## ­­­Анотація

Курсова робота написана на українській мові. Курсова робота містить 13 рисунків та складається з 30 сторінок

В цій курсовій роботі спроектована і реалізована база даних для фабрики суперкарів “Lamborghini”.

Зміст

Анотація……..………………………………………………………………………1

[Вступ 3](#_Toc437949506)

[1.1 Опис і аналіз предметної області 4](#_Toc437949507)

2[.1 Ієрархія потоків даних 4](#_Toc437949508)

[3.1 Діаграмма сутність-зв’язок 6](#_Toc437949509)

[3.2 Логічна модель даних 8](#_Toc437949510)

[3.3 Вибір СУБД 10](#_Toc437949507)

[3.4 Фізична модель даних 11](#_Toc437949508)

[3.5 Аналіз нормалізації схеми БД 12](#_Toc437949509)

[4.1 Форми вводу даних 14](#_Toc437949510)

[4.2 Запити до бази даних 16](#_Toc437949509)

[4.3 Звіт, вихідні данні 17](#_Toc437949510)

[5.1 Список використанної літератури 19](#_Toc437949511)

[5.2 Додатки 20](#_Toc437949512)

# Вступ

В сучасності, в будь-якої галузі, настає час, коли потрібно вирішувати проблему зберігання інформації. У нашому випадку, розробляючи базу даних заводу суперкарів "Lamborghini", нам потрібно зберігати дані о робітниках, клієнтах, автомобілях і мати можливість легко оперувати цими даними. Для цього існує мова структурованих запитів - SQL.

Набір інструментів для роботи з базами даних називається система управління базами даних (СУБД). У цій роботі буде використана одна з таких.

Для реалізації програми, яка буде використовувати ці дані, в цій роботі буде використано мову Java, тому що це легка мова, яка має всі необхідні механізми для роботи з базами даних.

## 1.1 Опис і аналіз предметної області

База даних фабрики суперкарів “Lamborghini” розробляється для ведення обліку даних о клієнтах, машинах, робітниках, та роботі, яку вони виконували, та для адміністрування цієї інформації.

Коли робітники розробляють машини, їм треба отримати таку інформацію від клієнта, як модель та колір машини, а робітники повинні дати клієнту інформацію о ціні цієї машини. З цього слідує, що наша база даних повинна зберігати такі дані о машинах, як: серійний номер, модель, колір, ціна.

База даних буде зберігати дані о клієнтах, щоб мати упорядковані дані для зв’язку з ними. Дані о клієнтах будуть такі, як: прізвище ім’я по батькові, телефон, адрес, та номера серійних машин які він хоче заказати.

Також база даних буде зберігати дані о робітниках для того, щоб директор фабрики легко відстежував кількість робітників, їх заслуги, та тримав своїх робітників на зв’язку. Дані о робітниках будуть вноситись такі, як: прізвище ім’я по батькові, телефон, та адрес.

Також в нашій базі даних треба зберігати інформацію о виконаної роботі. Коли над машиною виконується хоча б одна робота, то її можна заказати. Таку інформацію повинна зберігати о роботі наша база даних: ідентифікатор роботи, ідентифікатор робітника, серійний номер машини над якою була проведена робота, тип роботи, та дата роботи.

### 2.1. Ієрархія потоків даних

Діаграма потоків даних нульового рівню зображено на рисунку 1.

