

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №10

Специальность ПО5

Выполнил:
А.А. Игнатюк,
студент группы ПО-5

Проверил:
А.А. Крощенко,
ст. преп. кафедры ИИТ,
«__» _____ 2022 г.

Брест 2022

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

Задание.

На основе БД, разработанной в лабораторной работе №9, реализовать многооконное приложение-клиент, позволяющее выполнять основные операции над таблицей в БД (добавление, удаление, модификацию данных).

Основные требования к приложению:

- Для отображения выбирать таблицу с внешними ключами;
- Осуществлять вывод основных данных в табличном представлении;
- При выводе краткого представления записи в таблице (т.е. если выводятся не все поля), по щелчку мышкой на запись осуществлять вывод всех полей в подготовленные компоненты на форме;
- Для всех полей, представленных внешними ключами, выводить их текстовое представление из связанных таблиц (например, таблица-справочник “Времена года” содержит два поля - идентификатор и название сезона, в связанной таблице “Месяц года” есть внешний ключ на таблицу “Времена года”; в этом случае при выводе таблицы “Месяц года” нужно выводить название сезона, а не его идентификатор);
- При выводе предусмотреть упорядочивание по столбцу;
- Реализовать простейший фильтр данных по одному-двум полям;
- При добавлении новых данных в таблицу использовать дополнительное окно для ввода;
- При модификации данных можно использовать ту же форму, что и для добавления, но с внесенными актуальными значениями полей;
- При добавлении/модификации выводить варианты значений полей с внешним ключом с помощью выпадающего списка;
- При удалении данных осуществлять удаление записи, на которой в данный момент находится фокус.

Спецификация ввода: Ввод через взаимодействие с элементами интерфейса

Спецификация вывода: Вывод данных из базы в элементы графического интерфейса

Код программы и результаты тестирования:

```

└─ LAB
   └─ .vscode
      └─ src\main
         └─ java
            └─ com\example
               ├── App.java
               ├── CreateController.java
               ├── DB.java
               ├── MainController.java
               ├── Order.java
               ├── UpdateController.java
               └── module-info.java
            └─ resources\com\example
               ├── create.fxml
               ├── main.fxml
               └── update.fxml
            └─ target
               └─ pom.xml
```

Рисунок 1 - Структура проекта.

App.java 2 X

src > main > java > com > example > App.java > ...

```
1  package com.example;
2
3  import javafx.application.Application;
4  import javafx.fxml.FXMLLoader;
5  import javafx.scene.Parent;
6  import javafx.scene.Scene;
7  import javafx.stage.Stage;
8
9  import java.io.IOException;
10
11  /**
12   * JavaFX App
13   */
14  public final class App extends Application {
15      private static Scene scene;
16
17      @Override
18      public final void start(final Stage stage) throws IOException {
19          App.scene = new Scene(App.loadFXML("main"), 640, 480);
20          stage.setResizable(false);
21          stage.setScene(App.scene);
22          stage.show();
23      }
24
25      public final static Parent loadFXML(final String fxml) throws IOException {
26          FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource(fxml + ".fxml"));
27          return fxmlLoader.load();
28      }
29
30      Run | Debug
31      public final static void main(final String[] args) {
32          launch();
33      }
34  }
```

Рисунок 2 - Исходный код файла App.java.

CreateController.java X

src > main > java > com > example > CreateController.java > ...

```
1  package com.example;
2
3  import java.net.URL;
4  import java.sql.ResultSet;
5  import java.sql.SQLException;
6  import java.util.ResourceBundle;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9
10 import javafx.collections.FXCollections;
11 import javafx.collections.ObservableList;
12 import javafx.fxml.FXML;
13 import javafx.fxml.Initializable;
14 import javafx.scene.control.ChoiceBox;
15
16 public final class CreateController implements Initializable {
17     @FXML
18     private ChoiceBox<String> customersChoiceBox;
19     @FXML
20     private ChoiceBox<String> workersChoiceBox;
21     @FXML
22     private ChoiceBox<String> cpuChoiceBox;
23     @FXML
24     private ChoiceBox<String> gpuChoiceBox;
25
26     private DB db = null;
27
28     @Override
29     public void initialize(URL arg0, ResourceBundle arg1) {
30         try {
31             this.db = new DB();
32
33             ObservableList<String> customersList = FXCollections.observableArrayList();
34             ObservableList<String> workersList = FXCollections.observableArrayList();
35             ObservableList<String> cpuList = FXCollections.observableArrayList();
36             ObservableList<String> gpuList = FXCollections.observableArrayList();
37
38             ResultSet customersSet = this.db.getAll(DB.CUSTOMERS_TABLE);
39             ResultSet workersSet = this.db.getAll(DB.WORKERS_TABLE);
40             ResultSet cpuSet = this.db.getAll(DB.CPU_TABLE);
41             ResultSet gpuSet = this.db.getAll(DB.GPU_TABLE);
42
43             while (customersSet.next() && workersSet.next() && cpuSet.next() && gpuSet.next()) {
44                 customersList.add(customersSet.getString("email"));
45                 workersList.add(workersSet.getString("email"));
46                 cpuList.add(cpuSet.getString("name"));
47                 gpuList.add(gpuSet.getString("name"));
48             }
49
50             this.customersChoiceBox.setItems(customersList);
51             this.workersChoiceBox.setItems(workersList);
52             this.cpuChoiceBox.setItems(cpuList);
53             this.gpuChoiceBox.setItems(gpuList);
54         } catch (final SQLException exception) {
55             Logger.getLogger(CreateController.class.getName()).log(Level.SEVERE, msg: null, exception);
56         } catch (final Exception exception) {
57             Logger.getLogger(CreateController.class.getName()).log(Level.SEVERE, msg: null, exception);
58         }
59     }
60
61     @FXML
62     private final void create() {
63         if (this.customersChoiceBox.getSelectionModel().isEmpty()
64             || this.workersChoiceBox.getSelectionModel().isEmpty()
65             || this.cpuChoiceBox.getSelectionModel().isEmpty()
66             || this.gpuChoiceBox.getSelectionModel().isEmpty()) {
67             return;
68         }
69
70         this.db.addOrder(new Order(id: null,
71                                   this.customersChoiceBox.getValue(),
72                                   this.workersChoiceBox.getValue(),
73                                   date: null,
74                                   price: 0.0,
75                                   this.cpuChoiceBox.getValue(),
76                                   this.gpuChoiceBox.getValue()));
77     }
78
79 }
```

