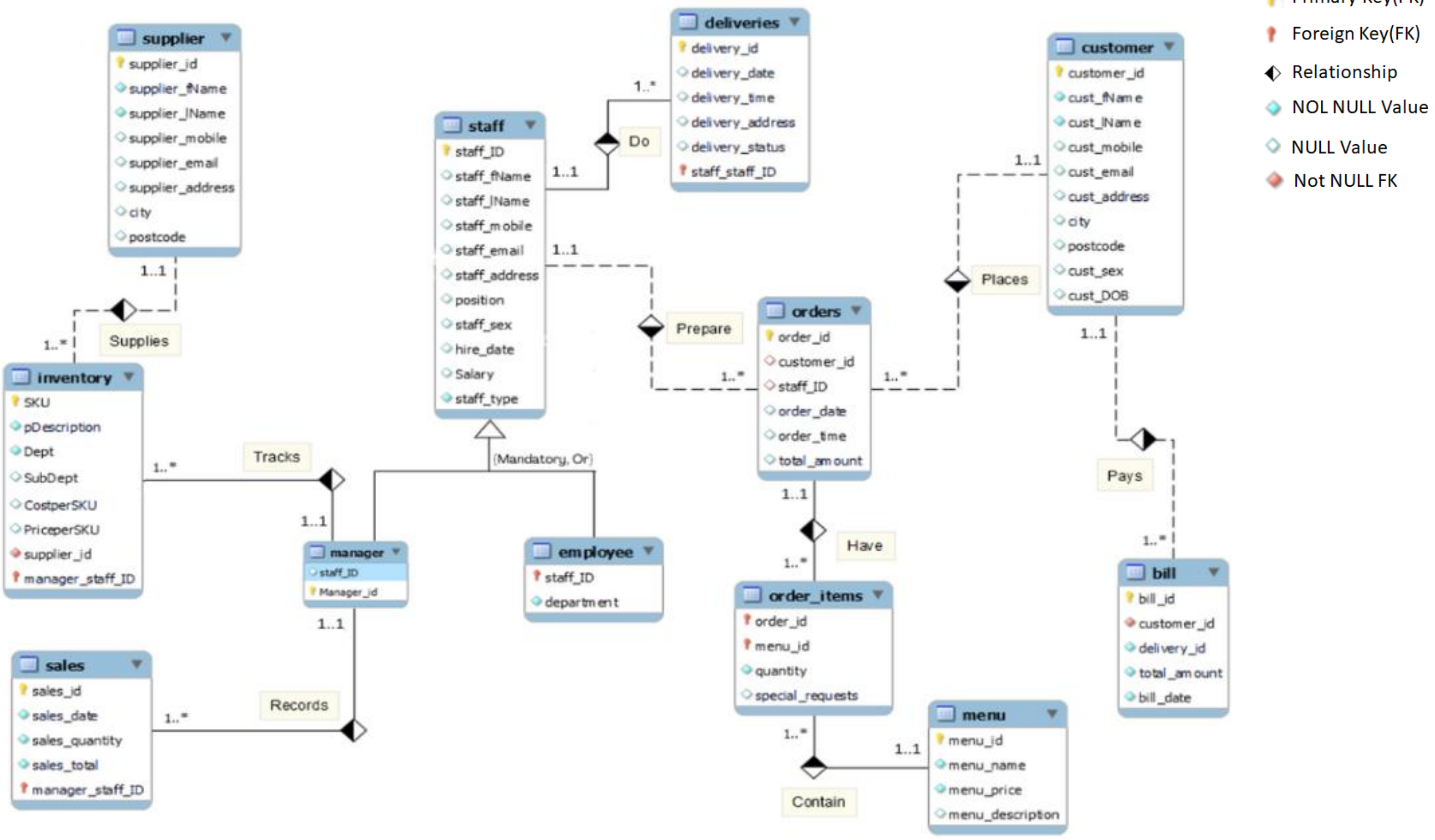
- To report on staff.
- To report on inventory.
- To report on customer.
- To report on suppliers
- To report on deliveries.
- To report on sales made.
- To report on expenses.

4. Major User Views

A cafe database management system may have different user views to support the needs of various stakeholders. Here are some potential user views that could be included:

Data	Access type	Owner	Manager	Staff	Customer	Finance Expert
All Staff	**Maintain**		**X**			
	Query	**X**	**X**			
	Report	**X**	**X**			
Customer	**Maintain**		**X**			
	Query	**X**	**X**		**X**	
	Report		**X**			
Suppliers	**Maintain**	**X**				
	Query	**X**	**X**			**X**
	Report	**X**				
Menu	**Maintain**	**X**	**X**			
	Query	**X**	**X**	**X**	**X**	
	Report		**X**	**X**		
Orders	**Maintain**		**X**			
	Query	**X**	**X**	**X**	**X**	
	Report		**X**	**X**		
Order_items	**Maintain**		**X**	**X**		
	Query	**X**	**X**	**X**	**X**	
	Report	**X**	**X**			
Deliveries	**Maintain**		**X**			
	Query		**X**	**X**	**X**	**X**
	Report	**X**	**X**	**X**		
Inventory	**Maintain**	**X**	**X**			
	Query	**X**	**X**			
	Report	**X**	**X**			
Sales	**Maintain**	**X**	**X**			
	Query	**X**	**X**			**X**
	Report	**X**	**X**			**X**
Invoices/Bills	**Maintain**	**X**				**X**
	Query	**X**				**X**
	Report	**X**	**X**			**X**

5. Complete E/R diagram



6. Relational Model

A. Customer

Attribute	Domain	Constraints	Default Value	Primary Key	Candidate Key	Foreign Key	Create table Statement
customer_id	char(6)	NOT NULL	N/A	Yes	Yes		CREATE TABLE customer (customer_id char(6) PRIMARY KEY NOT NULL, cust_fName varchar(10) NOT NULL, cust_lName varchar(10) NOT NULL, cust_mobile char(15), cust_email varchar(50), cust_address varchar(50), city varchar(10), postcode varchar(10), cust_sex char(1), cust_DOB date);
cust_fName	varchar(10)	NOT NULL	N/A				
cust_lName	varchar(10)	NOT NULL	N/A				
cust_mobile	char(15)	Check if a valid mobile number is entered	N/A				
cust_email	varchar(50)	Check if a valid email address is entered	N/A				
cust_address	varchar(50)	NOT NULL	N/A				
city	varchar(10)	NOT NULL	N/A				
postcode	varchar(10)	NOT NULL	N/A				
cust_sex	char(1)	Check if 'M', 'F' or 'O' (Male, Female, Other) is entered	N/A				
cust_DOB	date	Check if the date is in the past and not in the future	N/A				

Attribute Functional Dependencies:

Customer_id -> cust_fName, cust_lName, cust_mobile, cust_email, cust_address, city, postcode, cust_sex, cust_DOB
Cust_email -> Customer_id
Cust_mobile -> Customer_id

1NF: This table is already in 1NF, as each cell has only one data value from its domain.

2NF: For 2NF, we need to ensure that all non-primary key attributes are dependent on the entire primary key. In this case, all the non-primary key attributes are fully dependent on the primary key (Customer_id), so the table is already in 2NF.

3NF: For 3NF, we need to ensure that there are no transitive dependencies. In this table, there are no such dependencies, so the table is already in 3NF.

BCNF: In this table, all functional dependencies have Customer_id as the determinant, which is the primary key, so the table is already in BCNF.

B. Menu

Attribute	Domain	Constraints	Default Value	Primary Key	Candidate Key	Foreign Key	Create table Statement
menu_id	int	Unique, NOT NULL, > 0	N/A	Yes	Yes		CREATE TABLE menu (menu_id int PRIMARY KEY, menu_name varchar(50) NOT NULL, menu_price decimal(10,2) NOT NULL, menu_description varchar(255));
menu_name	varchar (50)	Unique, NOT NULL	N/A		Yes		
menu_price	decimal (10,2)	NOT NULL,> 0	N/A				
menu_description	varchar (255)		NULL				

Attribute Functional Dependencies:

Menu_id -> menu_name, menu_price, menu_description

1NF: This table is already in 1NF, as each cell has only one data value from its domain.

2NF: For 2NF, we need to ensure that all non-primary key attributes are dependent on the entire primary key. In this case, all the non-primary key attributes are fully dependent on the primary key (menu_id), so the table is already in 2NF.

3NF: For 3NF, we need to ensure that there are no transitive dependencies. In this table, there are no such dependencies, so the table is already in 3NF.

BCNF: In this table, all functional dependencies have menu id as the determinant, which is the primary key, so the table is already in BCNF.

C. Inventory

Attribute	Domain	Constraints	Default Value	Primary Key	Candidate Key	Foreign Key	Create table Statement
SKU	varchar(9)	NOT NULL	N/A	Yes	Yes		CREATE TABLE inventory (SKU varchar(9) NOT NULL PRIMARY KEY, pDescription varchar(50) NOT NULL, Dept varchar(9) NOT NULL, SubDept varchar(9), CostperSKU decimal(18, 2), PriceperSKU decimal(18, 2), supplier_id char(6), Manager_id varchar(6) NOT NULL, FOREIGN KEY (supplier_id) REFERENCES Supplier(supplier_id), FOREIGN KEY (Manager_id) REFERENCES Manager(Manager_id));
pDescription	varchar(50)	NOT NULL	N/A				
Dept	varchar(9)	NOT NULL	N/A				
SubDept	varchar(9)	N/A	NULL				
CostperSKU	decimal(18, 2)	>0	NULL				
PriceperSKU	decimal(18, 2)	>0	NULL				
supplier_id	char(6)	N/A	NULL			supplier_id From Supplier Table	
Manager_id	varchar(6)	NOT NULL	N/A			Manager_id From Manager Table	

Attribute Functional Dependencies:

SKU -> pDescription, Dept, subDept, CostPerSKU, PricePerSKU

SKU, supplier_id -> Manager_id

1NF: This table is already in 1NF, as each cell has only one data value from its domain.

2NF: For 2NF, we need to ensure that all non-primary key attributes are dependent on the entire primary key. In this case, all the non-primary key are fully dependent on the primary key (SKU), so the table is already in 2NF.

3NF: For 3NF, we need to ensure that there are no transitive dependencies. In this table, there are no such dependencies, so the table is already in 3NF.

BCNF: In this table, all functional dependencies have SKU as the determinant, which is the primary key, so the table is already in BCNF.

D. Sales

Attribute	Domain	Constraints	Default Value	Primary Key	Candidate Key	Foreign Key	Create table Statement
sales_id	int	NOT NULL	N/A	Yes	Yes		CREATE TABLE sales (sales_id INT PRIMARY KEY NOT NULL, menu_id INT NOT NULL, sales_date DATE NOT NULL, sales_quantity INT NOT NULL, sales_total decimal(10,2) NOT NULL, FOREIGN KEY (menu_id) REFERENCES menu(menu_id) FOREIGN KEY (Manager_id) REFERENCES Manager(Manager_id));
menu_id	int	NOT NULL	N/A	No	Yes		
sales_date	date	NOT NULL. Check if the date is in the past and not in the future	N/A	No			
sales_quantity	int	NOT NULL, > 0	N/A	No			
sales_total	decimal(10,2)	NOT NULL, > 0	N/A	No			
Manager_id	varchar(6)	NOT NULL	N/A			Manager_id From Manager table	

Attribute Functional Dependencies:

Sales_id -> menu_id, sales_date, sales_quantity, sales_total, Manager_id

1NF: This table is in 1NF because each cell has only one data from the attribute domain.

2NF: For 2NF, we need to ensure that all non-primary key attributes are dependent on the entire primary key. In this case, all the non-primary key attributes are fully dependent on the primary key (sales_id), so the table is already in 2NF.

3NF: For 3NF, we need to ensure that there are no transitive dependencies. In this table, there are no such dependencies, so the table is already in 3NF.

BCNF: In this table, all functional dependencies have sales id as the determinant, which is the primary key, so the table is already in BCNF.

E. Supplier

Attribute	Domain	Constraints	Default Value	Primary Key	Candidate Key	Foreign Key	Create table Statement
supplier_id	Char(6)	NOT NULL	N/A	Yes	Yes		CREATE TABLE Supplier (supplier_id char(6) PRIMARY KEY NOT NULL, supplier_fName varchar(10) NOT NULL, supplier_lName varchar(10) NOT NULL, supplier_mobile char(15), supplier_email varchar(50), supplier_address varchar(50), city varchar(10), postcode varchar(10));
Supplier_fName	Varchar(10)	NOT NULL	N/A				
Supplier_lName	Varchar(10)	NOT NULL	N/A				
Supplier_mobile	Char(15)	N/A	N/A				
Supplier_email	Varchar(50)	N/A	N/A				
Supplier_address	Varchar(50)	N/A	N/A				
city	Varchar(10)	N/A	N/A				
postcode	Varchar(10)	N/A	N/A				

Attribute Functional Dependencies:

Supplier_id -> supplier_fName, supplier_lName, supplier_mobile, supplier_email, supplier_address, city, postcode

1NF: This table is already in 1NF because each cell has only one data from the attribute domain.

2NF: This table is also in 2NF because all the non-key attributes are fully dependent on the primary key "supplier_id".

3NF: For 3NF, we need to ensure that there are no transitive dependencies. In this table, there are no such dependencies, so the table is already in 3NF.

BCNF: In this table, all functional dependencies have supplier id as the determinant, which is the primary key, so the table is already in BCNF.

F. Bill

Attribute	Domain	Constraints	Default Value	Primary Key	Candidate Key	Foreign Key	Create table Statement
bill_id	int	NOT NULL, unique, positive	N/A	Primary Key			CREATE TABLE bill (bill_id int PRIMARY KEY NOT NULL, customer_id char(6) NOT NULL, staff_ID char(6) NOT NULL, order_id int NOT NULL, delivery_id int NOT NULL, total_amount decimal(10,2) NOT NULL, bill_date date NOT NULL, FOREIGN KEY (customer_id) REFERENCES customer(customer_id), FOREIGN KEY (staff_ID) REFERENCES staff(staff_ID), FOREIGN KEY (order_id) REFERENCES orders(order_id));
customer_id	char(6)	NOT NULL	N/A			customer_id From Customer table	
staff_ID	char(6)	NOT NULL	N/A			staff_ID From Staff table	
order_id	int	NOT NULL, positive	N/A			order_id From Orders table	
delivery_id	int	NOT NULL, positive	N/A				
total_amount	decimal(10,2)	NOT NULL, positive, non-negative	N/A				
bill_date	date	NOT NULL	N/A				

Attribute Functional Dependencies:

Bill_id -> customer_id, staff_ID, order_id, delivery_id, total_amount, bill_date

Order_id -> customer_id, staff_ID, total_amount

Delivery_id -> customer_id, staff_ID

1NF: This table is in 1NF because each cell has only one data from the attribute domain.

2NF: This table is in 2NF because there is only one candidate key (bill_id), and all non-key attributes depend on the primary key.

3NF: For 3NF, we need to ensure that there are no transitive dependencies. In this table, there are no such dependencies, so the table is already in 3NF.

BCNF: In this table, all functional dependencies have bill id as the determinant, which is the primary key, so the table is already in BCNF.

G. Staff (Superclass)

Attribute	Domain	Constraints	Default Value	Primary Key	Candidate Key	Foreign Key	Create table Statement
staff_ID	char(5)	NOT NULL	N/A	Yes	Yes		CREATE TABLE staff (staff_ID char(5) PRIMARY KEY, staff_fName varchar(10), staff_lName varchar(10), staff_mobile char(15), staff_email varchar(50), staff_address varchar(50), position varchar(10), staff_sex char(1), hire_date date, Salary decimal(18, 2));
staff_fName	varchar(10)	NOT NULL	N/A				
staff_lName	varchar(10)	NOT NULL	N/A				
staff_mobile	char(15)	Check if a valid mobile number is entered	N/A				
staff_email	varchar(50)	Check if a valid email address is entered	N/A				
staff_address	varchar(50)	NOT NULL	N/A				
position	varchar(10)	NOT NULL	N/A				
staff_sex	char(1)	Check if 'M', 'F' or 'O' (Male, Female, Other) is entered	N/A				
hire_date	date	Check if the date is in the past and not in the future	N/A				
Salary	decimal(18, 2)	NOT NULL	N/A				

Attribute Functional Dependencies:

Staff_ID -> staff_fName, staff_lName, staff_mobile, staff_email, staff_address, position, staff_sex, hire_date, salary

Staff_email -> staff_ID, staff_fName, staff_lName, staff_mobile, staff_email, staff_address, position, staff_sex, hire_date, salary

Staff_mobile -> -> staff_ID, staff_fName, staff_lName, staff_email, staff_address, position, staff_sex, hire_date, salary

1NF: This table is in 1NF because each cell has only one data from the attribute domain.

2NF: For 2NF, we need to ensure that all non-primary key attributes are dependent on the entire primary key. In this case, all the non-primary key attributes are fully dependent on the primary key (staff_ID), so the table is already in 2NF.

3NF: For 3NF, we need to ensure that there are no transitive dependencies. In this table, there are no such dependencies, so the table is already in 3NF.

BCNF: In this table, all functional dependencies have staff id as the determinant, which is the primary key, so the table is already in BCNF.

H. Manager (Subclass)

Attribute	Domain	Constraints	Default Value	Primary Key	Candidate Key	Foreign Key	Create table Statement
staff_ID	char(5)	NOT NULL	N/A			staff_ID From Staff Table	CREATE TABLE Manager (staff_ID CHAR(5), Manager_id VARCHAR(6) NOT NULL PRIMARY KEY, FOREIGN KEY (staff_ID) REFERENCES staff(staff_ID));
Manager_id	varchar(6)	NOT NULL	N/A	Yes			

Attribute Functional Dependencies:

Staff_ID -> Manager_id

1NF: This table is in 1NF because each cell has only one data from the attribute domain.

2NF: This table is in 2NF because the non-key attribute depends on the entire primary key (staff_ID), and there are no partial dependencies.

