OOP-C00290922

Invoice System

Documentation

Douglas Kadzutu

3/31/2023

Contents

DESCRIPTION

The customer purchase management system is a software application that allows a business to keep track of their customers, invoices, and products. The system consists of a backend database and a front-end graphical user interface (GUI). The backend database contains at least three tables (Customer, Invoice, and Product) and provides inner join operations over multiple tables. The frontend GUI provides CRUD (create, retrieve, update, delete) operations on the database and utilizes a variety of Swing components such as buttons and text boxes. It is written in java using swing for GUI elements and the data is store in a relational database i.e. MySqL

REQUIREMENTS

1) **Backend database:**
   The system shall include a backend database that stores customer, invoice, and product data. The database shall be designed with at least three tables, including Customer, Invoice, and Product tables. The database shall provide inner join operations over multiple tables. The database shall use a Java database connector to allow communication between the frontend and backend.

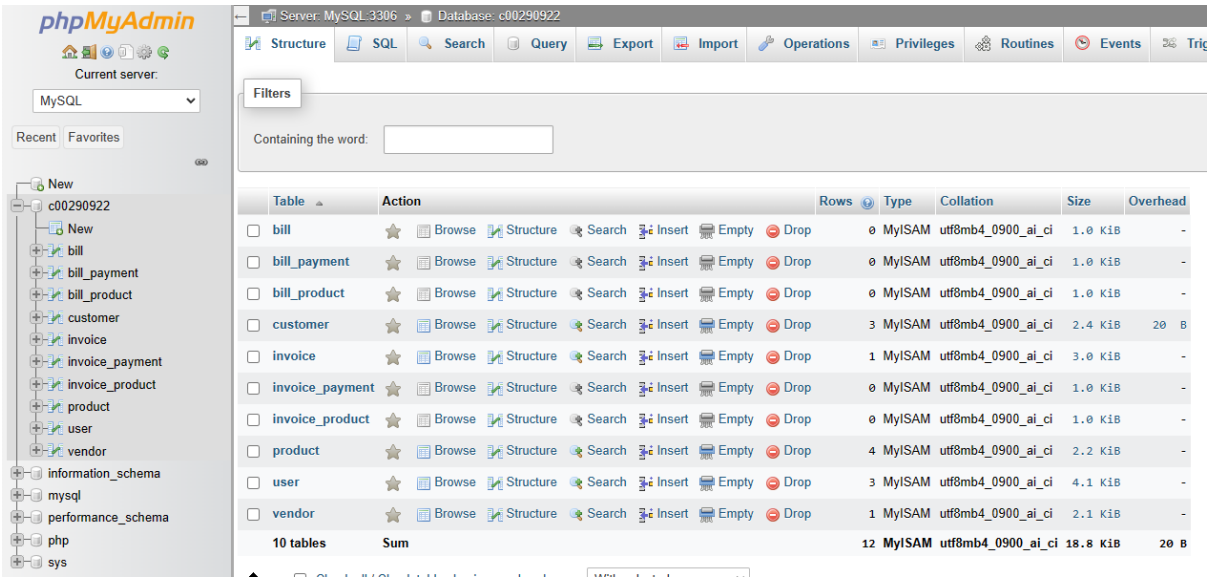
2) **Frontend GUI:**
   The system shall include a graphical user interface (GUI) that allows the user to interact with the backend database. The GUI shall provide CRUD operations on the database, including create, retrieve, update, and delete. The GUI shall be developed using Java Swing components, including buttons, text boxes, and tables. The GUI shall allow the user to search, sort, and filter data in the database. The GUI shall provide error handling and validation for user input.

3) **Security:**
   The system shall provide secure access to the database, requiring users to authenticate themselves with a valid username and password. The system shall limit access to certain functionalities based on user roles and permissions. The system shall implement security measures to prevent unauthorized access, data breaches, and data loss.

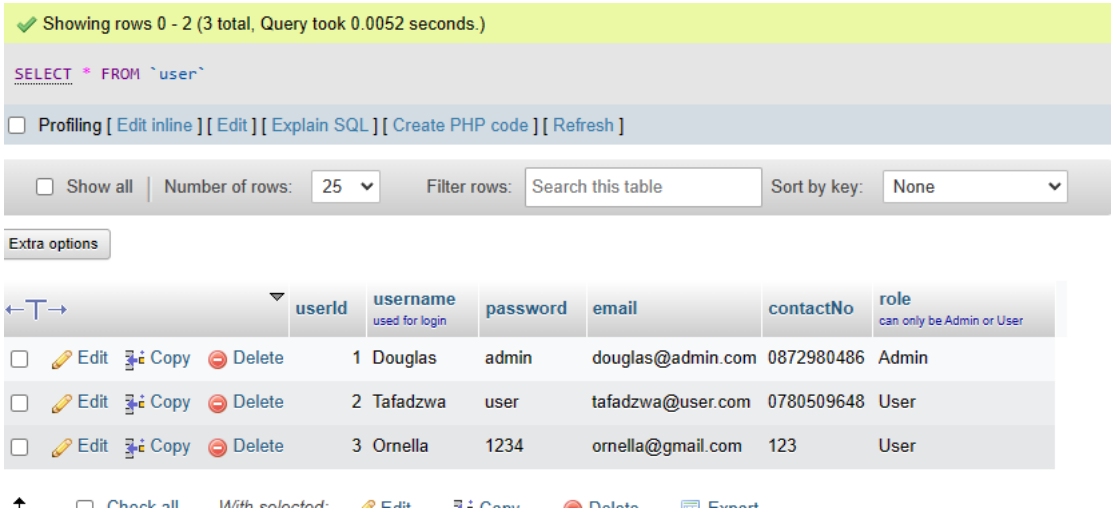
4) **Performance:**
   The system shall be designed to handle large amounts of data efficiently and quickly. The system shall minimize database queries and optimize query performance. The system shall implement caching mechanisms to reduce database load and improve performance.

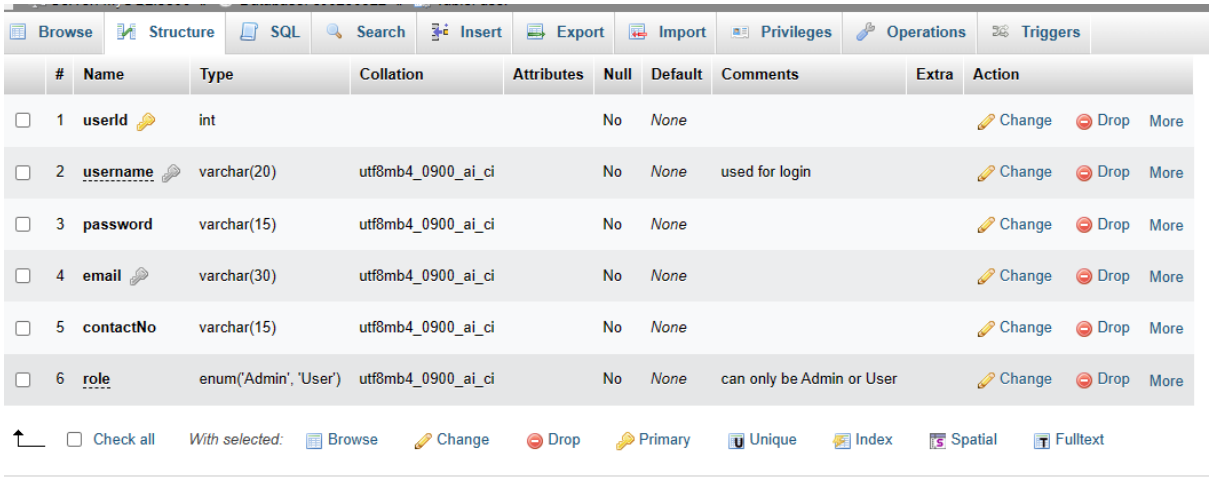
5) **Documentation:**
   The system shall include documentation that outlines its functionality, architecture, and design. The documentation shall provide instructions on how to install, configure, and use the system. The documentation shall include a user manual that explains how to use the frontend GUI to manage customer purchases.