Рисунок 3 - Исходный код файла CreateController.java.

DB.java 4 X

src > main > java > com > example > DB.java > ...

```
1  package com.example;
2
3  import java.sql.Connection;
4  import java.sql.DriverManager;
5  import java.sql.ResultSet;
6  import java.sql.SQLException;
7  import java.sql.Statement;
8  import java.util.logging.Level;
9  import java.util.logging.Logger;
10
11 public final class DB {
12     private final static String DATABASE_NAME = "lab";
13     public final static String CUSTOMERS_TABLE = "customers";
14     public final static String WORKERS_TABLE = "workers";
15     public final static String CPU_TABLE = "cpu";
16     public final static String GPU_TABLE = "gpu";
17     public final static String ORDERS_TABLE = "orders";
18
19     private Connection connection = null;
20
21     public DB() {
22         try {
23             final String HOST = "localhost";
24             final String PORT = "3307";
25             final String USERNAME = "root";
26             final String PASSWORD = "secret";
27             final String URL = "jdbc:mysql://" + HOST + ':' + PORT;
28
29             Class.forName("com.mysql.cj.jdbc.Driver");
30             this.connection = DriverManager.getConnection(URL + "?user=" + USERNAME + "&password=" + PASSWORD);
31             this.prepare();
32             // this.fill();
33         } catch (final ClassNotFoundException exception) {
34             Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
35         } catch (final SQLException exception) {
36             Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
37         } catch (final Exception exception) {
38             Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
39         }
40     }
41
42     public final void close() {
43         try {
44             this.connection.close();
45         } catch (final SQLException exception) {
46             Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
47         }
48     }
49
50     public final ResultSet getAll(final String table) {
51         try {
52             Statement statement = this.connection.createStatement();
53             statement.closeOnCompletion();
54
55             return statement
56                 .executeQuery(new String(
57                     "SELECT * FROM `" + DB.DATABASE_NAME + "`." + table + "` ORDER BY `id`");
58         } catch (final SQLException exception) {
59             Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
60         }
61
62         return null;
63     }
64 }
```

Рисунок 4 - Исходный код файла DB.java.