3NF: For 3NF, we need to ensure that there are no transitive dependencies. In this table, there are no such dependencies, so the table is already in 3NF.

BCNF: In this table, the single functional dependency has staff id as the determinant, which is the primary key, so the table is already in BCNF.

I. Employee (Subclass)

Attribute	Domain	Constraints	Default Value	Primary Key	Candidate Key	Foreign Key	Create table Statement
staff_ID	char(5)	NOT NULL	N/A	Yes		staff_ID From Staff Table	CREATE TABLE Employee (staff_ID CHAR(5), department VARCHAR(25) NOT NULL, FOREIGN KEY (staff_ID) REFERENCES staff(staff_ID));
Department	Varchar(25)	NOT NULL	N/A				

Attribute Functional Dependencies:

Staff_ID -> department

1NF: This table is in 1NF because each cell has only one data from the attribute domain.

2NF: For 2NF, we need to ensure that all non-primary key attributes are dependent on the entire primary key. In this case, the non-primary key attribute is fully dependent on the primary key (staff_ID), so the table is already in 2NF.

3NF: For 3NF, we need to ensure that there are no transitive dependencies. In this table, there are no such dependencies, so the table is already in 3NF.

BCNF: In this table, the single functional dependency has staff id as the determinant, which is the primary key, so the table is already in BCNF.

J. Deliveries

Attribute	Domain	Constraints	Default Value	Primary Key	Candidate Key	Foreign Key	Create table Statement
delivery_id	int	NOT NULL	N/A	Yes	Yes		CREATE TABLE deliveries (delivery_id int PRIMARY KEY, staff_ID char(5), order_id int, delivery_date date, delivery_time time, delivery_address varchar(50), delivery_status varchar(20), FOREIGN KEY (staff_ID) REFERENCES staff(staff_ID), FOREIGN KEY (order_id) REFERENCES orders(order_id));
staff_ID	char(5)	NOT NULL	N/A			staff_ID from Staff table	
order_id	int	NOT NULL	N/A			order_ID from orders table	
delivery_date	date	Check the date is not in the past	N/A				
delivery_time	time	Check delivery time is in past	N/A				
delivery_address	varchar(50)	NOT NULL	N/A				
delivery_status	varchar(20)	NOT NULL	N/A				

Attribute Functional Dependencies:

Delivery_id -> staff_ID, order_id, delivery_date, delivery_time, delivery_address, delivery_status

1NF: This table is in 1NF because each cell has only one data from the attribute domain.

2NF: For 2NF, we need to ensure that all non-primary key attributes are dependent on the entire primary key. In this case, all the non-primary key attributes are fully dependent on the primary key (delivery_id), so the table is already in 2NF.

3NF: For 3NF, we need to ensure that there are no transitive dependencies. In this table, there are no such dependencies, so the table is already in 3NF.

BCNF: In this table, all functional dependencies have delivery id as the determinant, which is the primary key, so the table is already in BCNF.

K. Orders

Attribute	Domain	Constraints	Default Value	Primary Key	Candidate Key	Foreign Key	Create table Statement
order_id	int	NOT NULL	N/A	Yes			CREATE TABLE orders (order_id int PRIMARY KEY, customer_id char(6), staff_ID char(5), order_date date, order_time time, total_amount decimal(18,2), CONSTRAINT fk_customer FOREIGN KEY (customer_id) REFERENCES customer(customer_id), CONSTRAINT fk_staff FOREIGN KEY (staff_ID) REFERENCES staff(staff_ID));
customer_id	char(6)	NOT NULL	N/A			customer_id from customer table	
staff_ID	char(5)	NOT NULL	N/A			staff_ID from staff table	
order_date	date	Check the date is not in the future	N/A				
order_time	time	Check order time is in past	N/A				
total_amount	Decimal(18,2)	NOT NULL	N/A				

Attribute Functional Dependency:

Order_id -> customer_id, staff_ID, order_date, order_time, total_amount

1NF: This table is in 1NF because each cell has only one data from the attribute domain.

2NF: For 2NF, we need to ensure that all non-primary key attributes are dependent on the entire primary key. In this case, all the non-primary key attributes are fully dependent on the primary key (order_id), so the table is already in 2NF.

3NF: For 3NF, we need to ensure that there are no transitive dependencies. In this table, there are no such dependencies, so the table is already in 3NF.

BCNF: In this table, all functional dependencies have order id as the determinant, which is the primary key, so the table is already in BCNF.

L. Order Items

Attribute	Domain	Constraints	Default Value	Primary Key	Candidate Key	Foreign Key	Create table Statement
order_id	int	NOT NULL	N/A	Yes		order_id from orders table	CREATE TABLE order_items (order_id INT NOT NULL, menu_id INT NOT NULL, quantity INT NOT NULL, special_requests varchar(255), PRIMARY KEY (order_id,menu_id), FOREIGN KEY (order_id) REFERENCES orders(order_id), FOREIGN KEY (menu_id) REFERENCES menu(menu_id));
menu_id	int	NOT NULL	N/A	Yes		menu_id from menu table	
quantity	int	NOT NULL	N/A				
special_requests	Varchar(255)	Check the date is not in the future	N/A				

Attribute Functional Dependencies:

Order_id, menu_id -> quantity, special_request

1NF: This table is in 1NF because each cell has only one data from the attribute domain.

2NF: For 2NF, we need to ensure that all non-primary key attributes are dependent on the entire primary key. In this case, all the non-primary key attributes are fully dependent on the primary key (order_id, menu_id), so the table is already in 2NF.

3NF: For 3NF, we need to ensure that there are no transitive dependencies. In this table, there are no such dependencies, so the table is already in 3NF.

BCNF: In this table, all functional dependencies have order id and menu id as the determinant, which is the primary key, so the table is already in BCNF.

7. Complete List of Use Cases and Realization

List all actors (i.e., users) of your database

1. **Customers** - who place orders, make payments and receive deliveries
2. **Staff** - who take orders, and process bills. Staff will also deliver orders.
3. **Managers** - who manage inventory, update menu and pricing, and monitor sales.
4. **System administrators** - who manage and maintain the database system.
5. **Owner** – View overall functioning of the system for their owned café
6. **Accountant** – View and download invoices, sales and expenses.

Enhance the use cases as follows: For each entity, you must have use cases that perform at least one aggregate query, one insert operation, one deletes operation, and one update operation; for each relationship, you must have use cases that perform at least one joint query. (Number your use cases. That's a minimum of 34 use cases for 7 entities and 2-person team, and 44 use cases for 9 entities and 3-person team)
Under each use case description, write down the complete SQL statement(s) needed to realize the use case

A. Use cases for customer table:

1. Aggregate Query:

Use Case Name:	**To find the total number of customers**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. Manager navigates to the "Customer" data view in the system. 3. Performs the below query 4. Generates the view for total number of customers.
Query:	SELECT COUNT(*) AS Customer_count FROM customer;

2. Insert Operation:

Use Case Name:	**To add a new customer**
Actor/User:	Manager/Customer
Steps:	1. The Manager/Customer logs into the database system. 2. Then navigates to the "Customer" data view in the system. 3. Performs the below query 4. Adds the details if they are a new customer. 5. System displays a confirmation message
Query:	INSERT INTO customer (customer_id, cust_fName, cust_lName, cust_mobile, cust_email, cust_address, city, postcode, cust_sex, cust_DOB) VALUES ('C10001', 'John', 'Doe', '+1-234-567-8901', 'johndoe@example.com', '123 Main St', 'Anytown', '12345', 'M', '1990-01-01');

3. Delete Operation:

Use Case Name:	**To delete a customer with a specific customer_id**
Actor/User:	Manager
Steps:	**1.** The Manager logs into the database system. **2.** Then navigates to the "Customer" data view in the system. **3.** Performs the below query **4.** System gets updated
Query:	DELETE FROM customer WHERE customer_id = 'C`10001';

4. Update Operation:

Use Case Name:	**To update the mobile number of a customer with a specific customer_id**
Actor/User:	Manager/Customer
Steps:	**5.** The Manager/Customer logs into the database system. **6.** Then navigates to the "Customer" data view in the system. **7.** Performs the below query **8.** System gets updated

Query:	UPDATE customer
	SET cust_mobile = '+1987654321', cust_address = '456 Second St', city = 'Chicago', postcode = '60601'
	WHERE customer_id = '10001';

B. Use cases for inventory table

5. Aggregate Query:

Use Case Name:	**Find the average cost per SKU for each department.**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. Manager navigates to the "inventory" data view in the system. 3. Performs the below query 4. Finds the average cost per SKU for each department.

Query:	SELECT Dept, AVG(CostperSKU) AS avg_cost_per_sku
	FROM inventory
	GROUP BY Dept;

6. Insert Operation:

Use Case Name:	**Add a new item to the inventory table.**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. Manager navigates to the "inventory" data view in the system. 3. Performs the below query 4. The new item is added to the inventory

Query:	INSERT INTO inventory (SKU, pDescription, Dept, SubDept, CostperSKU, PriceperSKU, supplier_id, Manager_id)
	VALUES ('ESP001', 'Espresso Beans', 'Coffee', 'Beans', 10.50, 15.99, 'S006', 'M00001');

7. Delete Operation:

Use Case Name:	**Remove all items from the inventory table with a price per SKU less than 5.00**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. Manager navigates to the "inventory" data view in the system. 3. Performs the below query 4. All items with price per SKU less than 5.00 is deleted from table.
Query:	DELETE FROM inventory WHERE PriceperSKU < 5.00;

8. Update Operation:

Use Case Name:	**Update the cost per SKU of a specific item in the inventory table**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. Manager navigates to the "inventory" data view in the system. 3. Performs the below query 4. The specific item cost is updated.

Query:	UPDATE inventory
	SET CostperSKU = 15.99
	WHERE SKU = ' ESP001';

C. Use cases for bill table

9. Aggregate Query:

Use Case Name:	**Retrieve the total amount of all bills in the system**
Actor/User:	Manager/Staff
Steps:	1. The Manager/staff logs into the database system. 2. Manager navigates to the "bill" data view in the system. 3. Performs the below query 4. The total amount of all bills is displayed
Query:	SELECT SUM(total_amount) as TOTAL_AMOUNT FROM bill;

10. Insert Operation:

Use Case Name:	**Add a new bill to the table.**
Actor/User:	Manager/Staff
Steps:	1. The Manager/Staff logs into the database system. 2. Manager navigates to the "bill" data view in the system. 3. Performs the below query 4. The total amount of all bills is displayed

Query:	INSERT INTO bill (bill_id, customer_id, staff_ID, order_id, delivery_id, total_amount, bill_date)
	VALUES (1, 'C001', 'S001', 1, 1, 52.00, '2022-04-07');

11. Delete Operation:

Use Case Name:	**Delete a bill with a specific bill_id.**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. Manager navigates to the "bill" data view in the system. 3. Performs the below query 4. Deletes a specific bill based on the ID
Query:	DELETE FROM bill WHERE bill_id = 1;

12. Update Operation:

Use Case Name:	**Update the total_amount of a bill with a specific bill_id.**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. Manager navigates to the "bill" data view in the system. 3. Performs the below query 4. The updated bill is shown in the table
Query:	UPDATE bill SET total_amount = 30.00 WHERE bill_id = 2;

D. Use cases for menu table

13. Aggregate Query:

Use Case Name:	**Retrieve the average price of all menu items:**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. Manager navigates to the "menu" data view in the system. 3. Performs the below query 4. The average price is shown for all menu items.
Query:	SELECT AVG(menu_price) FROM menu;

14. Insert Operation:

Use Case Name:	**To add a new item to the menu table**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. Manager navigates to the "menu" data view in the system. 3. Performs the below query. 4. The new item is added to the menu.
Query:	INSERT INTO menu (menu_id,menu_name, menu_price, menu_description) VALUES (1,'Grilled Sandwich', 19.99, 'Freshly grilled sandwich with steamed filling of chicken and roasted potatoes.');

15. Delete Operation:

Use Case Name:	**To delete an item from the menu table**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. Manager navigates to the "menu" data view in the system. 3. Performs the below query 4. The specific menu item is deleted from the table.
Query:	DELETE FROM menu WHERE menu_id = 3;

16. Update Operation:

Use Case Name:	**To update the price of an existing item in the menu table**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. Manager navigates to the "menu" data view in the system. 3. Performs the below query 4. The price is updated for the item.
Query:	UPDATE menu SET menu_price = 14.99 WHERE menu_id = 2;

E. Use cases for Staff table

17. Aggregate Query:

Use Case Name:	**To find the total staff working in the cafe**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. Manager navigates to the "Staff" data view in the system. 3. Performs the below query 4. Generates the view for total number of staff members.
Query:	SELECT COUNT(*) AS staff_count FROM Staff;

18. Insert Operation:

Use Case Name:	**To add a new staff member**
Actor/User:	Manager/Owner
Steps:	1. The Manager/Owner logs into the database system. 2. The user clicks on Staff button. User is prompted to staff details. 3. User clicks on “Add” button. User is prompted to enter the details of the new staff. 4. User enters the details and clicks on “Submit” button. 5. System displays a confirmation message.
Query:	INSERT INTO Staff (staff_ID , staff_fName , staff_lName , staff_mobile , staff_email , staff_address , position , staff_sex , hire_date , Salary) VALUES ('S1001','Sarah','Jones','01523-763871', 's_jones@gmail.com', '118 Main St','accountant','F','1989-11-21',30000.00);

19. Delete Operation:

Use Case Name:	**To delete a staff member detail with a specific staff_id**
Actor/User:	Manager/Owner
Steps:	1. The Manager/ Owner logs into the database system. 2. The user clicks on Staff button. User is prompted to staff details. 3. User clicks on “Remove” button. User is prompted to enter the staff id which is to be deleted. 4. User enters the staff id and clicks “Done” 5. System shows the message “Are you sure you want to remove this data” 6. User clicks “Yes” 7. System displays a confirmation message.
Query:	DELETE FROM staff WHERE staff_ID = 'S1001';

20. Update Operation:

Use Case Name:	**To update the position of a staff member with a specific staff_id**
Actor/User:	Manager/Owner
Steps:	1. The Manager/Owner logs into the database system. 2. The user clicks on Staff button. User is prompted to staff details. 3. User clicks on “Update” button. User is prompted to enter the staff id which is to be updated. 4. User enters the staff id and clicks “Done” 5. User is prompted to a new page to update the details of the staff. User enters the new position and clicks “save” button. 6. System gets updated
Query:	UPDATE staff SET position = 'Supervisor’ WHERE staff_ID = 'S1005';

F. Use cases for Deliveries table

21. Aggregate Query:

Use Case Name:	**To find the total deliveries made in a day**
Actor/User:	Manager/Owner
Steps:	1. The Manager/owner logs into the database system. 2. Manager/owner clicks on “Report” button. 3. User is prompted to select from various options. User selects Deliveries. 4. User is prompted to enter date. User enters the date. 5. Generates the view for total number of deliveries made on a particular date
Query:	SELECT COUNT(*) FROM deliveries WHERE delivery_date = '2022-03-15';

22. Insert Operation:

Use Case Name:	**To add details of a delivery to deliveries table**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. User clicks on “deliveries” button. User is prompted to the details of deliveries. User clicks on “New delivery” button. 3. User is prompted to enter the details of the delivery. 4. User enters the details and clicks on “Save” button. 5. System displays a confirmation message
Query:	INSERT INTO deliveries (delivery_id , staff_ID , order_id , delivery_date , delivery_time , delivery_address , delivery_status) VALUES (4, 'SG5', 6, '2022-03-15', '10:30:00', '118 El Mundo St, Houston, USA', 'delivered');

23. Delete Operation:

Use Case Name:	**To delete a delivery detail with a specific delivery_id**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. User clicks on “deliveries” button. User is prompted to the details of deliveries. User clicks on “Remove” button. 3. User is prompted to enter the delivery ID. 4. User enters the details and clicks on “Save” button. 5. System gets updated
Query:	DELETE FROM deliveries WHERE delivery_id = 3;

24. Update Operation:

Use Case Name:	**To update the delivery_status of a delivery with a delivery_id**
Actor/User:	Manager
Steps:	1. The Manager/Owner logs into the database system. 2. User clicks on “deliveries” button. User is prompted to the details of deliveries. User clicks on “Update” button. 3. User is prompted to enter the delivery ID. 4. User enters the delivery ID and system displays the delivery details. User updates the delivery status and clicks “Save” button. 5. System gets updated
Query:	UPDATE deliveries SET delivery_status = 'delivered’ WHERE delivery_id = 1

G. Use cases for Orders table

25. Aggregate Query:

Use Case Name:	**To find the total orders made in a day**
Actor/User:	Manager/Owner
Steps:	1. The Manager/owner logs into the database system. 2. Manager/owner clicks on “Report” button. 3. User is prompted to select from various options. User selects Orders. 4. User is prompted to enter date. User enters the date. 5. Generates the view for total number of orders made on a particular date
Query:	SELECT COUNT(*) FROM orders WHERE order_date = '2023-04-09';

26. Insert Operation:

Use Case Name:	**To add details of an order to orders table**
Actor/User:	Manager/Staff
Steps:	1. The Manager/Staff logs into the database system. 2. User clicks on “Orders” button. User is prompted to the details of Orders. User clicks on “New Order” button. 3. User is prompted to enter the details of the Order. 4. User enters the details and clicks on “Save” button. 5. System displays a confirmation message
Query:	INSERT INTO deliveries (order_id , customer_id , staff_ID , order_date , order_time , total_amount) VALUES (5, 'C00012', 'SG5', '2023-04-12', '16:28:04', 28.50);

27. Delete Operation:

Use Case Name:	**To delete an order detail with a specific order_id**
Actor/User:	Manager/Staff
Steps:	1. The Manager/Staff logs into the database system. 2. User clicks on “Orders” button. User is prompted to the details of Orders. User clicks on “Remove” button. 3. User is prompted to enter the order ID. 4. User enters the details and clicks on “Save” button. 5. System gets updated.
Query:	DELETE FROM orders WHERE order_id = 12;

