

UBER EATS DATABASE

*A
Report
Submitted in partial fulfilment of the*

***BE IV SEMESTER DATABASE MANAGEMENT
SYSTEM LAB***

INFORMATION TECHNOLOGY

By

Gollapalli Sai Madhu Samhita <1602-18-737-095>

Under the Guidance of

B. Leelavathy



Department of Information Technology

Vasavi College of Engineering (Autonomous)

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad-31

2020

BONAFIDE CERTIFICATE

This to Certify that the project report titled "**Uber Eats database management system**" project work of **Miss Gollpalli Sai Madhu Samhita** bearing Roll.no:**1602-18-737-095** who carried out this project under my supervision in the IV semester for the academic year 2019-2020.

Signature

external examiner

Signature

internal examiner

B. LEELAVATHY

Assistant Professor

Department of Information Technology

ABSTRACT

Different varieties of food have a growing demand these days. People want to enjoy different cuisines all over the world. But with increase of restaurants day-by-day dining out or takeaway is a difficult choice. An online food ordering system like “Uber Eats” shows an easy way out by bringing food to your doorstep. Customers can order food from any place and at any time provided network connection is available. “Uber Eats” provides customers with a variety of restaurants to order from. Various details of restaurant are given, like rating and food menu, making the choice of customer easy. Live tracking of order is provided. Apart from this, refund is provided when the correct order is not delivered or when the customer is not satisfied with the food. “Uber Eats” is the best choice for people looking for good food.

“Good food equals good mood”

REQUIREMENT ANALYSIS

List of tables:

- *Restaurant Details*
- *Customer Details*
- *Reservation*
- *Order Details*
- *Orders*
- *Payment*
- *Pays*
- *Order From*
- *Contains*
- *Reserve In*
- *Reserves*
- *Order By*

List of attributes with their domain types:

- *Customer*
 1. Customer Id – varchar (Primary key)
 2. Password - varchar
 3. Gmail account – varchar
 4. Name-char
 5. Phone number - Number
 6. Address – varchar

- *Uber Eats*

1. Opening and Closing Time – Time
2. Location – varchar
3. Food Item – char
4. Cost – Number
5. Restaurant Id – varchar (Primary key)

- *Order Details*

1. Location – varchar
2. Price – Number
3. Time of Delivery – Time
4. Order Id – Number (Primary Key)

- *Payment*

1. Date – date
2. Time – time
3. Type – varchar
4. Cash – Number
5. Transaction Id – Number (Primary Key)

- *Orders*

1. Order Id – varchar (Foreign key)
2. Customer Id – varchar (Foreign key)

- *Generates*

1. Order Id – varchar (Foreign key)

2. Transaction Id – varchar (Foreign key)

3

- *Order From*

1. Restaurant Id – varchar (Foreign key)
2. Customer Id – varchar (Foreign key)

- *Pays*

1. Customer Id – varchar2(Foreign key)
2. Transaction Id – varchar(Foreign key)

SPECIFIC GOAL OF THE PROJECT:

The main goal of this project is to provide an online based food ordering system, which ensures home delivery of food, chosen from a wide variety of restaurants. Details of different kind of restaurants are shown with their food menus. This allows the customer to choose food of their choice by sitting at home. An order of food from a specific restaurant can be placed via the Uber Eats Database. Payment can be done through different modes and order would be delivered to the required location.

SQL particular - Uber Eats, Customer, Order, Payment methods.

➤ **Architecture and technology used:**

SQL Plus is the most basic Oracle Database utility with a basic command-line interface, commonly used by users, administrators and programmers.

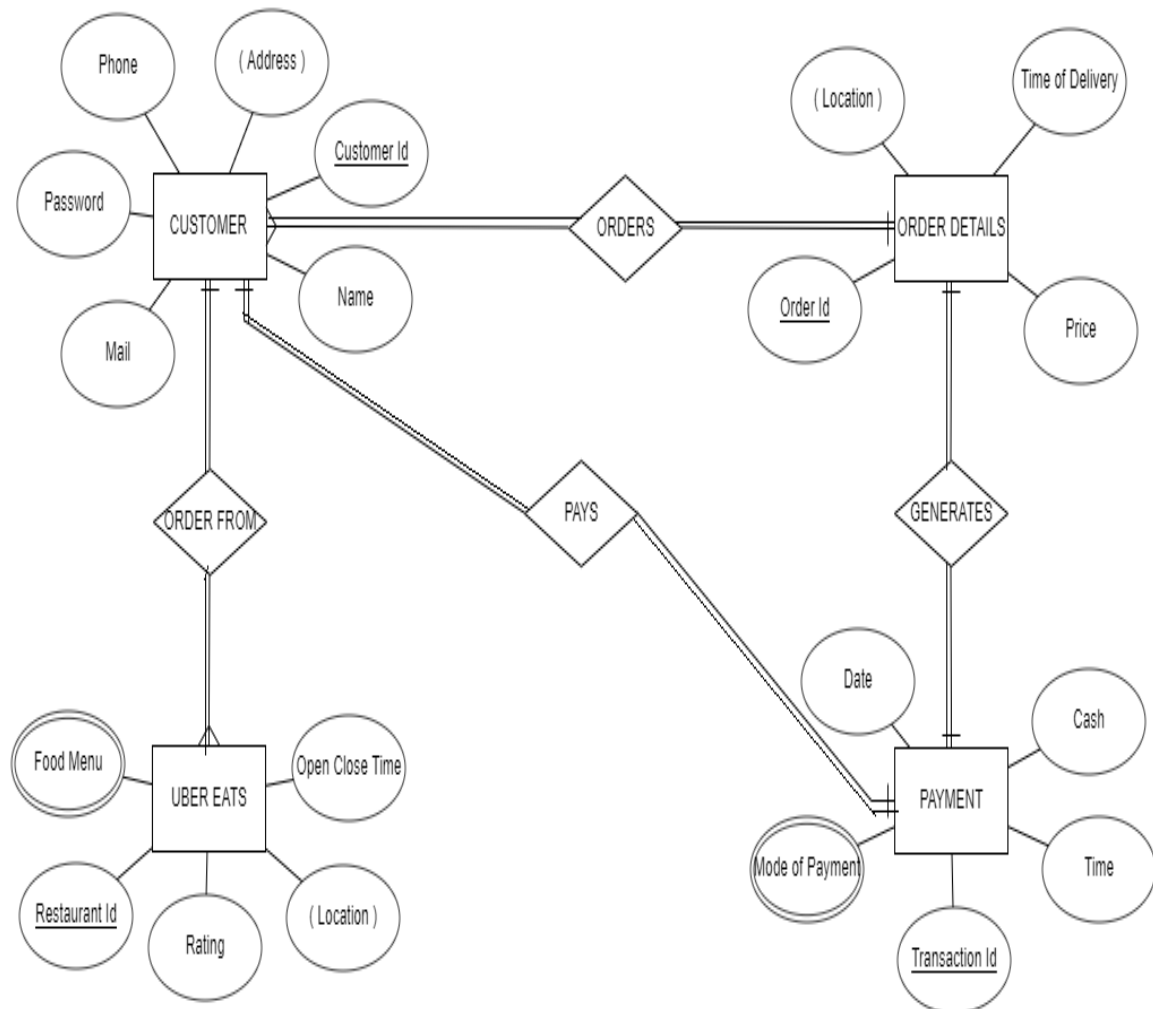
The interface of SQL Plus is used for creating the database. DDL and DML commands are implemented for operations being executed. The details of various Online MOOC's provider, courses, student, assignments, and results are stored in the form of tables in the database.