Продолжение рисунка 4.

```

65 private final void prepare() {
66     try {
67         final String[] preparation = {
68             new String("CREATE DATABASE IF NOT EXISTS `" + DB.DATABASE_NAME + "`;"),
69             new String("CREATE TABLE IF NOT EXISTS `" + DB.DATABASE_NAME + "`." +
70                 + DB.CUSTOMERS_TABLE
71                 + "` ( `id` INT UNSIGNED NOT NULL AUTO_INCREMENT , `first_name` VARCHAR(64) NOT NULL , "
72                 + "`last_name` VARCHAR(64) NOT NULL , `email` VARCHAR(64) NOT NULL , PRIMARY KEY (`id`) , "
73                 + "UNIQUE (`email`) ) ENGINE = InnoDB;"),
74             new String("CREATE TABLE IF NOT EXISTS `" + DB.DATABASE_NAME + "`." +
75                 + DB.WORKERS_TABLE
76                 + "` ( `id` INT UNSIGNED NOT NULL AUTO_INCREMENT , `first_name` VARCHAR(64) NOT NULL , "
77                 + "`last_name` VARCHAR(64) NOT NULL , `email` VARCHAR(64) NOT NULL , `position` VARCHAR(64) NOT NULL , "
78                 + "PRIMARY KEY (`id`) , UNIQUE (`email`) ) ENGINE = InnoDB;"),
79             new String("CREATE TABLE IF NOT EXISTS `" + DB.DATABASE_NAME + "`." +
80                 + DB.CPU_TABLE
81                 + "` ( `id` INT UNSIGNED NOT NULL AUTO_INCREMENT , `name` VARCHAR(64) NOT NULL , "
82                 + "`price` DOUBLE UNSIGNED NOT NULL , `description` TEXT NULL , PRIMARY KEY (`id`) , UNIQUE (`name`) ) ENGINE = InnoDB;"),
83             new String("CREATE TABLE IF NOT EXISTS `" + DB.DATABASE_NAME + "`." +
84                 + DB.GPU_TABLE
85                 + "` ( `id` INT UNSIGNED NOT NULL AUTO_INCREMENT , `name` VARCHAR(64) NOT NULL , "
86                 + "`price` DOUBLE UNSIGNED NOT NULL , `description` TEXT NULL , PRIMARY KEY (`id`) , UNIQUE (`name`) ) ENGINE = InnoDB;"),
87             new String("CREATE TABLE IF NOT EXISTS `" + DB.DATABASE_NAME + "`." +
88                 + DB.ORDERS_TABLE
89                 + "` ( `id` INT UNSIGNED NOT NULL AUTO_INCREMENT , `customer_id` INT UNSIGNED NOT NULL , "
90                 + "`worker_id` INT UNSIGNED NOT NULL , `date` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP , "
91                 + "`price` DOUBLE UNSIGNED NOT NULL , `cpu_id` INT UNSIGNED NOT NULL , `gpu_id` INT UNSIGNED NOT NULL , "
92                 + "PRIMARY KEY (`id`) , INDEX `customer_id_index` (`customer_id`) , INDEX `worker_id_index` (`worker_id`) , "
93                 + "INDEX `cpu_id_index` (`cpu_id`) , INDEX `gpu_id_index` (`gpu_id`) ) ENGINE = InnoDB;"),
94             new String("ALTER TABLE `" + DB.DATABASE_NAME + "`." + DB.ORDERS_TABLE
95                 + "` ADD FOREIGN KEY ( `customer_id` ) REFERENCES `"
96                 + DB.CUSTOMERS_TABLE
97                 + "` ( `id` ) ON DELETE CASCADE ON UPDATE RESTRICT ;"),
98             new String("ALTER TABLE `" + DB.DATABASE_NAME + "`." + DB.ORDERS_TABLE
99                 + "` ADD FOREIGN KEY ( `worker_id` ) REFERENCES `"
100                 + DB.WORKERS_TABLE
101                 + "` ( `id` ) ON DELETE CASCADE ON UPDATE RESTRICT ;"),
102             new String("ALTER TABLE `" + DB.DATABASE_NAME + "`." + DB.ORDERS_TABLE
103                 + "` ADD FOREIGN KEY ( `cpu_id` ) REFERENCES `" + DB.CPU_TABLE
104                 + "` ( `id` ) ON DELETE CASCADE ON UPDATE RESTRICT ;"),
105             new String("ALTER TABLE `" + DB.DATABASE_NAME + "`." + DB.ORDERS_TABLE
106                 + "` ADD FOREIGN KEY ( `gpu_id` ) REFERENCES `" + DB.GPU_TABLE
107                 + "` ( `id` ) ON DELETE CASCADE ON UPDATE RESTRICT ;"),
108             new String("USE `" + DB.DATABASE_NAME + "`;")
109         };
110
111         Statement statement = this.connection.createStatement();
112         statement.closeOnCompletion();
113
114         for (final String sql : preparation) {
115             statement.executeUpdate(sql);
116         }
117     } catch (final Exception exception) {
118         Logger.getLogger(DB.class.getName()).log(Level.SEVERE, msg: null, exception);
119     }
120 }
121
122 public final void deleteByID(final String table, final Integer id) {
123     try {
124         Statement statement = this.connection.createStatement();
125         statement.closeOnCompletion();
126         statement.executeUpdate(
127             "DELETE FROM `" + DB.DATABASE_NAME + "`." + table + "` WHERE `id` = " + id + ";\");");
128     } catch (final SQLException exception) {
129         Logger.getLogger(DB.class.getName()).log(Level.SEVERE, msg: null, exception);
130     }
131 }
132

```

Продолжение рисунка 4.

```
133 public final ResultSet getOrders(final String customerFilter, final String workerFilter, final Integer id) {
134     try {
135         Statement statement = this.connection.createStatement();
136         statement.closeOnCompletion();
137
138         String sql = new String("SELECT "
139             + "`orders`.`id` , "
140             + "`customers`.`email` AS `customer` , "
141             + "`workers`.`email` AS `worker` , "
142             + "`orders`.`date` , "
143             + "`orders`.`price` , "
144             + "`cpu`.`name` AS `cpu` , "
145             + "`gpu`.`name` AS `gpu` "
146             + "FROM `orders` "
147             + "INNER JOIN `customers` ON `orders`.`customer_id` = `customers`.`id` "
148             + "INNER JOIN `workers` ON `orders`.`worker_id` = `workers`.`id` "
149             + "INNER JOIN `cpu` ON `orders`.`cpu_id` = `cpu`.`id` "
150             + "INNER JOIN `gpu` ON `orders`.`gpu_id` = `gpu`.`id`");
151
152         if (id != null) {
153             sql += " WHERE `orders`.`id` = '" + Integer.toString(id) + "'";
154             return statement.executeQuery(sql);
155         }
156
157         if (customerFilter != null && workerFilter != null) {
158             sql += " WHERE `customers`.`email` = '" + customerFilter + "' AND `workers`.`email` = '" + workerFilter
159                 + "'";
160             return statement.executeQuery(sql);
161         }
162
163         if (customerFilter != null) {
164             sql += " WHERE `customers`.`email` = '" + customerFilter + "'";
165             return statement.executeQuery(sql);
166         }
167
168         if (workerFilter != null) {
169             sql += " WHERE `workers`.`email` = '" + workerFilter + "'";
170             return statement.executeQuery(sql);
171         }
172
173         return statement.executeQuery(sql + ';');
174     } catch (final SQLException exception) {
175         Logger.getLogger(DB.class.getName()).log(Level.SEVERE, msg: null, exception);
176     }
177
178     return null;
179 }
180
181 public final ResultSet getOrderDetails(final Integer id) {
182     try {
183         Statement statement = this.connection.createStatement();
184         statement.closeOnCompletion();
185
186         return statement
187             .executeQuery(new String("SELECT "
188                 + "`customers`.`first_name` AS `customer_first_name` , "
189                 + "`customers`.`last_name` AS `customer_last_name` , "
190                 + "`workers`.`first_name` AS `worker_first_name` , "
191                 + "`workers`.`last_name` AS `worker_last_name` , "
192                 + "`workers`.`position` AS `worker_position` , "
193                 + "`cpu`.`price` AS `cpu_price` , "
194                 + "`cpu`.`description` AS `cpu_description` , "
195                 + "`gpu`.`price` AS `gpu_price` , "
196                 + "`gpu`.`description` AS `gpu_description` "
197                 + "FROM `orders` "
198                 + "INNER JOIN `customers` ON `orders`.`customer_id` = `customers`.`id` "
199                 + "INNER JOIN `workers` ON `orders`.`worker_id` = `workers`.`id` "
200                 + "INNER JOIN `cpu` ON `orders`.`cpu_id` = `cpu`.`id` "
201                 + "INNER JOIN `gpu` ON `orders`.`gpu_id` = `gpu`.`id` "
202                 + "WHERE `orders`.`id` = '" + Integer.toString(id) + "'");
203     } catch (final SQLException exception) {
204         Logger.getLogger(DB.class.getName()).log(Level.SEVERE, msg: null, exception);
205     }
206
207     return null;
208 }
209 }
```