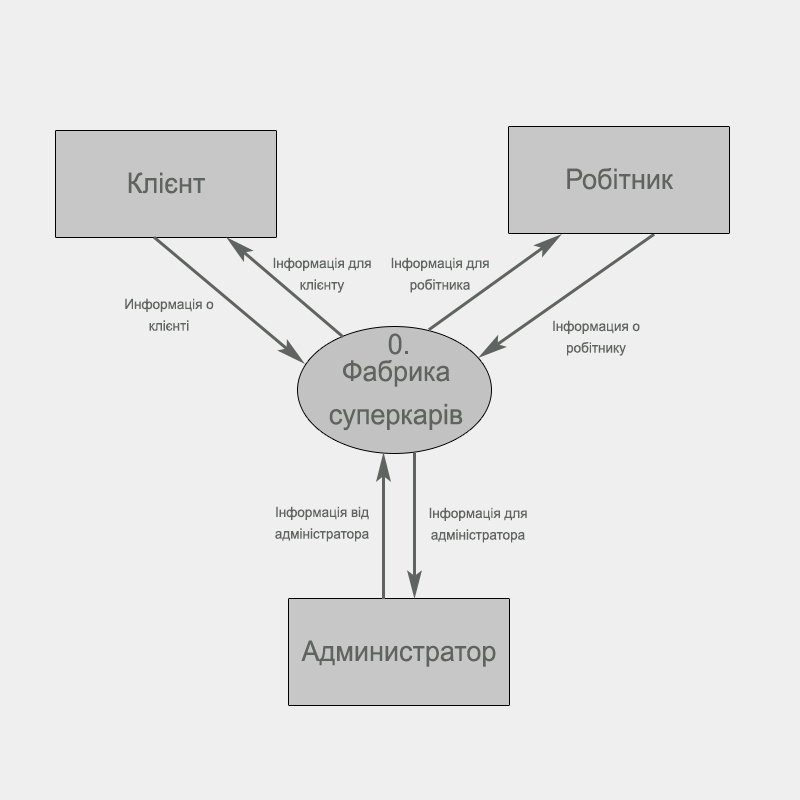


Рисунок 1 – Діаграма потоків даних нульового рівню

Діаграма потоків даних першого рівню зображена на рисунку 2.2.



Рисунок 2.2 – Діаграма потоків даних першого рівню

Відповідності потоків даних і атрибутів показано у таблиці 2.1.

|  |  |  |
| --- | --- | --- |
| Потоки даних 0 рівня | Потоки данних 1 рівня | Атрибути |
| Інформація від клієнту | Данні клієнта | ФІО, телефон, адреса |
| Інформація від робітника | Данні робітника | ID, ФІО, телефон, номер зарплатної карточки, серійні номера машин, логін, пароль |
| Данні о роботі над машиною | ID, тип роботи, ID робітника, Cерійний номер авто, дата |
| Інформація для адміністратора | Запит про інформацію о машині | Серійний номер |
| Відповідь на запит на інф о машині | Серійний номер, модель, колір, ціна |
| Інформація від адміністратора | Нова машина | Серійний номер, модель, колір, ціна |

Таблиця 2.1 - Відповідності потоків даних і атрибутів

## 3.1. Діаграма сутність-зв’язок

Концептуальна модель даних зображена на рисунку 3.1.