DATABASE

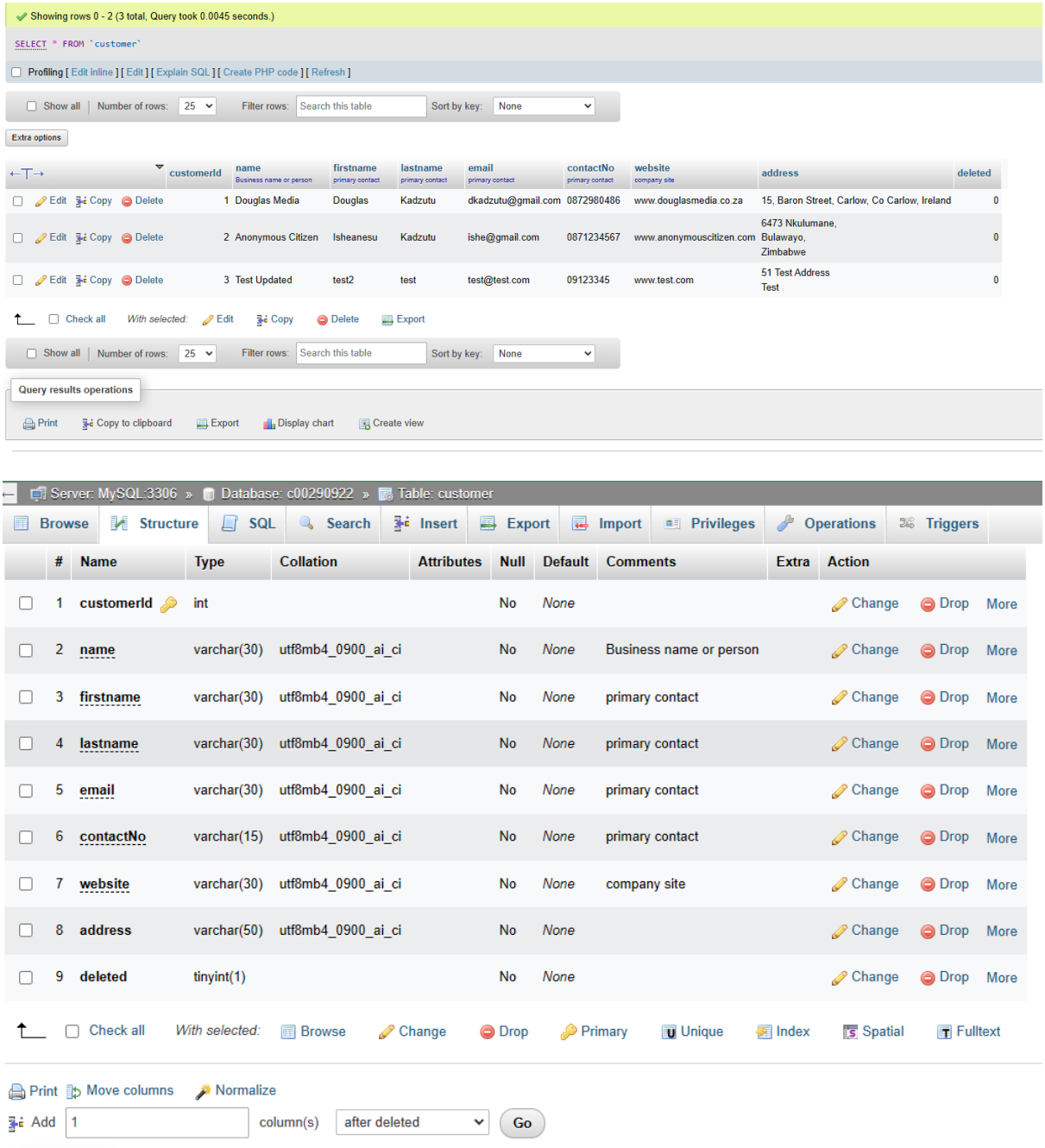


1) User





2) Customer



Showing rows 0 - 2 (3 total, Query took 0.0045 seconds.)

```
SELECT * FROM `customer`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

	customerId	name (Business name or person)	firstname (primary contact)	lastname (primary contact)	email (primary contact)	contactNo (primary contact)	website (company site)	address	deleted
Edit Copy Delete	1	Douglas Media	Douglas	Kadzutu	dkadzutu@gmail.com	0872980486	www.douglasmedia.co.za	15, Baron Street, Carlow, Co Carlow, Ireland	0
Edit Copy Delete	2	Anonymous Citizen	Isheanesu	Kadzutu	ishe@gmail.com	0871234567	www.anonymouscitizen.com	6473 Nkulumane, Bulawayo, Zimbabwe	0
Edit Copy Delete	3	Test Updated	test2	test	test@test.com	09123345	www.test.com	51 Test Address Test	0

Check all | With selected: Edit Copy Delete Export

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Query results operations

Print | Copy to clipboard | Export | Display chart | Create view

Server: MySQL:3306 » Database: c00290922 » Table: customer

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Triggers

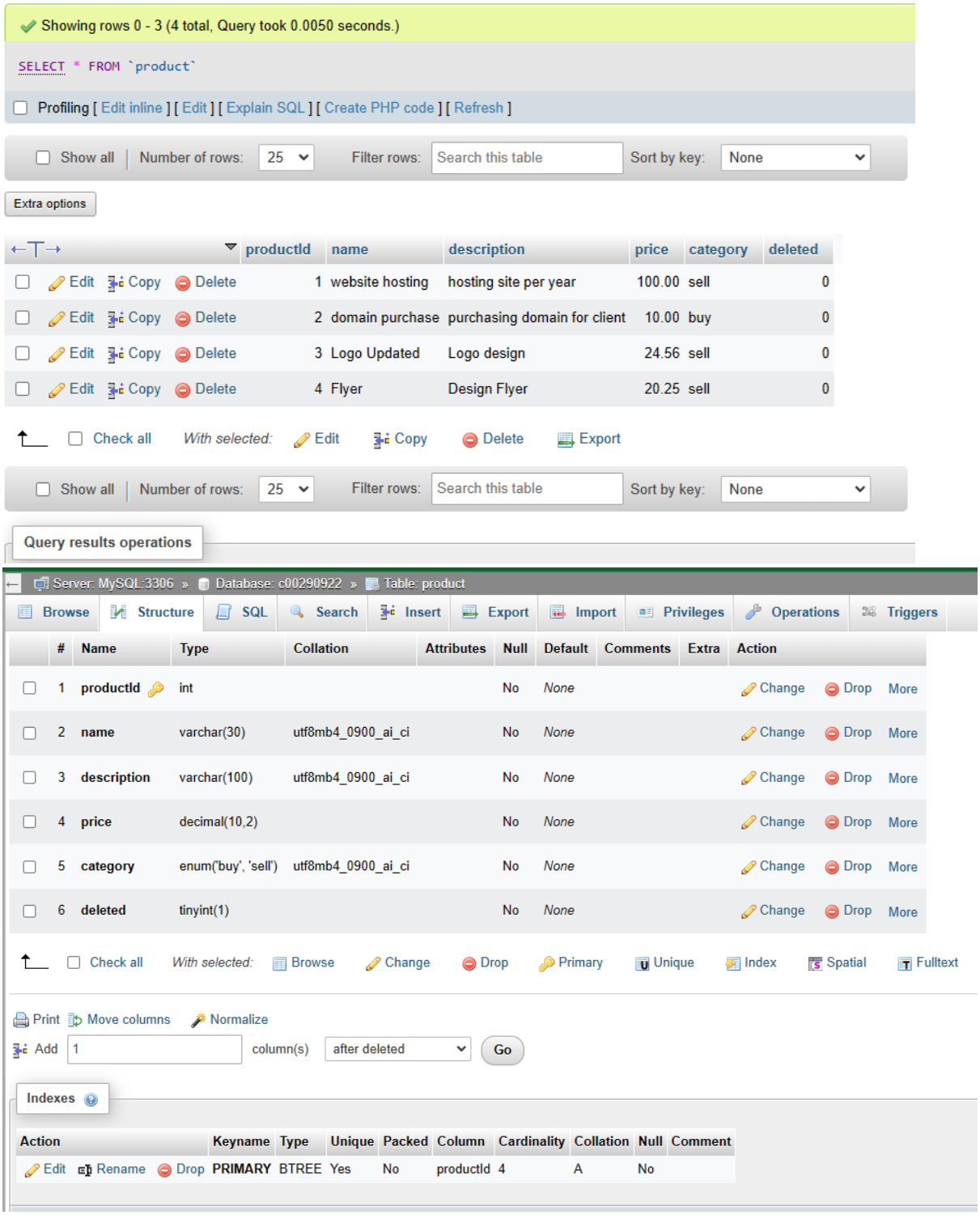
	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
	1	customerId	int			No	None			Change Drop More
	2	name	varchar(30)	utf8mb4_0900_ai_ci		No	None	Business name or person		Change Drop More
	3	firstname	varchar(30)	utf8mb4_0900_ai_ci		No	None	primary contact		Change Drop More
	4	lastname	varchar(30)	utf8mb4_0900_ai_ci		No	None	primary contact		Change Drop More
	5	email	varchar(30)	utf8mb4_0900_ai_ci		No	None	primary contact		Change Drop More
	6	contactNo	varchar(15)	utf8mb4_0900_ai_ci		No	None	primary contact		Change Drop More
	7	website	varchar(30)	utf8mb4_0900_ai_ci		No	None	company site		Change Drop More