Продолжение рисунка 4.

```
210 public final void addOrder(final Order order) {
211     try {
212         Statement statement = this.connection.createStatement();
213         statement.closeOnCompletion();
214         ResultSet resultSet = null;
215
216         resultSet = statement.executeQuery(
217             "SELECT `customers`.`id` FROM `customers` WHERE `email` = '" + order.getCustomer() + "';");
218         resultSet.next();
219         Integer customerId = resultSet.getInt(columnLabel: "id");
220
221         resultSet = statement
222             .executeQuery("SELECT `workers`.`id` FROM `workers` WHERE `email` = '" + order.getWorker() + "';");
223         resultSet.next();
224         Integer workerId = resultSet.getInt(columnLabel: "id");
225
226         resultSet = statement
227             .executeQuery("SELECT `cpu`.`price` FROM `cpu` WHERE `name` = '" + order.getCpu() + "';");
228         resultSet.next();
229         Double cpuPrice = resultSet.getDouble(columnLabel: "price");
230
231         resultSet = statement
232             .executeQuery("SELECT `gpu`.`price` FROM `gpu` WHERE `name` = '" + order.getGpu() + "';");
233         resultSet.next();
234         Double gpuPrice = resultSet.getDouble(columnLabel: "price");
235
236         resultSet = statement.executeQuery("SELECT `cpu`.`id` FROM `cpu` WHERE `name` = '" + order.getCpu() + "';");
237         resultSet.next();
238         Integer cpuId = resultSet.getInt(columnLabel: "id");
239
240         resultSet = statement.executeQuery("SELECT `gpu`.`id` FROM `gpu` WHERE `name` = '" + order.getGpu() + "';");
241         resultSet.next();
242         Integer gpuId = resultSet.getInt(columnLabel: "id");
243
244         String query = null;
245
246         if (order.getId() == null) {
247             query = new String("INSERT INTO `" + DB.ORDERS_TABLE
248                 + "` (`id`, `customer_id`, `worker_id`, `date`, `price`, `cpu_id`, `gpu_id`) "
249                 + "VALUES (NULL, '" + customerId + "', '" + workerId + "', current_timestamp(), "
250                 + Double.toString(cpuPrice + gpuPrice) + ", '" + Integer.toString(cpuId) + "', '"
251                 + Integer.toString(gpuId) + "');");
252
253             statement.executeUpdate(query);
254             return;
255         }
256
257         query = new String("UPDATE `" + DB.ORDERS_TABLE + "` SET "
258             + "`customer_id` = '" + Integer.toString(customerId)
259             + "', `worker_id` = '" + Integer.toString(workerId)
260             + "', `date` = current_timestamp(), "
261             + "`price` = '" + Double.toString(cpuPrice + gpuPrice)
262             + "', `cpu_id` = '" + Integer.toString(cpuId)
263             + "', `gpu_id` = '" + Integer.toString(gpuId)
264             + "` WHERE `orders`.`id` = '" + Integer.toString(order.getId()) + "';");
265
266         statement.executeUpdate(query);
267     } catch (final SQLException exception) {
268         Logger.getLogger(DB.class.getName()).log(Level.SEVERE, msg: null, exception);
269     }
270 }
271
272 private final void fill() {
273     this.fillCustomers();
274     this.fillWorkers();
275     this.fillCpu();
276     this.fillGpu();
277     this.fillOrders();
278 }
279
```