Eclipse is an integrated development environment (IDE) used in computer programming. It contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse is written mostly in java and its primary use is for developing Java applications, but it may also be used to develop applications in other programming languages via plug-ins, including Erlang, Java Scripts etc.

The front-end application code is written in "**Java**" using Eclipse. The portal for front end application is designed through Eclipse, runs and has the capacity to connect with the database which has data inserted using SQL.

DESIGN

ER DIAGRAM



Mapping Cardinalities and Constraints

- Customer(many) Order from Uber Eats(one)
One Customer can place an order from one Restaurant, but
One Restaurant can receive orders from many Restaurants.
- Customer(one) Orders Order Details(many)
One Customer can place many orders, but one order is places
by one Customer.
- Order Details(one) Generates Payment(one)
One Order generates one bill and one bill is generated by one
Order.
- Customer(one) Pays Payment(one)
One Customer can make one Payment regarding one order and
one Payment is made by only one Customer regarding one
order.

DDL COMMANDS

```
SQL> create table Customer(  
2 Cid varchar2(20),  
3 Password varchar2(16),  
4 Mail varchar2(16),  
5 Name char(20),  
6 Address varchar2(50),  
7 Phone number(12));
```

Table created.

```
SQL> create table UberEats(  
2 OpenCloseTime number(10),  
3 Location varchar2(50),  
4 Rating number(5),  
5 Rid varchar2(20),  
6 FoodMenu varchar2(20));
```

Table created.

```
SQL> create table OrderDetails(  
2 Location varchar2(50),  
3 Price number(10),  
4 Time number(10),  
5 Oid number(20));
```

Table created.

```
SQL> create table Payment(  
2 Dt date,  
3 Tm varchar2(7),  
4 Type varchar2(20),  
5 Cash number(6),  
6 Tid number(20));
```

Table created.

```
SQL> create table OrderFroms(  
2 Cid varchar2(20),  
3 Rid varchar2(20));
```

Table created.

DBMS ASSIGNMENT -1
UBER EATS DATABASE

```
SQL> create table Orders(  
2  Oid number(10),  
3  Cid varchar2(20));
```

Table created.

```
SQL> create table Pays(  
2  Cid varchar2(20),  
3  Tid number(20));
```

Table created.

```
SQL> create table Generates(  
2  Oid number(20),  
3  Tid number(20));
```

Table created.

```
SQL> alter table Customer add primary key(Cid);
```

Table altered.

```
SQL> alter table UberEats add primary key(Rid);
```

Table altered.

```
SQL> alter table Payment add primary key(Tid);
```

Table altered.

```
SQL> alter table OrderDetails add primary key(Oid);
```

Table altered.

```
SQL> alter table Pays add foreign key(Cid) references Customer;
```

Table altered.

```
SQL> alter table Pays add foreign key(Tid) references Payment;
```

Table altered.

```
SQL> alter table OrderFrom add foreign key(Cid) references Customer;
```

Table altered.

```
SQL> alter table OrderFrom add foreign key(Rid) references UberEats;
```

Table altered.

```
SQL> alter table Orders add foreign key(Cid) references Customer;
```

Table altered.

```
SQL> alter table Orders add foreign key(Oid) references OrderDetails;
```

Table altered.

```
SQL> alter table Generates add foreign key(Oid) references OrderDetails;
```

Table altered.

```
SQL> alter table Generates add foreign key(Tid) references Payment;
```

Table altered.

DBMS ASSIGNMENT -1
UBER EATS DATABASE

```
Run SQL Command Line

SQL> desc OrderDetails;
Name                                     Null?   Type
-----
LOCATION                                VARCHAR2(50)
PRICE                                NUMBER(10)
TIME                                NUMBER(10)
OID                                  NOT NULL NUMBER(20)

SQL> desc Payment;
Name                                     Null?   Type
-----
DT                                     DATE
TM                                     VARCHAR2(7)
TYPE                                VARCHAR2(20)
CASH                                NUMBER(6)
TID                                  NOT NULL NUMBER(20)

SQL> desc Customer;
Name                                     Null?   Type
-----
CID                                  NOT NULL VARCHAR2(20)
PASSWORD                            VARCHAR2(16)
MAIL                                VARCHAR2(16)
NAME                                CHAR(20)
ADDRESS                            VARCHAR2(50)
PHONE                                NUMBER(12)

SQL> desc UberEats;
Name                                     Null?   Type
-----
OPENCLOSETIME                        NUMBER(10)
LOCATION                                VARCHAR2(50)
RATING                              NUMBER(5)
RID                                  NOT NULL VARCHAR2(20)
FOODMENU                            VARCHAR2(20)
```

```
SQL> desc Pays;
```

Name	Null?	Type
CID		VARCHAR2(20)
TID		NUMBER(20)

```
SQL> desc Generates;
```

Name	Null?	Type
OID		NUMBER(20)
TID		NUMBER(20)

```
SQL> desc OrderFrom;
```

Name	Null?	Type
CID		VARCHAR2(20)
RID		VARCHAR2(20)

```
SQL> desc Orders;
```

Name	Null?	Type
OID		NUMBER(10)
CID		VARCHAR2(20)

```
SQL> █
```

DML COMMANDS

```
Run SQL Command Line

SQL> select * from UberEats;

OPENCLOSETIME LOCATION                                RATING
-----
RID            FOODMENU
-----
10 uppal              7
345              Biryani
12 tarnaka            6
1234              Kebab
11 lakdikapol        9
567              Pizza

OPENCLOSETIME LOCATION                                RATING
-----
RID            FOODMENU
-----
7 begumpet          8
002              Burger
12 mehdipatnam      5
148              Sandwich

SQL> select * from OrderFrom;

CID            RID
-----
576            345
9554           1234
123            567
737            002
001            148

SQL> _
```

```
SQL> select * from Customer;
```

CID	PASSWORD	MAIL	NAME
576	swert	samhita123	samhita
habsiguda			6303775736
9554	traffic	raghu34	raghu
kphb			8764523456
123	redflog	manasa56	manasa
gachibowli			7331109369

CID	PASSWORD	MAIL	NAME
737	great2	vamsi2345	vamsi
kukatpally			9948366219
001	forguветrt5	mohit73	mohit
uppal			9441109369


```
SQL> select * from Orders;
```

OID	CID
1	001
12	123
46	576
56	737
123	9554


```
SQL> select * from Payment;
```

DT	TM	TYPE	CASH	TID
11-JAN-20	3pm	cash	90	45
20-SEP-19	4pm	creditcard	500	7
18-OCT-20	8pm	debitcard	450	34
08-JUL-20	9pm	netbanking	750	33
21-JAN-20	4pm	cash	560	11

```
SQL> select * from OrderDetails;
```

LOCATION	PRICE	TIME
OID		
Narayanaguda 56	56	3

himayath nagar 123	45	4
vidyanagar 12	100	7

LOCATION	PRICE	TIME
OID		
amberpet 46	34	5
ameerpet 1	300	7

```
SQL> 
```

```
SQL> Run SQL Command Line
1 row created.

SQL> select * from Pays;

CID                                TID
-----
576                                45
9554                               7
123                                34
737                                33
001                                11

SQL> select * from Generates;

      OID      TID
-----
      1        7
     12       11
     46       33
     56       34
    123       45

SQL>
```