	8	address	varchar(50)	utf8mb4_0900_ai_ci		No	None			Change Drop More
	9	deleted	tinyint(1)			No	None			Change Drop More

Check all | With selected: Browse Change Drop Primary Unique Index Spatial Fulltext

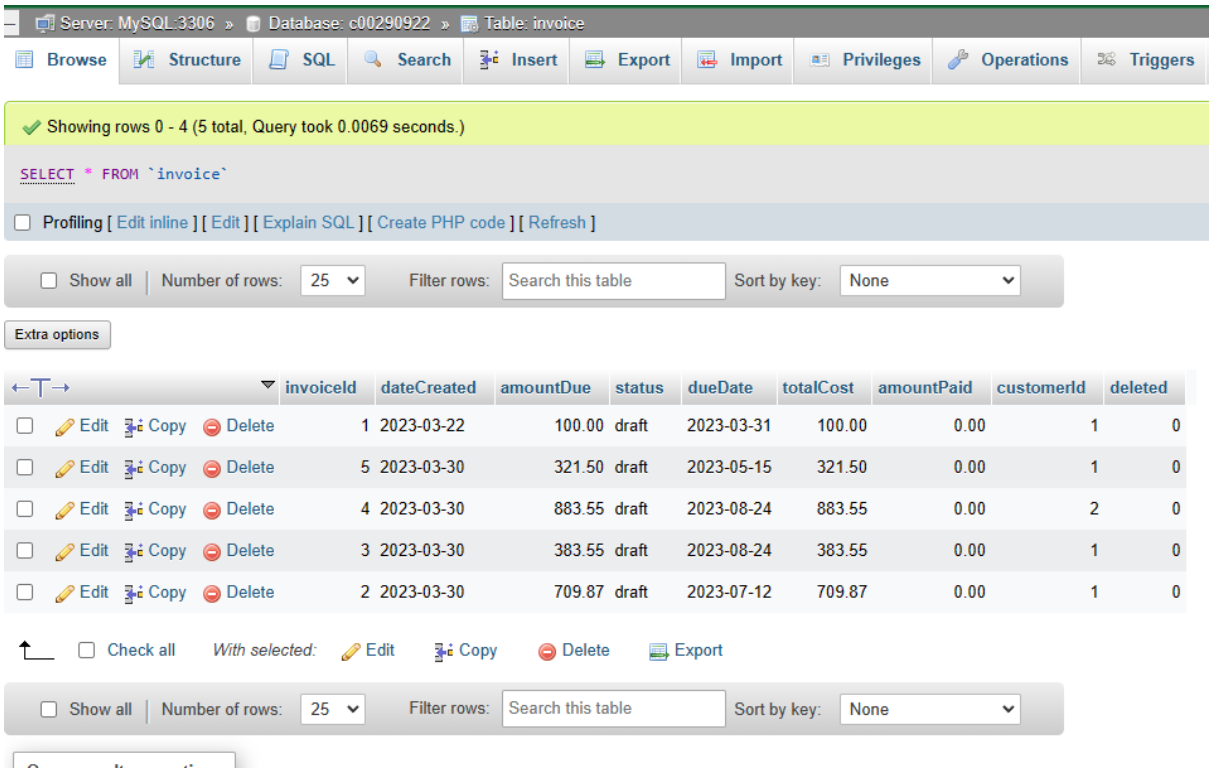
Print | Move columns | Normalize

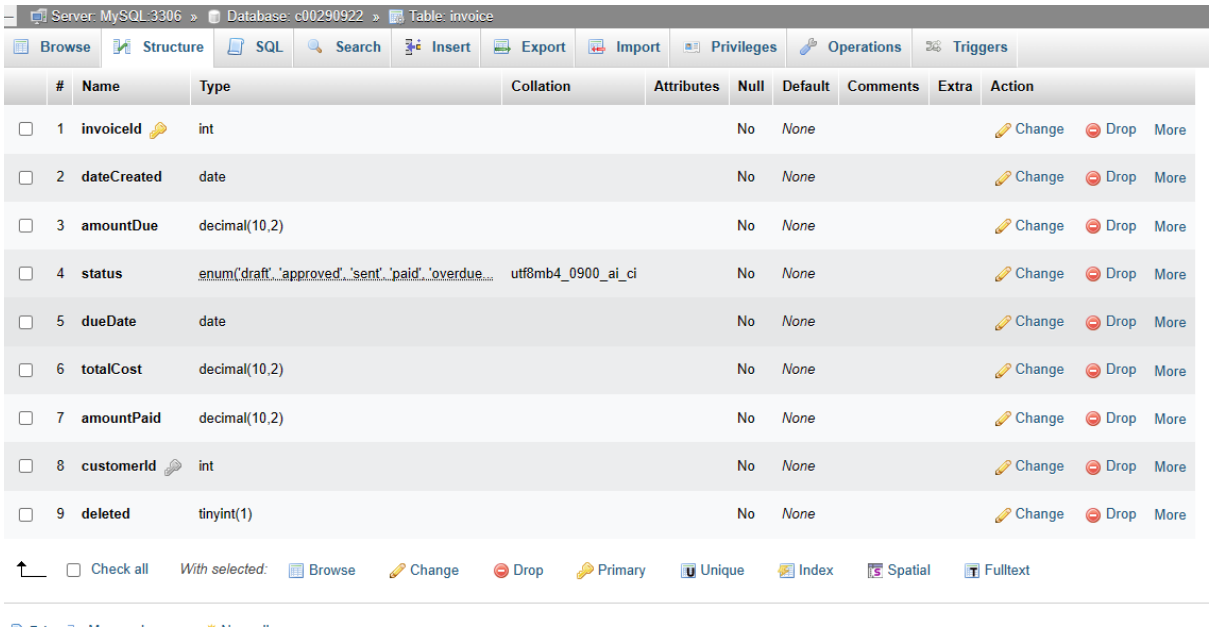
Add 1 column(s) after deleted Go

3) Product

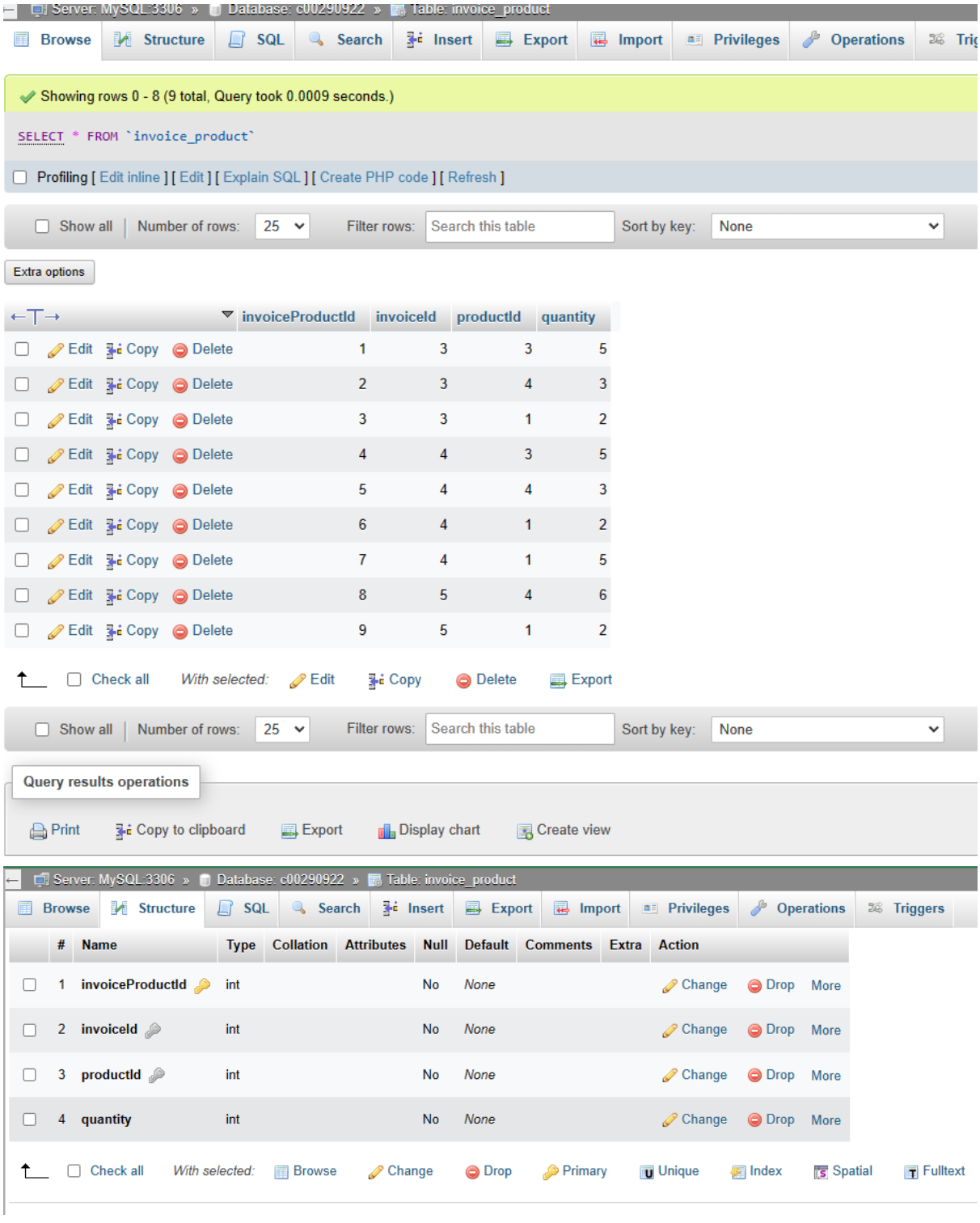

Showing rows 0 - 3 (4 total, Query took 0.0050 seconds.)
SELECT * FROM `product`
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
Extra options
productId name description price category deleted
Edit Copy Delete 1 website hosting hosting site per year 100.00 sell 0
Edit Copy Delete 2 domain purchase purchasing domain for client 10.00 buy 0
Edit Copy Delete 3 Logo Updated Logo design 24.56 sell 0
Edit Copy Delete 4 Flyer Design Flyer 20.25 sell 0
Check all With selected: Edit Copy Delete Export
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
Query results operations
Server: MySQL:3306 » Database: c00290922 » Table: product
Browse Structure SQL Search Insert Export Import Privileges Operations Triggers
Name Type Collation Attributes Null Default Comments Extra Action
1 productId int No None Change Drop More
2 name varchar(30) utf8mb4_0900_ai_ci No None Change Drop More
3 description varchar(100) utf8mb4_0900_ai_ci No None Change Drop More
4 price decimal(10,2) No None Change Drop More
5 category enum('buy', 'sell') utf8mb4_0900_ai_ci No None Change Drop More
6 deleted tinyint(1) No None Change Drop More
Check all With selected: Browse Change Drop Primary Unique Index Spatial Fulltext
Print Move columns Normalize
Add 1 column(s) after deleted Go
Indexes
Action Keyname Type Unique Packed Column Cardinality Collation Null Comment