● MainController.java X

src > main > java > com > example > ● MainController.java > ...

```
1  package com.example;
2
3  import java.io.IOException;
4  import java.net.URL;
5  import java.sql.Date;
6  import java.sql.ResultSet;
7  import java.sql.SQLException;
8  import java.util.ResourceBundle;
9  import java.util.logging.Level;
10 import java.util.logging.Logger;
11
12 import javafx.beans.value.ChangeListener;
13 import javafx.collections.FXCollections;
14 import javafx.collections.ObservableList;
15 import javafx.fxml.FXML;
16 import javafx.fxml.FXMLLoader;
17 import javafx.fxml.Initializable;
18 import javafx.scene.Parent;
19 import javafx.scene.Scene;
20 import javafx.scene.control.ChoiceBox;
21 import javafx.scene.control.TableColumn;
22 import javafx.scene.control.TableView;
23 import javafx.scene.control.TextArea;
24 import javafx.scene.control.TextField;
25 import javafx.scene.control.cell.PropertyValueFactory;
26 import javafx.stage.Stage;
27 import javafx.stage.StageStyle;
28
29 public final class MainController implements Initializable {
30     @FXML
31     private TableView<Order> ordersTableView;
32     @FXML
33     private TableColumn<Order, String> ordersTableViewId;
34     @FXML
35     private TableColumn<Order, String> ordersTableViewCustomer;
36     @FXML
37     private TableColumn<Order, String> ordersTableViewWorker;
38     @FXML
39     private TableColumn<Order, Date> ordersTableViewDate;
40     @FXML
41     private TableColumn<Order, Double> ordersTableViewPrice;
42     @FXML
43     private TableColumn<Order, String> ordersTableViewCpu;
44     @FXML
45     private TableColumn<Order, String> ordersTableViewGpu;
46     @FXML
47     private TableColumn<Order, String> ordersTableViewEdit;
48
49     @FXML
50     private TextField customerFirstNameTextField;
51     @FXML
52     private TextField customerLastNameTextField;
53     @FXML
54     private TextField workerFirstNameTextField;
55     @FXML
56     private TextField workerLastNameTextField;
57     @FXML
58     private TextField workerPositionTextField;
59     @FXML
60     private TextField cpuPriceTextField;
61     @FXML
62     private TextArea cpuDescriptionTextArea;
63     @FXML
64     private TextField gpuPriceTextField;
65     @FXML
66     private TextArea gpuDescriptionTextArea;
67
68     @FXML
69     private ChoiceBox<String> customersChoiceBox;
70     @FXML
71     private ChoiceBox<String> workersChoiceBox;
72
73     private ObservableList<Order> ordersList = null;
74     private DB db = null;
75
76     String customerFilter = null;
77     String workerFilter = null;
78 }
```

Рисунок 5 - Исходный код файла MainController.java.

Продолжение рисунка 5.

```
79 @Override
80 public void initialize(URL arg0, ResourceBundle arg1) {
81     this.ordersList = FXCollections.observableArrayList();
82     this.db = new DB();
83
84     this.ordersTableViewId.setCellValueFactory(new PropertyValueFactory<>("id"));
85     this.ordersTableViewCustomer.setCellValueFactory(new PropertyValueFactory<>("customer"));
86     this.ordersTableViewWorker.setCellValueFactory(new PropertyValueFactory<>("worker"));
87     this.ordersTableViewDate.setCellValueFactory(new PropertyValueFactory<>("date"));
88     this.ordersTableViewPrice.setCellValueFactory(new PropertyValueFactory<>("price"));
89     this.ordersTableViewCpu.setCellValueFactory(new PropertyValueFactory<>("cpu"));
90     this.ordersTableViewGpu.setCellValueFactory(new PropertyValueFactory<>("gpu"));
91
92     ChangeListener<Object> listener = (obs, oldValue, newValue) -> {
93         try {
94             Order order = this.ordersTableView.getSelectionModel().getSelectedItem();
95
96             if (order == null) {
97                 return;
98             }
99
100             ResultSet orderDetailsSet = this.db.getOrderDetails(order.getId());
101             orderDetailsSet.next();
102
103             this.customerFirstNameTextField.setText(orderDetailsSet.getString(columnLabel: "customer_first_name"));
104             this.customerLastNameTextField.setText(orderDetailsSet.getString(columnLabel: "customer_last_name"));
105             this.workerFirstNameTextField.setText(orderDetailsSet.getString(columnLabel: "worker_first_name"));
106             this.workerLastNameTextField.setText(orderDetailsSet.getString(columnLabel: "worker_last_name"));
107             this.workerPositionTextField.setText(orderDetailsSet.getString(columnLabel: "worker_position"));
108             this.cpuPriceTextField.setText(Double.toString(orderDetailsSet.getDouble(columnLabel: "cpu_price")));
109             this.cpuDescriptionTextArea.setText(orderDetailsSet.getString(columnLabel: "cpu_description"));
110             this.gpuPriceTextField.setText(Double.toString(orderDetailsSet.getDouble(columnLabel: "gpu_price")));
111             this.gpuDescriptionTextArea.setText(orderDetailsSet.getString(columnLabel: "gpu_description"));
112         } catch (final SQLException exception) {
113             Logger.getLogger(MainController.class.getName()).log(Level.SEVERE, msg: null, exception);
114         } catch (final Exception exception) {
115             Logger.getLogger(MainController.class.getName()).log(Level.SEVERE, msg: null, exception);
116         }
117     };
118
119     this.ordersTableView.focusedProperty().addListener(listener);
120     this.ordersTableView.getSelectionModel().selectedItemProperty().addListener(listener);
121 }
122
123 @FXML
124 private final void create() {
125     try {
126         Parent parent = App.loadFXML(fxml: "create");
127         Stage stage = new Stage();
128         stage.setScene(new Scene(parent));
129         stage.initStyle(StageStyle.UTILITY);
130         stage.show();
131     } catch (final IOException exception) {
132         Logger.getLogger(MainController.class.getName()).log(Level.SEVERE, msg: null, exception);
133     }
134 }
135
```