IMPLEMENTATION

Front End Programs:

1) Insert Customer-

```
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
public class InsertCustomer extends Panel
{
    Button insertCustomerButton;
    TextField cidText, cnameText, addressText,
    mailText,passwordText,phoneText;
    TextArea errorText;
    Connection connection;
    Statement statement;
    public InsertCustomer()
    {
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");
        }
        catch (Exception e)
        {
            System.err.println("Unable to find and load
driver");
            System.exit(1);
        }
        connectToDB();
    }
}
```

```
    }

    public void connectToDB()
    {
        try
        {
            connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:15
21:xe","system","OracleDBMS2090&");
            statement = connection.createStatement();

        }
        catch (SQLException connectException)
        {

System.out.println(connectException.getMessage());

System.out.println(connectException.getSQLState());

System.out.println(connectException.getErrorCode());
            System.exit(1);
        }
    }

    public void buildGUI()
    {
        insertCustomerButton = new Button("Submit");
        insertCustomerButton.addActionListener(new
ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
```

```
        try
        {
            Statement statement =
connection.createStatement();

            String query= "INSERT INTO Customer
VALUES('"+ cidText.getText() + "', " + "'" +
passwordText.getText() + "'," + "'" + mailText.getText() +
"',"+ "'" +cnameText.getText()+"',"
+"'" +addressText.getText()+"'," +phoneText.getText()+")";
            int i = statement.executeUpdate(query);
            errorText.append("\nInserted " + i + "
rows successfully");
        }
        catch (SQLException insertException)
        {
            displaySQLErrors(insertException);
        }
    }
});
```

```
        cnameText = new TextField(15);
        cidText = new TextField(15);
        addressText = new TextField(15);
        mailText = new TextField(15);
        passwordText = new TextField(15);
        phoneText = new TextField(15);
```

```
errorText = new TextArea(10,40);  
errorText.setEditable(false);
```

```
Panel first = new Panel();  
first.setLayout(new GridLayout(6,2));  
first.add(new Label("Customer ID:"));  
first.add(cidText);  
first.add(new Label("Name:"));  
first.add(cnameText);  
first.add(new Label("Address:"));  
first.add(addressText);  
first.add(new Label("Mail"));  
first.add(mailText);  
first.add(new Label("Password:"));  
first.add(passwordText);  
first.add(new Label("Phone:"));  
first.add(phoneText);
```

```
first.setBounds(125,90,300,150);
```

```
Panel second = new Panel(new GridLayout(4, 1));  
second.add(insertCustomerButton);  
second.setBounds(195,290,150,100);
```

```
Panel third = new Panel();  
third.add(errorText);  
third.setBounds(80,410,430,300);  
setLayout(null);
```

```
add(first);
```

```
        add(second);
        add(third);

        setSize(500,600);
        setVisible(true);
    }

    private void displaySQLExceptions(SQLException e)
    {
        errorText.append("\nSQLException: " +
e.getMessage() + "\n");
        errorText.append("SQLState:  " + e.getSQLState() +
"\n");
        errorText.append("VendorError: " +
e.getErrorCode() + "\n");
    }

    public static void main(String[] args)
    {
        InsertCustomer incus = new InsertCustomer();

        incus.buildGUI();
    }
}
```

2)Delete Customer-

```
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
public class DeleteCustomer extends Panel
{
    Button deleteCustomerButton;
    List CustomerIDList;
    TextField cidText, cnameText, mailText,
passwordText,addressText,phoneText;
    TextArea errorText;
    Connection connection;
    Statement statement;
    ResultSet rs;

    public DeleteCustomer()
    {
        try
        {

            Class.forName("oracle.jdbc.driver.OracleDriver");
        }
        catch (Exception e)
        {
            System.err.println("Unable to find and load
driver");
            System.exit(1);
        }
        connectToDB();
    }

    public void connectToDB()
```



```
{  
    try  
    {  
        connection =  
DriverManager.getConnection("jdbc:oracle:thin:@localhost:15  
21:xe","system","OracleDBMS2090&");  
        statement = connection.createStatement();  
    }  
    catch (SQLException connectException)  
    {  
  
System.out.println(connectException.getMessage());  
  
System.out.println(connectException.getSQLState());  
  
System.out.println(connectException.getErrorCode());  
        System.exit(1);  
    }  
}  
  
private void loadCustomer()  
{  
    try  
    {  
        rs = statement.executeQuery("SELECT * FROM  
Customer");  
        while (rs.next())  
        {  
            CustomerIDList.add(rs.getString("CID"));  
        }  
    }  
}
```

```
        }
        catch (SQLException e)
        {
            displaySQLErrors(e);
        }
    }

    public void buildGUI()
    {
        CustomerIDList = new List(10);
        loadCustomer();
        add(CustomerIDList);

        //When a list item is selected populate the text
fields
        CustomerIDList.addItemListener(new ItemListener()
        {
            public void itemStateChanged(ItemEvent e)
            {
                try
                {
                    rs =
statement.executeQuery("SELECT * FROM Customer");
                    while (rs.next())
                    {
                        if
(rs.getString("CID").equals(CustomerIDList.getSelectedItem()))
                            break;
                    }
                    if (!rs.isAfterLast())
                    {
```

```
        cidText.setText(rs.getString("CID"));

        passwordText.setText(rs.getString("Password"));

        mailText.setText(rs.getString("Mail"));

        cnameText.setText(rs.getString("Name"));

        addressText.setText(rs.getString("Address"));

        phoneText.setText(rs.getString("Phone"));
    }
}
catch (SQLException selectException)
{
    displaySQLErrors(selectException);
}
});

deleteCustomerButton = new Button("Delete");
deleteCustomerButton.addActionListener(new
ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        try
        {
```

```
Statement statement =  
connection.createStatement();  
int i =  
statement.executeUpdate("DELETE FROM Customer WHERE  
CID = ""  
+  
CustomerIDList.getSelectedItem()+"");  
errorText.append("\nDeleted " + i +  
" rows successfully");  
cidText.setText(null);  
passwordText.setText(null);  
mailText.setText(null);  
cnameText.setText(null);  
addressText.setText(null);  
phoneText.setText(null);  
CustomerIDList.removeAll();  
loadCustomer();  
}  
catch (SQLException insertException)  
{  
    displaySQLErrors(insertException);  
}  
}  
});  
  
cidText = new TextField(15);  
cnameText = new TextField(15);  
mailText = new TextField(15);  
passwordText = new TextField(15);  
addressText= new TextField(15);  
phoneText= new TextField(15);
```

```
errorText = new TextArea(10, 40);  
errorText.setEditable(false);
```

```
Panel first = new Panel();  
first.setLayout(new GridLayout(6, 1));  
first.add(new Label("Customer ID:"));  
first.add(cidText);  
cidText.setEditable(false);  
first.add(new Label("Name:"));  
first.add(cnameText);  
cnameText.setEditable(false);  
first.add(new Label("Mail:"));  
first.add(mailText);  
mailText.setEditable(false);  
first.add(new Label("Password:"));  
first.add(passwordText);  
passwordText.setEditable(false);  
first.add(new Label("Address:"));  
first.add(addressText);  
addressText.setEditable(false);  
first.add(new Label("Phones:"));  
first.add(phoneText);  
phoneText.setEditable(false);
```

```
Panel second = new Panel(new GridLayout(4, 1));  
second.add(deleteCustomerButton);
```

```
Panel third = new Panel();
```

```
        third.add(errorText);

        add(first);
        add(second);
        add(third);

        setSize(450, 600);
        setLayout(new FlowLayout());
        setVisible(true);

    }

    private void displaySQLExceptions(SQLException e)
    {
        errorText.append("\nSQLException: " +
e.getMessage() + "\n");
        errorText.append("SQLState:  " + e.getSQLState() +
"\n");
        errorText.append("VendorError: " +
e.getErrorCode() + "\n");
    }

    public static void main(String[] args)
    {
        DeleteCustomer delcus = new DeleteCustomer();
        delcus.buildGUI();
    }
}
```

```
    }  
}
```