Edit Rename Drop PRIMARY BTREE Yes No productId 4 A No

4) Invoice


Server: MySQL:3306 » Database: c00290922 » Table: invoice
Browse Structure SQL Search Insert Export Import Privileges Operations Triggers
Showing rows 0 - 4 (5 total, Query took 0.0069 seconds.)
SELECT * FROM `invoice`
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
Extra options
invoiceId dateCreated amountDue status dueDate totalCost amountPaid customerId deleted
Edit Copy Delete 1 2023-03-22 100.00 draft 2023-03-31 100.00 0.00 1 0
Edit Copy Delete 5 2023-03-30 321.50 draft 2023-05-15 321.50 0.00 1 0
Edit Copy Delete 4 2023-03-30 883.55 draft 2023-08-24 883.55 0.00 2 0
Edit Copy Delete 3 2023-03-30 383.55 draft 2023-08-24 383.55 0.00 1 0
Edit Copy Delete 2 2023-03-30 709.87 draft 2023-07-12 709.87 0.00 1 0
Check all With selected: Edit Copy Delete Export
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None


Server: MySQL:3306 » Database: c00290922 » Table: invoice
Browse Structure SQL Search Insert Export Import Privileges Operations Triggers
Name Type Collation Attributes Null Default Comments Extra Action
1 invoiceId int No None Change Drop More
2 dateCreated date No None Change Drop More
3 amountDue decimal(10,2) No None Change Drop More
4 status enum('draft', 'approved', 'sent', 'paid', 'overdue... utf8mb4_0900_ai_ci No None Change Drop More
5 dueDate date No None Change Drop More
6 totalCost decimal(10,2) No None Change Drop More
7 amountPaid decimal(10,2) No None Change Drop More
8 customerId int No None Change Drop More
9 deleted tinyint(1) No None Change Drop More
Check all With selected: Browse Change Drop Primary Unique Index Spatial Fulltext

5) Invoice_Product



6) Invoice_Payment

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Triggers

