

**Department of Computer Science**

## Final Term Project, Fall 2023-24

|  |  |  |  |
| --- | --- | --- | --- |
| Course | CSC 2108 - Introduction to Database [Section: A] | Group | 6 |

### Group Members:

|  |  |
| --- | --- |
| Student ID | Name |
| **22-49851-3** | **MD Junaid Alam Eusha** |
| **23-50864-1** | **Tanjila Habib Srabony** |
| **23-50857-1** | **Shirin Mahbuba** |
| **22-47644-2** | **Sarkar Romman Bin Waly Tonmoy** |

**COURSE INSTRUCTOR: RAZUAN KARIM**

**PROJECT TITLE:**

**ONLINE FOOD MANAGEMENT SYSTEM**

**Contents**

|  |  |  |
| --- | --- | --- |
| Topic  1. Contents | Page No  2 |  |
| 2. Case Study | 3 |  |
| 3. Gantt Chart | 4 |  |
| 4. ER-Diagram | 5 |  |
| 5. Normalization | 6 |  |
| 6. Finalization | 13 |  |
| 7. Table Creation | 13 |  |
| 8. Data Insertion | 18 |  |
| 9. Query Testing | 25 |  |
| 10.Database Connection  11. Conclusion | 34  37 |  |
|  |  | 2 |

Introduction

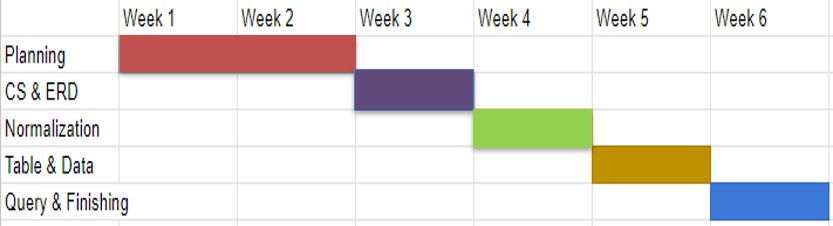
## Welcome to OpenTable! Our project brings tasty food and modern convenience together. With just a few clicks, you can explore a variety of delicious dishes, order them, and have them delivered to your doorstep. Our user-friendly system makes it easy to find what you are craving, track your order in real time, and pay securely. Join us as we make enjoying great food simpler and more enjoyable than ever before.

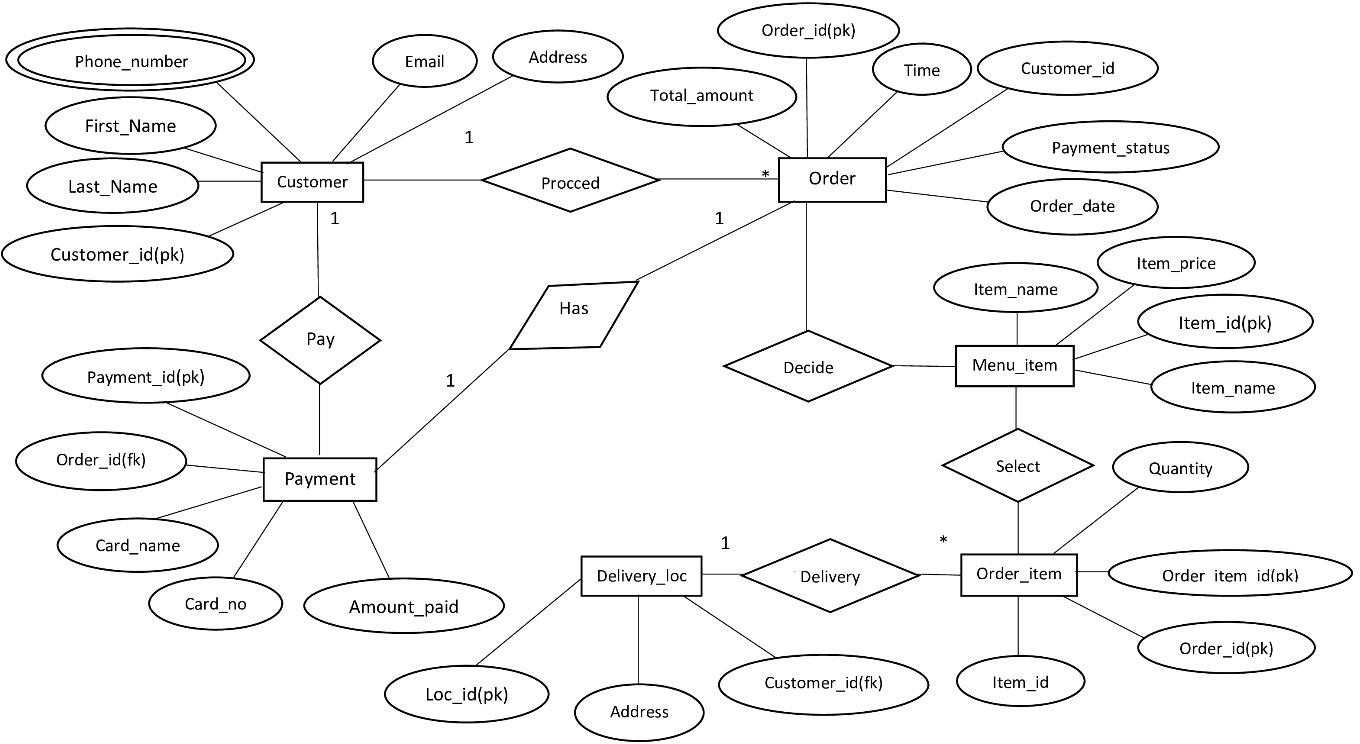
**CASE STUDY**

## In an online food management system, **customers** are able to place food orders conveniently after they have registered. To **register**, customers need to provide their information, including **Customer ID (Primary Key)**, First Name, Last Name, Email, Phone Number, and Address. The Customer ID serves as the primary key for identification. Each customer can place multiple orders, with each order having a unique Order Id. **Customers** select their desired food items from the menu. Once an order is completed, the system enables customers to make payments using online methods or credit/debit cards. The **payment details** include card number, card name, payment ID, and the amount paid. For each order placed, the customer must pay the corresponding amount. Additionally, customers' delivery locations are stored in the system. The **delivery location** entity includes a **Location ID** (Primary Key), linked to the respective **Customer ID**, and the delivery address. This setup ensures that customers can smoothly place and pay for orders, with the system effectively managing their information, orders, payments, and delivery locations.

3

**Gantt Chart**

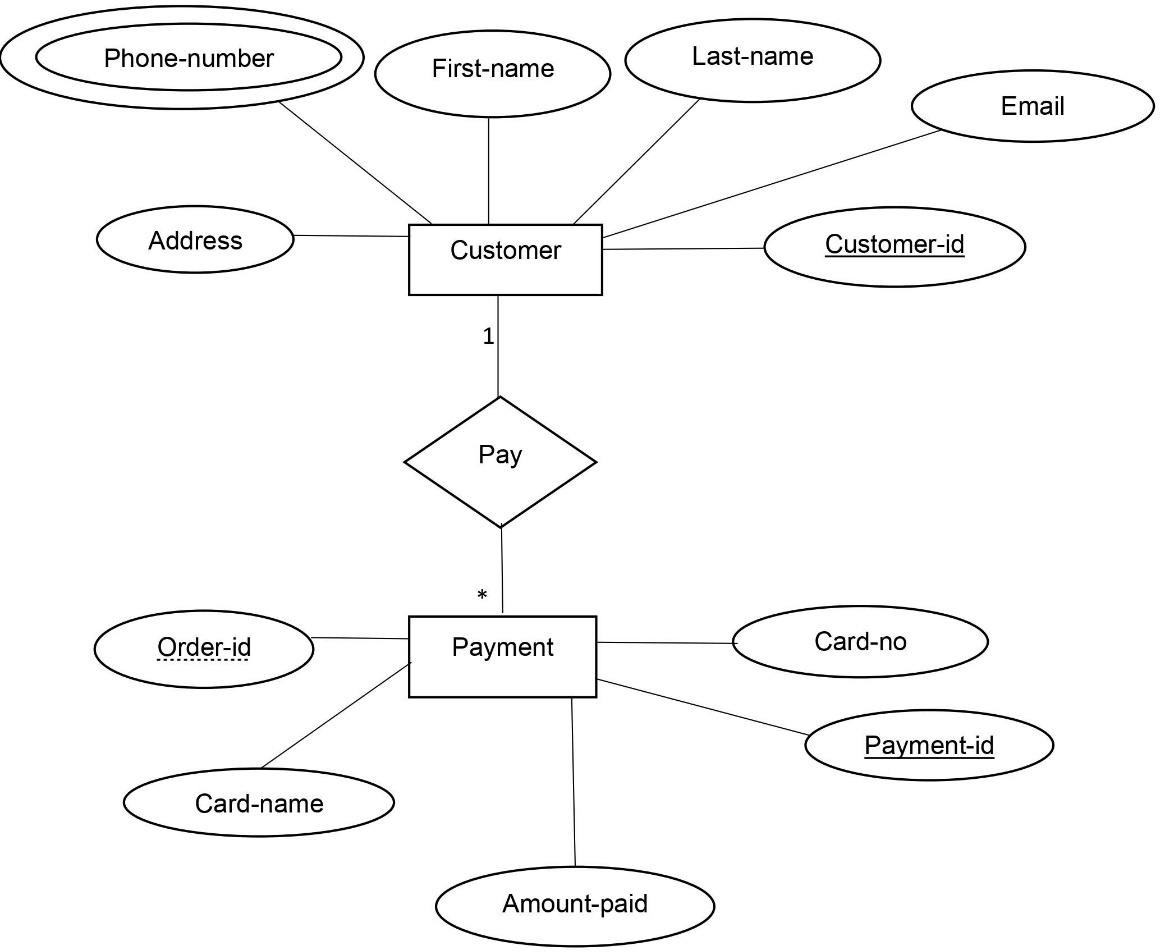




### Figure 1:ER diagram for food management system

Pay:

# Normalization



UNF:

### Figure 2: ER diagram for relationship between Customer and Payment

* Pay (Customer ID , First Name, Last Name, Email, Phone Number, Address, Payment ID

,Card no, Card Name, Amount Paid)

1NF : Phone Number is multivalued attribute

1. Customer ID ,First Name, Last Name, Email, Phone Number, Address, Payment ID ,Card no, Card Name, Amount Paid).

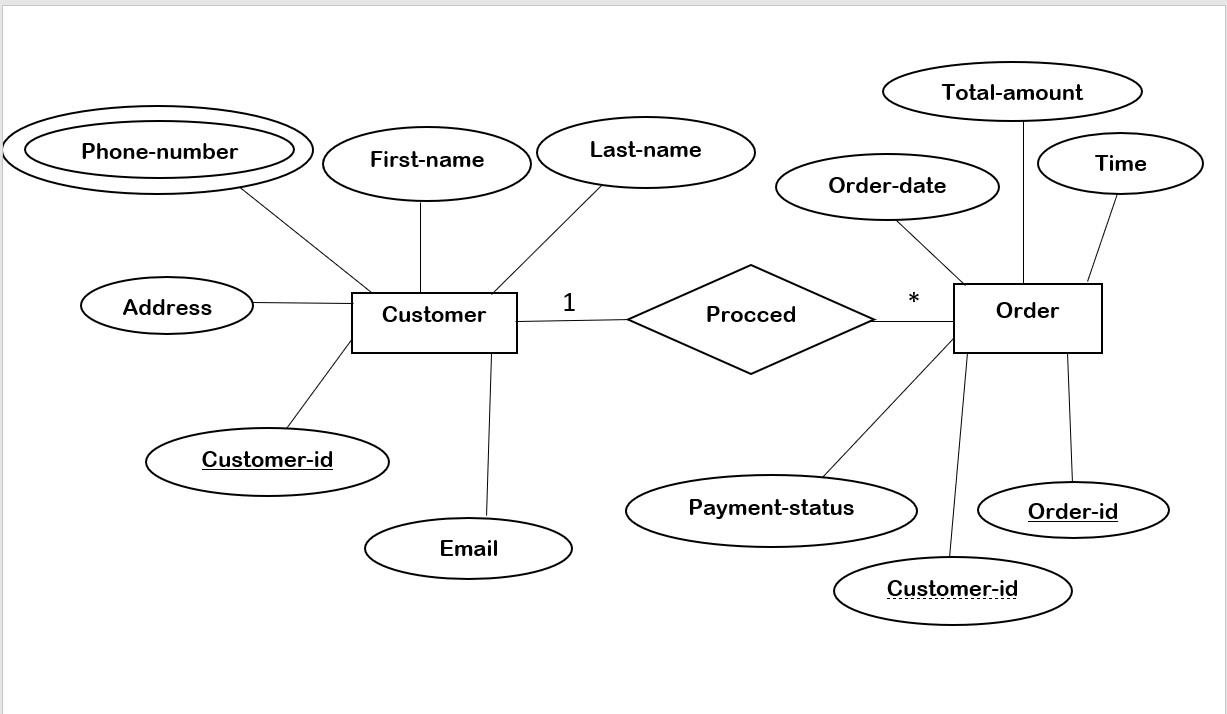
#### 2NF :