28. Update Operation:

Use Case Name:	**To update the total_amount of a customer with an given order_id**
Actor/User:	Manager
Steps:	1. The Manager/Staff logs into the database system. 2. User clicks on “Orders” button. User is prompted to the details of orders. User clicks on “Update” button. 3. User is prompted to enter the order ID. 4. User enters the order ID and system displays the order details. User updates the total amount and clicks “Save” button. 5. System gets updated
Query:	UPDATE orders SET total_amount = 32.45 WHERE order_id = 10;

H. Use cases for order_items Table

29. Aggregate Query:

Use Case Name:	**To find the total items ordered by a customer**
Actor/User:	Manager/Staff
Steps:	1. The Manager/Staff logs into the database system. 2. Manager/Staff clicks "Order" button. User is prompted to new page with orders details. User selects an order with given order ID. User then clicks on “details” button. 3. System displays the list of the items ordered and the total quantity.
Query:	SELECT COUNT(menu_id) AS items, SUM(quantity) AS totalitems FROM order_items WHERE order_id = 12;

30. Insert Operation:

Use Case Name:	**To add details of an order to order_items table**
Actor/User:	Manager/Staff
Steps:	1. The Manager/Staff logs into the database system. 2. User clicks on “Orders” button. User is prompted to the details of Orders. User clicks on “New Order” button. 3. User is prompted to enter the details of the Order. 4. User enters the details and clicks on “Save” button. 5. The details are logged in the order_items table.
Query:	INSERT INTO order_items (order_id , menu_id , quantity, special_requests) VALUES (5, 2, 1, 'With Icecream’);

31. Delete Operation:

Use Case Name:	**To delete an item from the order with a specific order_id and menu_id**
Actor/User:	Manager/Staff
Steps:	1. The Manager/staff logs into the database system. 2. User clicks on “Orders” button. User is prompted to the details of Orders. User clicks on “Remove” button. 3. User is prompted to enter the order ID. 4. User enters the details and clicks on “Save” button. 5. Order_items table gets updated automatically.
Query:	DELETE FROM order_items WHERE order_id = 2 AND menu_id = 6;

32. Update Operation:

Use Case Name:	**To update the Quantity of an item with an order_id**
Actor/User:	Manager/Staff
Steps:	1. The Manager/staff logs into the database system. 2. Then navigates to the " order_items " data view in the system. 3. Performs the below query 4. System gets updated
Query:	UPDATE order_item SET quantity = 3 WHERE order_id = 10 AND menu_id = 2;

I. Use cases for Supplier Table

33. Aggregate Query:

Use Case Name:	**To find the total number of suppliers, execute the following query**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. Manager navigates to the "Supplier" data view in the system. 3. Performs the below query 4. Generates the view for total number of suppliers.
Query:	SELECT COUNT(*) FROM Supplier;

34. Insert Operation:

Use Case Name:	**To add a new supplier, execute the following query**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. User clicks on “Suppliers” button. User is prompted to the details of Supplier. User clicks on “New Supplier” button. 3. User is prompted to enter the details of the Supplier. 4. User enters the details and clicks on “Save” button. 5. The details are logged in the Supplier table.
Query:	INSERT INTO Supplier (supplier_id, supplier_fName, supplier_lName, supplier_mobile, supplier_email, supplier_address, city, postcode) VALUES ('S00001', ‘Samantha’, 'Brown', '+1-281-572-8911', 'samanthabrown@gmail.com', 5678 Houston Boulevard, Suite 102', 'Houston', '77002');

35. Delete Operation:

Use Case Name:	**To delete a supplier with a specific supplier_id, execute the following query**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. User clicks on “Suppliers” button. User is prompted to the details of Suppliers. User clicks on “Remove” button. 3. User is prompted to enter the supplier ID to remove that supplier details. 4. User enters the supplier ID and clicks on “Save” button. 5. Supplier table gets updated automatically.
Query:	DELETE FROM Supplier WHERE supplier_id = 'S00001';

36. Update Operation:

Use Case Name:	**To update the email address of a supplier with a specific supplier_id, execute the following query**
Actor/User:	Manager
Steps:	6. The Manager logs into the database system. 7. Then navigates to the " Supplier " data view in the system. 8. Performs the below query 9. System gets updated
Query:	UPDATE Supplier SET supplier_email = 'samantha.brown@supplyfast.com' WHERE supplier_id = 'S00001';

J. Use cases for sales table:

37. Aggregate Query:

Use Case Name:	**To find the total sales for a specific menu item, execute the following query**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. Manager navigates to the "Sales" data view in the system. 3. Performs the below query 4. Generates the view for total Sales from menu id 123.
Query:	SELECT SUM(sales_total) FROM sales WHERE menu_id = 123;

38. Insert Operation:

Use Case Name:	**To add a new sales record, execute the following query**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. User clicks on “Sales” button. User is prompted to the details of Sales. User clicks on “New Sales” button. 3. The user is prompted to enter the details of the Sales. 4. User enters the details and clicks on “Save” button. 5. The details are logged in the Sales table.
Query:	INSERT INTO sales (sales_id, menu_id, sales_date, sales_quantity, sales_total) VALUES (1001, 123, '2023-04-07', 2, 25.98);

39. Delete Operation:

Use Case Name:	**To delete a sales record with a specific sales_id, execute the following query**
Actor/User:	Manager
Steps:	10. The Manager logs into the database system. 11. User clicks on “Sales” button. User is prompted to the details of Sales. User clicks on “Remove” button. 12. User is prompted to enter the Sales ID to remove that Sales details. 13. User enters the Sales ID and clicks on “Save” button. 14. Sales table gets updated automatically.
Query:	DELETE FROM sales WHERE sales_id = 1001;

40. Update Operation:

Use Case Name:	**To update the sales quantity for a specific sales record, execute the following query**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. Then navigates to the " Sales " data view in the system. 3. Performs the below query 4. System gets updated
Query:	UPDATE sales SET sales_quantity = 3 WHERE sales_id = 1001;

41. Relationship ‘tracks’ between manager and inventory

Use Case Name:	**To track the inventory details of all items managed by a specific manager**
Actor/User:	Manager/Staff
Steps:	1. The Manager/staff logs into the database system. 2. Then navigates to the " inventory " data view in the system. 3. Puts in manager id as prompted 4. System updates with the details of all items managed by the specific manager

Query:	SELECT SKU, pDescription, CostperSKU, PriceperSKU, Manager.Manager_id
	FROM inventory
	INNER JOIN Manager ON inventory.Manager_id = Manager.Manager_id
	WHERE Manager.Manager_id = 'M0001';

42. Relationship ‘supplies’ between supplier and inventory

Use Case Name:	**To retrieve a list of all items in inventory that are supplied by a supplier located in Anytown**
Actor/User:	Manager/Supplier
Steps:	1. Both Manager and Supplier can view this table of the system 2. Navigate to the " inventory " data view in the system. 3. Puts in supplier ID as prompted 4. System updates with the details of all items that are supplied by supplier in Houston

Query:	SELECT *
	FROM inventory i
	JOIN supplier s ON i.supplier_id = s.supplier_id
	WHERE s.city = 'Miami';

43. Relationship ‘prepare’ between staff and orders

Use Case Name:	**To prepare the order history of a specific customer**
Actor/User:	Staff
Steps:	1. The staff logs into the database system. 2. Then navigates to the "order " data view in the system. 3. Puts in the customer ID 4. System displays the order, bill and customer details as per below query
Query:	Select o.order_id,o.customer_id,o.order_time,s.staff_ID from orders , staff s JOIN orders o on o.staff_ID = s.staff_ID WHERE o.order_id = "1";

44. Relationship ‘places’ between customer and order

Use Case Name:	**Information about the customer who placed each order, as well as the items that were included in each order.**
Actor/User:	Customer/Staff
Steps:	1. Customer enters the system/application. 2. Then places the order. 3. System gets updated. 4. Following query returns information about customer who placed the order as well as the items.

Query:	SELECT * FROM orders
	JOIN customer ON orders.customer_id = customer.customer_id
	JOIN order_items ON orders.order_id = order_items.order_id;

45. Relationship ‘do’ between staff and deliveries on a specific date

Use Case Name:	**Retrieves information about the staff member who made deliveries**
Actor/User:	Manager/Staff
Steps:	1. The Manager/staff logs into the database system. 2. Then navigates to the " deliveries " data view in the system. 3. Staff ID is entered 4. The below query retrieves the information on the staff member who made particular delivery on particular date.

Query:	SELECT *
	FROM staff
	JOIN deliveries ON staff.staff_ID = deliveries.staff_ID
	WHERE deliveries.delivery_date = “2022-03-16”;

46. Relationship ‘pays’ between customer and bill

Use Case Name:	**This query will return all information for the customer associated with the specific bill, , as well as information about the bill itself**
Actor/User:	Manager/Staff/Customer
Steps:	1. The Manager/staff logs into the database system. 2. Then navigates to the " bills " data view in the system. 3. Enters the required bill ID 4. System generates all data related to the specific bill. 5. Customer can also view their bills on the system.

Query:	SELECT *
	FROM customer c
	JOIN bill b ON c.customer_ID = b.customer_ID
	WHERE b.bill_ID = ‘1’;

47. Relationship 'Have' between orders and order_items

Use Case Name:	**To see the items in an order placed by a customer**
Actor/User:	Staff
Steps:	1. The Staff member logs into the database system. 2. User clicks on the "orders" button. 3. User is prompted to enter the customer ID. User enters the customer ID. 4. System shows the items in an order placed by a customer.

Query:	SELECT o.customer_id, o.order_id, oi.menu_id, oi.quantity
	FROM orders o
	JOIN order_items oi ON o.order_id = oi.order_id
	WHERE o.customer_id = 'C10001';

48. Relationship 'contains' between menu and order_items

Use Case Name:	**To find the total number of a menu item ordered**
Actor/User:	Staff
Steps:	1. The Staff member logs into the database system. 2. User clicks "Reports" button. 3. Then the user is prompted to enter the criteria for which report is required. 4. User enters the criteria – menu and clicks on "submit" button. 5. The user is prompted to enter the menu item for which the report is required. User enters the menu item and clicks "Done". 6. The report showing the total number of times a particular menu item was ordered.

Query:	SELECT SUM(oi.quantity) AS total_orders
	FROM order_items oi
	JOIN menu m ON m.menu_id = oi.menu_id
	WHERE m.menu_name = "Espresso";

49. Relationship 'Records' between manager and sales

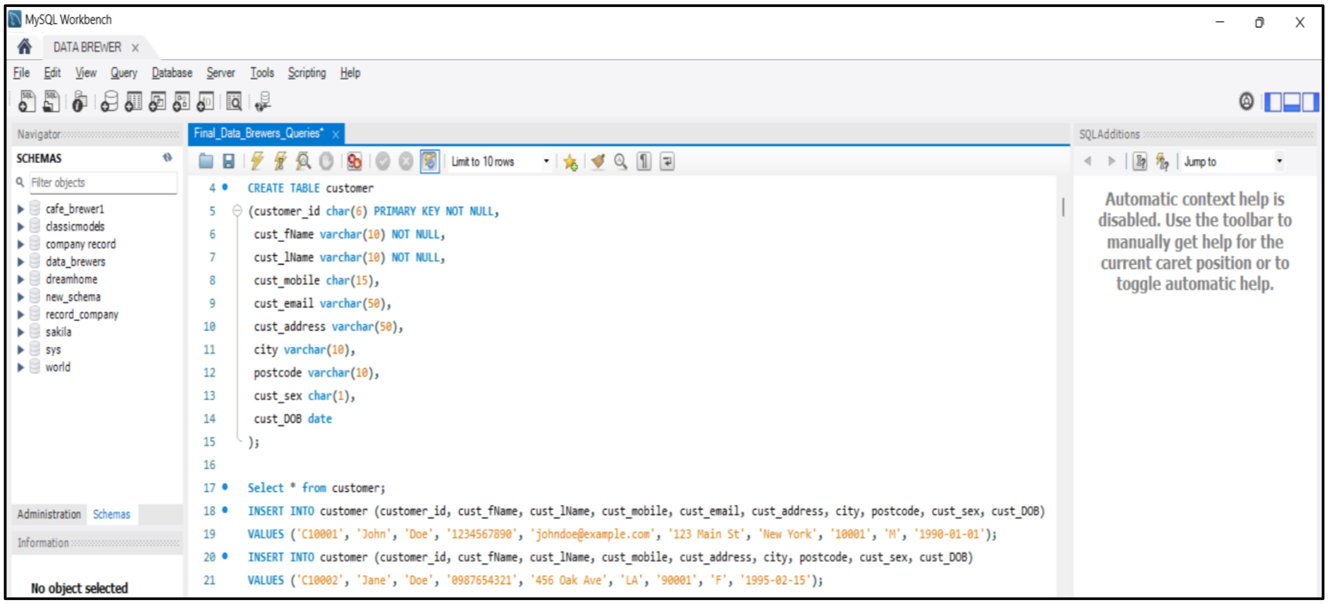
Use Case Name:	**To find the total sales and total amount of sales on a particular date**
Actor/User:	Manager
Steps:	1. The Manager logs into the database system. 2. Manager clicks on the "Report" button. 3. Then the user is prompted to enter the criteria for which report is required. 4. User enters the criteria - sales and clicks on "submit" button. 5. User is prompted to enter the date for which sales report is required. User enters the date and clicks "Done". 6. The report showing total sales and total amount will be generated.

Query:	SELECT SUM(s.sales_quantity) AS Total_quantity, SUM(s.sales_total) AS Total_amount
	From sales s
	JOIN manager m ON m.Manager_id = s.Manager_id;

8. Test Plan and Records

I. All data of each table

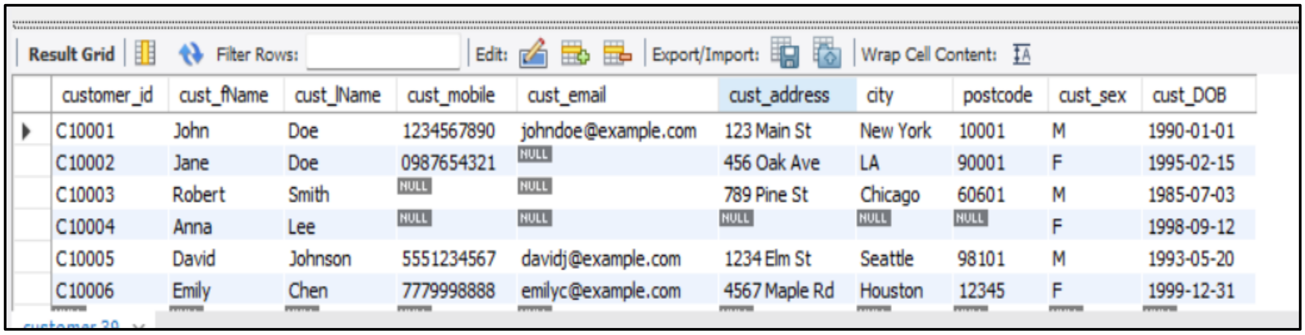
A. TABLE CUSTOMER:



```
CREATE TABLE customer
(customer_id char(6) PRIMARY KEY NOT NULL,
 cust_fName varchar(10) NOT NULL,
 cust_lName varchar(10) NOT NULL,
 cust_mobile char(15),
 cust_email varchar(50),
 cust_address varchar(50),
 city varchar(10),
 postcode varchar(10),
 cust_sex char(1),
 cust_DOB date
);

Select * from customer;
INSERT INTO customer (customer_id, cust_fName, cust_lName, cust_mobile, cust_email, cust_address, city, postcode, cust_sex, cust_DOB)
VALUES ('C10001', 'John', 'Doe', '1234567890', 'johndoe@example.com', '123 Main St', 'New York', '10001', 'M', '1990-01-01');
INSERT INTO customer (customer_id, cust_fName, cust_lName, cust_mobile, cust_address, city, postcode, cust_sex, cust_DOB)
VALUES ('C10002', 'Jane', 'Doe', '0987654321', '456 Oak Ave', 'LA', '90001', 'F', '1995-02-15');
```

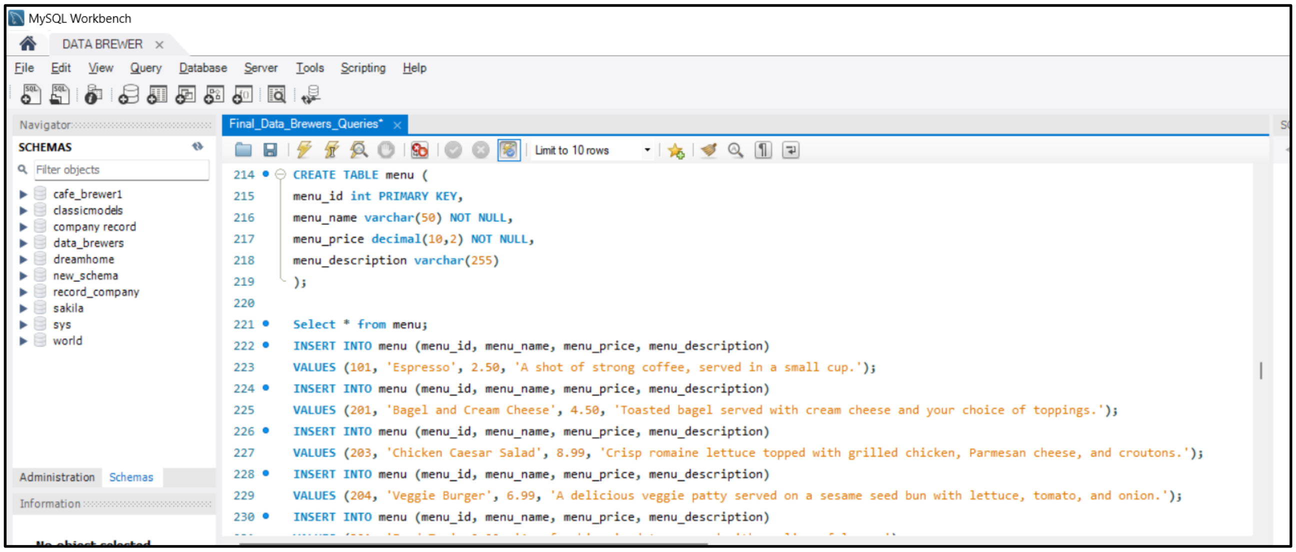
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

