МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №10

Специальность ПО-5(о)

Выполнил

Д. С. Бриштен,

студент группы ПО-5

Проверил

А. А. Крощенко

ст. преп. каф. ИИТ

Брест 2022

Цель работы: *приобрести практические навыки разработки многооконных приложений на JavaFX для работы с базами данных*.

Вариант 3

**Задание.** На основе БД, разработанной в лабораторной работе №9, реализовать многооконное приложение-клиент, позволяющее выполнять основные операции над таблицей в БД.

Для демонстрации работы были выбраны две таблицы из базы данных: **products** – таблица с *внешним ключом*, который ссылается на другую таблицу **manufacturers**.

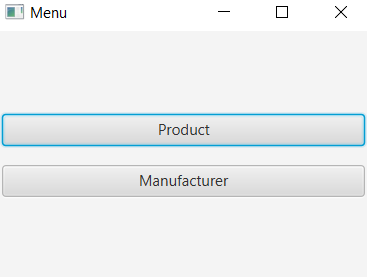
 

Рисунок 1.1 – Меню Рисунок 1.2 – Manufacturers

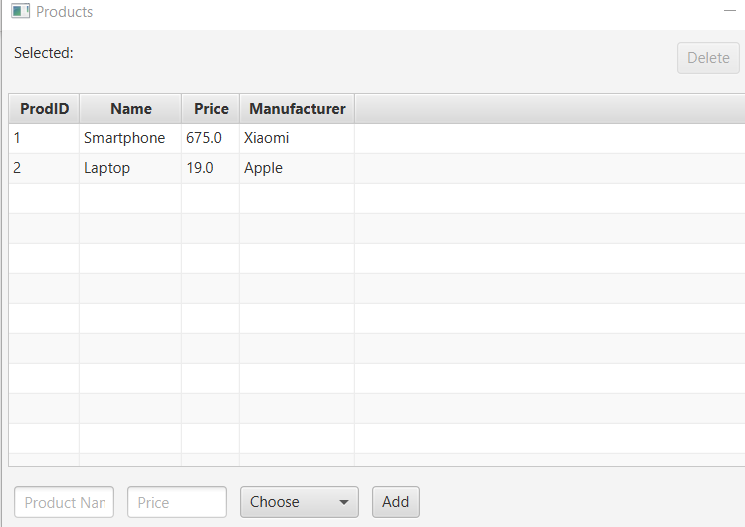


Рисунок 1.3 – Products

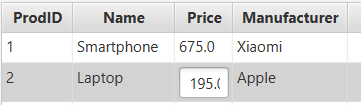


Рисунок 1.4 - Редактирование

Удалим Laptop и добавим TV:

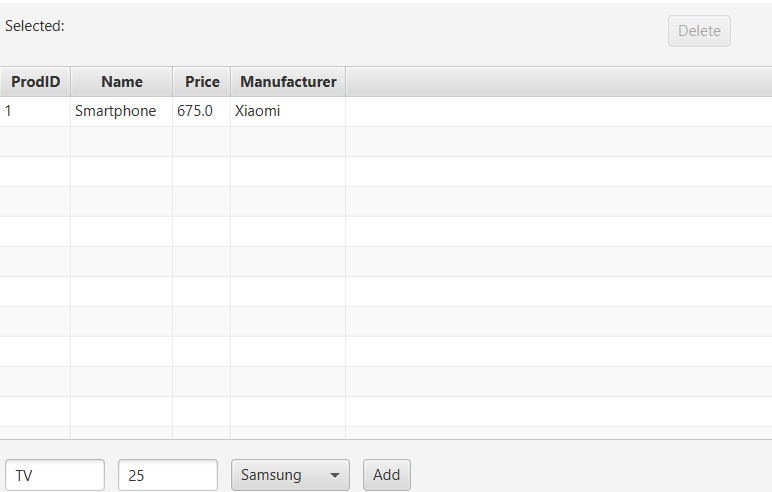


Рисунок 1.5 – Удаление и заполнение форм



Рисунок 1.6 – Добавленный продукт

**Код программы:**

**Main.java**

public class Main extends Application {  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
 @Override  
 public void start(Stage primaryStage) {  
 Stage stage = new Stage();  
 final Button productButton = new Button("Product");  
 productButton.setMinWidth(290);  
 productButton.setOnAction(actionEvent -> {  
 try {  
 stage.close();  
 ProductView productView = new ProductView();  
 productView.start(stage);  
 } catch (SQLException e) {  
  