* 1. Customer ID , First Name, Last Name, Email, Address
  2. Customer ID , Phone Number,
  3. Payment ID , Card no, Card Name, Amount Paid 3NF :

There is no Transitive dependency

1. Customer ID , First Name, Last Name, Email, Address
2. Customer ID , Phone Number,
3. Payment ID , Card no, Card Name, Amount Paid

Procced:



UNF:

### Figure 3: ER diagram for relationship between Customer and Order

PROCCED (Customer ID , First Name, Last Name, Email, Phone Number, Address, Order ID

,Customer ID(FK) ,Order Date, Total Amount, Payment Status, time ) 1NF :

Phone Number is multivalued attribute

Customer ID , First Name, Last Name, Email, Phone Number, Address, Order ID (PK), Customer ID(FK) , Order Date, Total Amount, Payment Status ,Time

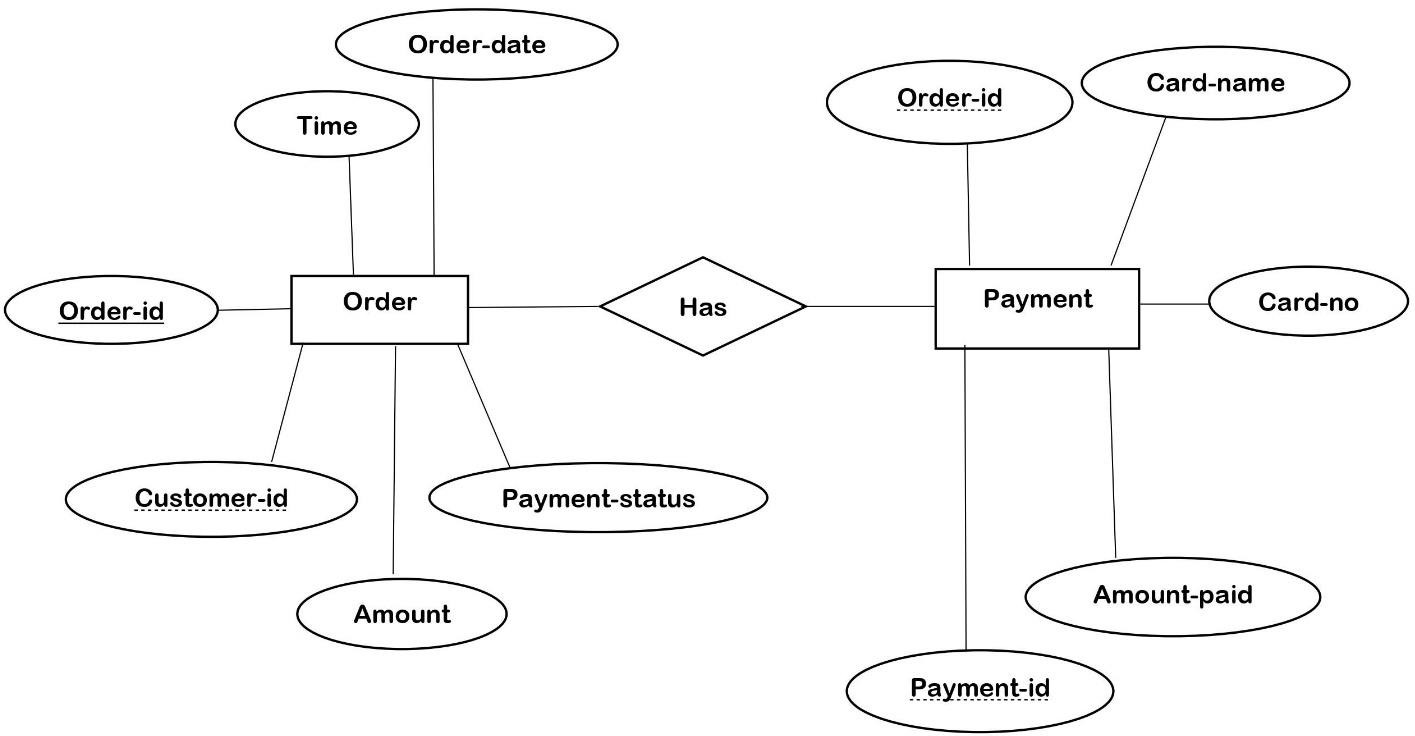
#### 2NF :

1. Order ID, Order Date, Total Amount, Payment Status, Time
2. Customer ID, First Name, Last Name, Email, Address
3. Customer ID, Phone Number

3NF : There is no Transitive dependency

1. Order ID, Order Date, Total Amount, Payment Status ,Time
2. Customer ID, First Name, Last Name, Email, Address
3. Customer ID, Phone Number

Has:



### Figure 4: ER diagram for relationship between Order and Payment

UNF : HAS(Order ID, Order Date, Total Amount, Payment Status, Payment ID ,Card no, Card Name, Amount Paid)

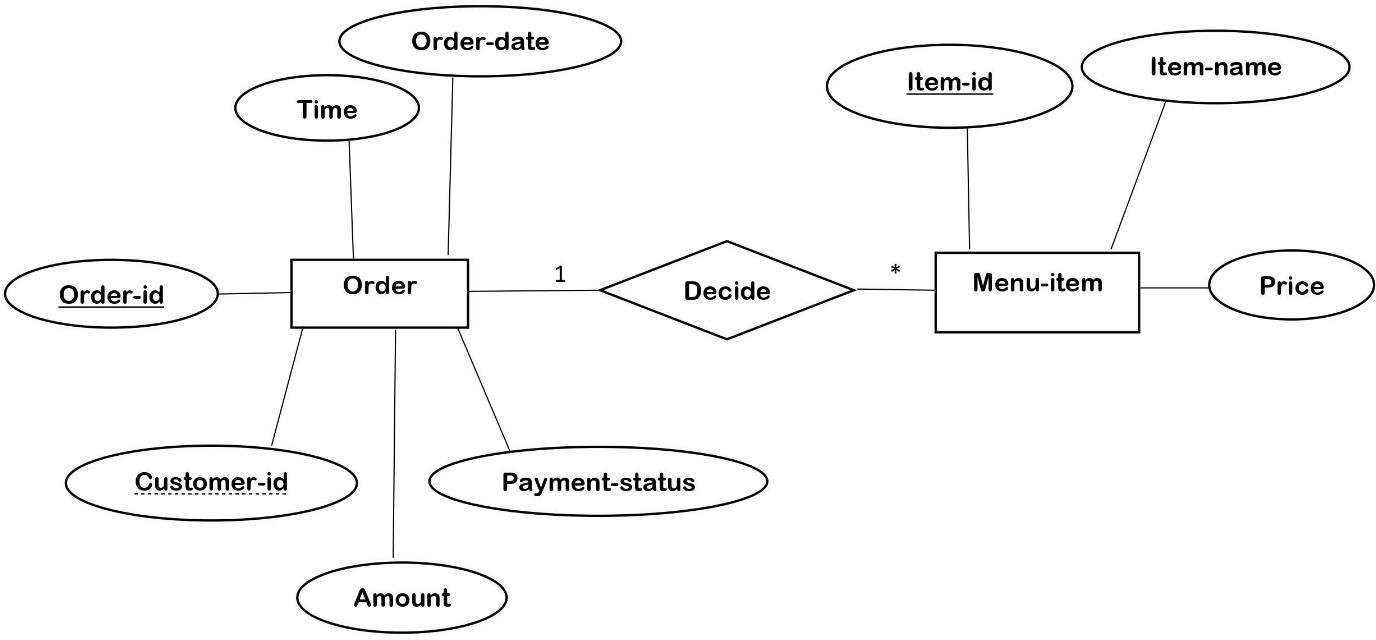
1NF : There is no multivalued attribute

1. Order ID , Order Date, Time, Total Amount, Payment Status, Payment ID , Card no, Card Name, Amount Paid

#### 2NF :

1. Order ID ,Order Date, Total Amount, Payment Status, Time
2. Payment ID , Card no, Card Name, Amount Paid 3NF : There is no Transitive dependency
3. Order ID ,Order Date, Total Amount, Payment Status, Time
4. Payment ID , Card no, Card Name, Amount Paid

Decide :



UNF :

### Figure 5:ER diagram for relationship between Order and Menu-item

Order ID ,Customer ID (Foreign Key), Order Date, Total Amount, Payment Status, Price , Item\_name , Item\_ID

#### 1NF :

The original relation is already in 1NF because all columns contain atomic values, and there are no repeating groups

1. Order ID , Customer ID(FK) ,Order Date, Total Amount, Payment Status, Price , Item\_name , Item\_ID

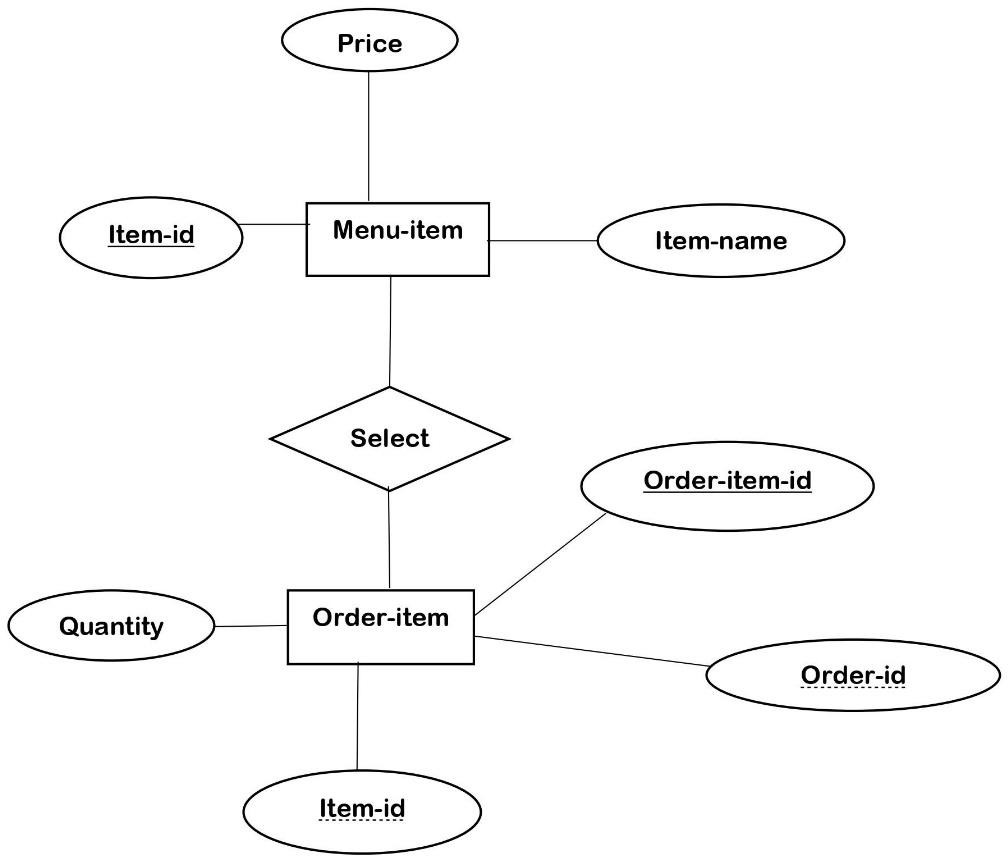
#### 2NF:

1. Order ID , Customer ID(FK) , Order Date, Total Amount, Payment Status, Price.
2. Customer id, Item\_name , Item\_ID 3NF :

There is no Transitive dependency

1. Order ID , Customer ID(FK) , Order Date, Total Amount, Payment Status, Price
2. Customer id, Item\_name , Item ID

Select:



### Figure 6: ER diagram for relationship between Menu-item and Order-item

#### UNF :

SELECT( Item\_name , Item ID ,Price, Order\_Item\_ID , Order ID(FK) , Item ID(FK) , Quantity )