Продолжение рисунка 5.

```
136 @FXML
137 private final void read() {
138     try {
139         this.clearTextFields();
140         this.ordersList.clear();
141
142         ResultSet ordersSet = this.db.getOrders(this.customerFilter, this.workerFilter, id: null);
143
144         while (ordersSet.next()) {
145             this.ordersList.add(new Order(
146                 ordersSet.getInt(columnLabel: "id"),
147                 ordersSet.getString(columnLabel: "customer"),
148                 ordersSet.getString(columnLabel: "worker"),
149                 ordersSet.getDate(columnLabel: "date"),
150                 ordersSet.getDouble(columnLabel: "price"),
151                 ordersSet.getString(columnLabel: "cpu"),
152                 ordersSet.getString(columnLabel: "gpu")));
153             this.ordersTableView.setItems(this.ordersList);
154         }
155
156         ObservableList<String> customersList = FXCollections.observableArrayList();
157         ObservableList<String> workersList = FXCollections.observableArrayList();
158
159         ResultSet customersSet = this.db.getAll(DB.CUSTOMERS_TABLE);
160         ResultSet workersSet = this.db.getAll(DB.WORKERS_TABLE);
161
162         while (customersSet.next() && workersSet.next()) {
163             customersList.add(customersSet.getString(columnLabel: "email"));
164             workersList.add(workersSet.getString(columnLabel: "email"));
165         }
166
167         this.customersChoiceBox.setItems(customersList);
168         this.workersChoiceBox.setItems(workersList);
169     } catch (final SQLException exception) {
170         Logger.getLogger(MainController.class.getName()).log(Level.SEVERE, msg: null, exception);
171     } catch (final Exception exception) {
172         Logger.getLogger(MainController.class.getName()).log(Level.SEVERE, msg: null, exception);
173     }
174 }
175
176 @FXML
177 private final void update() {
178     try {
179         Order order = this.ordersTableView.getSelectionModel().getSelectedItem();
180
181         if (order == null) {
182             return;
183         }
184
185         FXMLLoader loader = new FXMLLoader(getClass().getResource(name: "update.fxml"));
186         Parent parent = loader.load();
187
188         UpdateController updateController = loader.getController();
189         updateController.setUpdatingId(order.getId());
190
191         Stage stage = new Stage();
192         stage.setScene(new Scene(parent));
193         stage.initStyle(StageStyle.UTILITY);
194         stage.show();
195     } catch (final IOException exception) {
196         Logger.getLogger(MainController.class.getName()).log(Level.SEVERE, msg: null, exception);
197     } catch (final Exception exception) {
198         Logger.getLogger(MainController.class.getName()).log(Level.SEVERE, msg: null, exception);
199     }
200 }
201
```

Продолжение рисунка 5.

```
202 @FXML
203 private final void delete() {
204     Order order = this.ordersTableView.getSelectionModel().getSelectedItem();
205
206     if (order == null) {
207         return;
208     }
209
210     this.db.deleteByID(DB.ORDERS_TABLE, order.getId());
211 }
212
213 @FXML
214 private final void search() {
215     this.customerFilter = this.customersChoiceBox.getValue();
216     this.workerFilter = this.workersChoiceBox.getValue();
217     this.read();
218 }
219
220 private final void clearTextFields() {
221     this.customerFirstNameTextField.clear();
222     this.customerLastNameTextField.clear();
223     this.workerFirstNameTextField.clear();
224     this.workerLastNameTextField.clear();
225     this.workerPositionTextField.clear();
226     this.cpuPriceTextField.clear();
227     this.cpuDescriptionTextArea.clear();
228     this.gpuPriceTextField.clear();
229     this.gpuDescriptionTextArea.clear();
230 }
231 }
232
```

Order.java 3 X

src > main > java > com > example > Order.java > ...

```
1 package com.example;
2
3 import java.sql.Date;
4
5 public final class Order {
6     private Integer id;
7     private String customer;
8     private String worker;
9     private Date date;
10    private Double price;
11    private String cpu;
12    private String gpu;
13
14    public Order(final Integer id, final String customer, final String worker, final Date date, final Double price,
15                final String cpu, final String gpu) {
16        this.id = id;
17        this.customer = customer;
18        this.worker = worker;
19        this.date = date;
20        this.price = price;
21        this.cpu = cpu;
22        this.gpu = gpu;
23    }
24
```

Рисунок 6 - Исходный код файла Order.java.

Продолжение рисунка 6.

```
25     public final Integer getId() {
26         |     return this.id;
27     }
28
29     public final void setId(final Integer id) {
30         |     this.id = id;
31     }
32
33     public final String getCustomer() {
34         |     return this.customer;
35     }
36
37     public final void setCustomer(final String customer) {
38         |     this.customer = customer;
39     }
40
41     public final String getWorker() {
42         |     return this.worker;
43     }
44
45     public final void setWorker(final String worker) {
46         |     this.worker = worker;
47     }
48
49     public final Date getDate() {
50         |     return this.date;
51     }
52
53     public final void setDate(final Date date) {
54         |     this.date = date;
55     }
56
57     public final Double getPrice() {
58         |     return this.price;
59     }
60
61     public final void setPrice(final Double price) {
62         |     this.price = price;
63     }
64
65     public final String getCpu() {
66         |     return this.cpu;
67     }
68
69     public final void setCpu(final String cpu) {
70         |     this.cpu = cpu;
71     }
72
73     public final String getGpu() {
74         |     return this.gpu;
75     }
76
77     public final void setGpu(final String gpu) {
78         |     this.gpu = gpu;
79     }
80 }
81
```