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	invoicePayId	int			No	None			Change Drop More
2	method	enum('card', 'cash', 'eft')	utf8mb4_0900_ai_ci		No	None	payment method		Change Drop More
3	amount	decimal(10,2)			No	None			Change Drop More
4	datePayed	date			No	None			Change Drop More
5	notes	varchar(100)	utf8mb4_0900_ai_ci		No	None			Change Drop More
6	invoiceId	int			No	None			Change Drop More
7	deleted	tinyint(1)			No	None			Change Drop More

Check all With selected: Browse Change Drop Primary Unique Index Spatial Fulltext

7) Vendor

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Triggers

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	vendorId	int			No	None			Change Drop More
2	name	varchar(20)	utf8mb4_0900_ai_ci		No	None	business name		Change Drop More
3	firstname	varchar(20)	utf8mb4_0900_ai_ci		No	None	contact person		Change Drop More
4	lastname	varchar(20)	utf8mb4_0900_ai_ci		No	None	contact person		Change Drop More
5	email	varchar(30)	utf8mb4_0900_ai_ci		No	None	contact person		Change Drop More
6	address	varchar(100)	utf8mb4_0900_ai_ci		No	None	business address		Change Drop More
7	contactNo	varchar(15)	utf8mb4_0900_ai_ci		No	None	contact person		Change Drop More
8	website	varchar(30)	utf8mb4_0900_ai_ci		No	None			Change Drop More
9	deleted	tinyint(1)			No	None			Change Drop More

Check all With selected: Browse Change Drop Primary Unique Index Spatial Fulltext

8) Bill

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Triggers

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
☐	1	billId	int			No	None			Change Drop More
☐	2	dateCreated	date			No	None			Change Drop More
☐	3	amountDue	decimal(10,2)			No	None			Change Drop More
☐	4	status	enum('unpaid', 'paid', 'partial')	utf8mb4_0900_ai_ci		No	None			Change Drop More
☐	5	dateDue	date			No	None			Change Drop More