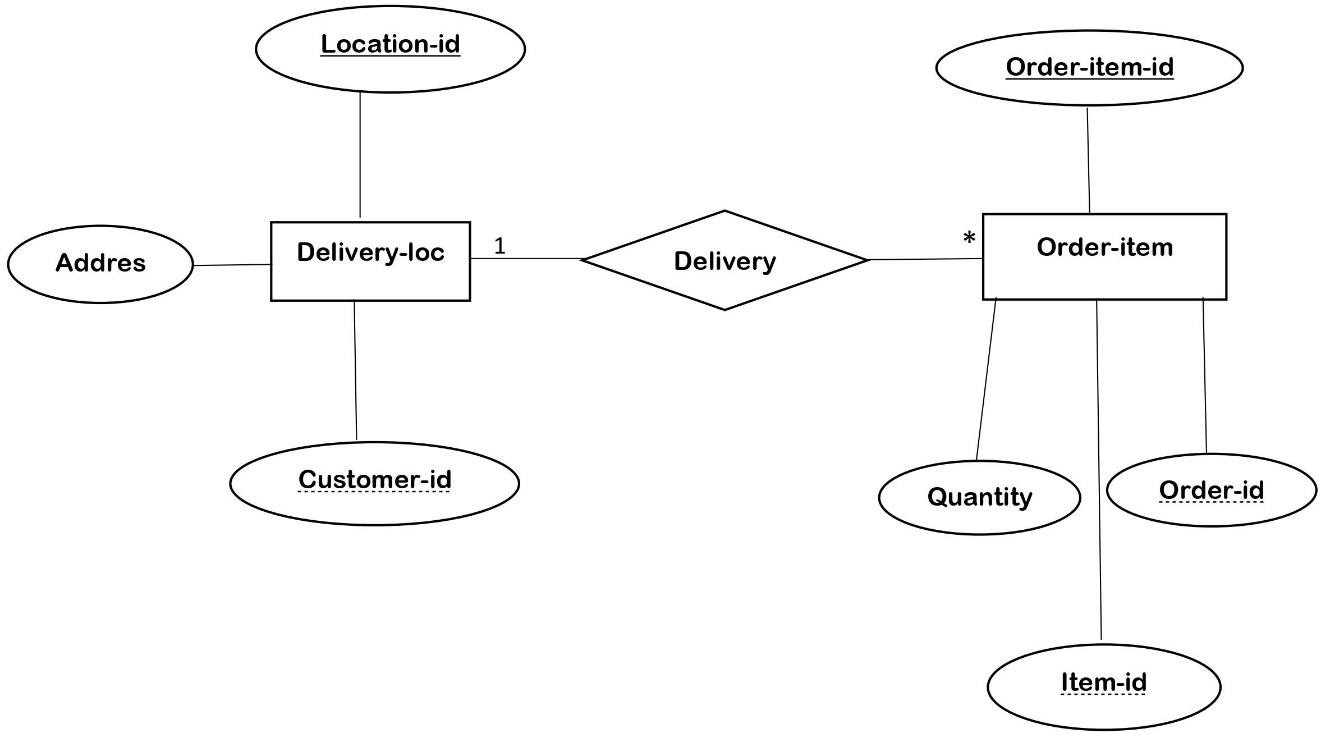
1NF : The original relation is already in 1NF because all columns contain atomic values, and there are no repeating groups

1. Item\_name , Item ID , Price, Order\_Item\_ID ,Order ID(FK), Item ID(FK), Quantity 2NF :
   1. Item\_ID, Item\_name, Price
   2. Order\_ID, Item ID , Order\_Item\_ID, Quantity

3NF : There are no transitive dependencies to eliminate. Therefore, the relation remains in 3NF

1. Item\_ID, Item\_name, Price
2. Order ID, Item ID , Order Item ID, Quantity

Delivery:



### Figure 7: ER diagram for relationship between Delivery-loc and Order-item

UNF : Delivery(Order Item ID , Order ID(FK) , Item ID(FK) , Quantity, Location ID , Customer ID (FK) ,Address )

1NF: we ensure that each column contains atomic (indivisible) values, and there are no repeating groups or arrays.

* 1. Delivery(Order Item ID , Order ID (FK), Item ID (FK) , Quantity, Location ID , customer ID(FK), Address)

#### 2NF:

1. Order\_ID, Item\_ID, Order\_Item\_ID, Quantity 2.Order\_ID, Item\_ID, Location ID , Customer\_ ID(FK)

3NF : There are no transitive dependencies to eliminate. Therefore, the relation remains in 3NF. 1.Order\_ID, Item\_ID, Order\_Item\_ID, Quantity

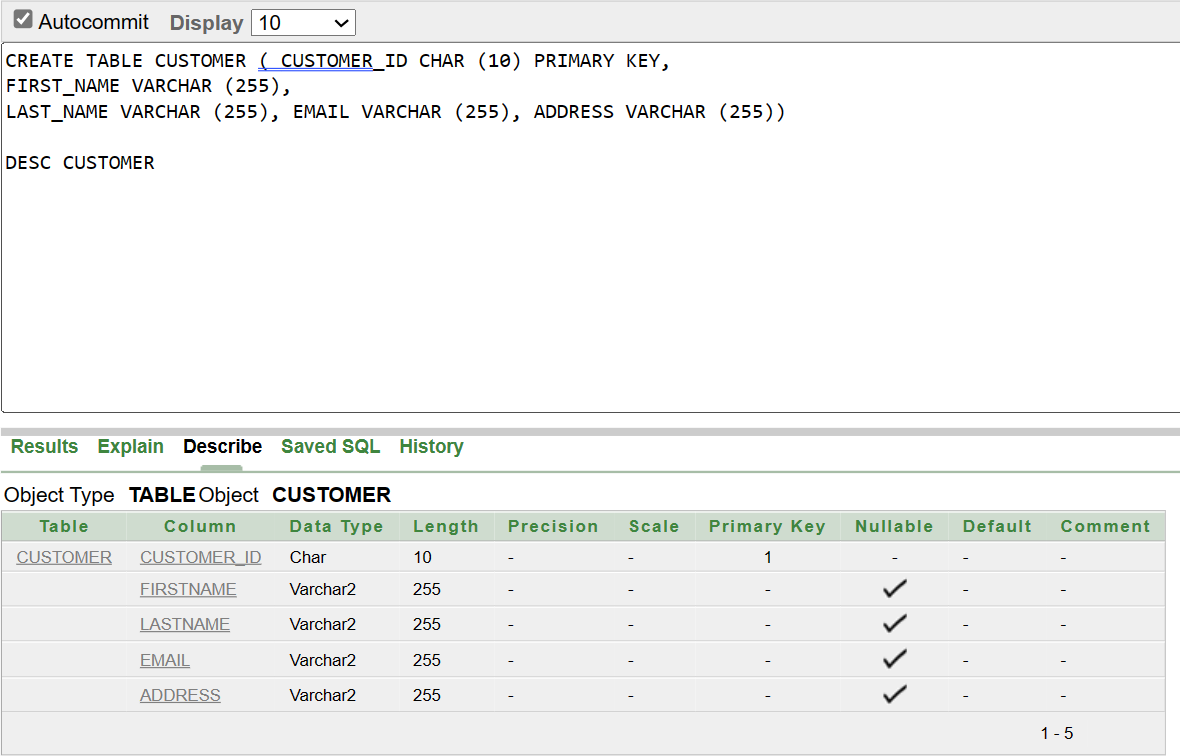
1. Order\_ID, Item\_ID, Location ID, Customer ID ( FK) , Address

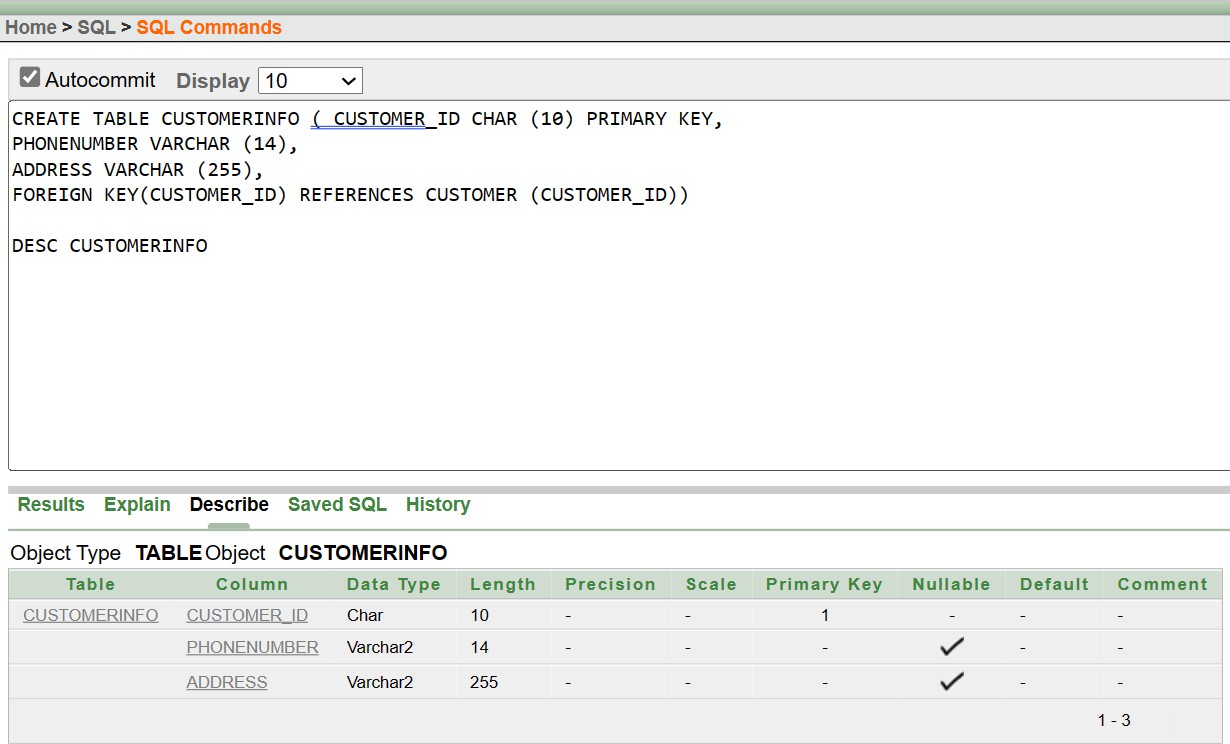
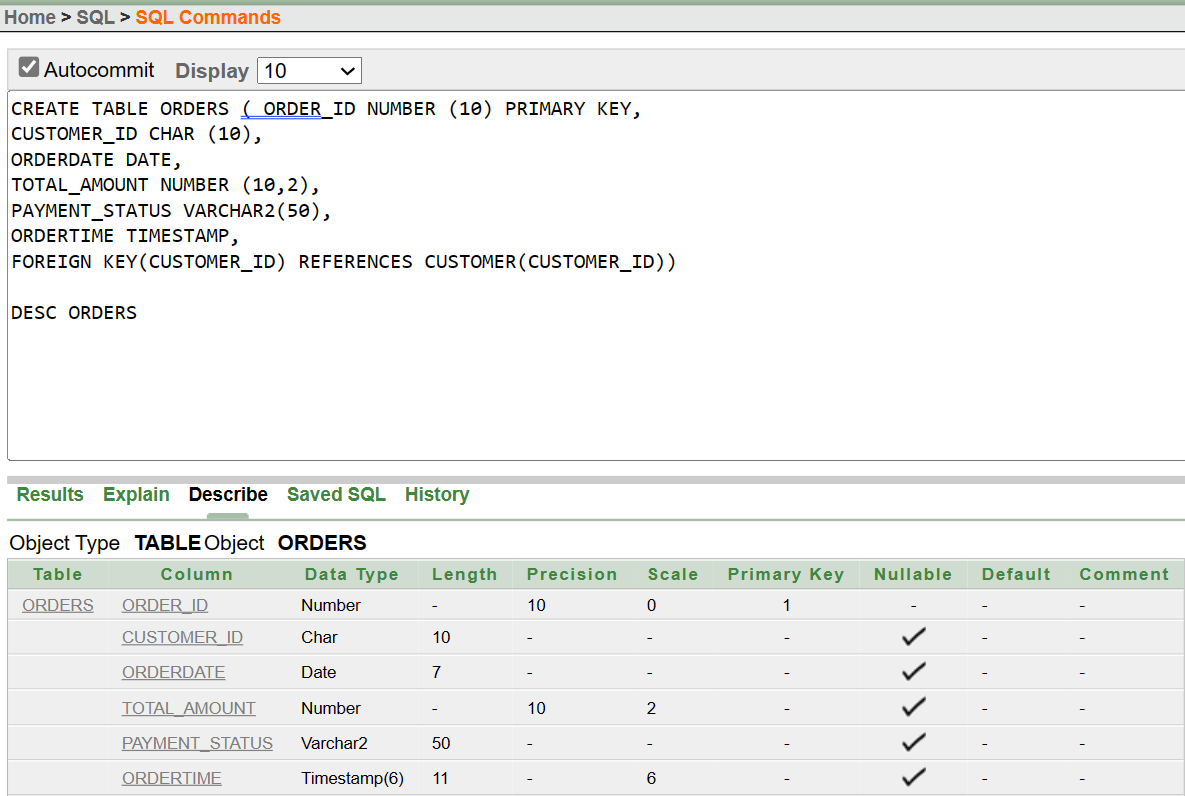
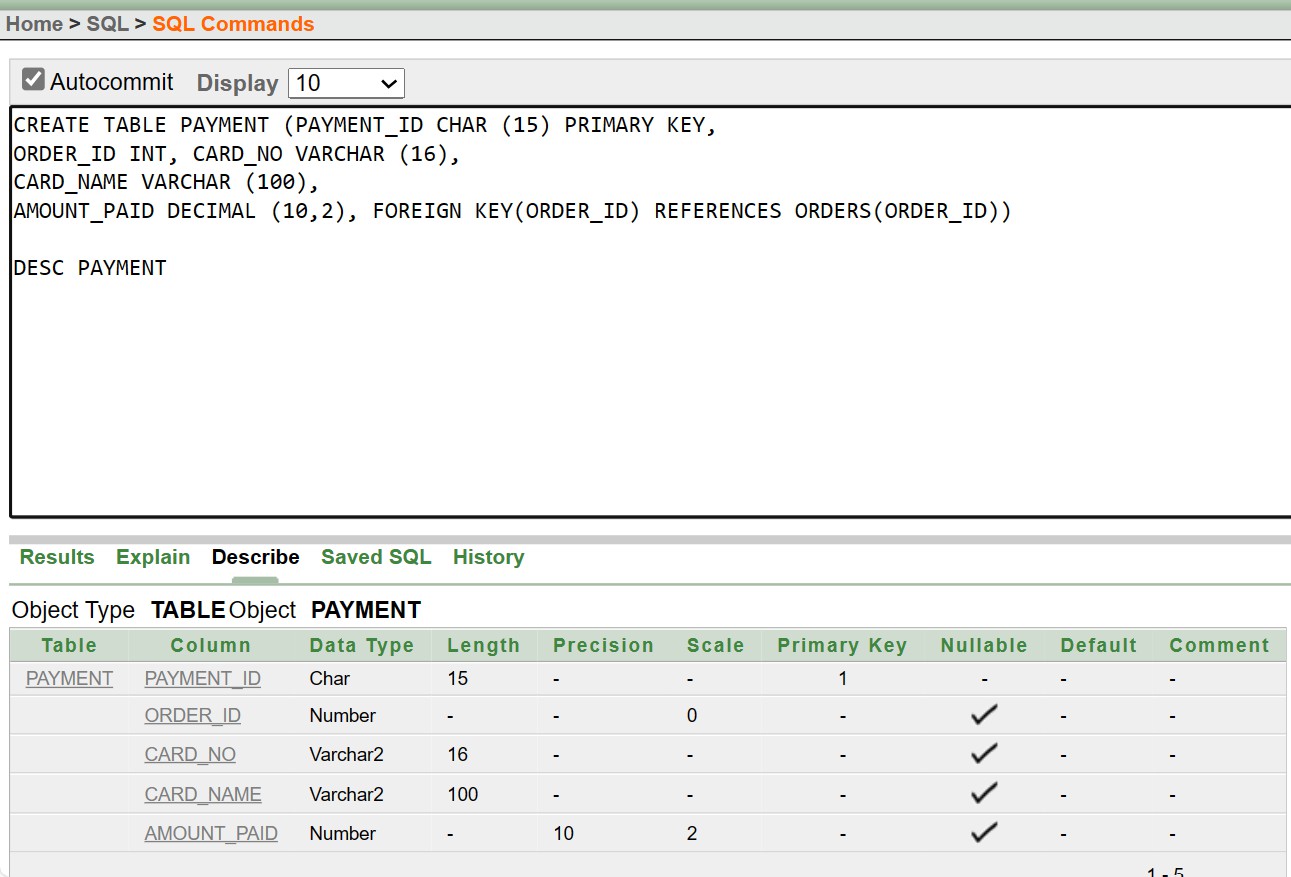
# Finalization