 e.printStackTrace();  
 }  
 });  
  
 final Button manufacturerButton = new Button("Manufacturer");  
 manufacturerButton.setMinWidth(290);  
 manufacturerButton.setOnAction(actionEvent -> {  
 try {  
 stage.close();  
 ManufacturerView manufacturerView = new ManufacturerView();  
 manufacturerView.start(stage);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 });  
  
 VBox root = new VBox();  
 root.setPadding(new Insets(5));  
 root.setSpacing(15);  
 root.setAlignment(Pos.*CENTER*);  
 root.getChildren().addAll(productButton, manufacturerButton);  
 primaryStage.setTitle("Menu");  
 Scene scene = new Scene(root, 300, 200);  
 primaryStage.setScene(scene);  
 primaryStage.show();  
 }  
}

**Manufacturer.java**

public class Manufacturer {  
 private SimpleIntegerProperty ID;  
 private SimpleStringProperty name;  
 private SimpleStringProperty establishDate;  
  
 public Manufacturer(int ID, String name, String establishDate) {  
 this.ID = new SimpleIntegerProperty(ID);  
 this.name = new SimpleStringProperty(name);  
 this.establishDate = new SimpleStringProperty(establishDate);  
 }  
  
 public int getID() {return ID.get();}  
 public String getName() {return name.get();}  
 public String getEstablishDate() {return establishDate.get();}  
  
 public void setID(int value) {ID.set(value);}  
 public void setName(String value) {name.set(value);}  
 public void setEstablishDate(String value) {establishDate.set(value);}  
}

**Product.java**

public class Product {  
 private SimpleIntegerProperty ID;  
 private SimpleStringProperty name;  
 private SimpleDoubleProperty price;  
 private Manufacturer manufacturer;  
  
 public Product(int ID, String name, double price, Manufacturer manufacturer) {  
 this.ID = new SimpleIntegerProperty(ID);  
 this.name = new SimpleStringProperty(name);  
 this.price = new SimpleDoubleProperty(price);  
 this.manufacturer = manufacturer;  
 }  
  
 public int getID() {return ID.get();}  
 public String getName() {return name.get();}  
 public double getPrice() {return price.get();}  
 public Manufacturer getManufacturer() {return this.manufacturer;}  
  
 public void setID(int value) {ID.set(value);}  
 public void setName(String value) {name.set(value);}  
 public void setPrice(double value) {price.set(value);}  
 public void setManufacturer(Manufacturer manufacturer) {this.manufacturer = manufacturer;}  
}

**ManufacturerDB.java**

public class ManufacturerDB {  
 private Connection dbConnection;  
 private Statement statement;  
  
 public ManufacturerDB() {  
 try {  
 dbConnection = *getDBConnection*();  
 statement = dbConnection.createStatement();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private static Connection getDBConnection() {  
 final String DB\_DRIVER, DB\_CONNECTION, DB\_USER, DB\_PASSWORD;  
 DB\_DRIVER = "com.mysql.cj.jdbc.Driver";  
 DB\_CONNECTION = "jdbc:mysql://127.0.0.1:3306/java";  
 DB\_USER = "root";  
 DB\_PASSWORD = "";  
 Connection dbconnection = null;  
  
 try {  
 Class.*forName*(DB\_DRIVER);  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
 try {  
 dbconnection = DriverManager.*getConnection*(DB\_CONNECTION, DB\_USER, DB\_PASSWORD);  
 } catch (SQLException e) {  
 System.*out*.println(e.getMessage());  
 }  
 return dbconnection;  
 }  
  
 public List selectFromTable() throws SQLException {  
 int ID;  
 String name, establishDate;  
 String selectTableSQL = "SELECT \* FROM `manufacturers`";  
 ResultSet rs = statement.executeQuery(selectTableSQL);  
 List<Manufacturer> manufacturers = new ArrayList<Manufacturer>();  
   
 while (rs.next()) {  
 ID = rs.getInt(1);  
 name = rs.getString("name");  
 establishDate = rs.getString(3);  
 manufacturers.add(new Manufacturer(ID, name, establishDate));  
 }  
 return manufacturers;  
 }  
}