☐	6	totalCost	decimal(10,2)			No	None			Change Drop More
☐	7	amountPaid	decimal(10,2)			No	None			Change Drop More
☐	8	vendorId	int			No	None			Change Drop More
☐	9	deleted	tinyint(1)			No	None			Change Drop More

Check all With selected: Browse Change Drop Primary Unique Index Spatial Fulltext

9) Bill_payment

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Triggers

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
☐	1	billPaymentId	int			No	None			Change Drop More
☐	2	method	enum('card', 'cash', 'eft')	utf8mb4_0900_ai_ci		No	None			Change Drop More
☐	3	amount	decimal(10,2)			No	None			Change Drop More
☐	4	paymentDate	date			No	None			Change Drop More
☐	5	notes	varchar(50)	utf8mb4_0900_ai_ci		No	None			Change Drop More
☐	6	billId	int			No	None			Change Drop More
☐	7	deleted	tinyint(1)			No	None			Change Drop More

Check all With selected: Browse Change Drop Primary Unique Index Spatial Fulltext

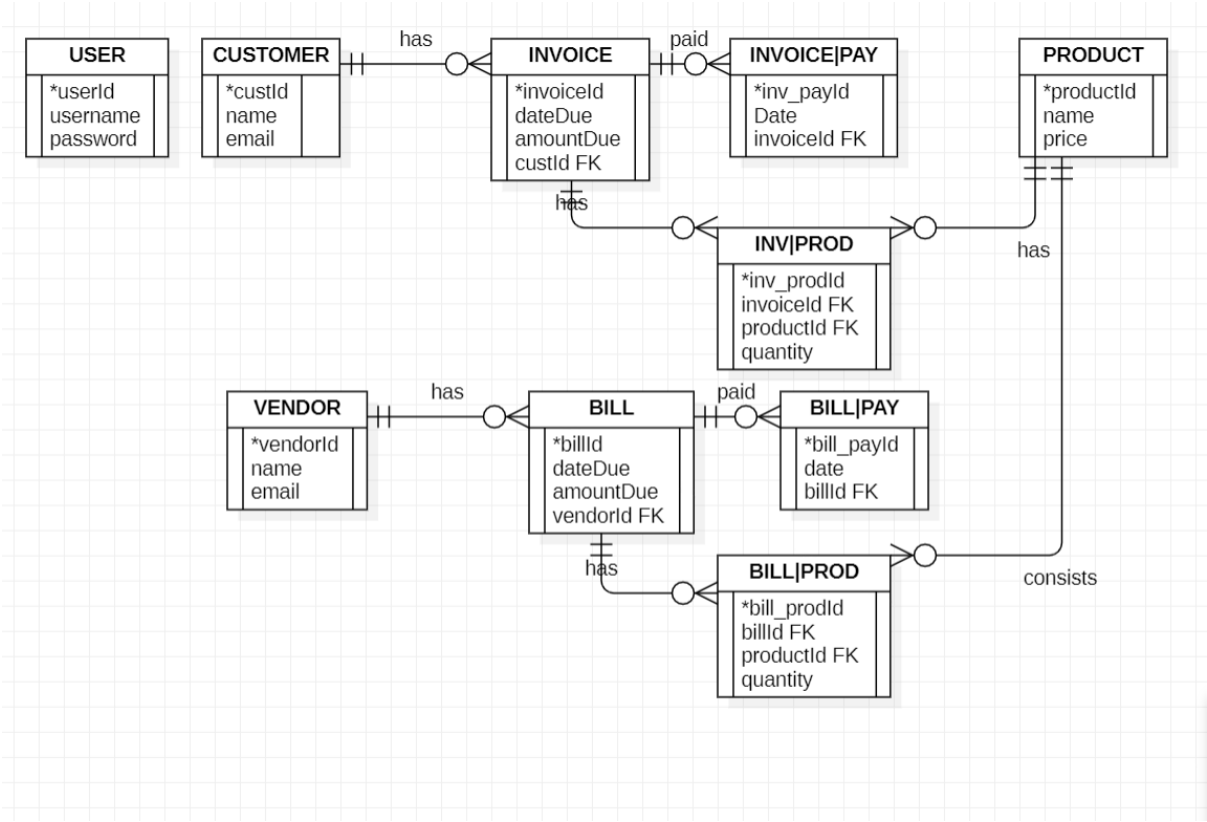
10) Bill_product

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
☐	1	billProductId	int			No	None			Change Drop More
☐	2	billId	int			No	None			Change Drop More
☐	3	productId	int			No	None			Change Drop More
☐	4	quantity	int			No	None			Change Drop More

Check all With selected: Browse Change Drop Primary Unique Index Sp

ER DIAGRAM

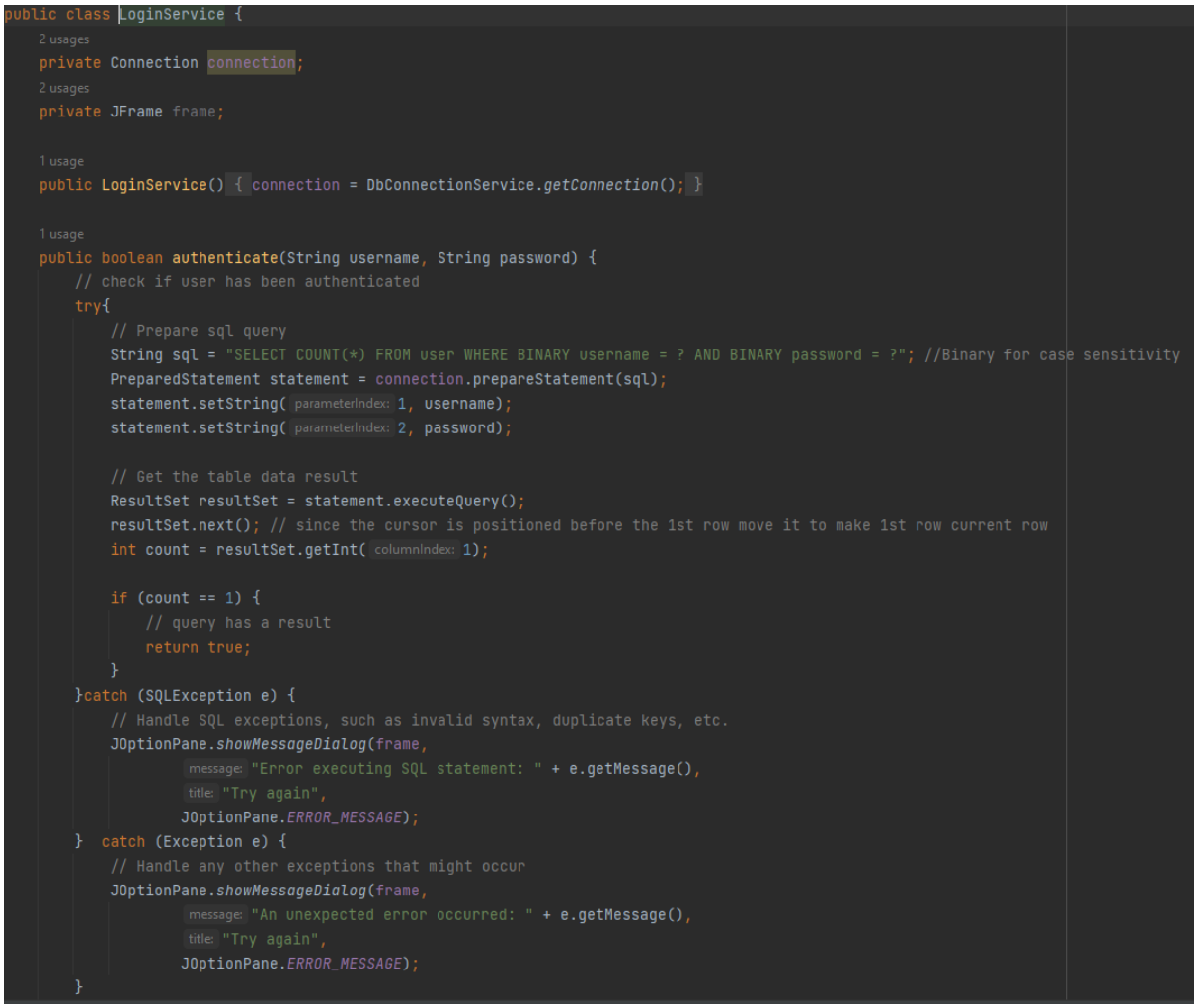


key: * = *Primary key*

FK = *Foreign Key*

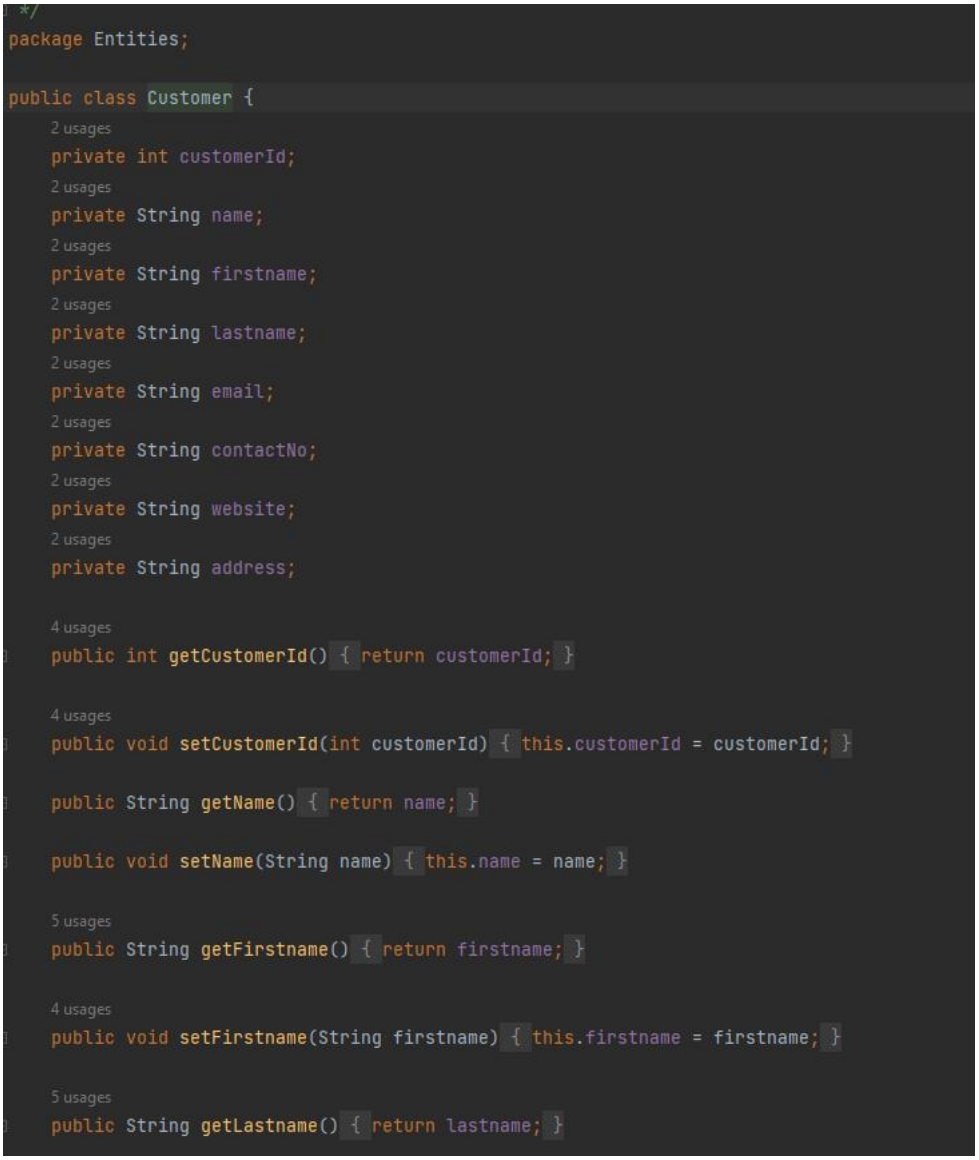
INTERESTING CODE SNIPET

1. ERROR HANDLING
   The program includes several error handling instances from database connection, user inputs and pattern validations. Example code in my Login service class this is how errors are handled and it applies across all service classes in the program and Jpane is used to display the errors to the user