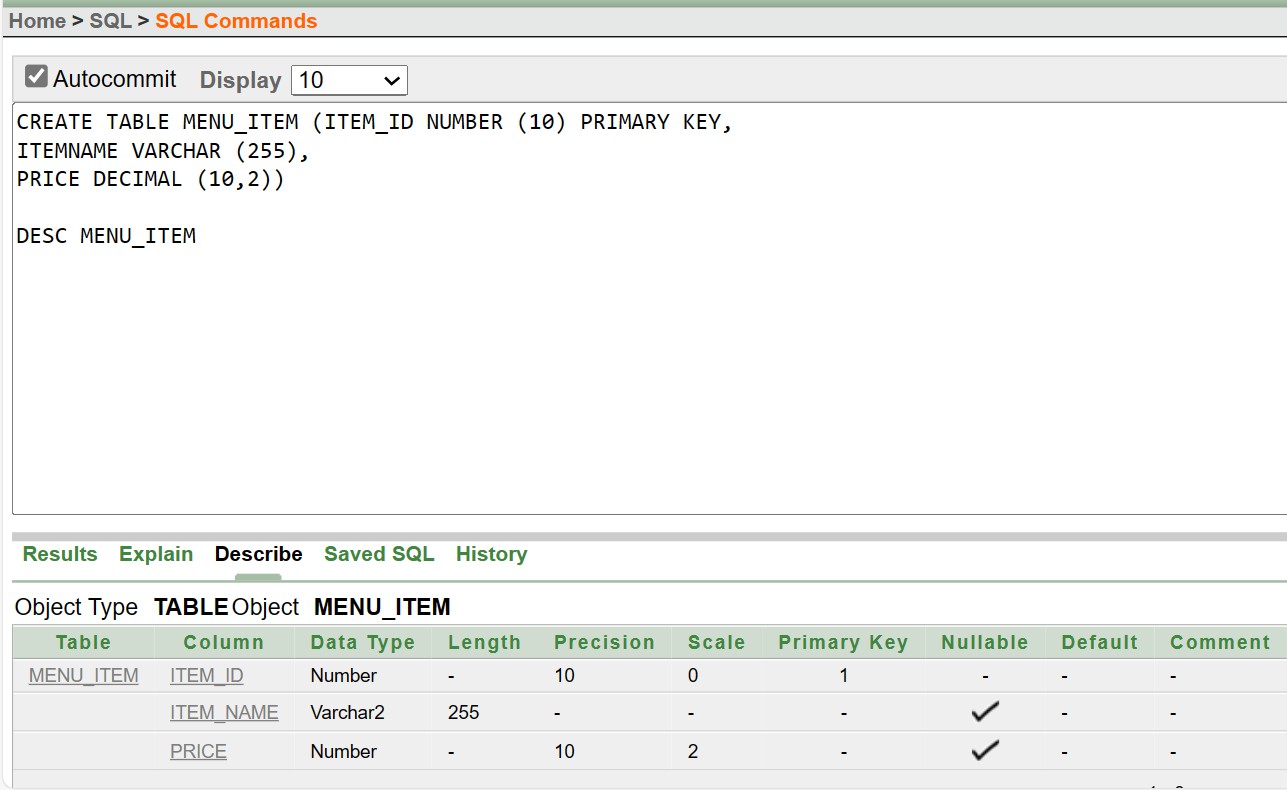
* 1. ~~Customer (Customer ID , First Name, Last Name, Email)~~
  2. CustomerInfo (Customer ID (PK, FK), Phone Number, Address)
  3. Customer (Customer ID, First Name, Last Name, Email, Phone Number, Address)
  4. Payment (Payment ID , Customer ID (FK), Order ID, Card no, Card Name, Amount Paid)
  5. Order(Order ID, Order Date, Total Amount, Payment Status, Customer ID (FK), Time)
  6. ~~Order ID , Order Date, Total Amount, Payment Status, Time~~
  7. ~~Payment ID (Primary Key), Card no, Card Name, Amount Paid~~
  8. Order ID , Customer ID (Foreign Key), Order Date, Total Amount, Payment Status, Price
  9. ~~Customer id, Item\_name , Item\_ID~~
  10. Item\_ID, Item\_name, Price
  11. ~~Order No, Item ID ,Order\_Item\_ID(FK), Quantity~~
  12. Order\_No, Item\_ID, Location\_ID ,Customer\_ID(FK)

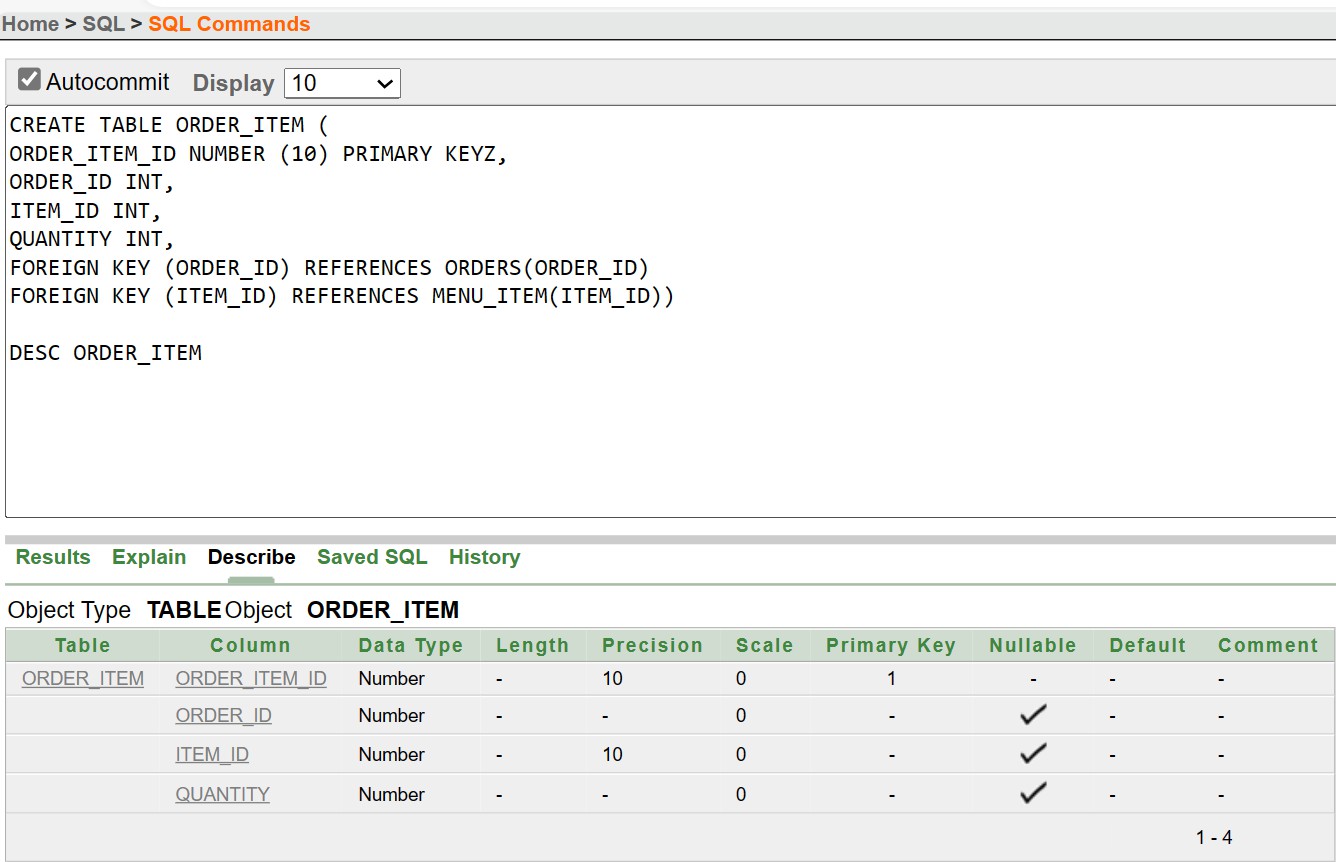
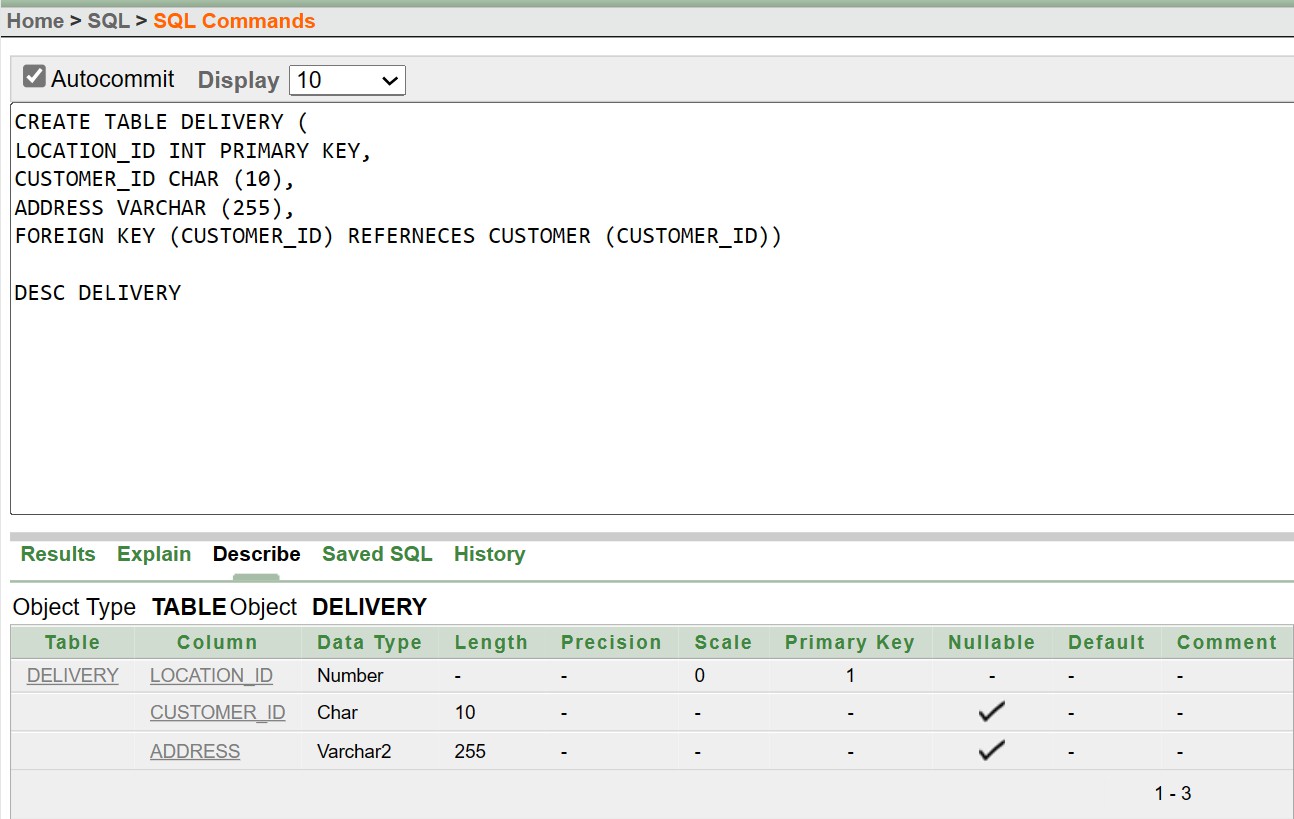
# Table Creation