**ProductDB.java**

private static Statement *statement*;  
 private static Connection *dbConnection*;  
  
 public ProductDB() {  
 try {  
 *dbConnection* = *getDBConnection*();  
 *statement* = *dbConnection*.createStatement();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private static Connection getDBConnection() {  
 final String DB\_DRIVER, DB\_CONNECTION, DB\_USER, DB\_PASSWORD;  
 DB\_DRIVER = "com.mysql.cj.jdbc.Driver";  
 DB\_CONNECTION = "jdbc:mysql://127.0.0.1:3306/java";  
 DB\_USER = "root";  
 DB\_PASSWORD = "";  
 Connection dbconnection = null;  
  
 try {  
 Class.*forName*(DB\_DRIVER);  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
 try {  
 dbconnection = DriverManager.*getConnection*(DB\_CONNECTION, DB\_USER, DB\_PASSWORD);  
 } catch (SQLException e) {  
 System.*out*.println(e.getMessage());  
 }  
 return dbconnection;  
 }  
  
 public List selectFromTable() throws SQLException {  
 int ID, manID;  
 String name;  
 double price;   
 String selectTableSQL = "SELECT \* FROM `products`";  
 ResultSet rs = *statement*.executeQuery(selectTableSQL);  
 List<Product> products = new ArrayList<Product>();  
  
 while (rs.next()) {  
 ID = rs.getInt(1);  
 name = rs.getString("name");  
 price = rs.getDouble(3);  
 manID = rs.getInt("man\_id");  
 PreparedStatement pr = *dbConnection*.prepareStatement("SELECT " +  
 "man\_id, name, establish\_date FROM `manufacturers` where man\_id=?");  
 pr.setInt(1, manID);  
 ResultSet rs2 = pr.executeQuery();  
 Manufacturer manufacturer = null;  
 if (rs2.next()) {  
 manufacturer = new Manufacturer(rs2.getInt(1),  
 rs2.getString(2), rs2.getString(3));  
 }  
 products.add(new Product(ID, name, price, manufacturer));  
 }  
 return products;  
 }  
  
 public static void addProduct(String name, Double price, Manufacturer manufacturer) throws SQLException {  
 PreparedStatement preparedStatement = *dbConnection*.prepareStatement("INSERT " +  
 "`products`(name, price, man\_id) VALUES (?, ?, ?)");  
 preparedStatement.setString(1, name);  
 preparedStatement.setDouble(2, price);  
 preparedStatement.setInt(3, manufacturer.getID());  
 preparedStatement.executeUpdate();  
 }  
  
 public static void editProduct(Product product) throws SQLException {  
 PreparedStatement preparedStatement = *dbConnection*.prepareStatement("UPDATE " +  
 "`products` SET name=?, price=?, man\_id=? WHERE pr\_id=?");  
 preparedStatement.setString(1, product.getName());  
 preparedStatement.setDouble(2, product.getPrice());  
 preparedStatement.setInt(3, product.getManufacturer().getID());  
 preparedStatement.setInt(4, product.getID());  
 preparedStatement.executeUpdate();  
 }  
  
 public static void deleteProduct(int ID) throws SQLException {  
 PreparedStatement preparedStatement = *dbConnection*.prepareStatement("DELETE " +  
 "FROM `products` WHERE pr\_id=?");  
 preparedStatement.setInt(1, ID);  
 preparedStatement.executeUpdate();  
 }  
}

**ManufacturerView.java**

public class ManufacturerView extends Application {  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
  
 @Override  
 public void start(Stage primaryStage) throws SQLException {  
 ManufacturerDB manufacturerDB = new ManufacturerDB();  
 List<Manufacturer> manufacturers = manufacturerDB.selectFromTable();  
 ObservableList<Manufacturer> manufacturerObservableList = FXCollections.*observableArrayList*(manufacturers);  
 TableView<Manufacturer> table = new TableView<Manufacturer>(manufacturerObservableList);  
 table.setMinWidth(500);  
 table.setEditable(false);  
  