```
public class LoginService {
    2 usages
    private Connection connection;
    2 usages
    private JFrame frame;

    1 usage
    public LoginService() { connection = DbConnectionService.getConnection(); }

    1 usage
    public boolean authenticate(String username, String password) {
        // check if user has been authenticated
        try{
            // Prepare sql query
            String sql = "SELECT COUNT(*) FROM user WHERE BINARY username = ? AND BINARY password = ?"; //Binary for case sensitivity
            PreparedStatement statement = connection.prepareStatement(sql);
            statement.setString( parameterIndex: 1, username);
            statement.setString( parameterIndex: 2, password);

            // Get the table data result
            ResultSet resultSet = statement.executeQuery();
            resultSet.next(); // since the cursor is positioned before the 1st row move it to make 1st row current row
            int count = resultSet.getInt( columnIndex: 1);

            if (count == 1) {
                // query has a result
                return true;
            }
        }catch (SQLException e) {
            // Handle SQL exceptions, such as invalid syntax, duplicate keys, etc.
            JOptionPane.showMessageDialog(frame,
                    message: "Error executing SQL statement: " + e.getMessage(),
                    title: "Try again",
                    JOptionPane.ERROR_MESSAGE);
        }  catch (Exception e) {
            // Handle any other exceptions that might occur
            JOptionPane.showMessageDialog(frame,
                    message: "An unexpected error occurred: " + e.getMessage(),
                    title: "Try again",
                    JOptionPane.ERROR_MESSAGE);
        }
```

2. CUSTOM ENTITY DEFINING CLASS
   Invoices, customers and products are stored in a specific structure there to ensure that the data sent to the database matches the data that will be received. Example is the Customer Entity:



```
*/
package Entities;

public class Customer {
    2 usages
    private int customerId;
    2 usages
    private String name;
    2 usages
    private String firstname;
    2 usages
    private String lastname;
    2 usages
    private String email;
    2 usages
    private String contactNo;
    2 usages
    private String website;
    2 usages
    private String address;

    4 usages
    public int getCustomerId() { return customerId; }

    4 usages
    public void setCustomerId(int customerId) { this.customerId = customerId; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    5 usages
    public String getFirstname() { return firstname; }

    4 usages
    public void setFirstname(String firstname) { this.firstname = firstname; }

    5 usages
    public String getLastname() { return lastname; }
```

Then this how it is used in the Customer service class

```
10 usages
public CustomerService() { connection = DbConnectionService.getConnection(); }

2 usages
public Customer addCustomer(String name, String firstname, String lastname, String email, String contactNo, String website, String address){
    Customer customer = new Customer();
    try{
        int deleted = 0; // new customer deleted attribute set to 0
        Statement smt = connection.createStatement();
        // get id
        String query = "SELECT customerId FROM customer ORDER BY customerId DESC LIMIT 1";
        ResultSet resultSet = smt.executeQuery(query);

        // get last customer entered id if no customer set it to zero
        int lastCustomerId = resultSet.next() ? resultSet.getInt( columnLabel: "customerId") : 0;

        // set new customerId
        int newCustomerId = lastCustomerId + 1;

        // Insert new customer to database
        String sql = "INSERT INTO customer (customerId, name, firstname, lastname, email, contactNo, website, address, deleted)" +
                "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";
        PreparedStatement statement = connection.prepareStatement(sql);
        statement.setInt( parameterIndex: 1, newCustomerId);
        statement.setString( parameterIndex: 2, name);
        statement.setString( parameterIndex: 3, firstname);
        statement.setString( parameterIndex: 4, lastname);
        statement.setString( parameterIndex: 5, email);
        statement.setString( parameterIndex: 6, contactNo);
        statement.setString( parameterIndex: 7, website);
        statement.setString( parameterIndex: 8, address);
        statement.setInt( parameterIndex: 9, deleted);

        int rowsInserted = statement.executeUpdate();
        if (rowsInserted > 0) {
            customer.setCustomerId(newCustomerId);
            customer.setName(name);
            customer.setFirstname(firstname);
```

This then returns a customer in the format that matches the structure of the database. Also each primary key is generated manually as follows

```
int deleted = 0; // new customer deleted attribute set to 0
Statement smt = connection.createStatement();
// get id
String query = "SELECT customerId FROM customer ORDER BY customerId DESC LIMIT 1";
ResultSet resultSet = smt.executeQuery(query);

// get last customer entered id if no customer set it to zero
int lastCustomerId = resultSet.next() ? resultSet.getInt( columnLabel: "customerId") : 0;

// set new customerId
int newCustomerId = lastCustomerId + 1;

// Insert new customer to database
String sql = "INSERT INTO customer (customerId, name, firstname, lastname, email, contactNo, website, address, deleted)" +
        "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";
PreparedStatement statement = connection.prepareStatement(sql);
statement.setInt( parameterIndex: 1, newCustomerId);
```

3. DATA VALIDATION

   There is utils class in the helper package that ised to validate data entries. has static methods to avoid instantiating the class each time we need to validate data

```java
package Helper;

public class Utils {
    4 usages
    public static boolean validateEmail(String email){
        String regex = "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$"; // email pattern
        return email.matches(regex);
    }