1. Customer Table:

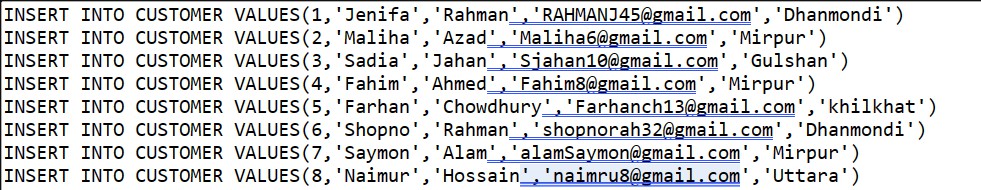


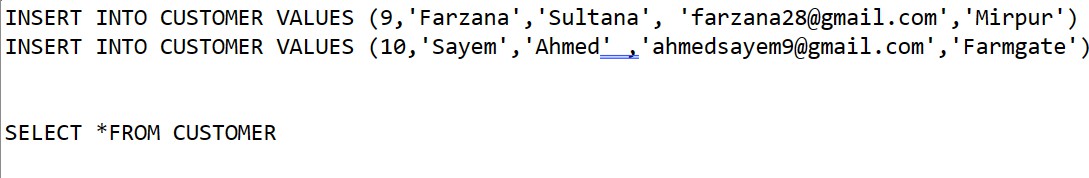
1. Customer-info Table:
2. Orders Table:
3. Payment Table:
4. Menu-item Table:

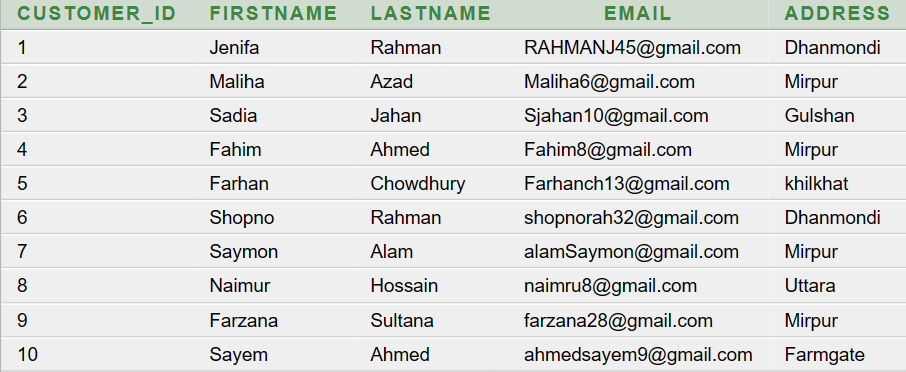


1. Order-item Table:
2. Delivery Table:
3. **Customer Table:**

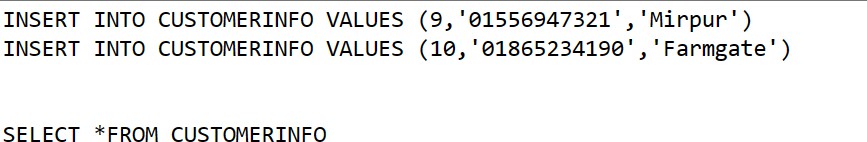
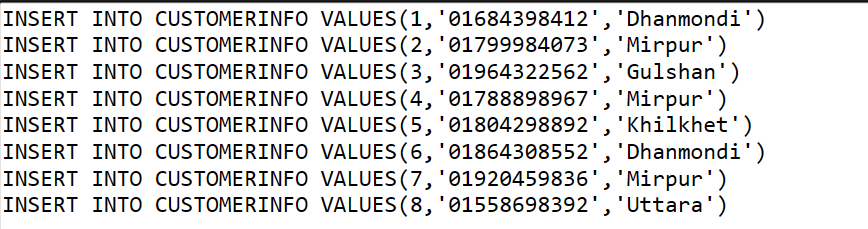
# Value Insertion

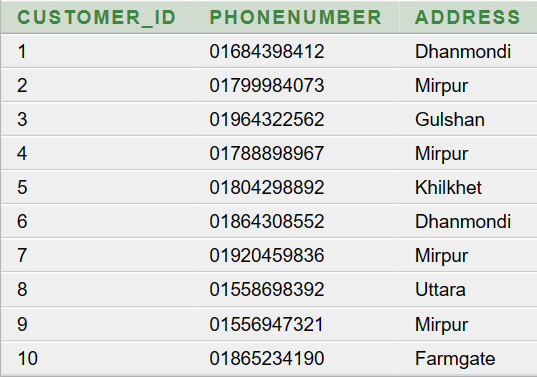




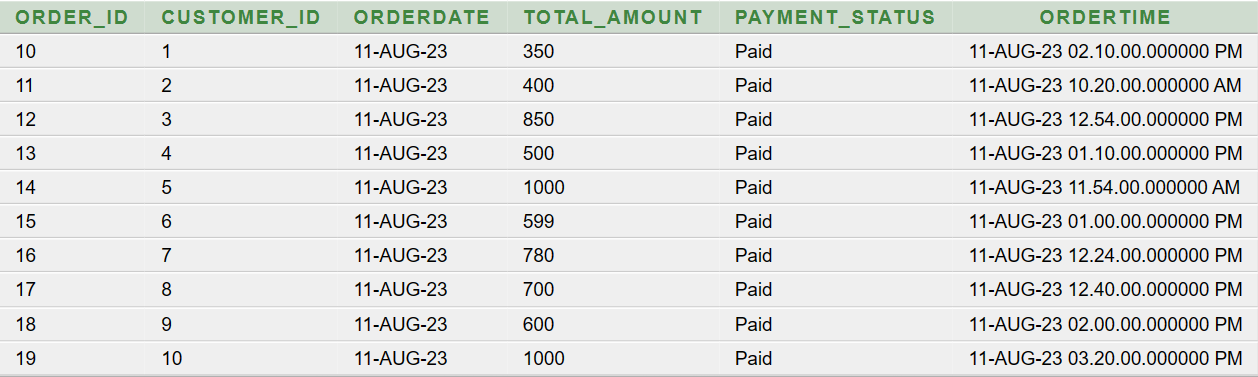
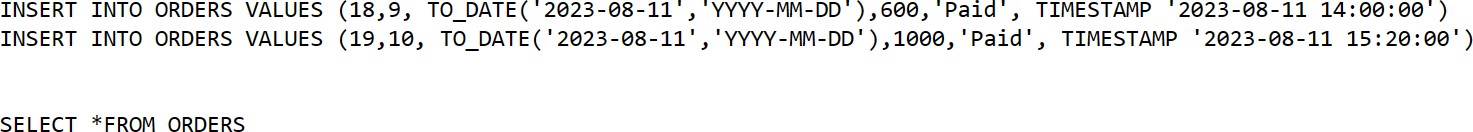
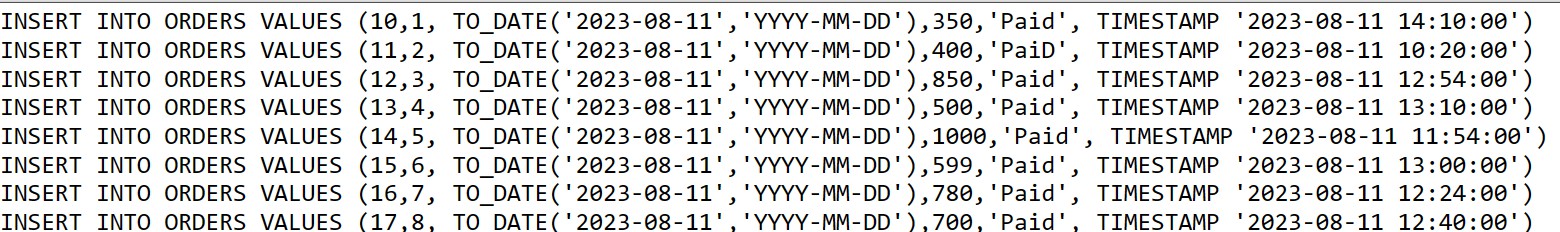


1. **Customerinfo Table:**

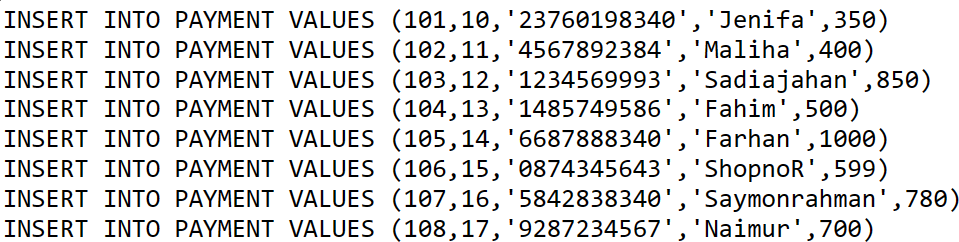


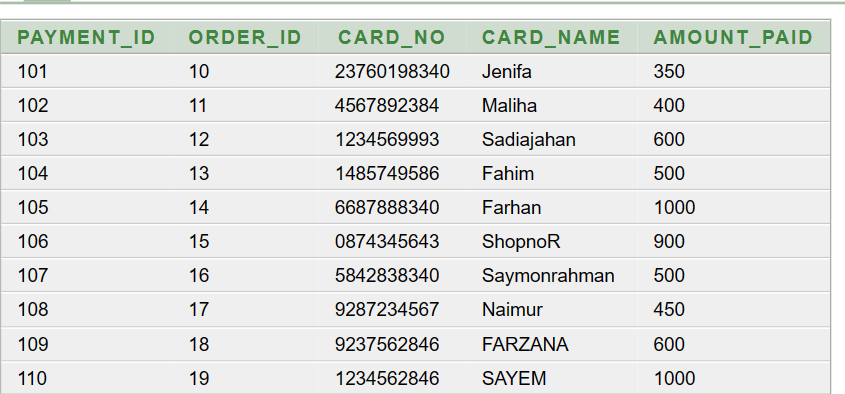
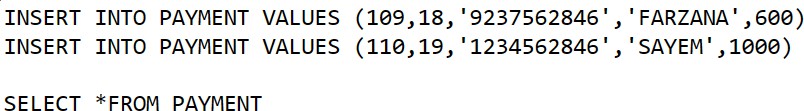


1. **Orders Table:**

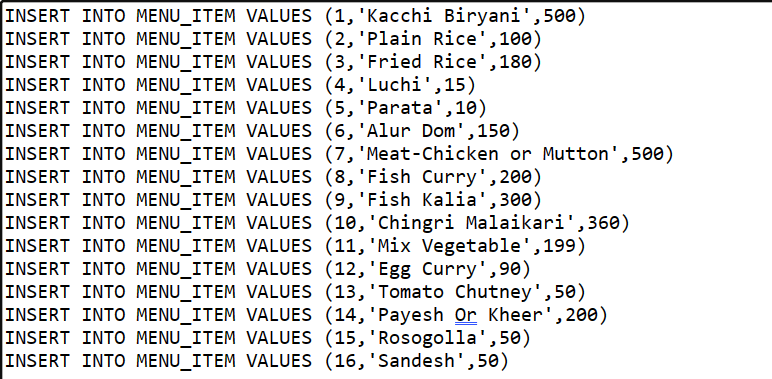


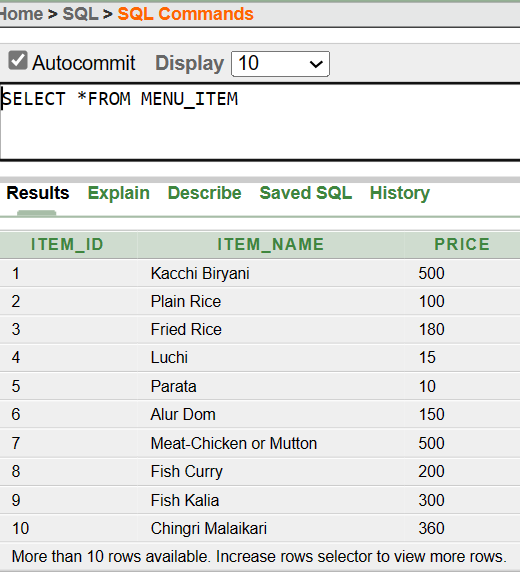
1. **Payment Table:**

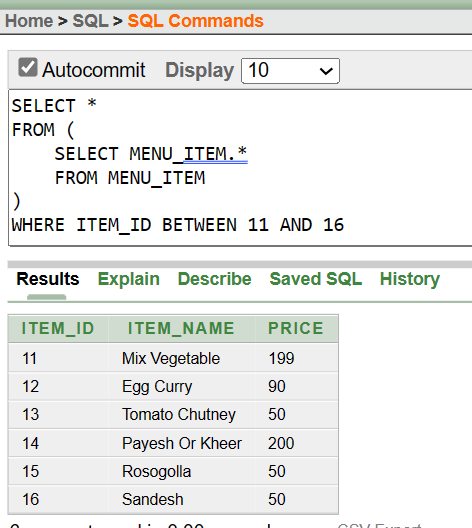




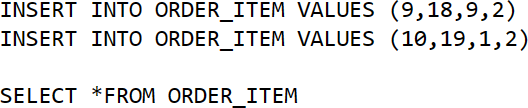
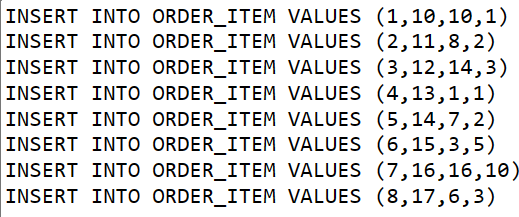
1. **Menu\_item Table:**