 TableColumn<Manufacturer, Integer> manufacturerIdColumn = new TableColumn<Manufacturer, Integer>("ManID");  
 manufacturerIdColumn.setCellValueFactory(new PropertyValueFactory<Manufacturer, Integer>("ID"));  
 manufacturerIdColumn.setEditable(false);  
  
 TableColumn<Manufacturer, String> manufacturerName = new TableColumn<Manufacturer, String>("Name");  
 manufacturerName.setCellValueFactory(new PropertyValueFactory<Manufacturer, String>("name"));  
 manufacturerName.setCellFactory(TextFieldTableCell.<Manufacturer>*forTableColumn*());  
  
 TableColumn<Manufacturer, String> establishDate = new TableColumn<Manufacturer, String>("Date");  
 establishDate.setCellValueFactory(new PropertyValueFactory<Manufacturer, String>("establishDate"));  
 establishDate.setCellFactory(TextFieldTableCell.<Manufacturer>*forTableColumn*());  
  
 table.getColumns().addAll(manufacturerIdColumn, manufacturerName, establishDate);  
  
 VBox root = new VBox();  
 root.setPadding(new Insets(5));  
 root.setSpacing(10);  
 root.getChildren().addAll(table);  
 primaryStage.setTitle("Manufacturers");  
 Scene scene = new Scene(root, 600, 400);  
 primaryStage.setScene(scene);  
 primaryStage.show();  
 }  
}

**ProductView.java**

public class ProductView extends Application {  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
 @Override  
 public void start(Stage primaryStage) throws SQLException {  
 ProductDB productDB = new ProductDB();  
 ManufacturerDB manufacturerDB = new ManufacturerDB();  
  
 List<Product> products = productDB.selectFromTable();  
 List<Manufacturer> manufacturers = manufacturerDB.selectFromTable();  
 List<String> manufacturersNames = new ArrayList<>();  
  
 for (Manufacturer manufacturer : manufacturers) {  
 manufacturersNames.add(manufacturer.getName());  
 }  
  
 ObservableList<Product> productObservableList = FXCollections.*observableArrayList*(products);  
 ObservableList<String> manufacturersList = FXCollections.*observableArrayList*(manufacturersNames);  
   
 TableView<Product> table = new TableView<Product>(productObservableList);  
  
 table.setMinWidth(600);  
 table.setEditable(true);  
 Label label = new Label();  
 label.setText("Selected: ");  
 label.setMinWidth(520);  
  
 TableColumn<Product, Integer> productIdColumn = new TableColumn<Product, Integer>("ProdID");  
 productIdColumn.setCellValueFactory(new PropertyValueFactory<Product, Integer>("ID"));  
 productIdColumn.setEditable(false);  
  
 TableColumn<Product, String> productName = new TableColumn<Product, String>("Name");  
 productName.setCellValueFactory(new PropertyValueFactory<Product, String>("name"));  
 productName.setCellFactory(TextFieldTableCell.<Product>*forTableColumn*());  
 productName.setOnEditCommit((TableColumn.CellEditEvent<Product, String> event) -> {  
 TablePosition<Product, String> position = event.getTablePosition();  
 String name = event.getNewValue();  
 if (!name.equals("")) {  
 int row = position.getRow();  
 Product product = event.getTableView().getItems().get(row);  
 product.setName(name);  
 try {  
 ProductDB.*editProduct*(product);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 });  
  
 TableColumn<Product, Double> price = new TableColumn<Product, Double>("Price");  
 price.setCellValueFactory(new PropertyValueFactory<Product, Double>("price"));  
 price.setCellFactory(TextFieldTableCell.<Product, Double>*forTableColumn*(new DoubleStringConverter()));  
 price.setOnEditCommit((TableColumn.CellEditEvent<Product, Double> event) ->  
 {  
 TablePosition<Product, Double> position = event.getTablePosition();  
 Double productPrice = event.getNewValue();  
 if (productPrice >= 0.0) {  
 int row = position.getRow();  
 Product product = event.getTableView().getItems().get(row);  
 product.setPrice(productPrice);  
 try {  
 ProductDB.*editProduct*(product);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 });  
   