    4 usages
    public static boolean validateContact(String contactNo){
        // 0871234561
        String regex = "^[0-9]{1,15}$"; // 15 digits max
        return contactNo.matches(regex);
    }

    3 usages
    public static boolean validateUrl(String website){
        // www.example.com
        String regex = "^(https?://)?[a-zA-Z0-9-]+(\\.[a-zA-Z0-9-]+)*(/[a-zA-Z0-9-]+)*$";
        return website.matches(regex);
    }

    4 usages
    public static boolean validateName(String name) {
        //String
        String regex = "^[a-zA-Z\\s'-]+$";
        return name.matches(regex);
    }

    2 usages
    public  static boolean validatePrice(String price) {
        //Double
        String regex = "^\\d+(\\.\\d{1,2})?$";
        return price.matches(regex);
    }
```

   Use

```
public void addCustomer() {
    Components.Customer.AddCustomer
    public void addCustomer()
    c00290922
                                    getText();
                                    ld.getText();
                                    .getText();
                                    ld.getText();
    String contactNo = customerContactTxtField.getText();
    String website = customerWebsiteTxtField.getText();
    String address = addressTxtArea.getText();

    try{
        if (name.isEmpty() || email.isEmpty() || contactNo.isEmpty() || firstname.isEmpty() ||
                lastname.isEmpty() || website.isEmpty() || address.isEmpty()) {
            JOptionPane.showMessageDialog(frame,
                    message: "Please Fill all Fields",
                    title: "Try again",
                    JOptionPane.ERROR_MESSAGE);
            return;
        }

        if(!Utils.validateEmail(email)){
            JOptionPane.showMessageDialog(frame,
                    message: "Please Enter Email Address In Correct Format",
                    title: "Try again",
                    JOptionPane.ERROR_MESSAGE);
            return;
        }

        if(!Utils.validateContact(contactNo)){
            JOptionPane.showMessageDialog(frame,
                    message: "Please Enter Contact number in correct format. Only digits, no spaces or + symbol",
                    title: "Try again",
                    JOptionPane.ERROR_MESSAGE);
            return;
        }
```

4. DATABASE INTEGRATION:
   There is a separate database connection service Note that we use static variable and method here. We only need one instance of the connection object to ensure that only one connection is made to the database to avoid connection conflicts. Also the close connection is static so that we can call it without creating an instance of the class

```
public class DbConnectionService {

    6 usages
    private static Connection connection = null;

    5 usages
    public static Connection getConnection() {
        try {
            if (connection == null || connection.isClosed()) {
                // Create a new database connection
                Class.forName( className: "com.mysql.cj.jdbc.Driver");
                String url = "jdbc:mysql://localhost/c00290922";
                String user = "root";
                String password = "";
                connection = DriverManager.getConnection(url, user, password);
            }
        } catch (SQLException | ClassNotFoundException e) {
            e.printStackTrace();
        }

        return connection;
    }

    public static void closeConnection() {
        // Close the database connection
        try {
            if (connection != null) {
                connection.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Use

```
try{
    CustomerService customerService = new CustomerService();
    boolean updated = customerService.updateCustomer(customer);
    DbConnectionService.closeConnection();

    if(updated) {
        JOptionPane.showMessageDialog(frame,
                message: "Customer " + name + " has been successfully update",
                title: "Success",
                JOptionPane.PLAIN_MESSAGE);
        frame.dispose();
        UpdateCustomer updateCustomer = new UpdateCustomer();
    } else {
        JOptionPane.showMessageDialog(frame,
                message: "Error Updating Customer ",

    }
}catch (Exceptio
    e.printStack
}
```

Candidates for method call **JOptionPane.showMessageDialog(frame, "Error Updating Customer ", "Try again", JOptionPane.ERROR_MESSAGE)** are:

void showMessageDialog(Component, Object)
void showMessageDialog(Component, Object, String, int)
void showMessageDialog(Component, Object, String, int, Icon)

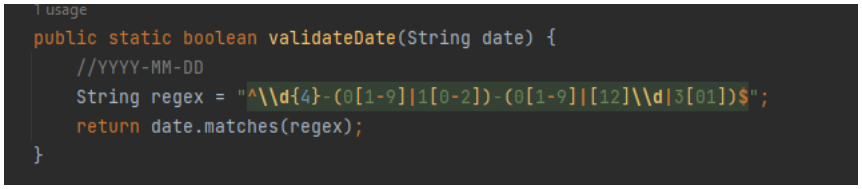
Each service class starts a connection to the database when instantiated hence we close it after performing transaction

5. INNER JOIN
   When retrieving invoice details an inner join to get invoice number and also the customer details such as name and email as customerId is the foreign key in that table

```
1 usage
public List<InvCust> getInvoices() {
    List<InvCust> invCusts = new ArrayList<>();
    try{
        String sql = "SELECT invoice.invoiceId, invoice.dateCreated, invoice.amountDue, customer.name, customer.email " +
                "FROM invoice " +
                "INNER JOIN customer ON invoice.customerId = customer.customerId " +
                "WHERE invoice.customerId = customer.customerId";
        PreparedStatement statement = connection.prepareStatement(sql);
        ResultSet rs = statement.executeQuery();

        while (rs.next()) {
            InvCust invCust = new InvCust();
            invCust.setInvoiceId(rs.getInt( columnLabel: "invoiceId"));
            invCust.setDateCreated(String.valueOf(rs.getDate( columnLabel: "dateCreated")));
            invCust.setAmountDue(rs.getDouble( columnLabel: "amountDue"));
            invCust.setName(rs.getString( columnLabel: "name"));
            invCust.setEmail(rs.getString( columnLabel: "email"));
            invCusts.add(invCust);
            // display the retrieved data in a Java Swing component, such as a table or list
        }
    }catch (SQLException e) {
        // Handle SQL exceptions, such as invalid syntax, duplicate keys, etc.
        JOptionPane.showMessageDialog(frame,
                message: "Error executing SQL statement: " + e.getMessage(),
                title: "Try again",
                JOptionPane.ERROR_MESSAGE);

    }  catch (Exception e) {
        // Handle any other exceptions that might occur
        JOptionPane.showMessageDialog(frame,
                message: "An unexpected error occurred: " + e.getMessage(),
                title: "Try again",
                JOptionPane.ERROR_MESSAGE);
    }

    return invCusts;
}
```

UNIT TESTS

In the utils class there is a static method that validates date format



```
1 usage
public static boolean validateDate(String date) {
    //YYYY-MM-DD
    String regex = "^\\d{4}-(0[1-9]|1[0-2])-(0[1-9]|[12]\\d|3[01])$";
    return date.matches(regex);
}
```

The corresponding unit Test to validatesDate

```
import org.junit.Test;

import static org.junit.Assert.assertTrue;

import static org.junit.Assert.assertFalse;


public class DateValidatorTest {

    @Test

    public void testValidateDate() {

        assertTrue(DateValidator.validateDate("2022-03-31"));

        assertFalse(DateValidator.validateDate("2022-02-31"));

        assertFalse(DateValidator.validateDate("2022-04-31"));

        assertFalse(DateValidator.validateDate("2022-13-01"));

        assertFalse(DateValidator.validateDate("2022-01-32"));

        assertFalse(DateValidator.validateDate("2022-01-0"));

    }

}
```