<u>DATA in table:</u>



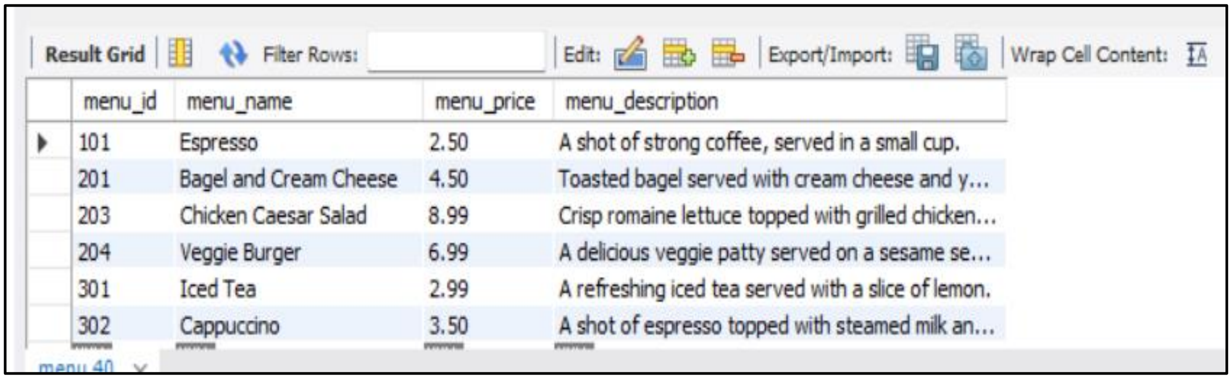
customer_id	cust_fName	cust_lName	cust_mobile	cust_email	cust_address	city	postcode	cust_sex	cust_DOB
C10001	John	Doe	1234567890	johndoe@example.com	123 Main St	New York	10001	M	1990-01-01
C10002	Jane	Doe	0987654321	NULL	456 Oak Ave	LA	90001	F	1995-02-15
C10003	Robert	Smith	NULL	NULL	789 Pine St	Chicago	60601	M	1985-07-03
C10004	Anna	Lee	NULL	NULL	NULL	NULL	NULL	F	1998-09-12
C10005	David	Johnson	5551234567	davidj@example.com	1234 Elm St	Seattle	98101	M	1993-05-20
C10006	Emily	Chen	7779998888	emilyc@example.com	4567 Maple Rd	Houston	12345	F	1999-12-31

B. TABLE MENU:



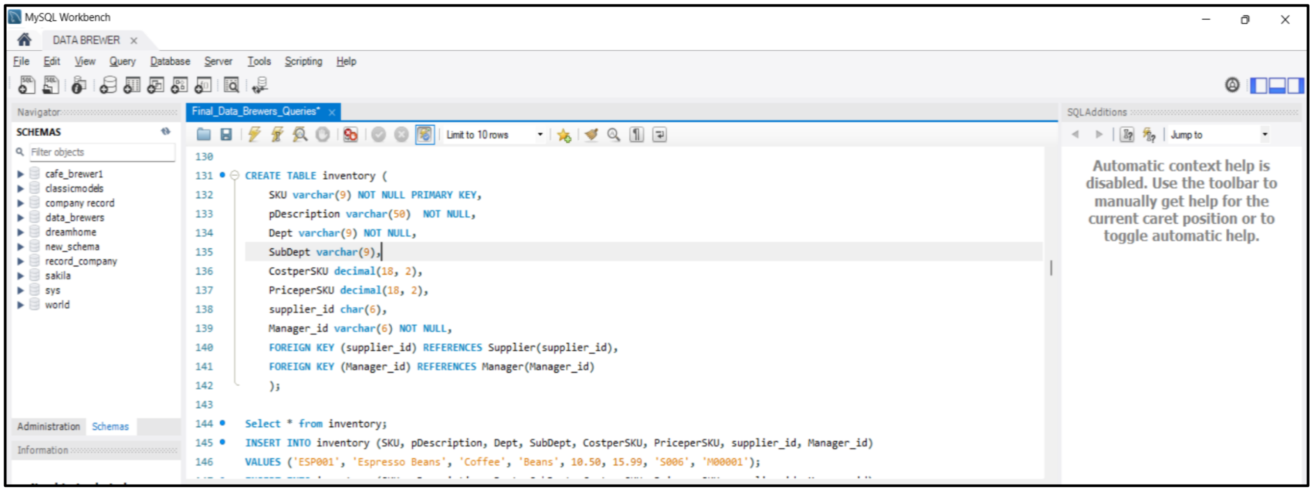
```sql
CREATE TABLE menu (
menu_id int PRIMARY KEY,
menu_name varchar(50) NOT NULL,
menu_price decimal(10,2) NOT NULL,
menu_description varchar(255)
);

Select * from menu;
INSERT INTO menu (menu_id, menu_name, menu_price, menu_description)
VALUES (101, 'Espresso', 2.50, 'A shot of strong coffee, served in a small cup.');
INSERT INTO menu (menu_id, menu_name, menu_price, menu_description)
VALUES (201, 'Bagel and Cream Cheese', 4.50, 'Toasted bagel served with cream cheese and your choice of toppings.');
INSERT INTO menu (menu_id, menu_name, menu_price, menu_description)
VALUES (203, 'Chicken Caesar Salad', 8.99, 'Crisp romaine lettuce topped with grilled chicken, Parmesan cheese, and croutons.');
INSERT INTO menu (menu_id, menu_name, menu_price, menu_description)
VALUES (204, 'Veggie Burger', 6.99, 'A delicious veggie patty served on a sesame seed bun with lettuce, tomato, and onion.');
INSERT INTO menu (menu_id, menu_name, menu_price, menu_description)
```

DATA in table:



menu_id	menu_name	menu_price	menu_description
101	Espresso	2.50	A shot of strong coffee, served in a small cup.
201	Bagel and Cream Cheese	4.50	Toasted bagel served with cream cheese and y...
203	Chicken Caesar Salad	8.99	Crisp romaine lettuce topped with grilled chicken...
204	Veggie Burger	6.99	A delicious veggie patty served on a sesame se...
301	Iced Tea	2.99	A refreshing iced tea served with a slice of lemon.
302	Cappuccino	3.50	A shot of espresso topped with steamed milk an...

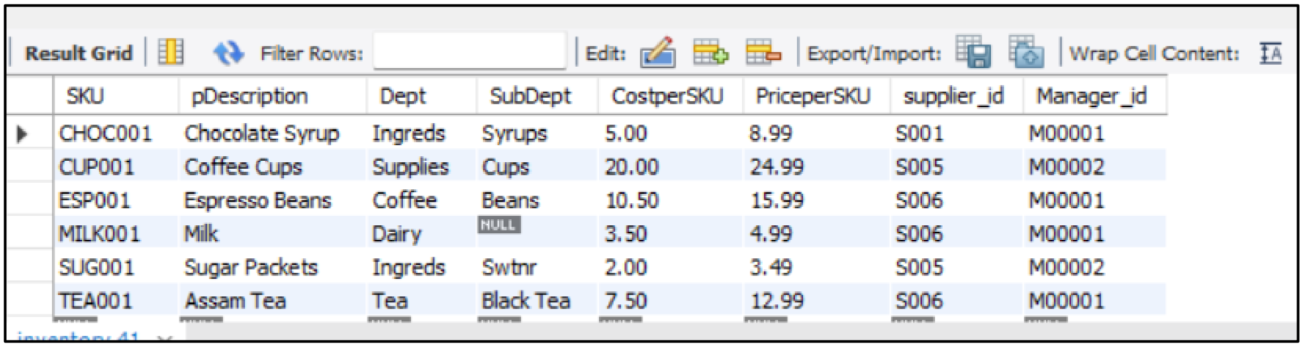
C. TABLE INVENTORY:



```
CREATE TABLE inventory (
    SKU varchar(9) NOT NULL PRIMARY KEY,
    pDescription varchar(50)  NOT NULL,
    Dept varchar(9) NOT NULL,
    SubDept varchar(9),
    CostperSKU decimal(18, 2),
    PriceperSKU decimal(18, 2),
    supplier_id char(6),
    Manager_id varchar(6) NOT NULL,
    FOREIGN KEY (supplier_id) REFERENCES Supplier(supplier_id),
    FOREIGN KEY (Manager_id) REFERENCES Manager(Manager_id)
    );

Select * from inventory;
INSERT INTO inventory (SKU, pDescription, Dept, SubDept, CostperSKU, PriceperSKU, supplier_id, Manager_id)
VALUES ('ESP001', 'Espresso Beans', 'Coffee', 'Beans', 10.50, 15.99, 'S006', 'M00001');
```

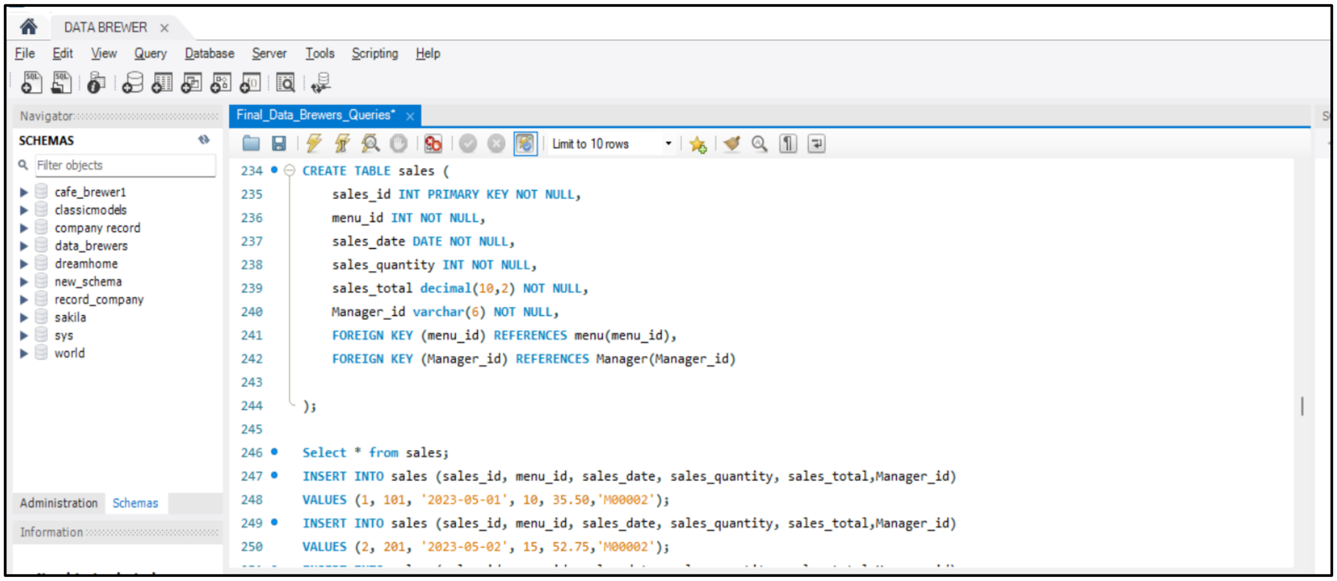
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

DATA in table:



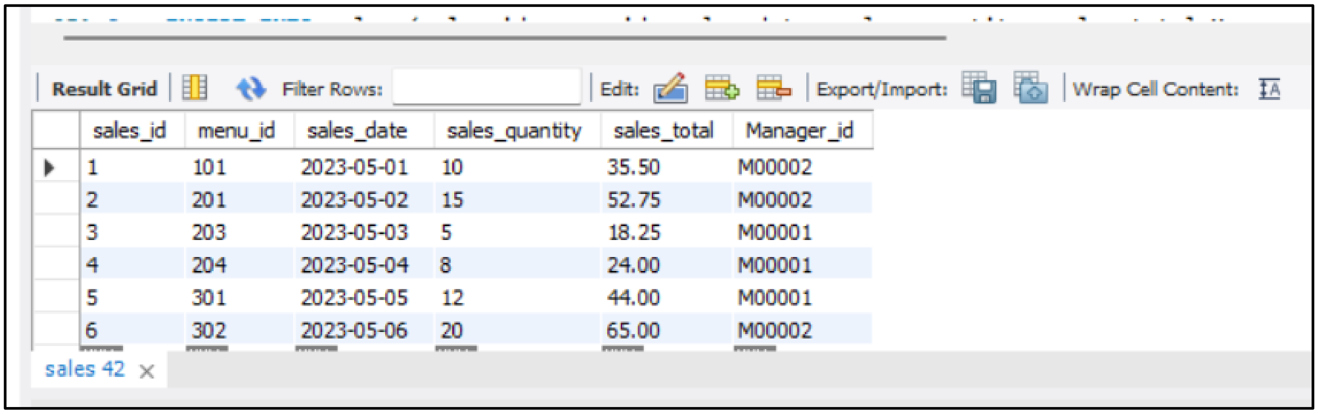
SKU	pDescription	Dept	SubDept	CostperSKU	PriceperSKU	supplier_id	Manager_id
CHOC001	Chocolate Syrup	Ingreds	Syrups	5.00	8.99	S001	M00001
CUP001	Coffee Cups	Supplies	Cups	20.00	24.99	S005	M00002
ESP001	Espresso Beans	Coffee	Beans	10.50	15.99	S006	M00001
MILK001	Milk	Dairy	NULL	3.50	4.99	S006	M00001
SUG001	Sugar Packets	Ingreds	Swtnr	2.00	3.49	S005	M00002
TEA001	Assam Tea	Tea	Black Tea	7.50	12.99	S006	M00001

D. TABLE SALES:



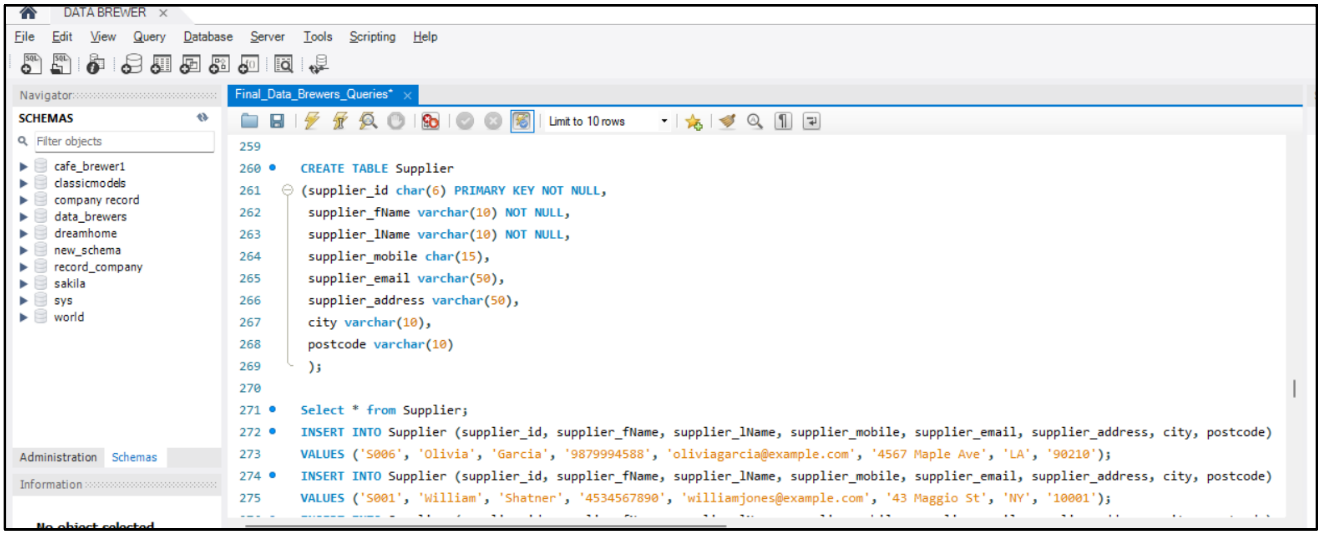
```
CREATE TABLE sales (
    sales_id INT PRIMARY KEY NOT NULL,
    menu_id INT NOT NULL,
    sales_date DATE NOT NULL,
    sales_quantity INT NOT NULL,
    sales_total decimal(10,2) NOT NULL,
    Manager_id varchar(6) NOT NULL,
    FOREIGN KEY (menu_id) REFERENCES menu(menu_id),
    FOREIGN KEY (Manager_id) REFERENCES Manager(Manager_id)

);

Select * from sales;
INSERT INTO sales (sales_id, menu_id, sales_date, sales_quantity, sales_total,Manager_id)
VALUES (1, 101, '2023-05-01', 10, 35.50,'M00002');
INSERT INTO sales (sales_id, menu_id, sales_date, sales_quantity, sales_total,Manager_id)
VALUES (2, 201, '2023-05-02', 15, 52.75,'M00002');
```

DATA in table:



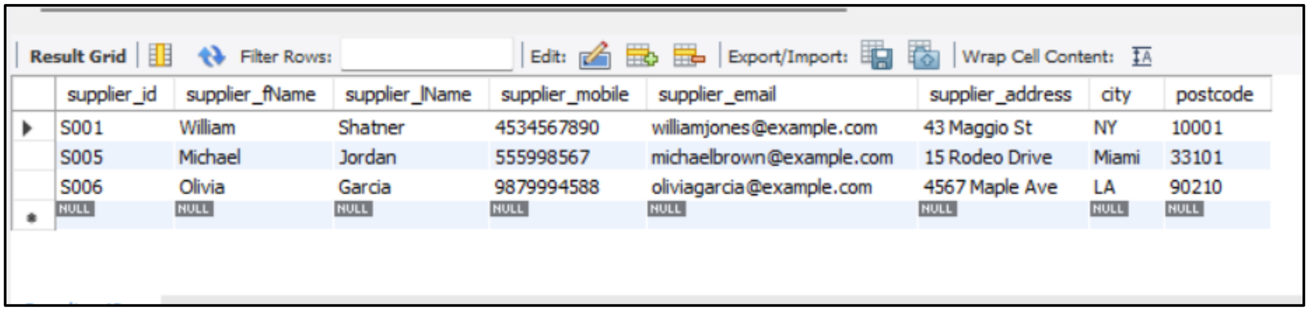
sales_id	menu_id	sales_date	sales_quantity	sales_total	Manager_id
1	101	2023-05-01	10	35.50	M00002
2	201	2023-05-02	15	52.75	M00002
3	203	2023-05-03	5	18.25	M00001
4	204	2023-05-04	8	24.00	M00001
5	301	2023-05-05	12	44.00	M00001
6	302	2023-05-06	20	65.00	M00002

E. TABLE SUPPLIER:



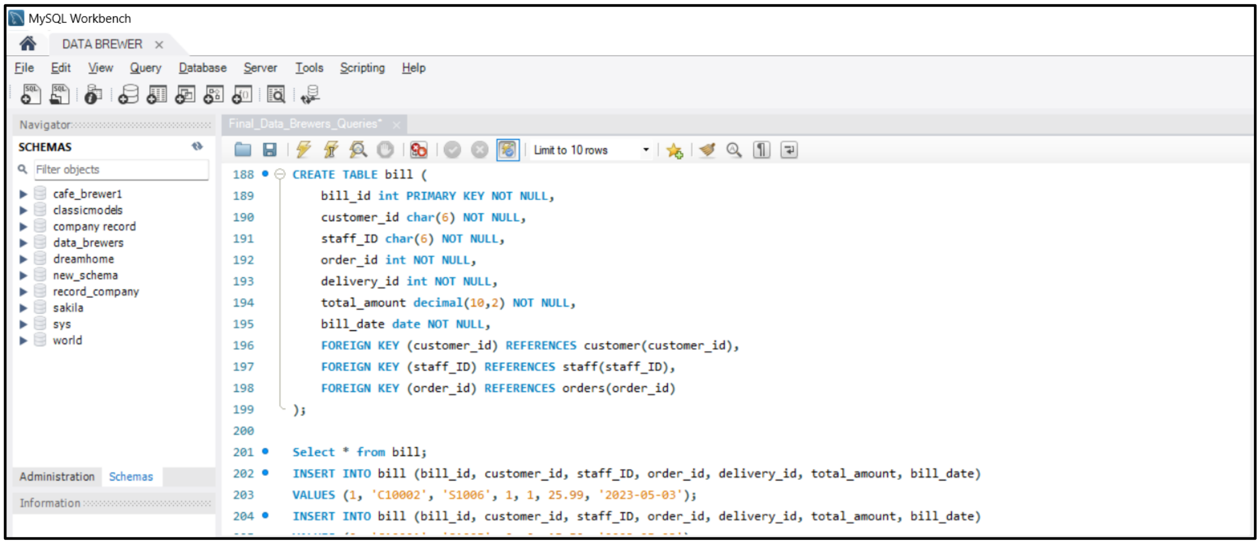
```
CREATE TABLE Supplier
(supplier_id char(6) PRIMARY KEY NOT NULL,
 supplier_fName varchar(10) NOT NULL,
 supplier_lName varchar(10) NOT NULL,
 supplier_mobile char(15),
 supplier_email varchar(50),
 supplier_address varchar(50),
 city varchar(10),
 postcode varchar(10)
 );

Select * from Supplier;
INSERT INTO Supplier (supplier_id, supplier_fName, supplier_lName, supplier_mobile, supplier_email, supplier_address, city, postcode)
VALUES ('S006', 'Olivia', 'Garcia', '9879994588', 'oliviagarcia@example.com', '4567 Maple Ave', 'LA', '90210');
INSERT INTO Supplier (supplier_id, supplier_fName, supplier_lName, supplier_mobile, supplier_email, supplier_address, city, postcode)
VALUES ('S001', 'William', 'Shatner', '4534567890', 'williamjones@example.com', '43 Maggio St', 'NY', '10001');
```

DATA in table:



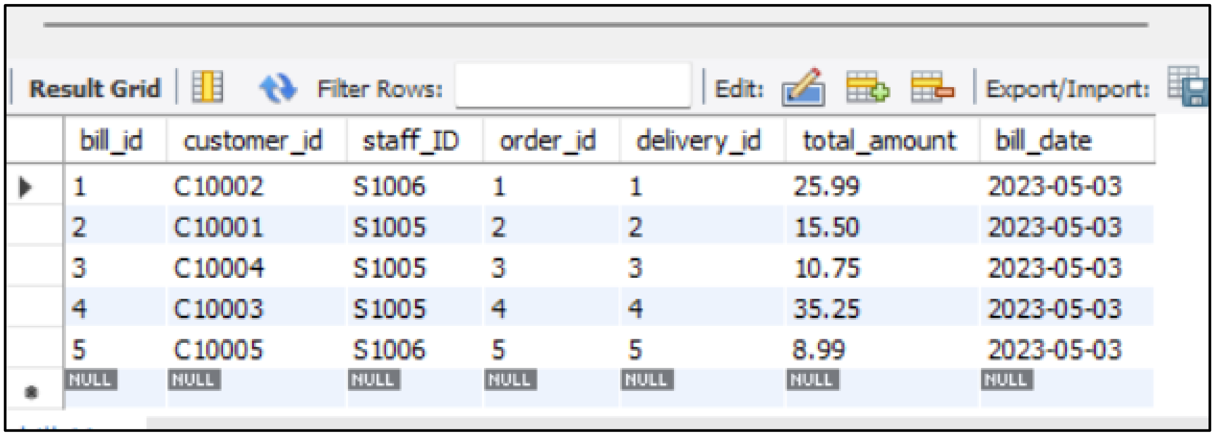
supplier_id	supplier_fName	supplier_lName	supplier_mobile	supplier_email	supplier_address	city	postcode
S001	William	Shatner	4534567890	williamjones@example.com	43 Maggio St	NY	10001
S005	Michael	Jordan	555998567	michaelbrown@example.com	15 Rodeo Drive	Miami	33101
S006	Olivia	Garcia	9879994588	oliviagarcia@example.com	4567 Maple Ave	LA	90210
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

F. TABLE BILL:



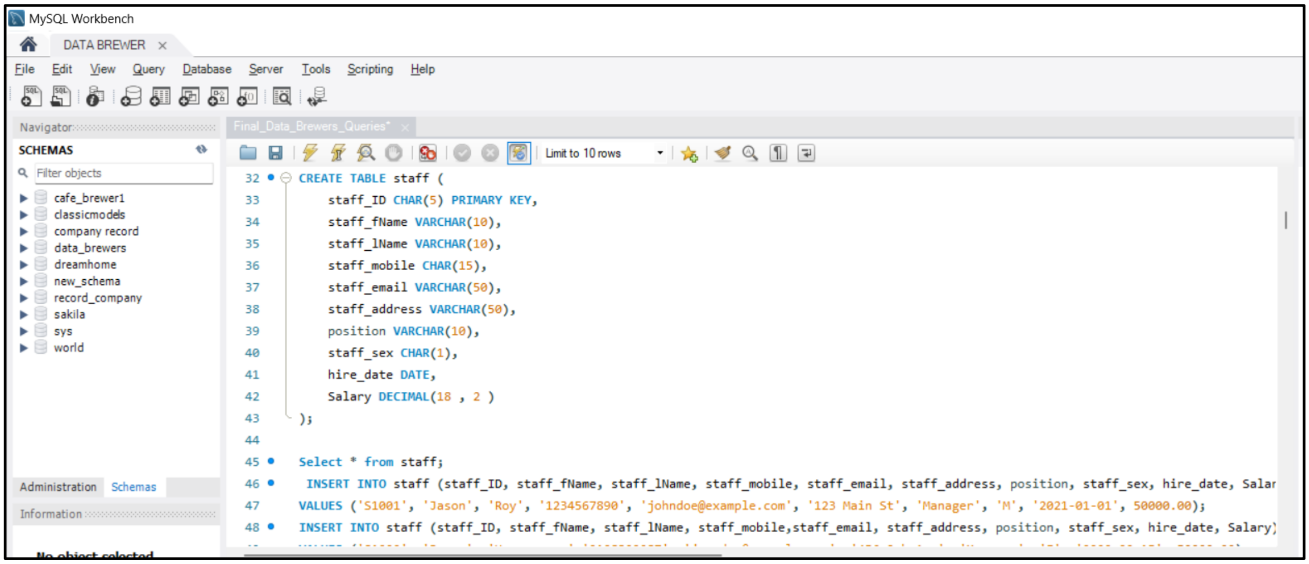
```sql
CREATE TABLE bill (
    bill_id int PRIMARY KEY NOT NULL,
    customer_id char(6) NOT NULL,
    staff_ID char(6) NOT NULL,
    order_id int NOT NULL,
    delivery_id int NOT NULL,
    total_amount decimal(10,2) NOT NULL,
    bill_date date NOT NULL,
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id),
    FOREIGN KEY (staff_ID) REFERENCES staff(staff_ID),
    FOREIGN KEY (order_id) REFERENCES orders(order_id)
);

Select * from bill;
INSERT INTO bill (bill_id, customer_id, staff_ID, order_id, delivery_id, total_amount, bill_date)
VALUES (1, 'C10002', 'S1006', 1, 1, 25.99, '2023-05-03');
INSERT INTO bill (bill_id, customer_id, staff_ID, order_id, delivery_id, total_amount, bill_date)
```

DATA in table:



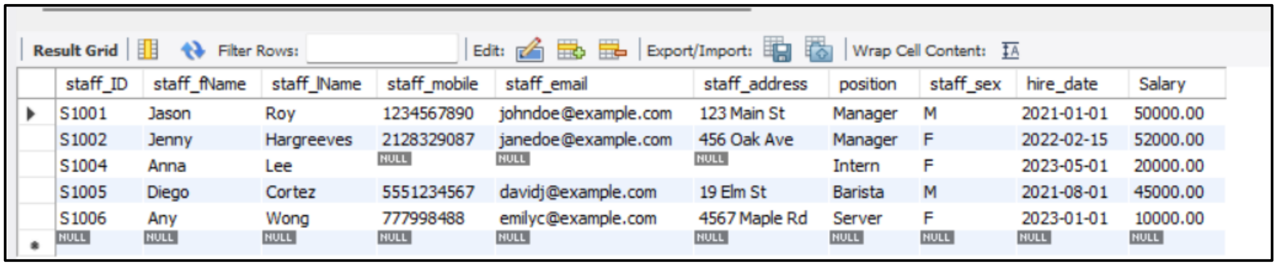
bill_id	customer_id	staff_ID	order_id	delivery_id	total_amount	bill_date
1	C10002	S1006	1	1	25.99	2023-05-03
2	C10001	S1005	2	2	15.50	2023-05-03
3	C10004	S1005	3	3	10.75	2023-05-03
4	C10003	S1005	4	4	35.25	2023-05-03
5	C10005	S1006	5	5	8.99	2023-05-03
NULL	NULL	NULL	NULL	NULL	NULL	NULL

G. TABLE STAFF:



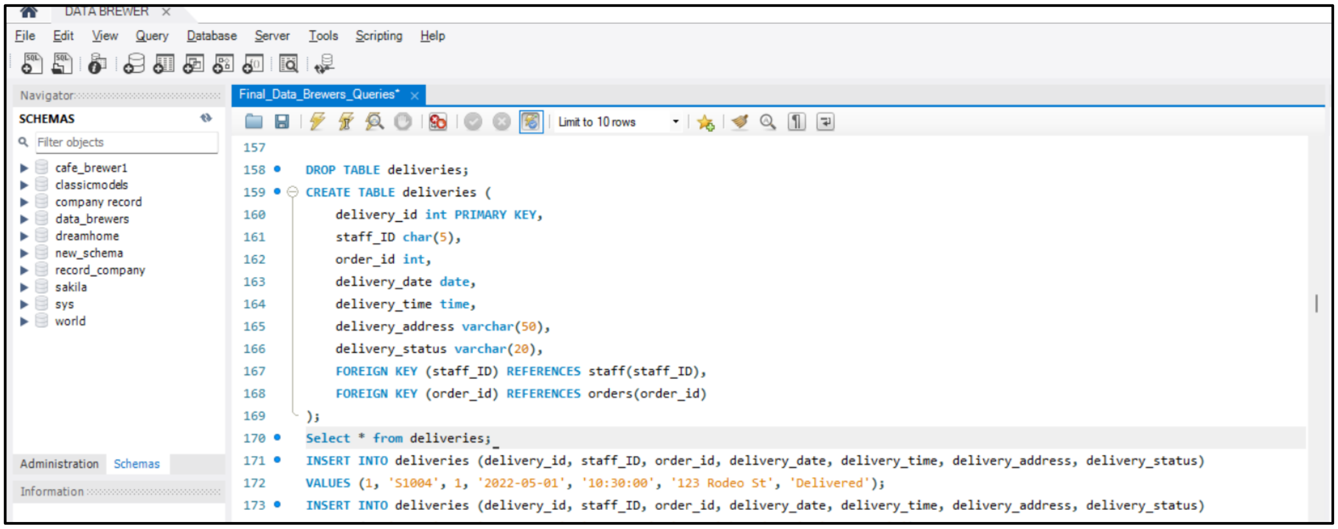
```sql
CREATE TABLE staff (
    staff_ID CHAR(5) PRIMARY KEY,
    staff_fName VARCHAR(10),
    staff_lName VARCHAR(10),
    staff_mobile CHAR(15),
    staff_email VARCHAR(50),
    staff_address VARCHAR(50),
    position VARCHAR(10),
    staff_sex CHAR(1),
    hire_date DATE,
    Salary DECIMAL(18 , 2 )
);

Select * from staff;
 INSERT INTO staff (staff_ID, staff_fName, staff_lName, staff_mobile, staff_email, staff_address, position, staff_sex, hire_date, Salar
VALUES ('S1001', 'Jason', 'Roy', '1234567890', 'johndoe@example.com', '123 Main St', 'Manager', 'M', '2021-01-01', 50000.00);
INSERT INTO staff (staff_ID, staff_fName, staff_lName, staff_mobile,staff_email, staff_address, position, staff_sex, hire_date, Salary)
```

DATA in table:



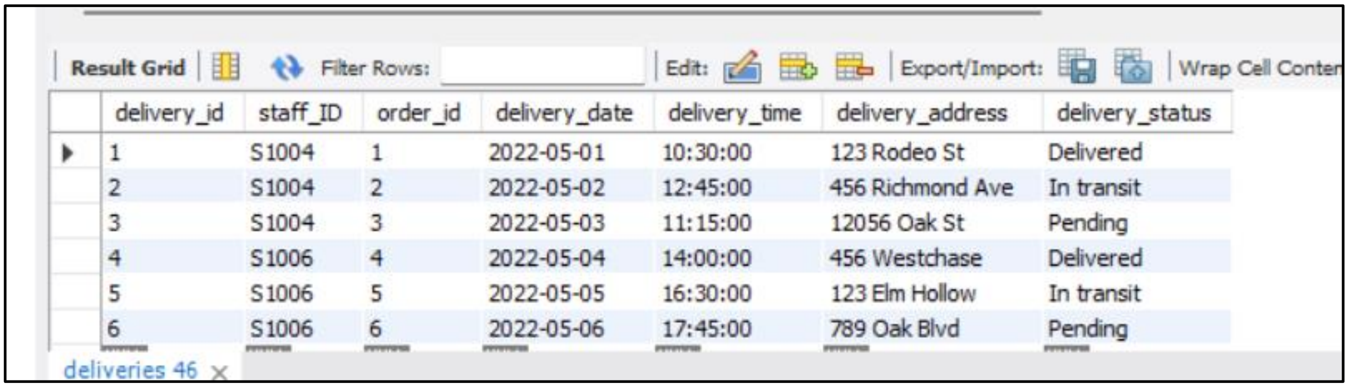
staff_ID	staff_fName	staff_lName	staff_mobile	staff_email	staff_address	position	staff_sex	hire_date	Salary
S1001	Jason	Roy	1234567890	johndoe@example.com	123 Main St	Manager	M	2021-01-01	50000.00
S1002	Jenny	Hargreeves	2128329087	janedoe@example.com	456 Oak Ave	Manager	F	2022-02-15	52000.00
S1004	Anna	Lee	NULL	NULL	NULL	Intern	F	2023-05-01	20000.00
S1005	Diego	Cortez	5551234567	davidj@example.com	19 Elm St	Barista	M	2021-08-01	45000.00
S1006	Any	Wong	777998488	emilyc@example.com	4567 Maple Rd	Server	F	2023-01-01	10000.00
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