 TableColumn<Product, String> manufacturerName = new TableColumn<Product, String>("Manufacturer");  
 manufacturerName.setCellValueFactory(param -> {  
 Product product = param.getValue();  
 Manufacturer manufacturer = product.getManufacturer();  
 return new SimpleObjectProperty<String>(manufacturer.getName());  
 });  
 manufacturerName.setCellFactory(ComboBoxTableCell.*forTableColumn*(manufacturersList));  
 manufacturerName.setOnEditCommit((TableColumn.CellEditEvent<Product, String>  
 event) -> {  
 TablePosition<Product, String> position = event.getTablePosition();  
 String newRenter = event.getNewValue();  
 int manufacturerIdInList = 0;  
 for (int i = 0; i < manufacturersNames.size(); i++) {  
 if (newRenter.equals(manufacturersNames.get(i))) {  
 manufacturerIdInList = i;  
 break;  
 }  
 }  
 Manufacturer manufacturer = manufacturers.get(manufacturerIdInList);  
  
 int row = position.getRow();  
 Product product = event.getTableView().getItems().get(row);  
 product.setManufacturer(manufacturer);  
 try {  
 ProductDB.*editProduct*(product);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 });  
  
 table.getColumns().addAll(productIdColumn, productName, price, manufacturerName);  
  
 final ComboBox<String> manufacturerComboBox = new ComboBox<>(manufacturersList);  
 manufacturerComboBox.setValue("Choose");  
 final Button addButton = new Button("Add");  
 final Button deleteButton = new Button("Delete");  
  
 TableView.TableViewSelectionModel<Product> selectionModel = table.getSelectionModel();  
 selectionModel.selectedItemProperty().addListener((observableValue, oldProduct, newProduct) -> {  
 if (newProduct != null) {  
 label.setText("Selected: " + newProduct.getName());  
 deleteButton.setDisable(false);  
 }  
 });  
  
 final TextField addName = new TextField();  
 final TextField addPrice = new TextField();  
 addName.setPromptText("Product Name");  
 addName.setMaxWidth(productName.getPrefWidth());  
 addPrice.setPromptText("Price");  
 addPrice.setMaxWidth(price.getPrefWidth());  
 addButton.setOnAction(e -> {  
 if (!addName.getText().isEmpty() && !manufacturerComboBox.getValue().equals("Choose") &&  
 !addPrice.getText().isEmpty() && Double.*parseDouble*(addPrice.getText()) >= 0.0) {  
 try {  
 String newManufacturer = manufacturerComboBox.getValue();  
 int manufacturerIdInList = 0;  
 for (int i = 0; i < manufacturersNames.size(); i++) {  
 if (newManufacturer.equals(manufacturersNames.get(i))) {  
 manufacturerIdInList = i;  
 break;  
 }  
 }  
 Manufacturer manufacturer = manufacturers.get(manufacturerIdInList);  
 ProductDB.*addProduct*(addName.getText(), Double.*valueOf*(addPrice.getText()), manufacturer);  
 start(primaryStage);  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
 });  
  
 deleteButton.setDisable(true);  
 deleteButton.setOnAction(actionEvent -> {  
 if (table.getSelectionModel().getSelectedItem() != null) {  
 Product selectedItem = table.getSelectionModel().getSelectedItem();  
 try {  
 ProductDB.*deleteProduct*(selectedItem.getID());  
 start(primaryStage);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 });  
  
 VBox root = new VBox();  
 HBox deleteBox = new HBox();  
 HBox addBox = new HBox();  
 root.setPadding(new Insets(5));  
 root.setSpacing(10);  
 root.getChildren().addAll(deleteBox, table, addBox);  
 deleteBox.setPadding(new Insets(5));  
 deleteBox.setSpacing(10);  
 deleteBox.getChildren().addAll(label, deleteButton);  
 addBox.setPadding(new Insets(5));  
 addBox.setSpacing(10);  
 addBox.getChildren().addAll(addName, addPrice, manufacturerComboBox, addButton);  
 primaryStage.setTitle("Products");  
 Scene scene = new Scene(root, 700, 400);  
 primaryStage.setScene(scene);  
 primaryStage.show();  
 }  
}

Вывод: Создано приложение для работы с таблицами баз данных, в котором реализован пользовательский интерфейс, позволяющий выполнять основные операции над таблицей.