UpdateController.java X

src > main > java > com > example > UpdateController.java > ...

```
1  package com.example;
2
3  > import java.net.URL; ...
15
16 public final class UpdateController implements Initializable {
17     @FXML
18     private ChoiceBox<String> customersChoiceBox;
19     @FXML
20     private ChoiceBox<String> workersChoiceBox;
21     @FXML
22     private ChoiceBox<String> cpuChoiceBox;
23     @FXML
24     private ChoiceBox<String> gpuChoiceBox;
25
26     private Integer updatingId = null;
27     private DB db = null;
28
29     @Override
30     public void initialize(URL arg0, ResourceBundle arg1) {
31         try {
32             this.db = new DB();
33
34             ObservableList<String> customersList = FXCollections.observableArrayList();
35             ObservableList<String> workersList = FXCollections.observableArrayList();
36             ObservableList<String> cpuList = FXCollections.observableArrayList();
37             ObservableList<String> gpuList = FXCollections.observableArrayList();
38
39             ResultSet customersSet = this.db.getAll(DB.CUSTOMERS_TABLE);
40             ResultSet workersSet = this.db.getAll(DB.WORKERS_TABLE);
41             ResultSet cpuSet = this.db.getAll(DB.CPU_TABLE);
42             ResultSet gpuSet = this.db.getAll(DB.GPU_TABLE);
43
44             while (customersSet.next() && workersSet.next() && cpuSet.next() && gpuSet.next()) {
45                 customersList.add(customersSet.getString(columnLabel: "email"));
46                 workersList.add(workersSet.getString(columnLabel: "email"));
47                 cpuList.add(cpuSet.getString(columnLabel: "name"));
48                 gpuList.add(gpuSet.getString(columnLabel: "name"));
49             }
50
51             this.customersChoiceBox.setItems(customersList);
52             this.workersChoiceBox.setItems(workersList);
53             this.cpuChoiceBox.setItems(cpuList);
54             this.gpuChoiceBox.setItems(gpuList);
55         } catch (final SQLException exception) {
56             logger.getLogger(CreateController.class.getName()).log(Level.SEVERE, msg: null, exception);
57         } catch (final Exception exception) {
58             logger.getLogger(CreateController.class.getName()).log(Level.SEVERE, msg: null, exception);
59         }
60     }
61
62     public final void setUpdatingId(final Integer updatingId) {
63         try {
64             this.updatingId = updatingId;
65
66             ResultSet order = this.db.getOrders(customerFilter: null, workerFilter: null, this.updatingId);
67             order.next();
68
69             this.customersChoiceBox.setValue(order.getString(columnLabel: "customer"));
70             this.workersChoiceBox.setValue(order.getString(columnLabel: "worker"));
71             this.cpuChoiceBox.setValue(order.getString(columnLabel: "cpu"));
72             this.gpuChoiceBox.setValue(order.getString(columnLabel: "gpu"));
73         } catch (final SQLException exception) {
74             logger.getLogger(CreateController.class.getName()).log(Level.SEVERE, msg: null, exception);
75         } catch (final Exception exception) {
76             logger.getLogger(CreateController.class.getName()).log(Level.SEVERE, msg: null, exception);
77         }
78     }
79
80     @FXML
81     private final void update() {
82         if (this.customersChoiceBox.getSelectionModel().isEmpty()
83             || this.workersChoiceBox.getSelectionModel().isEmpty()
84             || this.cpuChoiceBox.getSelectionModel().isEmpty()
85             || this.gpuChoiceBox.getSelectionModel().isEmpty()
86             || this.updatingId == null) {
87             return;
88         }
89
90         this.db.addOrder(new Order(this.updatingId,
91             this.customersChoiceBox.getValue(),
92             this.workersChoiceBox.getValue(),
93             date: null,
94             price: 0.0,
95             this.cpuChoiceBox.getValue(),
96             this.gpuChoiceBox.getValue()));
97     }
98
99 }
```

Рисунок 7 - Исходный код файла UpdateController.java.

ID	Customer	Worker	Date	Price	CPU	GPU
No content in table						

Customer First Name:

Worker First Name:

Worker Position:

CPU Price:

CPU Description:

Customer Last Name:

Worker Last Name:

GPU Price:

GPU Description:

Search By Customer:

Search By Worker:

Create

Read

Update

Delete

Search

ID	Customer	Worker	Date	Price	CPU	GPU
1	Roosevelt_Glov...	Abilene_Deبرا...	2022-05-02	549.0	Intel Core i5-10...	NVIDIA Geforc...
2	Terry_Duffy@g...	Lilac_Rick@gm...	2022-05-02	748.0	AMD Ryzen 7 5...	NVIDIA Geforc...
3	Avery_Schaefer...	Joanie_Julianne...	2022-05-02	478.0	AMD Ryzen 3 4...	AMD Radeon R...
4	Abel_Wood@g...	Caroline_Jean@...	2022-05-02	1078.0	Intel Core i9-12...	AMD Radeon R...
5	Gale_Bean@gm...	Kiki_Gene@gm...	2022-05-02	731.0	AMD FX-6300	NVIDIA Geforc...

Customer First Name:

Worker First Name:

Worker Position:

CPU Price:

CPU Description:

Customer Last Name:

Worker Last Name:

GPU Price:

GPU Description:

Search By Customer:

Search By Worker:

Create

Read

Update

Delete

Search

ID	Customer	Worker	Date	Price	CPU	GPU
1	Roosevelt_Glov...	Abilene_Deبرا...	2022-05-02	549.0	Intel Core i5-10...	NVIDIA Geforc...
2	Terry_Duffy@g...	Lilac_Rick@gm...	2022-05-02	748.0	AMD Ryzen 7 5...	NVIDIA Geforc...
3	Avery_Schaefer...	Joanie_Julianne...	2022-05-02	478.0	AMD Ryzen 3 4...	AMD Radeon R...
4	Abel_Wood@g...	Caroline_Jean@...	2022-05-02	1078.0	Intel Core i9-12...	AMD Radeon R...
5	Gale_Bean@gm...	Kiki_Gene@gm...	2022-05-02	731.0	AMD FX-6300	NVIDIA Geforc...