H. TABLE DELIVERIES:



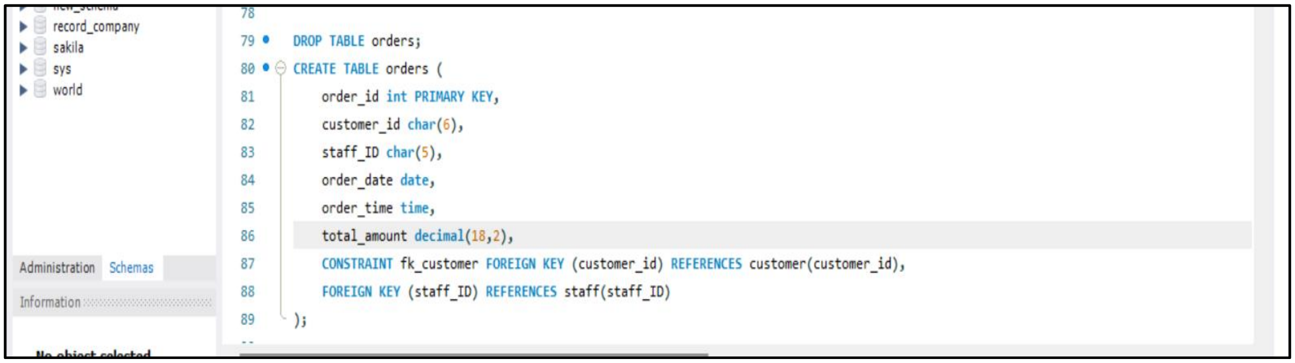
```
DROP TABLE deliveries;
CREATE TABLE deliveries (
    delivery_id int PRIMARY KEY,
    staff_ID char(5),
    order_id int,
    delivery_date date,
    delivery_time time,
    delivery_address varchar(50),
    delivery_status varchar(20),
    FOREIGN KEY (staff_ID) REFERENCES staff(staff_ID),
    FOREIGN KEY (order_id) REFERENCES orders(order_id)
);
Select * from deliveries;
INSERT INTO deliveries (delivery_id, staff_ID, order_id, delivery_date, delivery_time, delivery_address, delivery_status)
VALUES (1, 'S1004', 1, '2022-05-01', '10:30:00', '123 Rodeo St', 'Delivered');
INSERT INTO deliveries (delivery_id, staff_ID, order_id, delivery_date, delivery_time, delivery_address, delivery_status)
```

DATA in table:



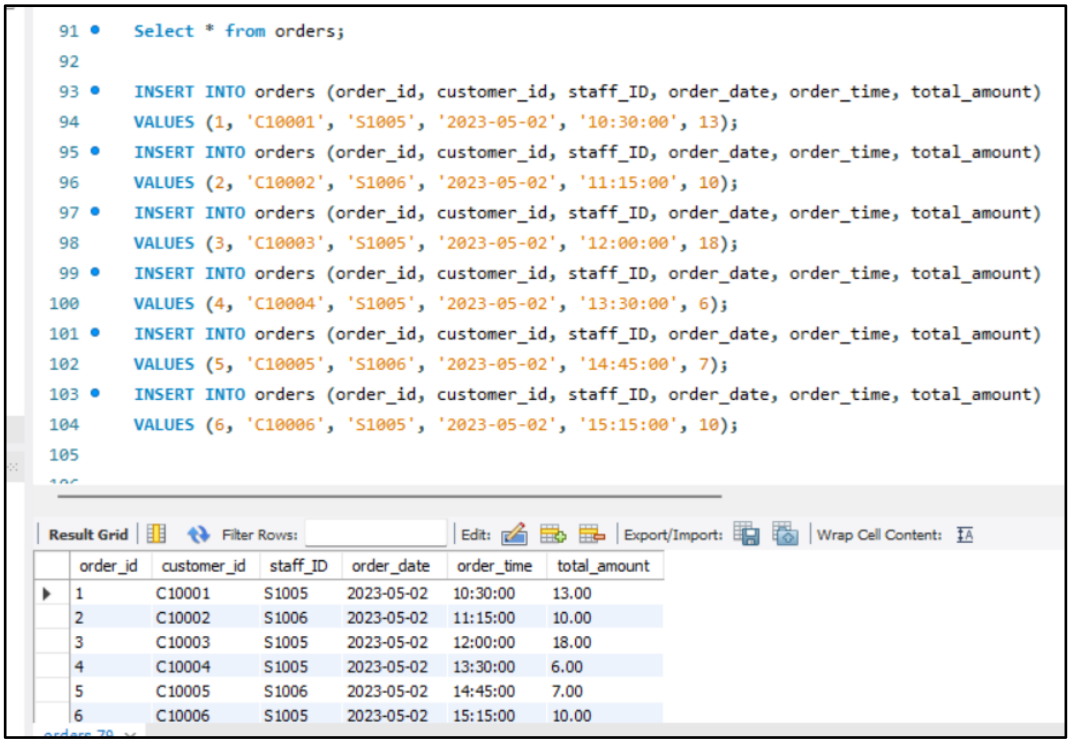
delivery_id	staff_ID	order_id	delivery_date	delivery_time	delivery_address	delivery_status
1	S1004	1	2022-05-01	10:30:00	123 Rodeo St	Delivered
2	S1004	2	2022-05-02	12:45:00	456 Richmond Ave	In transit
3	S1004	3	2022-05-03	11:15:00	12056 Oak St	Pending
4	S1006	4	2022-05-04	14:00:00	456 Westchase	Delivered
5	S1006	5	2022-05-05	16:30:00	123 Elm Hollow	In transit
6	S1006	6	2022-05-06	17:45:00	789 Oak Blvd	Pending

I. TABLE ORDER:



```
DROP TABLE orders;
CREATE TABLE orders (
    order_id int PRIMARY KEY,
    customer_id char(6),
    staff_ID char(5),
    order_date date,
    order_time time,
    total_amount decimal(18,2),
    CONSTRAINT fk_customer FOREIGN KEY (customer_id) REFERENCES customer(customer_id),
    FOREIGN KEY (staff_ID) REFERENCES staff(staff_ID)
);
```

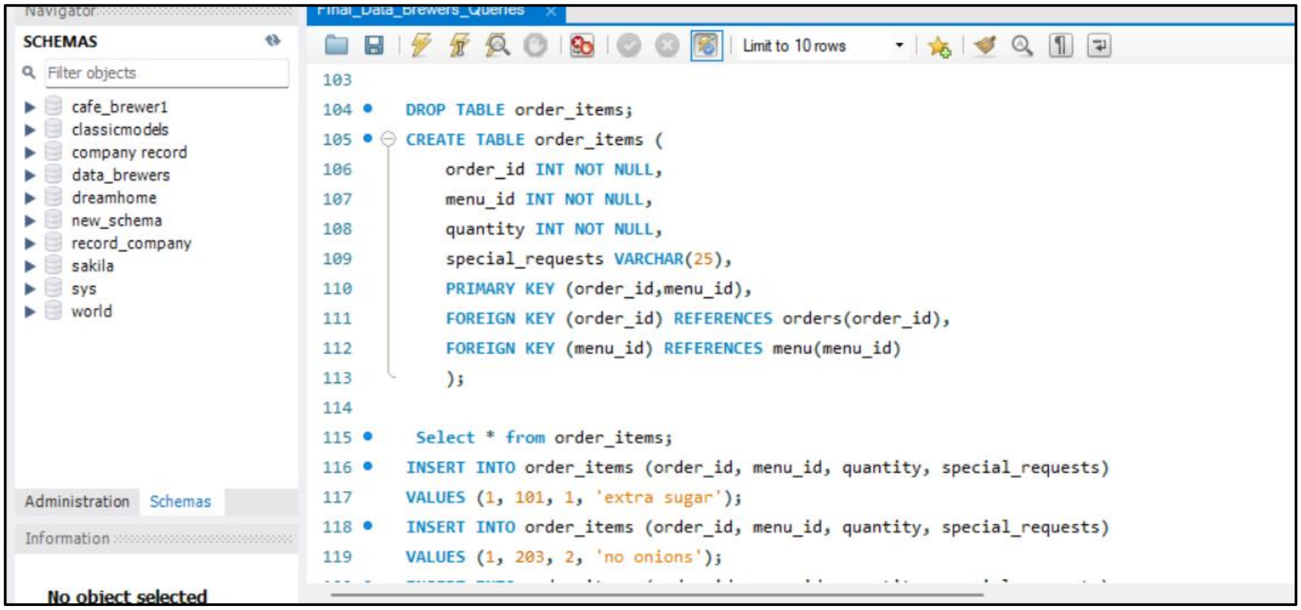
DATA in table:



```
Select * from orders;

INSERT INTO orders (order_id, customer_id, staff_ID, order_date, order_time, total_amount)
VALUES (1, 'C10001', 'S1005', '2023-05-02', '10:30:00', 13);
INSERT INTO orders (order_id, customer_id, staff_ID, order_date, order_time, total_amount)
VALUES (2, 'C10002', 'S1006', '2023-05-02', '11:15:00', 10);
INSERT INTO orders (order_id, customer_id, staff_ID, order_date, order_time, total_amount)
VALUES (3, 'C10003', 'S1005', '2023-05-02', '12:00:00', 18);
INSERT INTO orders (order_id, customer_id, staff_ID, order_date, order_time, total_amount)
VALUES (4, 'C10004', 'S1005', '2023-05-02', '13:30:00', 6);
INSERT INTO orders (order_id, customer_id, staff_ID, order_date, order_time, total_amount)
VALUES (5, 'C10005', 'S1006', '2023-05-02', '14:45:00', 7);
INSERT INTO orders (order_id, customer_id, staff_ID, order_date, order_time, total_amount)
VALUES (6, 'C10006', 'S1005', '2023-05-02', '15:15:00', 10);
```

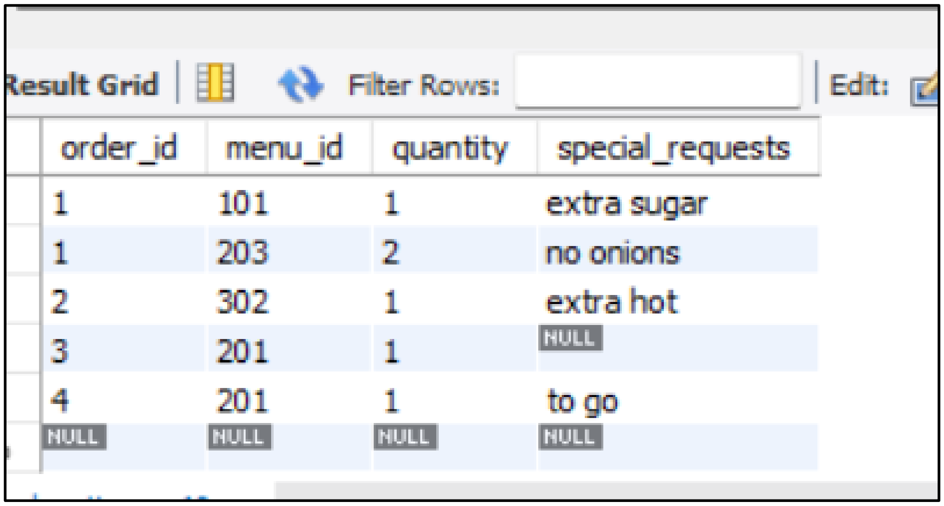
order_id	customer_id	staff_ID	order_date	order_time	total_amount
1	C10001	S1005	2023-05-02	10:30:00	13.00
2	C10002	S1006	2023-05-02	11:15:00	10.00
3	C10003	S1005	2023-05-02	12:00:00	18.00
4	C10004	S1005	2023-05-02	13:30:00	6.00
5	C10005	S1006	2023-05-02	14:45:00	7.00
6	C10006	S1005	2023-05-02	15:15:00	10.00

J. TABLE ORDER ITEMS:



```
DROP TABLE order_items;
CREATE TABLE order_items (
    order_id INT NOT NULL,
    menu_id INT NOT NULL,
    quantity INT NOT NULL,
    special_requests VARCHAR(25),
    PRIMARY KEY (order_id,menu_id),
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    FOREIGN KEY (menu_id) REFERENCES menu(menu_id)
    );

 Select * from order_items;
INSERT INTO order_items (order_id, menu_id, quantity, special_requests)
VALUES (1, 101, 1, 'extra sugar');
INSERT INTO order_items (order_id, menu_id, quantity, special_requests)
VALUES (1, 203, 2, 'no onions');
```

DATA in table:

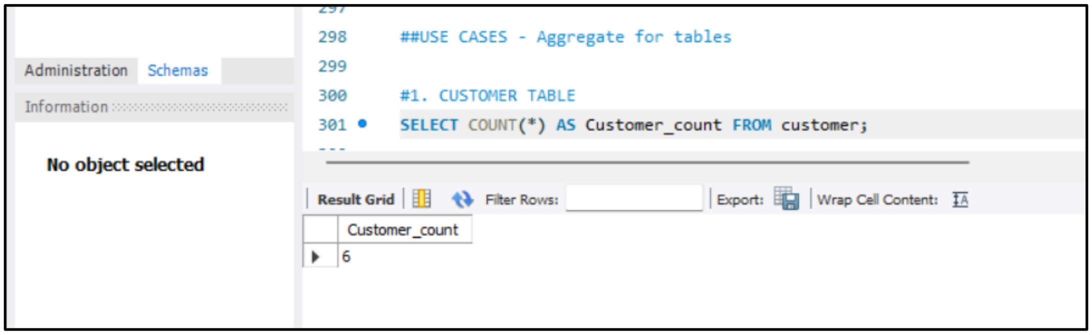


order_id	menu_id	quantity	special_requests
1	101	1	extra sugar
1	203	2	no onions
2	302	1	extra hot
3	201	1	NULL
4	201	1	to go
NULL	NULL	NULL	NULL

II. Aggregate query for each table

A. TABLE CUSTOMER:

SELECT COUNT(*) AS Customer_count FROM customer;

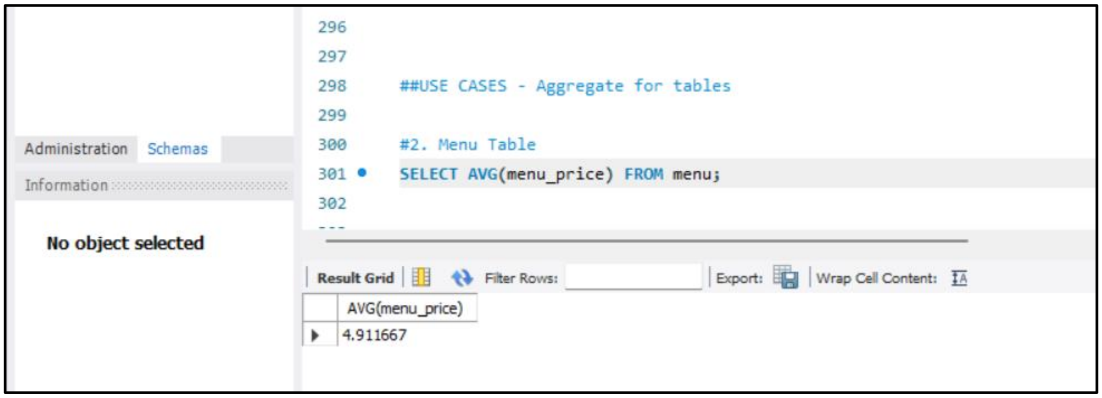


```
##USE CASES - Aggregate for tables

#1. CUSTOMER TABLE
SELECT COUNT(*) AS Customer_count FROM customer;
```

Customer_count
6

B. TABLE MENU:

SELECT AVG(menu_price) FROM menu;

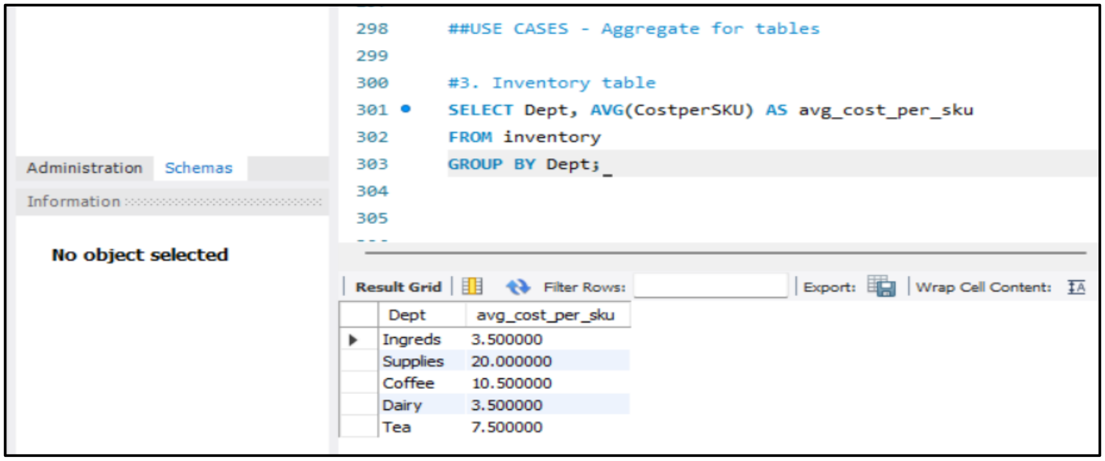


```


##USE CASES - Aggregate for tables

#2. Menu Table
SELECT AVG(menu_price) FROM menu;

```

AVG(menu_price)
4.911667

C. TABLE INVENTORY:

SELECT Dept, AVG(CostperSKU) AS avg_cost_per_sku
FROM inventory
GROUP BY Dept;

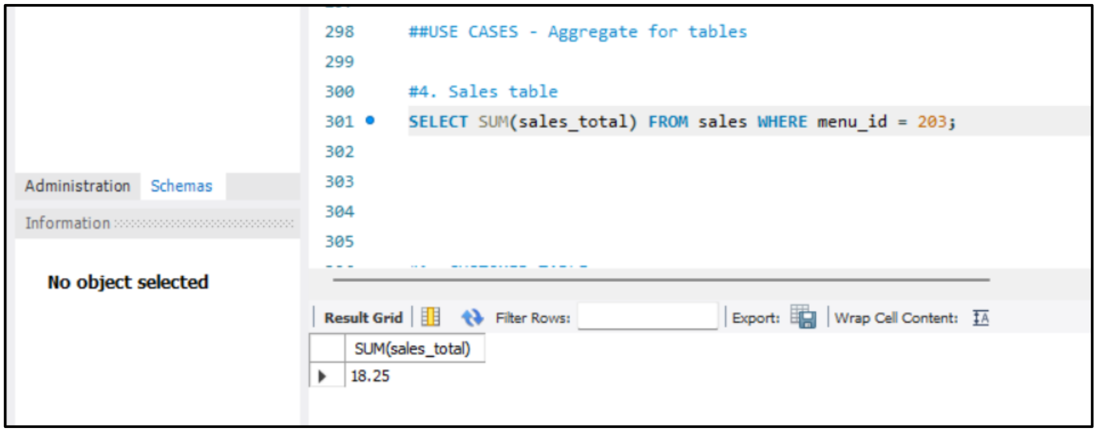


```
##USE CASES - Aggregate for tables

#3. Inventory table
SELECT Dept, AVG(CostperSKU) AS avg_cost_per_sku
FROM inventory
GROUP BY Dept;
```