3)Update Customer-

```
import java.awt.*;  
import java.awt.event.*;  
import java.sql.*;  
public class UpdateCustomer extends Panel  
{  
    Button updateCustomerButton;  
    List CustomerIDList;  
    TextField cidText,cnameText, mailText,  
passwordText,addressText,phoneText;  
    TextArea errorText;  
    Connection connection;  
    Statement statement;  
    ResultSet rs;  
  
    public UpdateCustomer()  
  
    {  
        try  
        {  
  
            Class.forName("oracle.jdbc.driver.OracleDriver");  
        }  
        catch (Exception e)  
        {  
            System.err.println("Unable to find and load  
driver");  
            System.exit(1);  
        }  
    }  
}
```

```
        }
        connectToDB();
    }

    public void connectToDB()
    {
        try
        {
            connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:15
21:xe","system","OracleDBMS2090&");
            statement = connection.createStatement();

        }
        catch (SQLException connectException)
        {

System.out.println(connectException.getMessage());

System.out.println(connectException.getSQLState());

System.out.println(connectException.getErrorCode());
            System.exit(1);
        }
    }

    private void loadCustomer()
    {
        try
        {
```



```
        rs = statement.executeQuery("SELECT CID FROM
Customer");
        while (rs.next())
        {
            CustomerIDList.add(rs.getString("CID"));
        }
    }
    catch (SQLException e)
    {
        displaySQLErrors(e);
    }
}

public void buildGUI()
{
    CustomerIDList = new List(10);
    loadCustomer();
    add(CustomerIDList);

    //When a list item is selected populate the text
fields
    CustomerIDList.addItemListener(new ItemListener()
    {
        public void itemStateChanged(ItemEvent e)
        {
            try
            {
                rs = statement.executeQuery("SELECT *
FROM Customer");
                while (rs.next())
                {
```

```
                if
(rs.getString("CID").equals(CustomerIDList.getSelectedItem()))
                    break;
            }
            if (!rs.isAfterLast())
            {
                cidText.setText(rs.getString("CID"));

                passwordText.setText(rs.getString("Password"));

                mailText.setText(rs.getString("Mail"));

                cnameText.setText(rs.getString("Name"));

                addressText.setText(rs.getString("Address"));

                phoneText.setText(rs.getString("Phone"));
            }
        }
        catch (SQLException selectException)
        {
            displaySQLErrors(selectException);
        }
    }
});

updateCustomerButton = new Button("Modify");
updateCustomerButton.addActionListener(new
ActionListener()
{
```

```
        public void actionPerformed(ActionEvent e)
        {
            try
            {
                Statement statement =
connection.createStatement();
                int i =
statement.executeUpdate("UPDATE Customer "
                        + "SET password='" +
passwordText.getText() + "', "
                        + "mail='" + mailText.getText() + "',
"
                        + "name ='"+
cnameText.getText()+"',"
                        + "address ='"+
addressText.getText()+"',"
                        + "phone=" + phoneText.getText() + "
WHERE Cid = '"
                        +
CustomerIDList.getSelectedItem()+"'");
                errorText.append("\nUpdated " + i
+ " rows successfully");

                CustomerIDList.removeAll();
                loadCustomer();
            }
            catch (SQLException insertException)
            {
                displaySQLErrors(insertException);
            }
        }
    });
```

```
cidText = new TextField(15);  
cidText.setEditable(false);  
cnameText = new TextField(15);  
mailText = new TextField(15);  
passwordText = new TextField(15);  
addressText=new TextField(15);  
phoneText=new TextField(15);
```

```
errorText = new TextArea(10, 40);  
errorText.setEditable(false);
```

```
Panel first = new Panel();  
first.setLayout(new GridLayout(6, 2));  
first.add(new Label("Customer ID:"));  
first.add(cidText);  
first.add(new Label("Name:"));  
first.add(cnameText);  
first.add(new Label("Mail:"));  
first.add(mailText);  
first.add(new Label("Password:"));  
first.add(passwordText);  
first.add(new Label("Address:"));  
first.add(addressText);  
first.add(new Label("Phone:"));  
first.add(phoneText);
```

```
Panel second = new Panel(new GridLayout(4, 1));
```

```
        second.add(updateCustomerButton);

        Panel third = new Panel();
        third.add(errorText);

        add(first);
        add(second);
        add(third);

        setSize(500, 600);
        setLayout(new FlowLayout());
        setVisible(true);
    }

    private void displaySQLExceptions(SQLException e)
    {
        errorText.append("\nSQLException: " +
e.getMessage() + "\n");
        errorText.append("SQLState:  " + e.getSQLState() +
"\n");
        errorText.append("VendorError: " +
e.getErrorCode() + "\n");
    }

    public static void main(String[] args)
    {
        UpdateCustomer upc = new UpdateCustomer();

        upc.buildGUI();
    }
}
```

```
}  
}
```