Customer First Name:

Worker First Name:

Worker Position:

CPU Price:

CPU Description:

Customer Last Name:

Worker Last Name:

GPU Price:

GPU Description:

Search By Customer:

Search By Worker:

Create

Read

Update

Delete

Search

ID	Customer	Worker	Date	Price	CPU	GPU
1	Roosevelt_Glov...	Abilene_Deبرا...	2022-05-02	549.0	Intel Core i5-10...	NVIDIA Geforc...
3	Avery_Schaefer...	Joanie_Julianne...	2022-05-02	478.0	AMD Ryzen 3 4...	AMD Radeon R...
4	Abel_Wood@g...	Caroline_Jean@...	2022-05-02	1078.0	Intel Core i9-12...	AMD Radeon R...
5	Gale_Bean@gm...	Kiki_Gene@gm...	2022-05-02	731.0	AMD FX-6300	NVIDIA Geforc...

Customer First Name:

Worker First Name:

Worker Position:

CPU Price:

CPU Description:

Customer Last Name:

Worker Last Name:

GPU Price:

GPU Description:

Search By Customer:

Search By Worker:

Create

Read

Update

Delete

Search

ID	Customer	Worker	Date	Price	CPU	GPU
1	Roosevelt_Glov...	Abilene_Deبرا...	2022-05-02	549.0	Intel Core i5-10...	NVIDIA Geforc...
3	Avery_Schaefer...	Joanie_Julianne...	2022-05-02	478.0	AMD Ryzen 3 4...	AMD Radeon R...
4	Abel_Wood@g...	Caroline_Jean@...	2022-05-02	1078.0	Intel Core i9-12...	AMD Radeon R...
5	Gale_Bean@gm...	Kiki_Gene@gm...	2022-05-02	731.0	AMD FX-6300	NVIDIA Geforc...

Customer First Name:

Worker First Name:

Worker Position:

CPU Price:

CPU Description:

Customer Last Name:

Worker Last Name:

GPU Price:

GPU Description:

Search By Customer:

Search By Worker:

Create

Read

Update

Delete

Search

ID	Customer	Worker	Date	Price	CPU	GPU
1	Roosevelt_Glov...	Abilene_Deبرا...	2022-05-02	549.0	Intel Core i5-10...	NVIDIA Geforc...
3	Avery_Schaefer...	Joanie_Julianne...	2022-05-02	478.0	AMD Ryzen 3 4...	AMD Radeon R...
4	Abel_Wood@g...	Caroline_Jean@...	2022-05-02	1078.0	Intel Core i9-12...	AMD Radeon R...
5	Gale_Bean@gm...	Kiki_Gene@gm...	2022-05-02	731.0	AMD FX-6300	NVIDIA Geforc...

Customer First Name:

Worker First Name:

Worker Position:

CPU Price:

CPU Description:

Customer Last Name:

Worker Last Name:

GPU Price:

GPU Description:

Search By Customer:

Search By Worker:

Create

Read

Update

Delete

Search

ID	Customer	Worker	Date	Price	CPU	GPU
1	Roosevelt_Glov...	Abilene_Deبرا...	2022-05-02	549.0	Intel Core i5-10...	NVIDIA Geforc...
3	Avery_Schaefer...	Joanie_Julianne...	2022-05-02	478.0	AMD Ryzen 3 4...	AMD Radeon R...
4	Abel_Wood@g...	Caroline_Jean@...	2022-05-02	1078.0	Intel Core i9-12...	AMD Radeon R...
5	Gale_Bean@gm...	Kiki_Gene@gm...	2022-05-02	731.0	AMD FX-6300	NVIDIA Geforc...
6	Terry_Duffy@g...	Lilac_Rick@gm...	2022-05-02	748.0	AMD Ryzen 7 5...	NVIDIA Geforc...

Customer First Name:

Worker First Name:

Worker Position:

CPU Price:

CPU Description:

Customer Last Name:

Worker Last Name:

GPU Price:

GPU Description:

Search By Customer:

Search By Worker:

Create

Read

Update

Delete

Search

Рисунок 8 - Результат работы программы.

Продолжение рисунка 8.

The application displays a table of computer parts with the following columns: ID, Customer, Worker, Date, Price, CPU, and GPU. The data shown in the screenshots is as follows:

ID	Customer	Worker	Date	Price	CPU	GPU
1	Roosevelt_Glov...	Abilene_Debra...	2022-05-02	549.0	Intel Core i5-10...	NVIDIA GeForc...
3	Avery_Schaefer...	Joanie_Julianne...	2022-05-02	478.0	AMD Ryzen 3 4...	AMD Radeon R...
4	Abel_Wood@g...	Caroline_Jean@...	2022-05-02	1078.0	Intel Core i9-12...	AMD Radeon R...
5	Gale_Bean@gm...	Kiki_Gene@gm...	2022-05-02	731.0	AMD FX-6300	NVIDIA GeForc...
6	Terry_Duffy@g...	Lilac_Rick@gm...	2022-05-02	748.0	AMD Ryzen 7 5...	NVIDIA GeForc...

The application also includes input fields for Customer First Name, Customer Last Name, Worker First Name, Worker Last Name, Worker Position, CPU Price, CPU Description, GPU Price, and GPU Description. The CPU and GPU dropdown menus show the following options:

- CPU: None, Gale_Bean@gmail.com, Abel_Wood@gmail.com, Roosevelt_Glover@gmail.com, Avery_Schaefer@gmail.com, Terry_Duffy@gmail.com (selected)
- GPU: The GeForce GTX 1080 (selected)

The application also includes a search bar and buttons for Create, Read, Update, Delete, and Search.

Вывод: Приобрел практические навыки разработки многооконных приложений на JavaFX для работы с базами данных.