Dept	avg_cost_per_sku
Ingreds	3.500000
Supplies	20.000000
Coffee	10.500000
Dairy	3.500000
Tea	7.500000

D. TABLE SALES:

SELECT SUM(sales_total) FROM sales WHERE menu_id = 203;

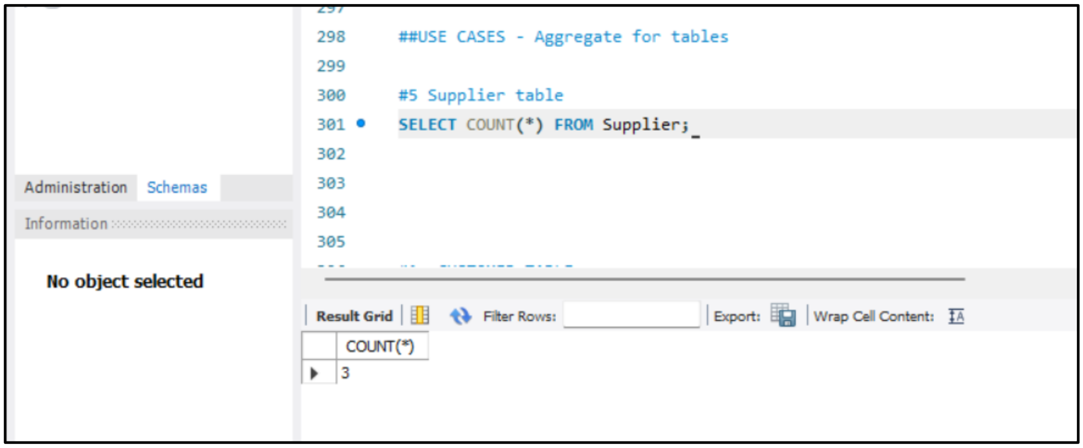


```
##USE CASES - Aggregate for tables

#4. Sales table
SELECT SUM(sales_total) FROM sales WHERE menu_id = 203;
```

```
SUM(sales_total)
18.25
```

E. TABLE SUPPLIER:

SELECT COUNT(*) FROM Supplier;

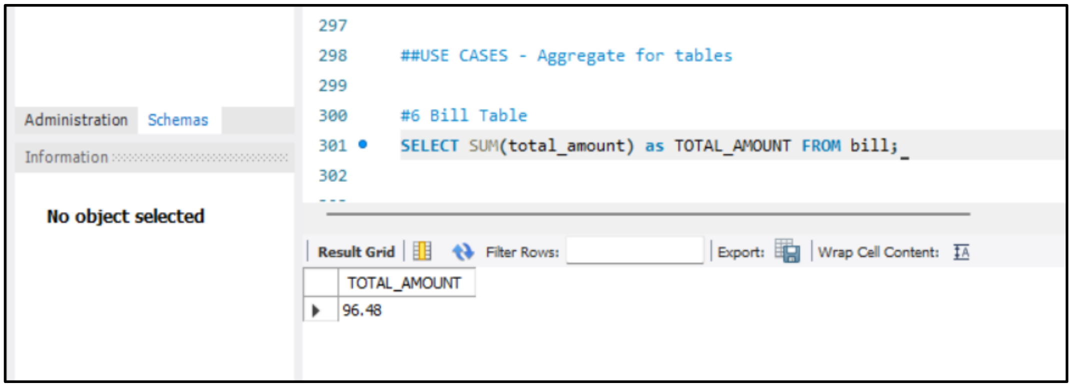


```
##USE CASES - Aggregate for tables

#5 Supplier table
SELECT COUNT(*) FROM Supplier;
```

```
COUNT(*)
3
```

F. TABLE BILL:

SELECT SUM(total_amount) as TOTAL_AMOUNT FROM bill;

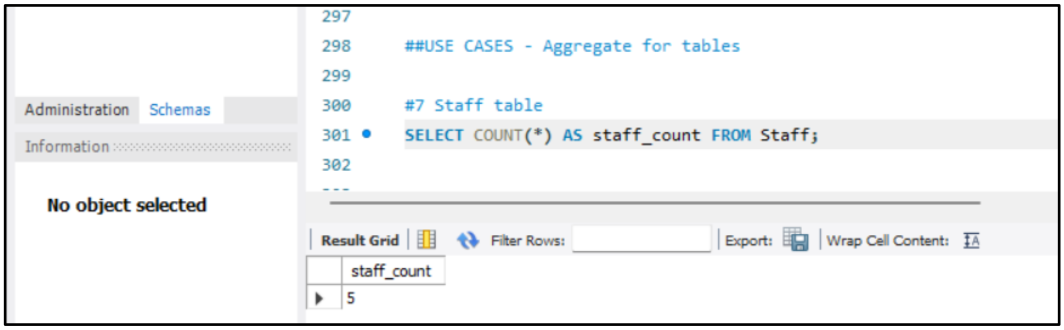


```
##USE CASES - Aggregate for tables

#6 Bill Table
SELECT SUM(total_amount) as TOTAL_AMOUNT FROM bill;
```

TOTAL_AMOUNT
96.48

G. TABLE STAFF:

SELECT COUNT(*) AS staff_count FROM Staff;

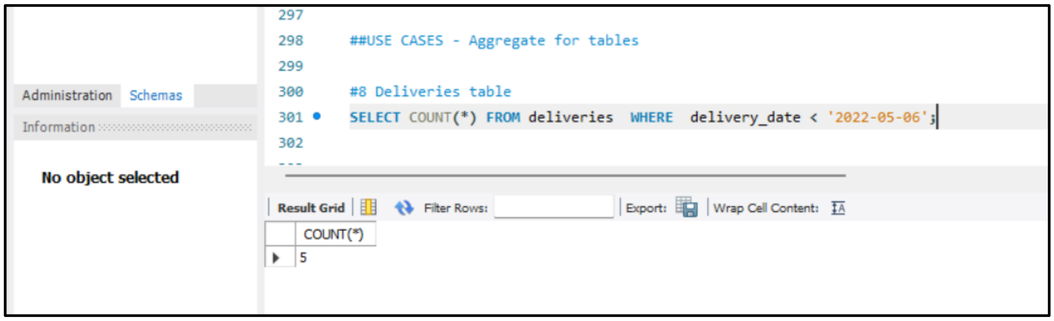


```
##USE CASES - Aggregate for tables

#7 Staff table
SELECT COUNT(*) AS staff_count FROM Staff;
```

staff_count
5

H. TABLE DELIVERIES:

SELECT COUNT(*) FROM deliveries WHERE delivery_date = '2022-05-06';

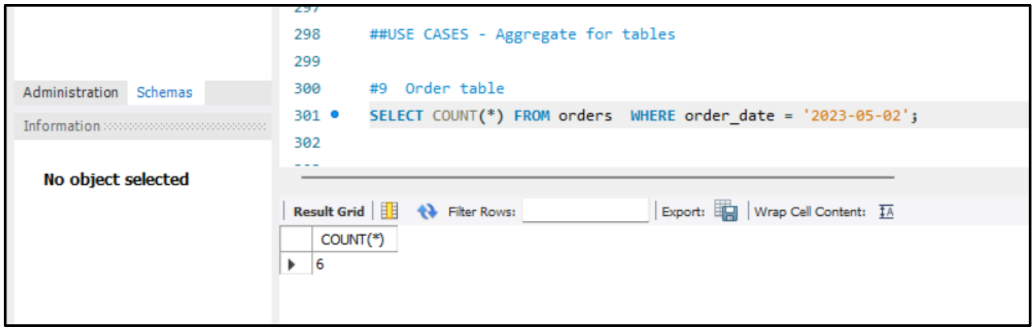


```
##USE CASES - Aggregate for tables

#8 Deliveries table
SELECT COUNT(*) FROM deliveries  WHERE  delivery_date < '2022-05-06';
```

COUNT(*)
5

I. TABLE ORDER:

SELECT COUNT(*) FROM orders WHERE order_date = '2023-05-02';

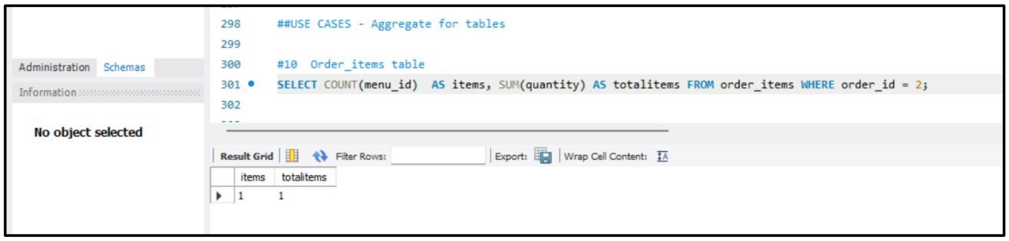


```
##USE CASES - Aggregate for tables

#9  Order table
SELECT COUNT(*) FROM orders  WHERE order_date = '2023-05-02';
```

```
COUNT(*)
6
```

J. TABLE ORDER_ITEMS:

SELECT COUNT(menu_id) AS items, SUM(quantity) AS totalitems FROM order_items WHERE order_id = 2;



```
##USE CASES - Aggregate for tables

#10  Order_items table
SELECT COUNT(menu_id)  AS items, SUM(quantity) AS totalitems FROM order_items WHERE order_id = 2;
```

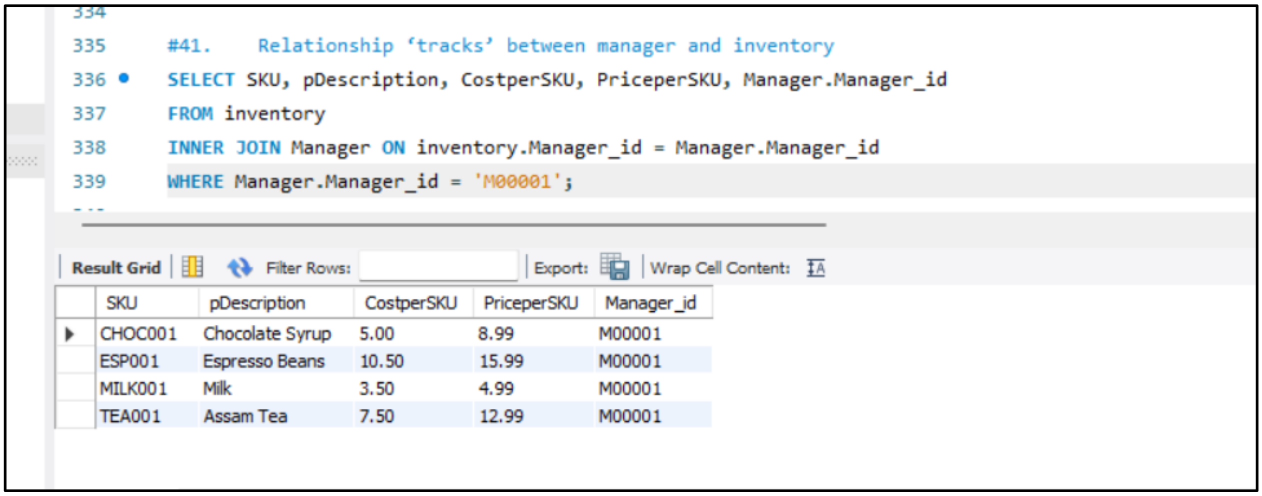
```
items   totalitems
1       1
```

III. Joint query for each set of entities that have a direct relationship

1. Relationship ‘tracks’ between manager and inventory

Use Case Name:	**To track the inventory details of all items managed by a specific manager**
Actor/User:	Manager/Staff
Steps:	5. The Manager/staff logs into the database system. 6. Then navigates to the " inventory " data view in the system. 7. Puts in manager id as prompted 8. System updates with the details of all items managed by the specific manager

Query:	SELECT SKU, pDescription, CostperSKU, PriceperSKU, Manager.Manager_id
	FROM inventory
	INNER JOIN Manager ON inventory.Manager_id = Manager.Manager_id
	WHERE Manager.Manager_id = 'M0001';



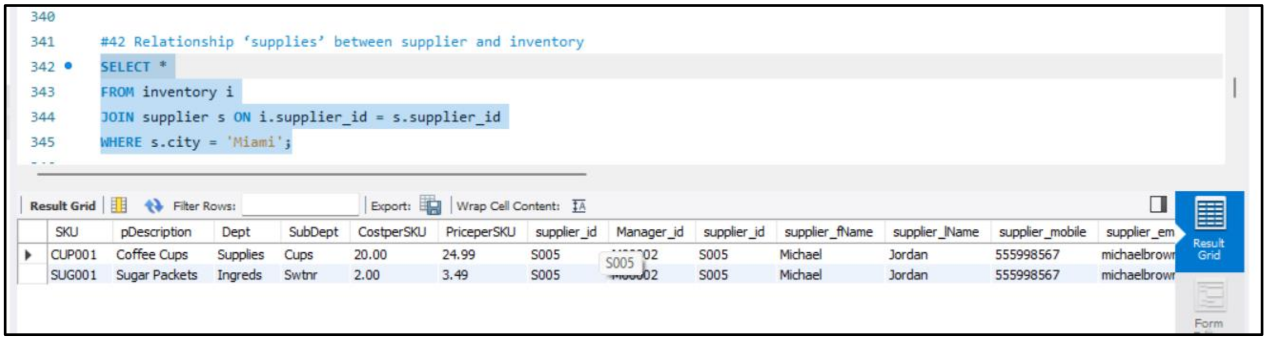
```
#41.    Relationship ‘tracks’ between manager and inventory
SELECT SKU, pDescription, CostperSKU, PriceperSKU, Manager.Manager_id
FROM inventory
INNER JOIN Manager ON inventory.Manager_id = Manager.Manager_id
WHERE Manager.Manager_id = 'M00001';
```

SKU	pDescription	CostperSKU	PriceperSKU	Manager_id
CHOC001	Chocolate Syrup	5.00	8.99	M00001
ESP001	Espresso Beans	10.50	15.99	M00001
MILK001	Milk	3.50	4.99	M00001
TEA001	Assam Tea	7.50	12.99	M00001

2. Relationship ‘supplies’ between supplier and inventory

Use Case Name:	**To retrieve a list of all items in inventory that are supplied by a supplier located in Anytown**
Actor/User:	Manager/Supplier
Steps:	5. Both Manager and Supplier can view this table of the system 6. Navigate to the " inventory " data view in the system. 7. Puts in supplier ID as prompted 8. System updates with the details of all items that are supplied by supplier in Houston

Query:	SELECT *
	FROM inventory i
	JOIN supplier s ON i.supplier_id = s.supplier_id
	WHERE s.city = 'Miami';



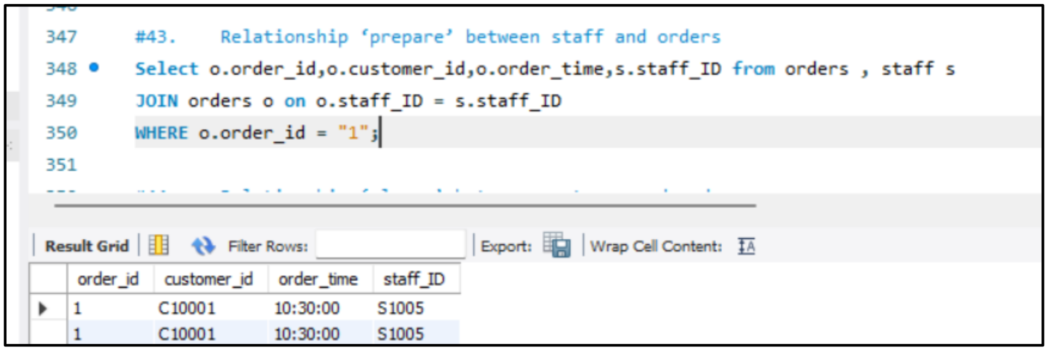
```
#42 Relationship ‘supplies’ between supplier and inventory
SELECT *
FROM inventory i
JOIN supplier s ON i.supplier_id = s.supplier_id
WHERE s.city = 'Miami';
```

SKU	pDescription	Dept	SubDept	CostperSKU	PriceperSKU	supplier_id	Manager_id	supplier_id	supplier_fName	supplier_lName	supplier_mobile	supplier_em
CUP001	Coffee Cups	Supplies	Cups	20.00	24.99	S005	M00002	S005	Michael	Jordan	555998567	michaelbrowr
SUG001	Sugar Packets	Ingreds	Swtnr	2.00	3.49	S005	M00002	S005	Michael	Jordan	555998567	michaelbrowr

S005

3. Relationship ‘prepare’ between staff and orders

Use Case Name:	**To prepare the order history of a specific customer**
Actor/User:	Staff
Steps:	5. The staff logs into the database system. 6. Then navigates to the "order " data view in the system. 7. Puts in the customer ID 8. System displays the order, bill and customer details as per below query
Query:	Select o.order_id,o.customer_id,o.order_time,s.staff_ID from orders , staff s JOIN orders o on o.staff_ID = s.staff_ID WHERE o.order_id = "1";



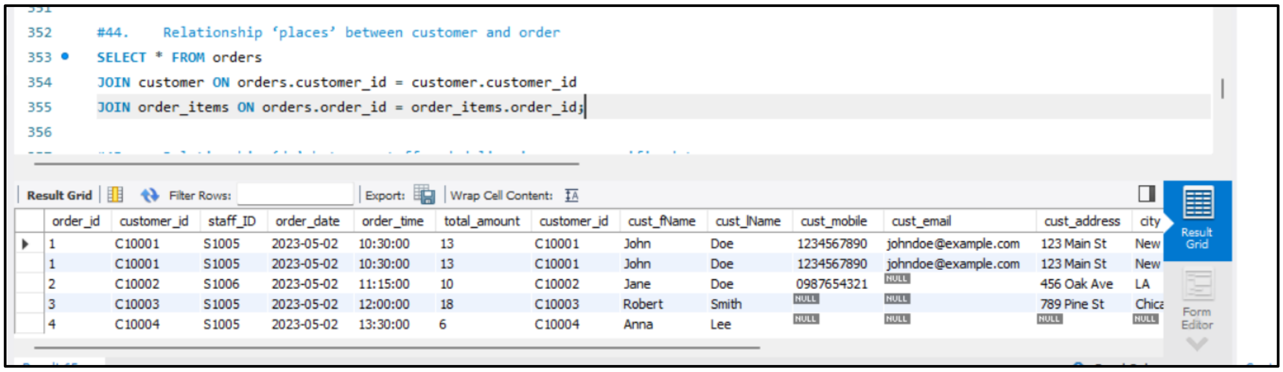
```
#43.    Relationship ‘prepare’ between staff and orders
Select o.order_id,o.customer_id,o.order_time,s.staff_ID from orders , staff s
JOIN orders o on o.staff_ID = s.staff_ID
WHERE o.order_id = "1";
```

order_id	customer_id	order_time	staff_ID
1	C10001	10:30:00	S1005
1	C10001	10:30:00	S1005

4. Relationship ‘places’ between customer and order

Use Case Name:	**Information about the customer who placed each order, as well as the items that were included in each order.**
Actor/User:	Customer/Staff
Steps:	5. Customer enters the system/application. 6. Then places the order. 7. System gets updated. 8. Following query returns information about customer who placed the order as well as the items.