4)Main Method-

```
import java.awt.*;  
import java.awt.event.*;  
  
class UberEatsDatabase extends Frame implements  
ActionListener  
{  
    String msg = "";  
    Label l1,l2;  
  
    CardLayout cardLO;  
  
    InsertCustomer incus;  
    UpdateCustomer upcus;  
    DeleteCustomer delcus;  
    InsertRestaurant inres;  
    UpdateRestaurant upres;  
    DeleteRestaurant delres;  
    InsertOrder ino;  
    DeleteOrder delo;  
    UpdateOrder upo;  
    InsertPayment inpay;  
    UpdatePayment uppay;  
    DeletePayment delpay;  
    InsertOrders inords;  
    UpdateOrders upords;  
    DeleteOrders delords;  
    InsertOrderFrom inorf;
```

```
UpdateOrderFrom uporf;  
DeleteOrderFrom delorf;  
InsertPays inpays;  
UpdatePays uppays;  
DeletePays delpays;  
InsertGenerates ingen;  
UpdateGenerates upgen;  
DeleteGenerates delgen;
```

```
Panel home,welcome;
```

```
UberEatsDatabase()  
{  
    cardLO = new CardLayout();  
  
    home = new Panel();  
    home.setLayout(cardLO);  
  
    l1 = new Label();  
    l2 =new Label();  
    l1.setAlignment(Label.CENTER);  
    l2.setAlignment(Label.CENTER);  
    l1.setText("Welcome to UBER EATS");  
    l2.setText("\nAll @rights are reserved");  
    //Create welcome panel and add the label to it  
    welcome = new Panel();  
    welcome.add(l1);  
    welcome.add(l2);
```

```
//create panels for each of our menu items
and build them with respective components
        incus = new InsertCustomer(); incus.buildGUI();
        upcus = new UpdateCustomer();
upcus.buildGUI();
        delcus = new DeleteCustomer();
delcus.buildGUI();
        inres = new
InsertRestaurant();inres.buildGUI();
        upres= new
UpdateRestaurant();upres.buildGUI();
        delres = new
DeleteRestaurant();delres.buildGUI();
        ino = new InsertOrder();ino.buildGUI();
        delo = new DeleteOrder();delo.buildGUI();
        upo= new UpdateOrder();upo.buildGUI();
        inpay= new InsertPayment();
inpay.buildGUI();
        uppay= new
UpdatePayment();uppay.buildGUI();
        delpay = new DeletePayment();
delpay.buildGUI();
        inords = new InsertOrders();inords.buildGUI();
        upords = new
UpdateOrders();upords.buildGUI();
        delords = new
DeleteOrders();delords.buildGUI();
        inorf = new
InsertOrderFrom();inorf.buildGUI();
        delorf = new
DeleteOrderFrom();delorf.buildGUI();
```



```
        uporf = new  
UpdateOrderFrom();uporf.buildGUI();  
        inpays = new InsertPays();inpays.buildGUI();  
        delpays = new DeletePays();delpays.buildGUI();  
        uppays = new UpdatePays();uppays.buildGUI();  
        ingen = new  
InsertGenerates();ingen.buildGUI();  
        delgen = new  
DeleteGenerates();delgen.buildGUI();  
        upgen = new  
UpdateGenerates();upgen.buildGUI();
```

```
        //add all the panels to the home panel which  
has a cardlayout
```

```
        home.add(welcome, "Welcome");  
        home.add(incus, "InsertCustomer");  
        home.add(upcus, "UpdateCustomer");  
        home.add(delcus, "DeleteCustomer");  
        home.add(inres, "InsertRestaurant");  
        home.add(upres, "UpdateRestaurant");  
        home.add(delres, "DeleteRestaurant");  
        home.add(ino, "InsertOrder");  
        home.add(deo, "DeleteOrder");  
        home.add(upo, "UpdateOrder");  
        home.add(inpay, "InsertPayment");  
        home.add(uppay, "UpdatePayment");  
        home.add(delpay, "DeletePayment");  
        home.add(inords, "InsertOrders");  
        home.add(upords, "UpdateOrders");
```

```
home.add(delords,"DeleteOrders");
home.add(inpays,"InsertPays");
home.add(delpays,"DeletePays");
home.add(uppays,"UpdatePays");
home.add(inorf,"InsertOrderFrom");
home.add(delorf,"DeleteOrderFrom");
home.add(uporf,"UpdateOrderFrom");
home.add(ingen,"InsertGenerates");
home.add(delgen,"DeleteGenerates");
home.add(upgen,"UpdateGenerates");

// add home panel to main frame
add(home);

// create menu bar and add it to frame
MenuBar mbar = new MenuBar();
setMenuBar(mbar);

// create the menu items and add it to Menu
Menu customer= new Menu("Customer
Details");

MenuItem item1, item2, item3;
customer.add(item1 = new MenuItem("Insert
Customer"));

customer.add(item2 = new MenuItem("View
Customer"));

customer.add(item3 = new MenuItem("Delete
Customer"));
```

```
mbar.add(customer);

Menu res = new Menu("UberEats");
MenuItem item4, item5, item6;
res.add(item4 = new MenuItem("Insert
Restaurant"));
res.add(item5 = new MenuItem("View
Restaurant"));
res.add(item6 = new MenuItem("Delete
Restaurant"));
mbar.add(res);

Menu order = new Menu("Order Details");
MenuItem item7, item8, item9;
order.add(item7 = new MenuItem("Insert
Order"));
order.add(item8 = new MenuItem("View
Order"));
order.add(item9 = new MenuItem("Delete
Order"));
mbar.add(order);

Menu payment= new Menu("Payment
Details");
MenuItem item10, item11, item12;
payment.add(item10 = new MenuItem("Insert
Payment"));
payment.add(item11= new MenuItem("View
Payment"));
payment.add(item12 = new MenuItem("Delete
Payment"));
```

```
mbar.add(payment);

Menu orders= new Menu("Orders");
MenuItem item13, item14, item15;
orders.add(item13 = new MenuItem("Insert
Orders"));
orders.add(item14= new MenuItem("View
Orders"));
orders.add(item15 = new MenuItem("Delete
Orders"));
mbar.add(orders);

Menu orderFrom= new Menu("Order From");
MenuItem item16, item17, item18;
orderFrom.add(item16 = new
MenuItem("Insert Order From"));
orderFrom.add(item17= new MenuItem("View
Order From"));
orderFrom.add(item18 = new
MenuItem("Delete Order From"));
mbar.add(orderFrom);

Menu pays= new Menu("Pays");
MenuItem item19, item20, item21;
pays.add(item19 = new MenuItem("Insert
Pays"));
pays.add(item20= new MenuItem("View
Pays"));
pays.add(item21 = new MenuItem("Delete
Pays"));
mbar.add(pays);
```

```
Menu generates= new Menu("Generates");  
MenuItem item22, item23, item24;  
generates.add(item22 = new  
MenuItem("Insert Generates"));  
generates.add(item23= new MenuItem("View  
Generates"));  
generates.add(item24 = new  
MenuItem("Delete Generates"));  
mbar.add(generates);
```

```
// register listeners  
item1.addActionListener(this);  
item2.addActionListener(this);  
item3.addActionListener(this);  
item4.addActionListener(this);  
item5.addActionListener(this);  
item6.addActionListener(this);  
item7.addActionListener(this);  
item8.addActionListener(this);  
item9.addActionListener(this);  
item10.addActionListener(this);  
item11.addActionListener(this);  
item12.addActionListener(this);  
item13.addActionListener(this);  
item14.addActionListener(this);  
item15.addActionListener(this);  
item16.addActionListener(this);  
item17.addActionListener(this);  
item18.addActionListener(this);  
item19.addActionListener(this);
```

```
        item20.addActionListener(this);
        item21.addActionListener(this);
        item22.addActionListener(this);
        item23.addActionListener(this);
        item24.addActionListener(this);

        // Anonymous inner class which extends
        WindowAdaptor to handle the Window event: windowClosing
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent
we)

                {
                    System.exit(0);
                }
        });

        //Frame properties
        setTitle("UBER EATS");
        Color clr = new Color(255, 102, 102);
        setBackground(clr);
        setFont(new Font("Monaco", Font.BOLD, 20));
        setSize(900, 1000);

        setVisible(true);

    }

    public void actionPerformed(ActionEvent ae)
    {
        String arg = ae.getActionCommand();
```

```
        if(arg.equals("Insert Customer"))
        {

            cardLO.show(home, "InsertCustomer");

        }

        else if(arg.equals("View Customer"))
        {

            cardLO.show(home, "UpdateCustomer");

        }

        else if(arg.equals("Delete Customer"))
        {

            cardLO.show(home, "DeleteCustomer");

        }
        else if(arg.equals("Insert Restaurant"))
        {

            cardLO.show(home, "InsertRestaurant");

        }
        else if(arg.equals("Delete Restaurant"))
        {

            cardLO.show(home, "DeleteRestaurant");

        }
        else if(arg.equals("View Restaurant"))
```

```
{  
  
    cardLO.show(home, "UpdateRestaurant");  
}  
else if(arg.equals("Insert Order"))  
{  
    cardLO.show(home, "InsertOrder");  
}  
else if(arg.equals("Delete Order"))  
{  
    cardLO.show(home, "DeleteOrder");  
}  
else if(arg.equals("View Order"))  
{  
    cardLO.show(home, "UpdateOrder");  
}  
else if(arg.equals("Insert Payment"))  
{  
    cardLO.show(home, "InsertPayment");  
}  
else if(arg.equals("View Payment"))  
{  
    cardLO.show(home, "UpdatePayment");  
}  
else if(arg.equals("Delete Payment"))  
{  
    cardLO.show(home, "DeletePayment");  
}  
else if(arg.equals("Insert Orders"))  
{  
    cardLO.show(home, "InsertOrders");  
}
```



```
}  
else if(arg.equals("View Orders"))  
{  
    cardLO.show(home, "UpdateOrders");  
}  
else if(arg.equals("Delete Orders"))  
{  
    cardLO.show(home, "DeleteOrders");  
}  
else if(arg.equals("Insert Order From"))  
{  
    cardLO.show(home, "InsertOrderFrom");  
}  
else if(arg.equals("View Order From"))  
{  
    cardLO.show(home, "UpdateOrderFrom");  
}  
else if(arg.equals("Delete Order From"))  
{  
    cardLO.show(home, "DeleteOrderFrom");  
}  
else if(arg.equals("Insert Pays"))  
{  
    cardLO.show(home, "InsertPays");  
}  
else if(arg.equals("View Pays"))  
{  
    cardLO.show(home, "UpdatePays");  
}  
else if(arg.equals("Delete Pays"))  
{
```

```
        cardLO.show(home, "DeletePays");
    }
    else if(arg.equals("Insert Generates"))
    {
        cardLO.show(home, "InsertGenerates");
    }
    else if(arg.equals("View Generates"))
    {
        cardLO.show(home, "UpdateGenerates");
    }
    else if(arg.equals("Delete Generates"))
    {
        cardLO.show(home, "DeleteGenerates");
    }
}

public static void main(String ... args)
{
    new UberEatsDatabase();
}
}
```