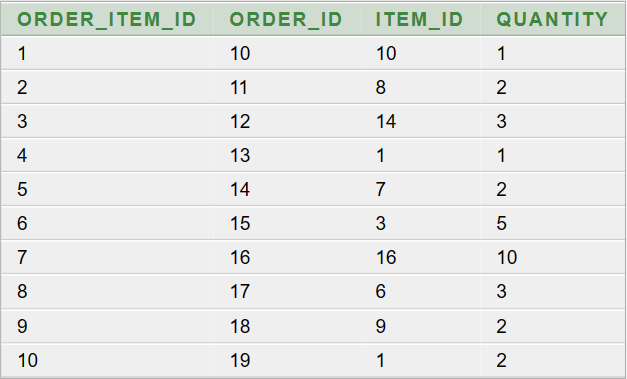




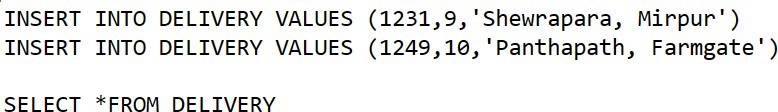
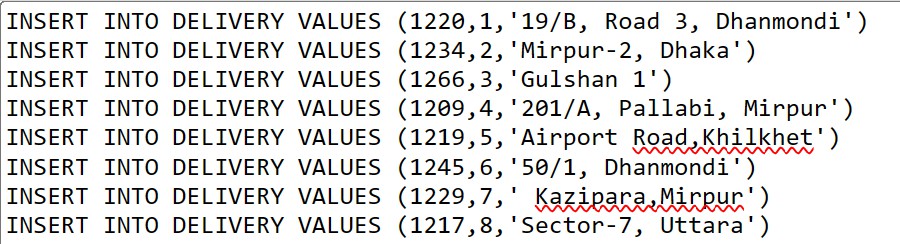


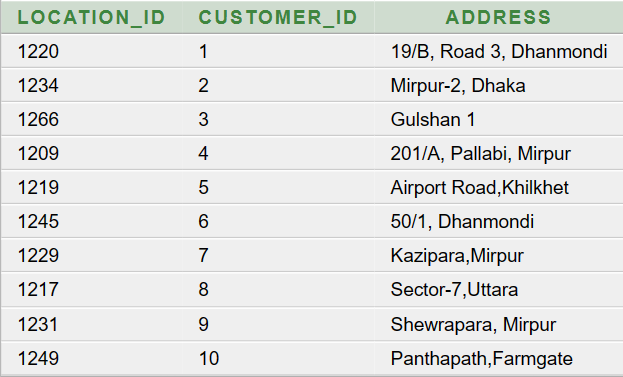
1. **Order\_Item Table:**





1. **Delivery Table:**

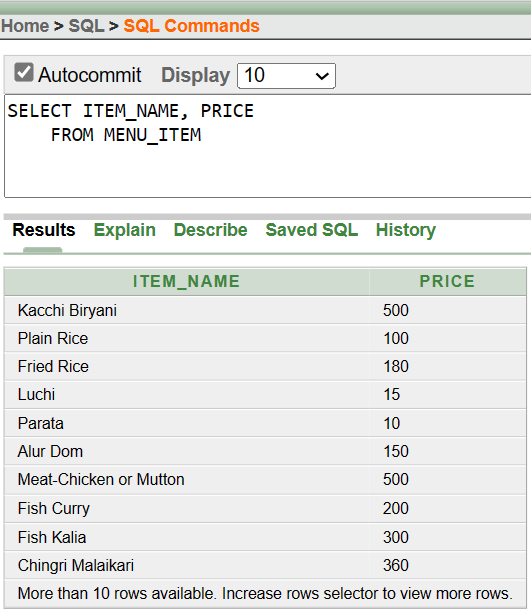




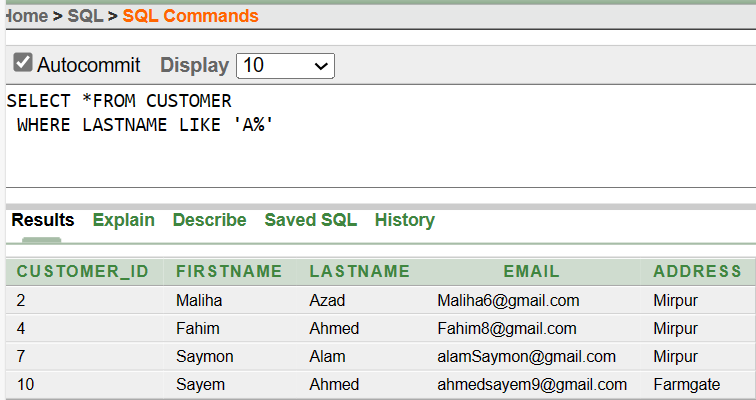
**Query Testing**

# Simple Query:

1. Display item name and price from menu item table.

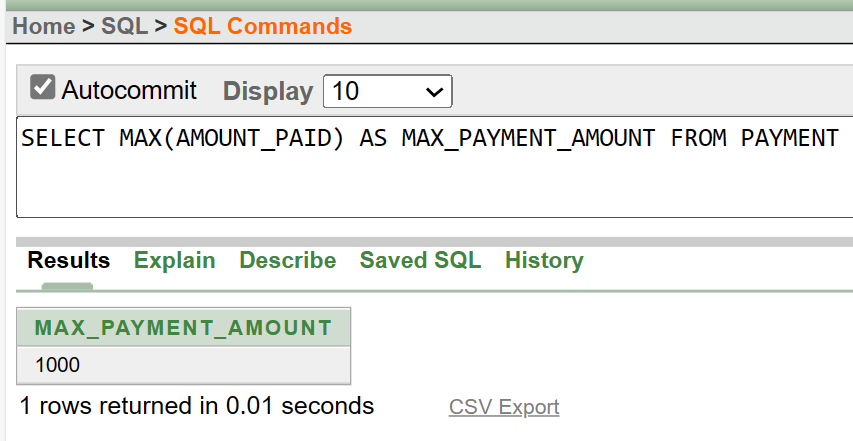


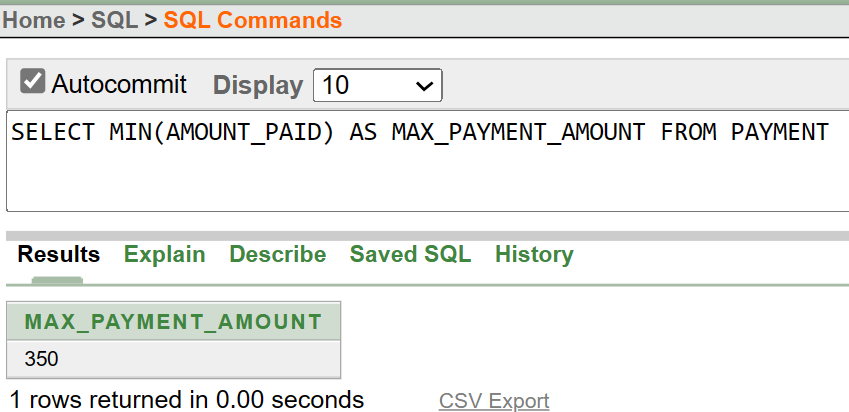
1. Display all customers whose last name start with A.

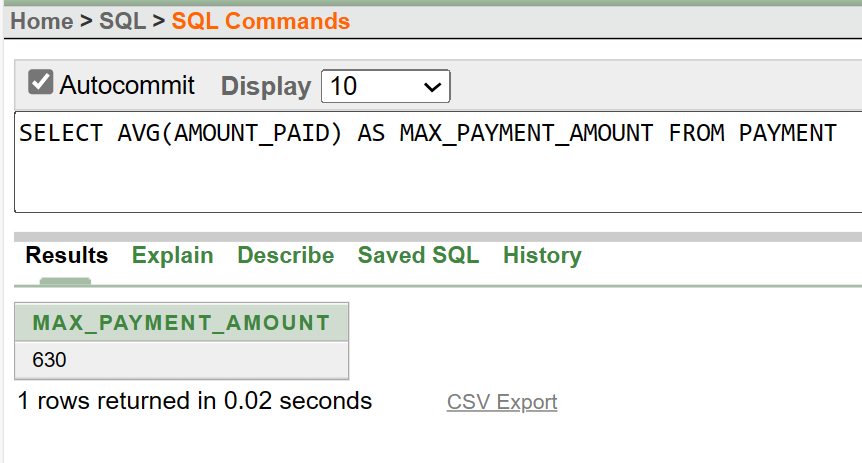


# Aggregate Query:

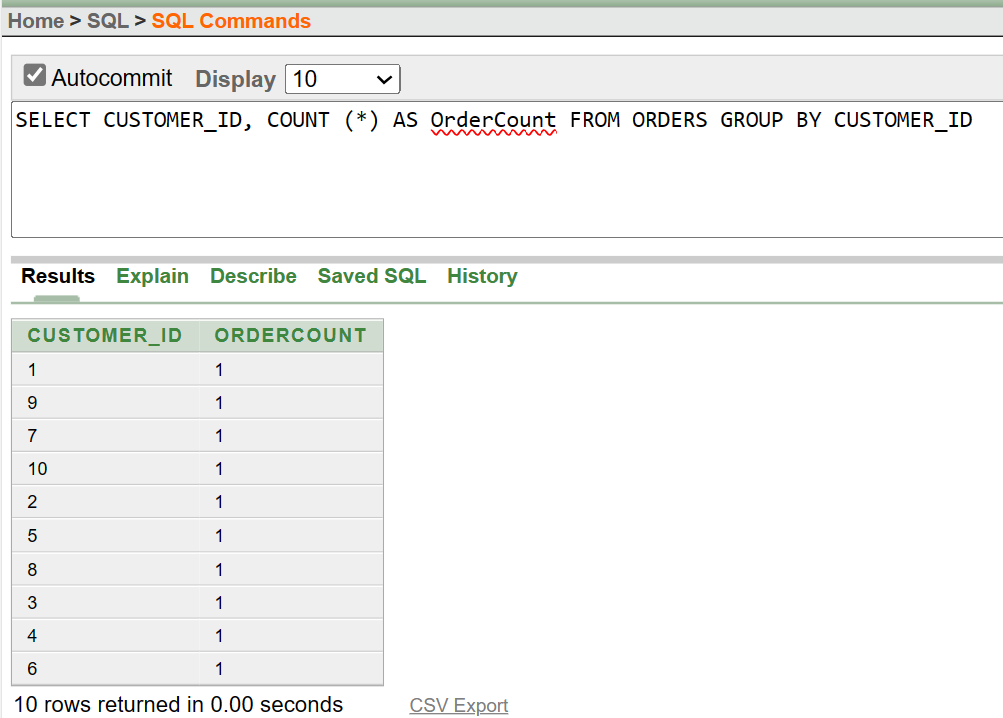
1. Display MAXIMUM, MINIMUM AND AVERAGE paid amount.