Query:	SELECT * FROM orders
	JOIN customer ON orders.customer_id = customer.customer_id
	JOIN order_items ON orders.order_id = order_items.order_id;



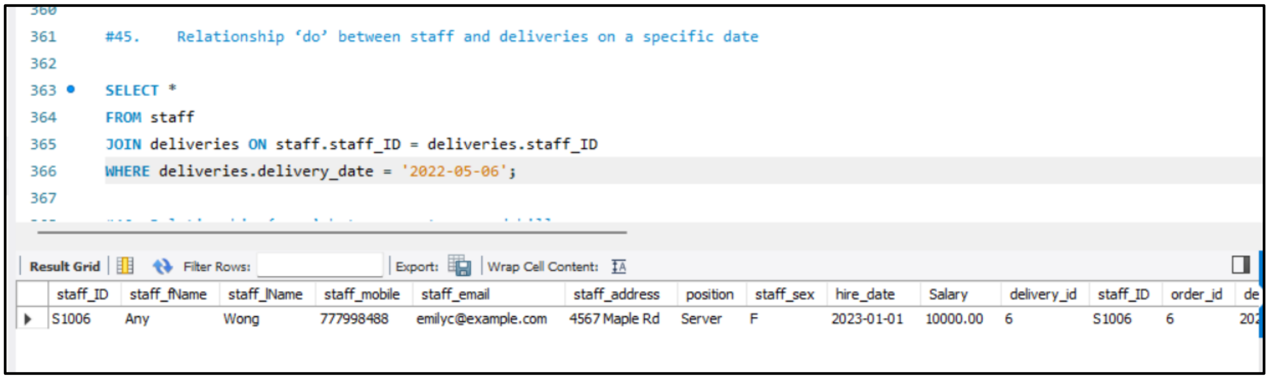
```
#44.    Relationship ‘places’ between customer and order
SELECT * FROM orders
JOIN customer ON orders.customer_id = customer.customer_id
JOIN order_items ON orders.order_id = order_items.order_id;
```

order_id	customer_id	staff_ID	order_date	order_time	total_amount	customer_id	cust_fName	cust_lName	cust_mobile	cust_email	cust_address	city
1	C10001	S1005	2023-05-02	10:30:00	13	C10001	John	Doe	1234567890	johndoe@example.com	123 Main St	New
1	C10001	S1005	2023-05-02	10:30:00	13	C10001	John	Doe	1234567890	johndoe@example.com	123 Main St	New
2	C10002	S1006	2023-05-02	11:15:00	10	C10002	Jane	Doe	0987654321	NULL	456 Oak Ave	LA
3	C10003	S1005	2023-05-02	12:00:00	18	C10003	Robert	Smith	NULL	NULL	789 Pine St	Chica
4	C10004	S1005	2023-05-02	13:30:00	6	C10004	Anna	Lee	NULL	NULL	NULL	NULL

5. Relationship ‘do’ between staff and deliveries on a specific date

Use Case Name:	**Retrieves information about the staff member who made deliveries**
Actor/User:	Manager/Staff
Steps:	5. The Manager/staff logs into the database system. 6. Then navigates to the " deliveries " data view in the system. 7. Staff ID is entered 8. The below query retrieves the information on the staff member who made particular delivery on particular date.

Query:	SELECT *
	FROM staff
	JOIN deliveries ON staff.staff_ID = deliveries.staff_ID
	WHERE deliveries.delivery_date = “2022-03-16”;



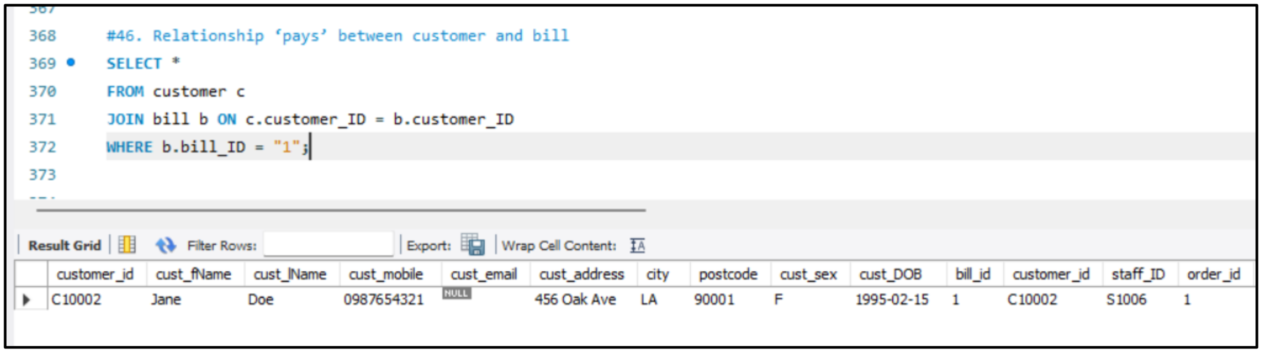
```
#45.   Relationship ‘do’ between staff and deliveries on a specific date

SELECT *
FROM staff
JOIN deliveries ON staff.staff_ID = deliveries.staff_ID
WHERE deliveries.delivery_date = '2022-05-06';
```

staff_ID	staff_fName	staff_lName	staff_mobile	staff_email	staff_address	position	staff_sex	hire_date	Salary	delivery_id	staff_ID	order_id	de
S1006	Any	Wong	777998488	emilyc@example.com	4567 Maple Rd	Server	F	2023-01-01	10000.00	6	S1006	6	202

6. Relationship ‘pays’ between customer and bill

Use Case Name:	**This query will return all information for the customer associated with the specific bill, , as well as information about the bill itself**
Actor/User:	Manager/Staff/Customer
Steps:	6. The Manager/staff logs into the database system. 7. Then navigates to the " bills " data view in the system. 8. Enters the required bill ID 9. System generates all data related to the specific bill. 10. Customer can also view their bills on the system.

Query:	SELECT *
	FROM customer c
	JOIN bill b ON c.customer_ID = b.customer_ID
	WHERE b.bill_ID = ‘1’;



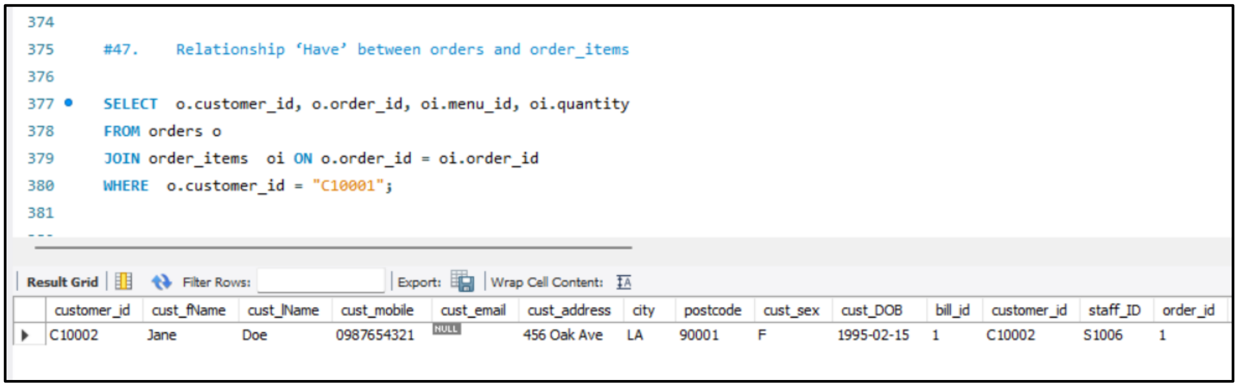
```
#46. Relationship ‘pays’ between customer and bill
SELECT *
FROM customer c
JOIN bill b ON c.customer_ID = b.customer_ID
WHERE b.bill_ID = "1";
```

	customer_id	cust_fName	cust_lName	cust_mobile	cust_email	cust_address	city	postcode	cust_sex	cust_DOB	bill_id	customer_id	staff_ID	order_id
▶	C10002	Jane	Doe	0987654321	NULL	456 Oak Ave	LA	90001	F	1995-02-15	1	C10002	S1006	1

7. Relationship ‘Have’ between orders and order_items

Use Case Name:	To see the items in an order placed by a customer
Actor/User:	Staff
Steps:	5. The Staff member logs into the database system. 6. User clicks on the “orders” button. 7. User is prompted to enter the customer ID. User enters the customer ID. 8. System shows the items in an order placed by a customer.

Query:	SELECT o.customer_id, o.order_id, oi.menu_id, oi.quantity
	FROM orders o
	JOIN order_items oi ON o.order_id = oi.order_id
	WHERE o.customer_id = ‘C00001’;



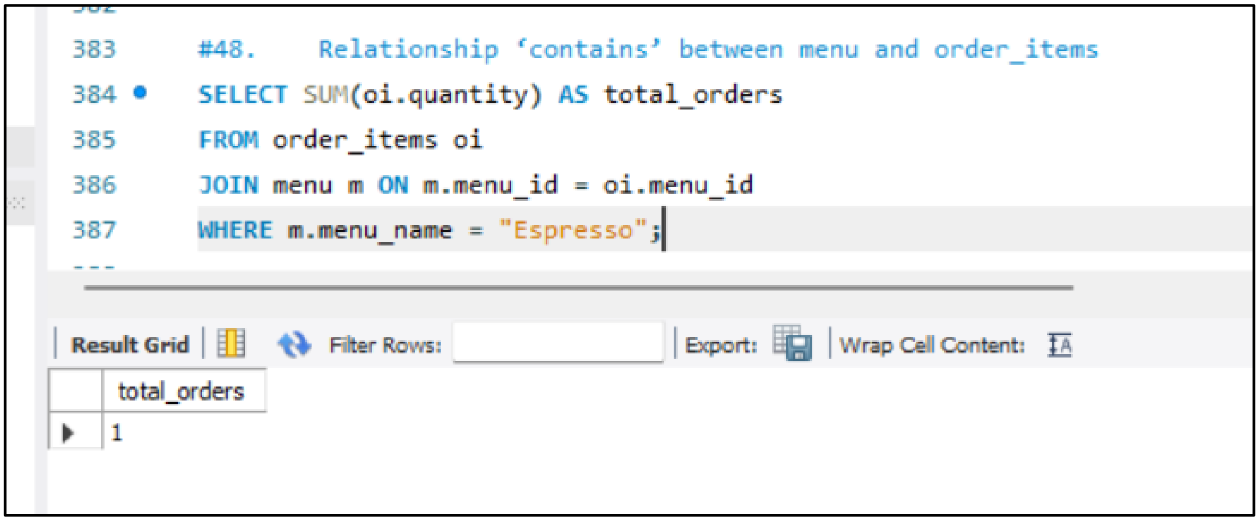
```
#47.   Relationship ‘Have’ between orders and order_items

SELECT  o.customer_id, o.order_id, oi.menu_id, oi.quantity
FROM orders o
JOIN order_items  oi ON o.order_id = oi.order_id
WHERE  o.customer_id = "C10001";
```

customer_id	cust_fName	cust_lName	cust_mobile	cust_email	cust_address	city	postcode	cust_sex	cust_DOB	bill_id	customer_id	staff_ID	order_id
C10002	Jane	Doe	0987654321	NULL	456 Oak Ave	LA	90001	F	1995-02-15	1	C10002	S1006	1

8. Relationship ‘contains’ between menu and order_items

Use Case Name:	**To find the total number of a menu item ordered**
Actor/User:	Staff
Steps:	7. The Staff member logs into the database system. 8. User clicks “Reports” button. 9. Then the user is prompted to enter the criteria for which report is required. 10. User enters the criteria – menu and clicks on “submit” button. 11. The user is prompted to enter the menu item for which the report is required. User enters the menu item and clicks “Done”. 12. The report showing the total number of times a particular menu item was ordered.

Query:	SELECT SUM(oi.quantity) AS total_orders
	FROM order_items oi
	JOIN menu m ON m.menu_id = oi.menu_id
	WHERE m.menu_name = “Latte”;



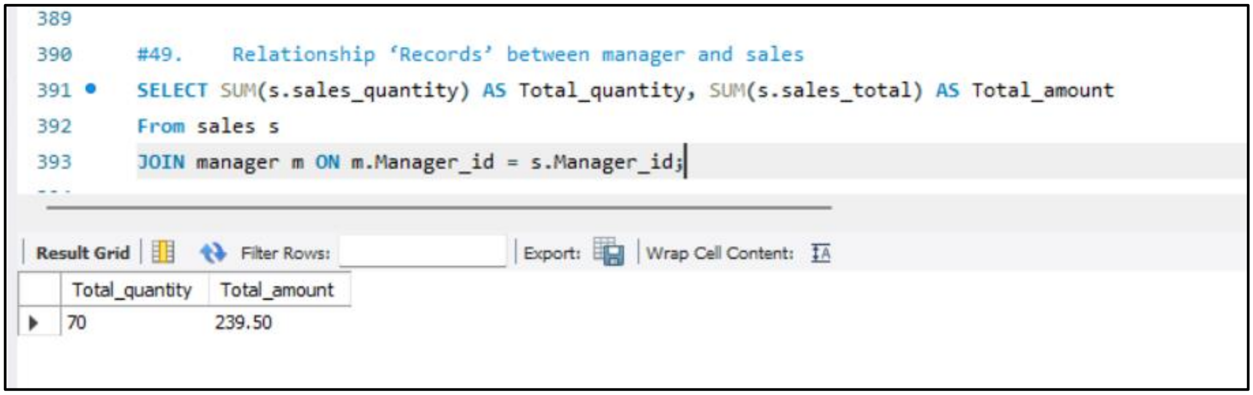
```
#48.    Relationship ‘contains’ between menu and order_items
SELECT SUM(oi.quantity) AS total_orders
FROM order_items oi
JOIN menu m ON m.menu_id = oi.menu_id
WHERE m.menu_name = "Espresso";
```

```
total_orders
1
```

9. Relationship 'Records' between manager and sales

Use Case Name:	**To find the total sales and total amount of sales on a particular date**
Actor/User:	Manager
Steps:	7. The Manager logs into the database system. 8. Manager clicks on the "Report" button. 9. Then the user is prompted to enter the criteria for which report is required. 10. User enters the criteria - sales and clicks on "submit" button. 11. User is prompted to enter the date for which sales report is required. User enters the date and clicks "Done". 12. The report showing total sales and total amount will be generated.

Query:	SELECT SUM(s.sales_quantity) AS Total_quantity, SUM(s.sales_total) AS Total_amount
	From sales s
	JOIN manager m ON m.Manager_id = s.Manager_id;



```
#49.    Relationship 'Records' between manager and sales
SELECT SUM(s.sales_quantity) AS Total_quantity, SUM(s.sales_total) AS Total_amount
From sales s
JOIN manager m ON m.Manager_id = s.Manager_id;
```

Total_quantity	Total_amount
70	239.50

9. Link records:

On One drive:

CS 5318 UHD Final Phase Demo_Data Brewers.mp4

On Youtube:

https://youtu.be/P5wqc82Ia_M

Two above links have the same content (just in case one of them cannot open)

10. Conclusion

In conclusion, our cafe database management system can become an important tool for managing the various aspects of a cafe business, such as inventory, orders, deliveries, bills, and sales. The system includes several tables that are interrelated through various primary and foreign keys, allowing for efficient data retrieval and manipulation.

The tables in the above database are mostly normalized, meaning they are in 1NF, 2NF, 3NF, or BCNF. One can write queries to retrieve data from the tables, allowing for analysis and reporting. For example, a query can be used to retrieve the total sales quantity and amount for a specific manager. However, care must be taken when constructing queries to ensure that they are correct and efficient.

Overall, the cafe database management system provides an efficient and effective way to manage the various aspects of a cafe business. With proper maintenance and optimization, it can help streamline operations and increase profitability.

11. Reference

Dishman, D., & Owen, J. (2011). CS342-Phase-5 Drew Dishman Jacob Owen. California State University Bakersfield, Department of Computer and Electrical Engineering and Computer Science. Retrieved from https://www.cs.csub.edu/~hwang/CS342/Student_Project/2011/CS342-Phase-5%20Drew%20Dishman%20Jacob%20Owen.pdf