Connectivity with the Database:

Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access a database. It is a Java-based data access technology used for Java database connectivity. It is a part of the Java Standard Edition platform, from Oracle Corporation. It provides methods to query and

update data in a database and is oriented towards relational databases.

Block of Code for JAVA-SQL connectivity with JDBC:

```
public void connectToDB()
{
    try
    {
        connection =
        DriverManager.getConnection("jdbc:oracle:thin:@lo
calhost:1521:xe","system","OracleDBMS2090&");
        statement = connection.createStatement();

    }
    catch (SQLException connectException)
    {

        System.out.println(connectException.getMessage());

        System.out.println(connectException.getSQLState());

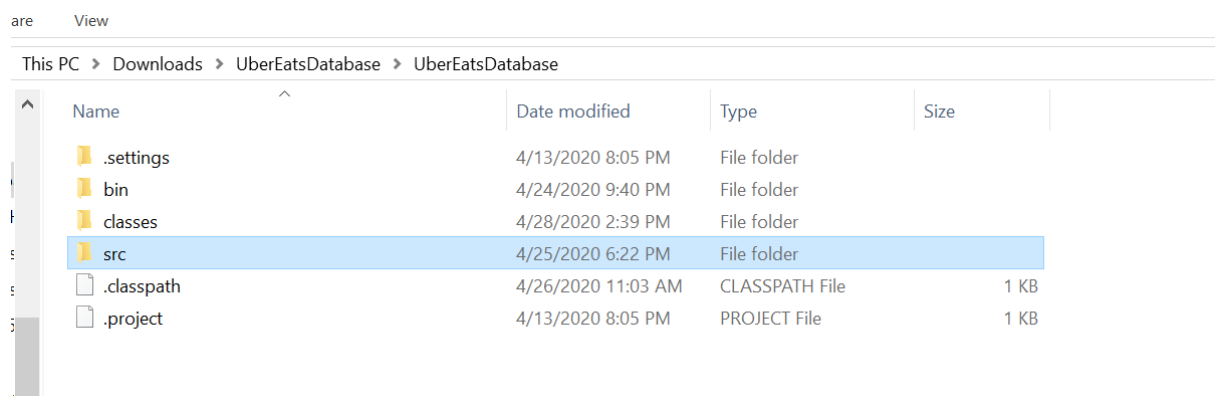
        System.out.println(connectException.getErrorCode());
        System.exit(1);
    }
}
```

GITHUB LINK:

<https://github.com/Samhita20/DBMS-ASSIGNMENT>

Folder Structure:

This project consists of a folder named src which has 25 .java files. The files are for 8 different tables, including four relation tables. The programs include insert, update, delete functionalities and the main function. By which we can navigate easily to reach the java code and we can make changes easily.




























Name	Date modified	Type	Size
.settings	4/13/2020 8:05 PM	File folder	
bin	4/24/2020 9:40 PM	File folder	
classes	4/28/2020 2:39 PM	File folder	
src	4/25/2020 6:22 PM	File folder	
.classpath	4/26/2020 11:03 AM	CLASSPATH File	1 KB
.project	4/13/2020 8:05 PM	PROJECT File	1 KB