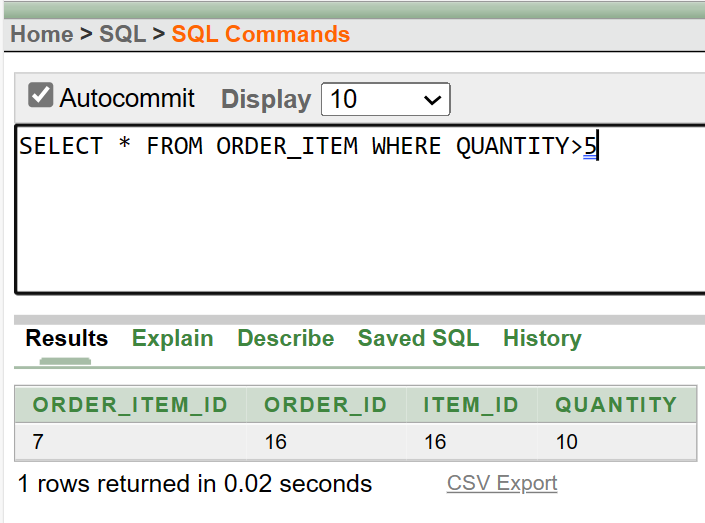


1. Display customer id, COUNT (\*) AS OrderCount from orders table using GROUP BY function:

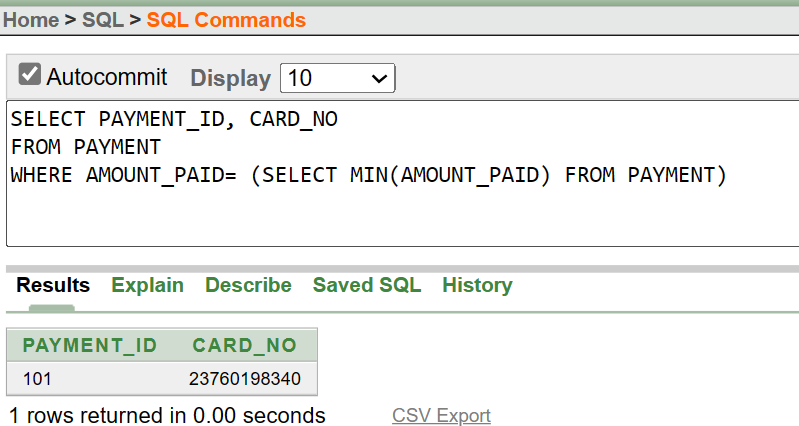


# Single-row Subquery:

1. Display all from order item table where quantity is greater than 5:

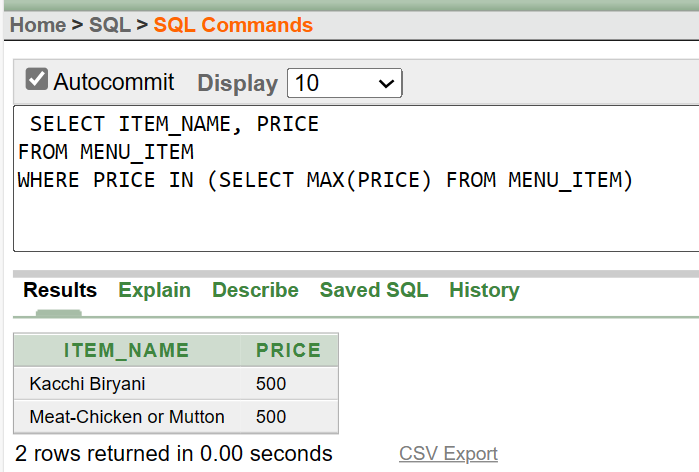


1. Display payment id, card no where lowest amount paid in payment table:

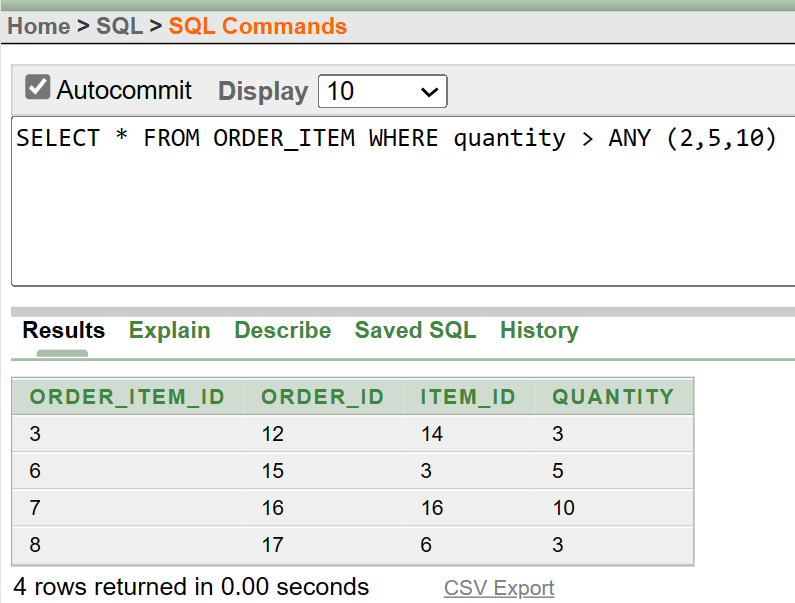


# Multiple-row Subquery:

1. Display the item name of highest price item from menu item table:

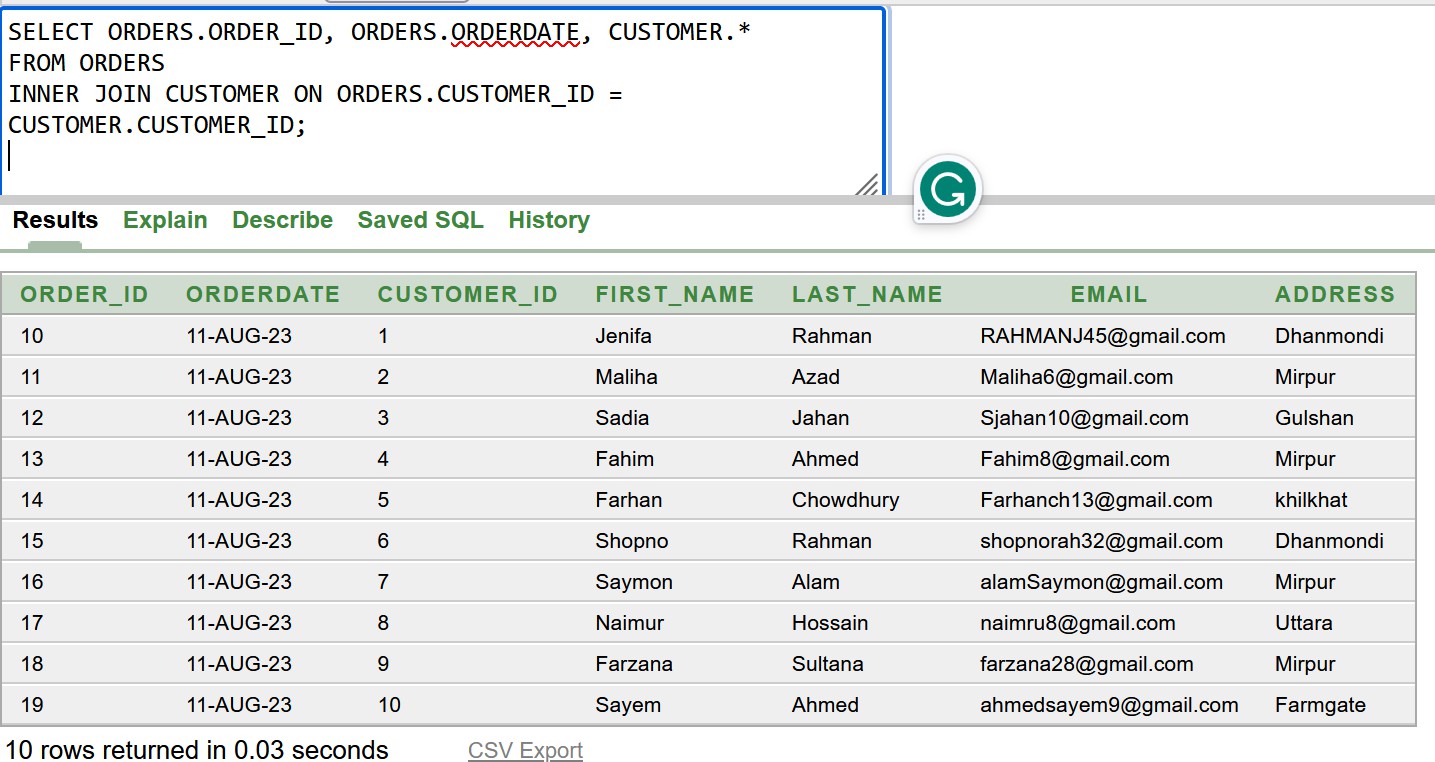


1. Display all from order item table using ANY operator in quantity column.

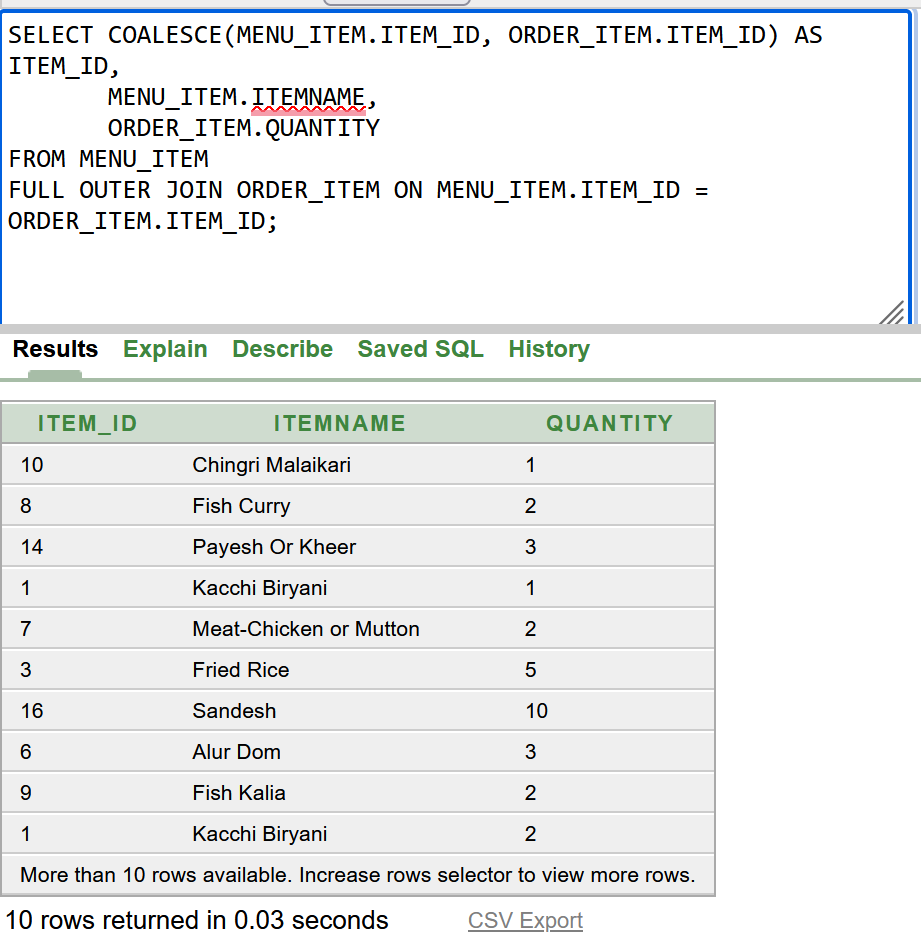


# Joining:

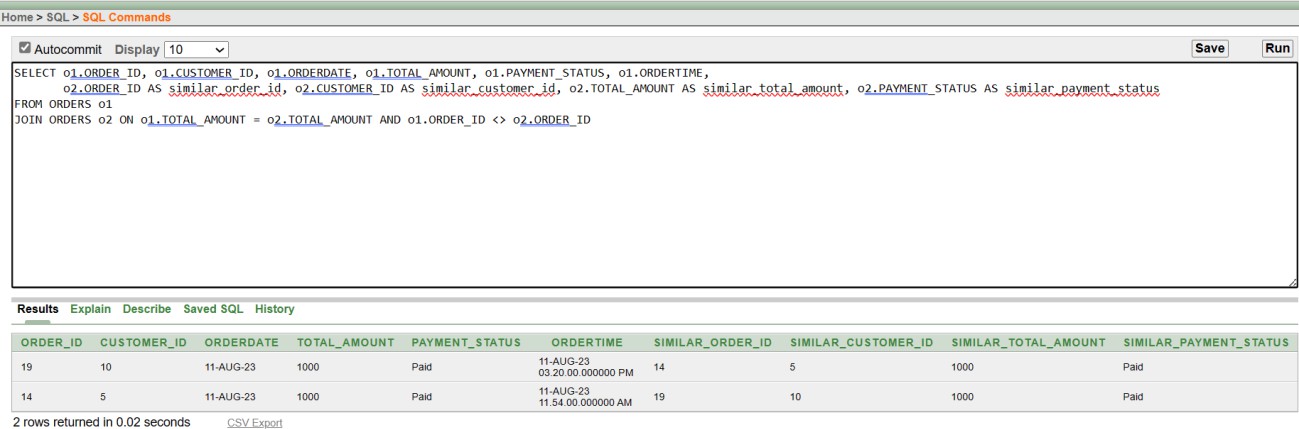
1. Display all the CUSTOMERINFO for all the ORDERS.



1. Get all the matching and nonmatching records from Menu Item and Order Item tables

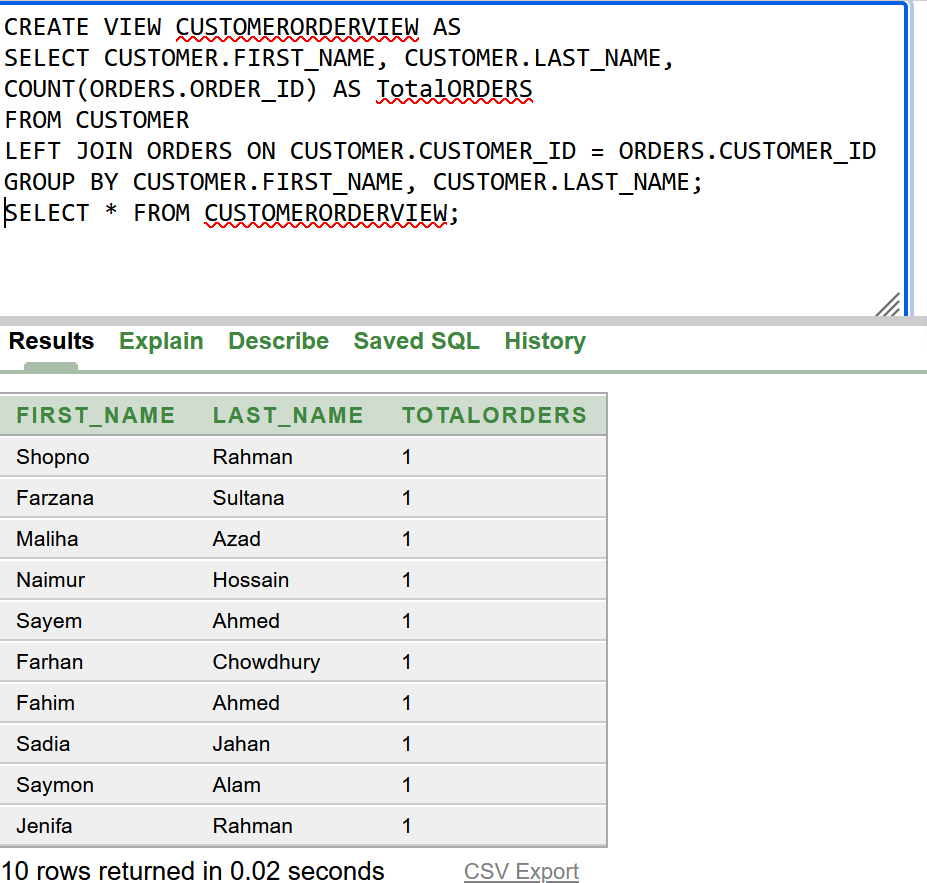


1. Display all the similar order id, customer id, total amount, payment status from orders table

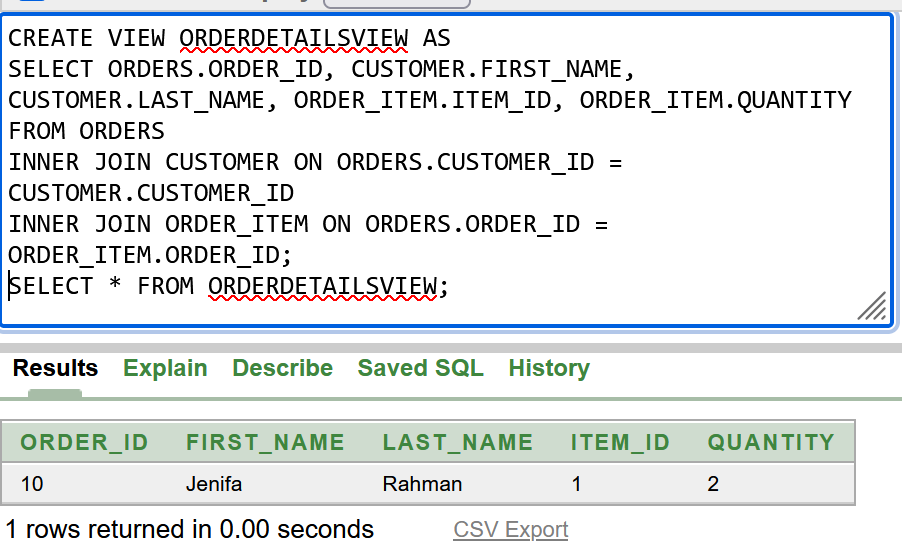


### Views:

* + 1. Create a view called CUSTOMERORDERBIEW based on first name ,last name and count the total orders from customer and orders table



* + 1. Create a view called ORDERDETAILSVIEW based on order id, first name, last name, item id, quantity using INNER JOIN from customer and orders table



# Database Connection

### Setting Up Java-MySQL Connection

* Getting the Connector/J:

To establish a connection between Java and MySQL, I acquired the essential Connector/J JAR file directly from the official MySQL website.

Navigating to "<https://dev.mysql.com/downloads/connector/j/>", I carefully selected the appropriate version tailored to the requirements of my project.

* Creating Database and Tables:
* In the MySQL Shell I smartly set up the database structure using short but effective SQL commands.

To make things easier, I kept a script with the DDL statements for reference.

* Setting Up in IntelliJ IDEA:

I tidied up my project in IntelliJ IDEA by neatly organizing its components. Through the "Project Structure" settings, I seamlessly integrated my Java code, resources, and external elements for a well-arranged setup.

* Strong Connection with Connector/J:

To make a solid connection, I added a new library to my project.

I picked the right Connector/J JAR file that's good at connecting Java and MySQL.

### Setting Up MySQL Connectivity

* Registering the Driver:

I began by registering the MySQL JDBC driver, a crucial step that enables my Java code to comprehend how to communicate effectively with the MySQL database.

* Connecting to the Database:

Following that, I set up a connection to the MySQL database. This connection serves as a secure pathway, allowing my Java code to seamlessly interact with the data.

* Creating a Statement:

With the connection in place, I moved onto create a statement object. This object is like a messenger that conveys my SQL queries to the database.

* Executing Queries and Obtaining Results:

I successfully executed my SQL queries using the statement object. This action led to the creation of a ResultSet, which held the necessary data retrieved from the database.

* Closing the Connection:

Upon completing the essential operations, I took care of proper resource management by closing both the statement and connection. This precautionary step prevents any unnecessary resource usage.

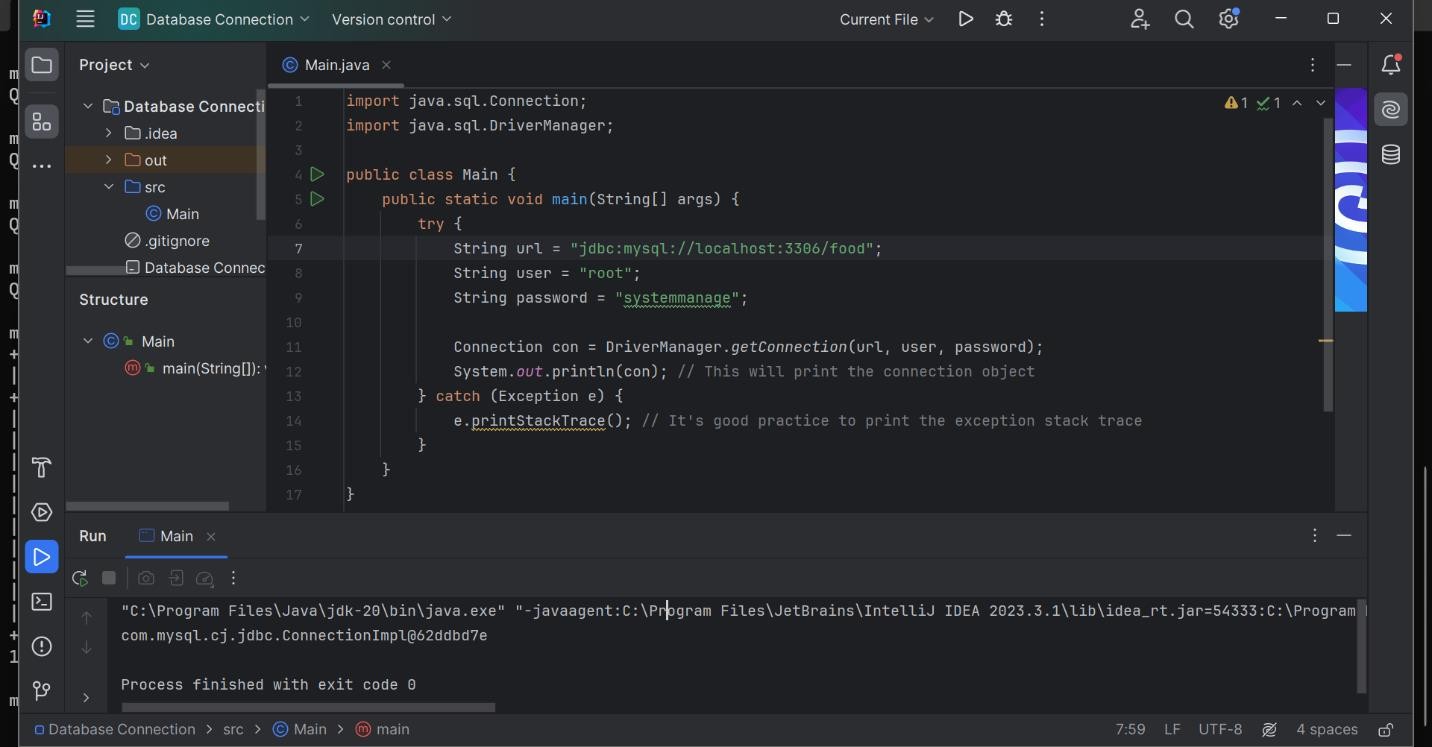


Fig 8: Java-MySQL connection using IntelliJ

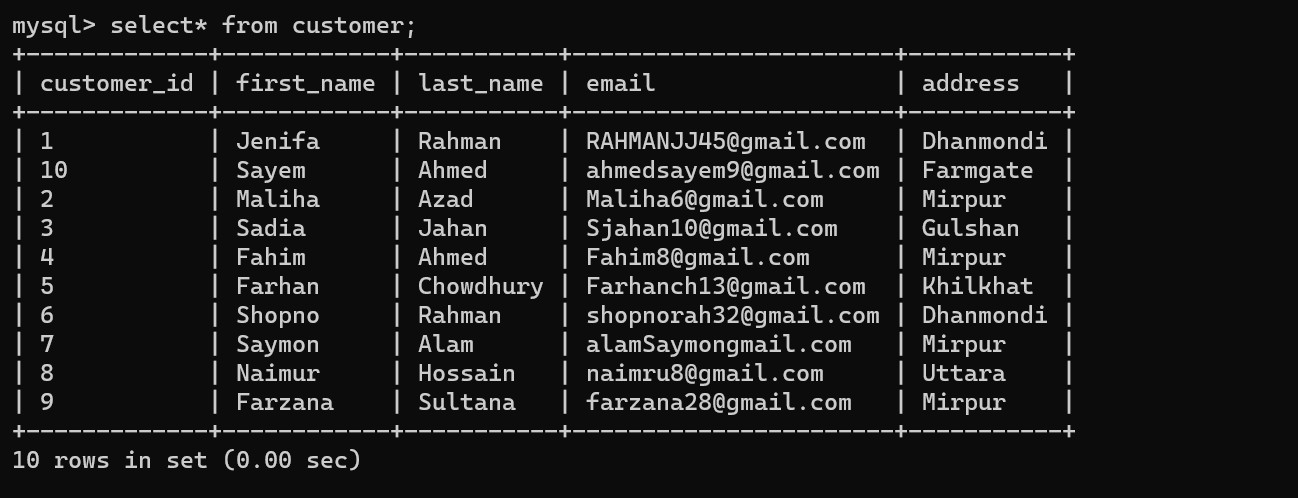


Fig 9: Customer table using mySQL

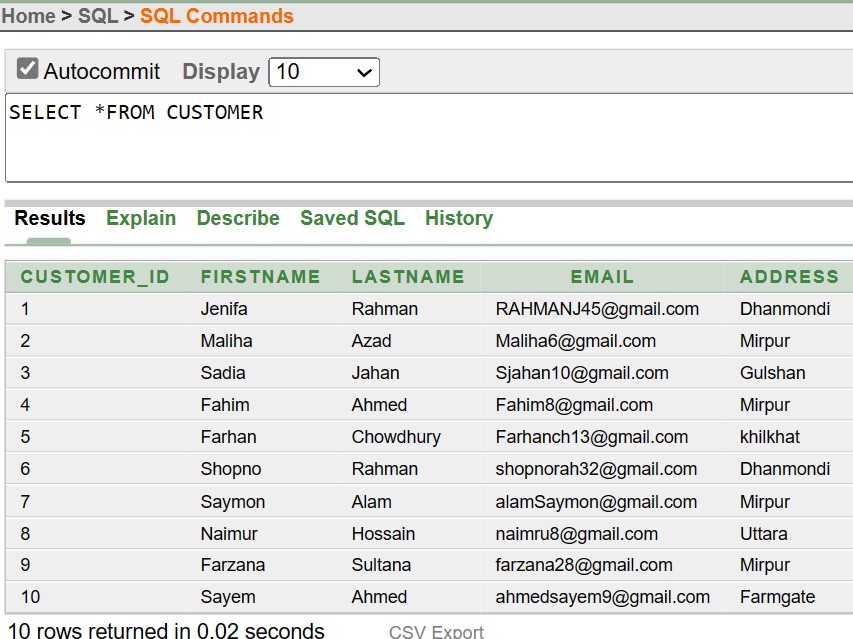


Fig 10: Customer table using Oracle SQL

**Conclusion**

Our Online Food Management System is a user-friendly app that lets you easily order delicious food. You can look at menus, choose what you want, pay online, and even track your order in real time as it's delivered to your door.

Future Plans: In the future, we want to make our system even better for you. We are planning to do things like suggesting meals you might like, creating mobile apps for phones, and helping restaurants manage their menus better.

Improvement Plans: To keep getting better, we will listen to your feedback and make things faster and safer. We will also work with other companies to make payments easier and help you find the best food. And we will do all of this while thinking about the environment and making sure we're doing good for our planet.