DBMS ASSIGNMENT -1

UBER EATS DATABASE

This PC > Downloads > UberEatsDatabase > UberEatsDatabase > src

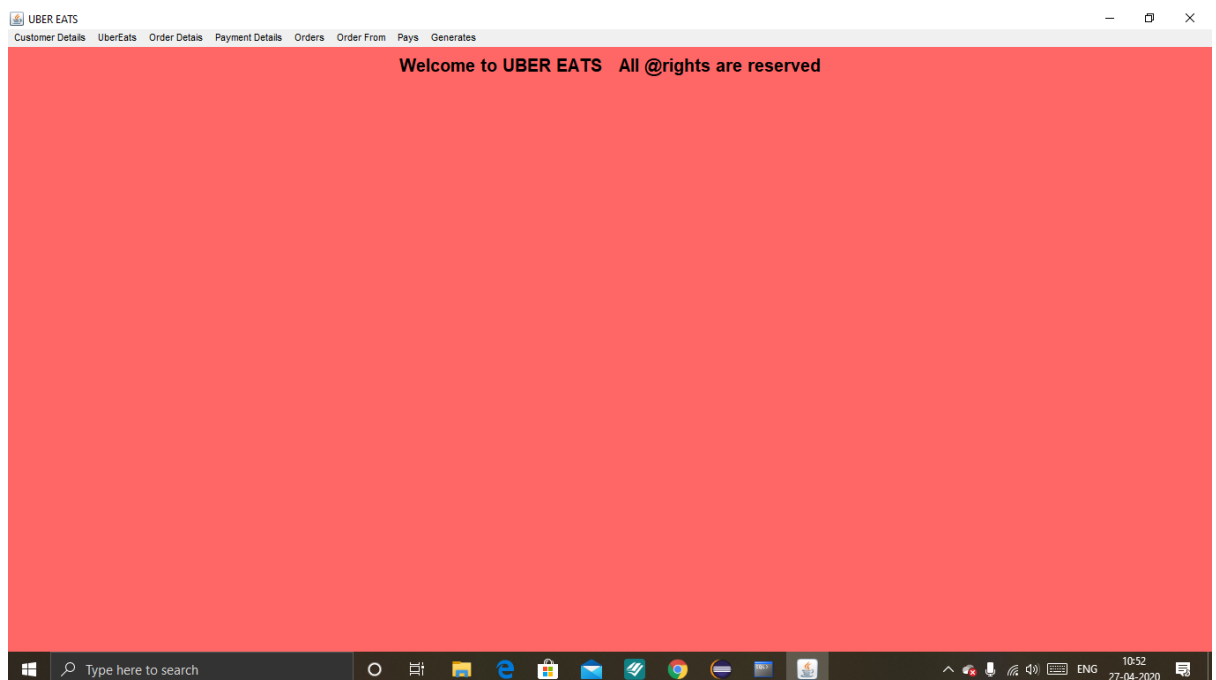
Name	Date modified	Type	Size
 DeleteCustomer	4/27/2020 10:15 AM	Java Source File	5 KB
 DeleteGenerates	4/28/2020 3:10 PM	Java Source File	4 KB
 DeleteOrder	4/27/2020 10:15 AM	Java Source File	5 KB
 DeleteOrderFrom	4/27/2020 3:03 PM	Java Source File	4 KB
 DeleteOrders	4/27/2020 10:16 AM	Java Source File	4 KB
 DeletePayment	4/28/2020 2:59 PM	Java Source File	5 KB
 DeletePays	4/27/2020 2:58 PM	Java Source File	4 KB
 DeleteRestaurant	4/27/2020 10:20 AM	Java Source File	5 KB
 InsertCustomer	4/27/2020 12:13 PM	Java Source File	4 KB
 InsertGenerates	4/27/2020 10:23 AM	Java Source File	4 KB
 InsertOrder	4/28/2020 3:16 PM	Java Source File	3 KB
 InsertOrderFrom	4/27/2020 10:26 AM	Java Source File	4 KB
 InsertOrders	4/27/2020 10:27 AM	Java Source File	4 KB
 InsertPayment	4/27/2020 10:28 AM	Java Source File	4 KB
 InsertPays	4/27/2020 10:29 AM	Java Source File	4 KB
 InsertRestaurant	4/27/2020 10:29 AM	Java Source File	4 KB
 UberEatsdatabase	4/28/2020 3:17 PM	Java Source File	10 KB
 UpdateCustomer	4/28/2020 2:41 PM	Java Source File	5 KB
 UpdateGenerates	4/27/2020 1:13 PM	Java Source File	4 KB
 UpdateOrder	4/27/2020 10:33 AM	Java Source File	4 KB
 UpdateOrderFrom	4/27/2020 1:22 PM	Java Source File	4 KB
 UpdateOrders	4/27/2020 1:24 PM	Java Source File	4 KB
 UpdatePayment	4/28/2020 3:00 PM	Java Source File	5 KB
 UpdatePays	4/27/2020 1:12 PM	Java Source File	4 KB
 UpdateRestaurant	4/28/2020 2:46 PM	Java Source File	5 KB

TESTING

The program runs for the three basic operations of insertion, updating, and deletion on 8 different tables. Along with this, it also has a output column which gives the information about how many rows have been edited, Errors, syntactical or exceptional will be shown if occurred.

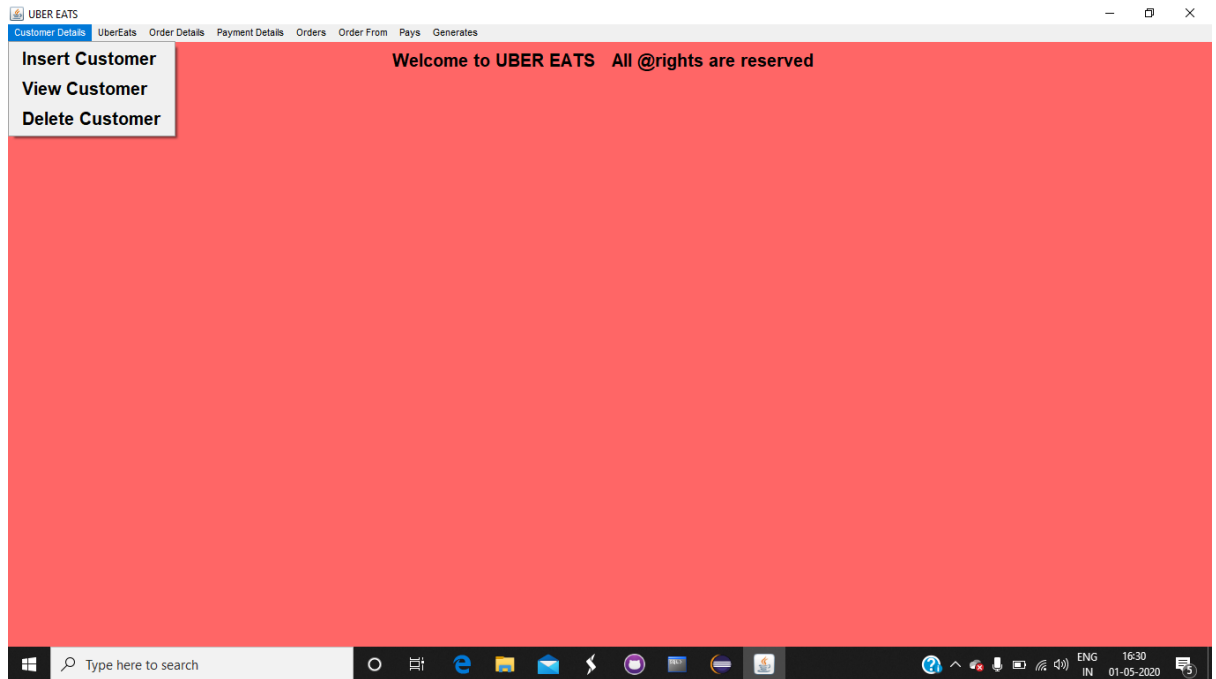
The code written for building GUI and connecting with database ensures that the values entered by the users are of correct data types. It prompts an error message in the text message box.

Home Page:



DBMS ASSIGNMENT -1

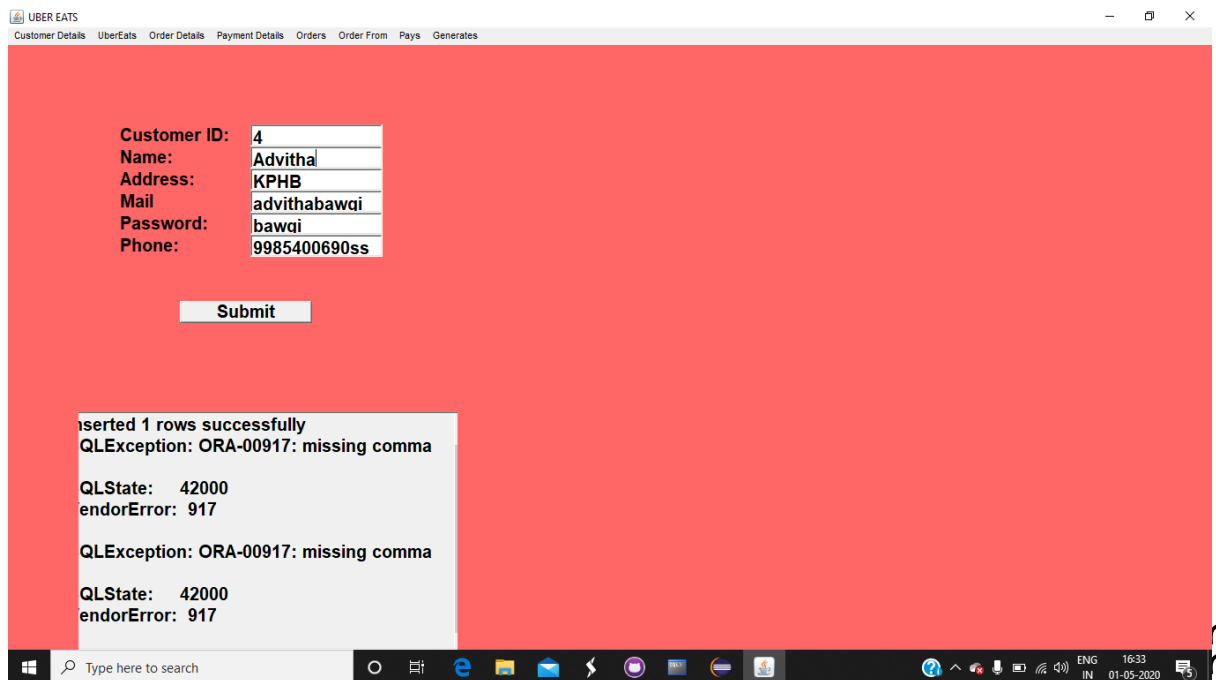
UBER EATS DATABASE



Insertion:

Error-

If user given invalid content it gives an error.



G Samhita
1602-18-737-095

DBMS ASSIGNMENT -1
UBER EATS DATABASE

Proper Entry-

UBER EATS

Customer Details UberEats Order Details Payment Details Orders Order From Pays Generates

Customer ID: 4
Name: Advitha
Address: KPHB
Mail: advithabawqi
Password: bawqi
Phone: 9985400690

Submit

Inserted 1 rows successfully

```
SQL> select *from customer;
```

CID	PASSWORD	MAIL	NAME
105 Hapsiguda	samhita	samhitagolla	Samhita 6303775756
95 Secunderabad	narthaki	ssathwikareddy	Sathwika 9701740017
93 Uppal	renu	renuakanksha	Akanksha 9381929435

CID	PASSWORD	MAIL	NAME
89 Bandlaguda	curie	mpranathi	Pranathi 8886502990
107 Kukatpally	taaadi	tsujithaa	Sujitha 9849529440
4 KPHB	bawgi	advithabawgi	Advitha 9985400690

6 rows selected.

G Samhita
1602-18-737-095

Deletion:

Proper Entry-

DBMS ASSIGNMENT -1
UBER EATS DATABASE

The screenshot displays a web application titled "UBER EATS" with a navigation bar containing "Customer Details", "UberEats", "Order Details", "Payment Details", "Orders", "Order From", "Pays", and "Generates". The main content area has a red background. On the left, a list of customer IDs (105, 95, 93, 89, 107, 4) is shown, with "4" selected. To the right, the details for Customer ID 4 are displayed: Name: Advitha, Mail: advithabawgi, Password: bawgi, Address: KPHB, and Phones: 9985400690. A "Delete" button is located next to the details. Below the details is a large empty text area.

Customer ID	Name	Mail	Password	Address	Phones
4	Advitha	advithabawgi	bawgi	KPHB	9985400690

Deleted 1 rows successfully

G Samhita
1602-18-737-095

DBMS ASSIGNMENT -1
UBER EATS DATABASE

```
SQL> select *from customer;
```

CID	PASSWORD	MAIL	NAME	ADDRESS	PHONE
105	samhita	samhitagolla	Samhita	Hapsiguda	6303775756
95	dancer	ssathwikareddy	Sathwika	Secunderabad	9701740071
93	renu	renuakanksha	Akanksha	Uppal	9381929435
89	curie	mpranathi	Pranathi	Bandlaguda	8886502990
107	taaadi	tsujithaa	Sujitha	Kukatpally	9849529440

Update:

Error-

UBER EATS

Customer Details UberEats Order Details Payment Details Orders Order From Pays Generates

105
95
93
89
107
4

Customer ID: 95
Name: Sathwika
Mail: ssathwikareddy
Password: dancer
Address: Secunderabad
Phone: 9701740071gh

Modify

Updated 1 rows successfully
SQLException: ORA-00933: SQL command not properly ended
SQLState: 42000
VendorError: 933

Type here to search

G Samhita
1602-18-737-095

Proper Entry-

The entry of Customer Id 95 was updated from narthaki to dancer in the field of password.

UBER EATS

Customer Details | UberEats | Order Details | Payment Details | Orders | Order From | Pays | Generates

105
95
93
89
107
4

Customer ID: 95
Name: Sathwika
Mail: ssathwikareddy
Password: dancer
Address: Secunderabad
Phone: 9701740071

Modify

Updated 1 rows successfully

```
SQL> select *from customer;
```

CID	PASSWORD	MAIL	NAME	ADDRESS	PHONE
105	samhita	samhitagolla	Samhita	Hapsiguda	6303775756
95	narthaki	ssathwikareddy	Sathwika	Secunderabad	9701740017
93	renu	renuakanksha	Akanksha	Uppal	9381929435
89	curie	mpranathi	Pranathi	Bandlaguda	8886502990
107	taaadi	tsujithaa	Sujitha	Kukatpally	9849529440
4	bawgi	advithabawgi	Advitha	KPHB	9985400690

6 rows selected.

G Samhita
1602-18-737-095

DBMS ASSIGNMENT -1
UBER EATS DATABASE

```
SQL> select *from customer;
```

CID	PASSWORD	MAIL	NAME
105	samhita	samhitagolla	Samhita
Hapsiguda			6303775756
95	dancer	ssathwikareddy	Sathwika
Secunderabad			9701740071
93	renu	renuakanksha	Akanksha
Uppal			9381929435
CID	PASSWORD	MAIL	NAME
39	curie	mpranathi	Pranathi
Bandlaguda			8886502990
107	taaadi	tsujithaa	Sujitha
Kukatpally			9849529440
4	bawgi	advithabawgi	Advitha
KPHB			9985400690

```
5 rows selected.
```

G Samhita
1602-18-737-095

RESULTS:

The DML commands, insert, update and delete for one of the tables are given below:

For Customer table (in java as per the application):

```
INSERT - "INSERT INTO Customer VALUES('" + cidText.getText() + ", " + "'" + passwordText.getText() + "', " + "'" + mailText.getText() + "', " + "'" + cnameText.getText() + "', " + "'" + addressText.getText() + "', " + "'" + phoneText.getText() + "');" ;
```

```
DELETE - "DELETE FROM Customer WHERE CID = '" + CustomerIDList.getSelectedItem() + "' "
```

```
UPDATE - "UPDATE Customer " + "SET password='" + passwordText.getText() + "', " + "mail='" + mailText.getText() + "', " + "name ='" + cnameText.getText() + "', " + "address ='" + addressText.getText() + "', " + "phone=" + phoneText.getText() + " WHERE Cid = '" + CustomerIDList.getSelectedItem() + "' "
```

1. Connection with database is established.
2. The values given for tables in the GUI components by the user are saved in the database.

REFERENCES

1. https://en.wikipedia.org/wiki/Uber_Eats
2. <https://eng.uber.com/uber-eats-query-understanding/>
3. <https://github.com/Samhita20/DBMS-ASSIGNMENT>