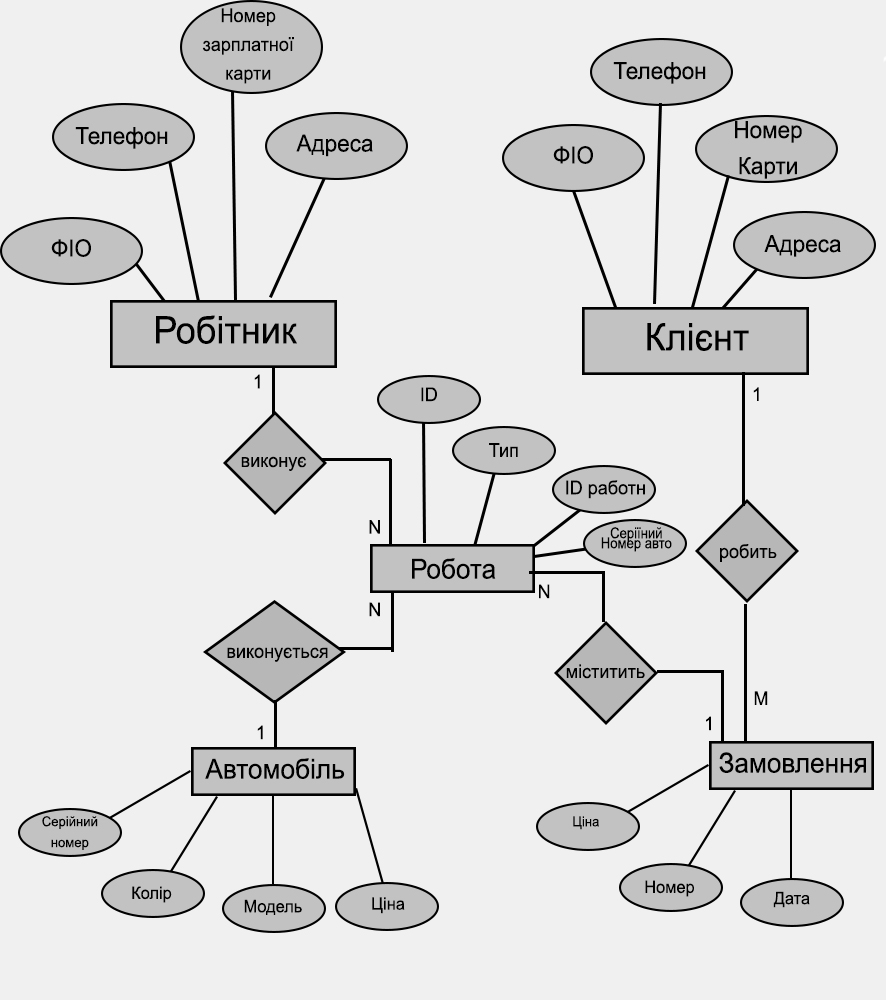


Рисунок 3.1 – Діаграма сутність-зв’язок

**3.2. Логічна модель даних**

Логічна модель даних зображена на рисунку 4.1

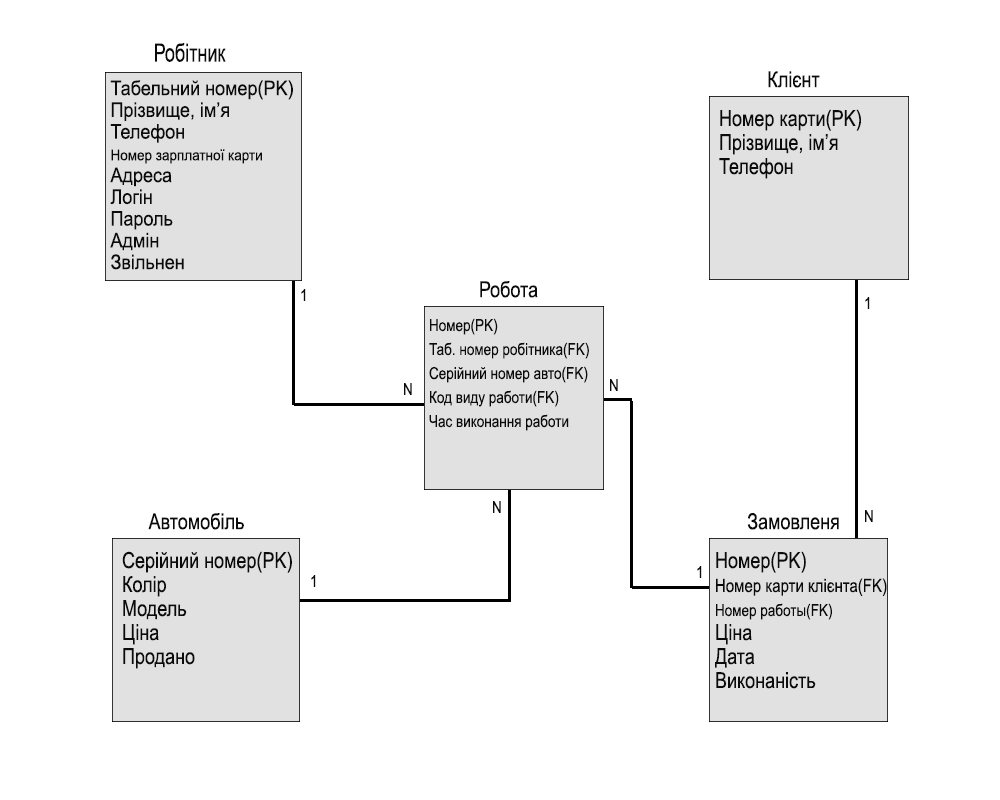


Рисунок 4.1 – Логічна модель даних

**3.3. Вибір СУБД**

СУБД повинна забезпечувати реляційну модель роботи з даними. Сама модель має на увазі певний тип зв'язку між сутностями з різних таблиць. Щоб зберігати і працювати з даними, такий тип СУБД повинен мати певну структуру (таблиці). У таблицях кожен стовпець може містити дані різного типу. Кожен запис складається з безлічі атрибутів (стовпців) і має унікальний ключ, що зберігається в тій же таблиці - всі ці дані взаємопов'язані між собою, як описано в реляційної моделі.

Ось три основних вільно розповсюджуваних СУБД:

## SQLite - дуже потужна вбудована система управління

## MySQL - найпопулярніша і поширена СУБД

## PostgreSQL - найбільш просунута СУБД

Ми оберемо СУБД MySQL, тому що вона має:

Багатий функціонал - MySQL підтримує більшість функціоналу SQL.

Простота в роботі - встановити MySQL досить просто.

Безпечність - велика кількість функцій забезпечують безпеку, які підтримується за замовчуванням

Масштабованість - MySQL легко працює з великими обсягами даних і легко масштабується

Швидкість - спрощення деяких стандартів дозволяє MySQL значно збільшити продуктивність.

Але, як і у всіх поширених СУБД, MySQL має свої недоліки:

В MySQL закладені деякі обмеження функціонала, які іноді необхідні в особливо вимогливих додатках.

Проблеми з надійністю - через деяких способів обробки даних MySQL (зв'язку, транзакції, аудити) іноді поступається іншим СУБД по надійності.

Повільна розробка - Хоча MySQL технічно відкрите ПЗ, існують скарги на процес розробки. Варто зауважити, що існують інші досить успішні СУБД створені на базі MySQL, наприклад MariaDB.

**3.4. Фізична модель даних**

Фізична модель даних, що відповідає логічній моделі з рисунку 4.1 зображена на рисунку 5.1

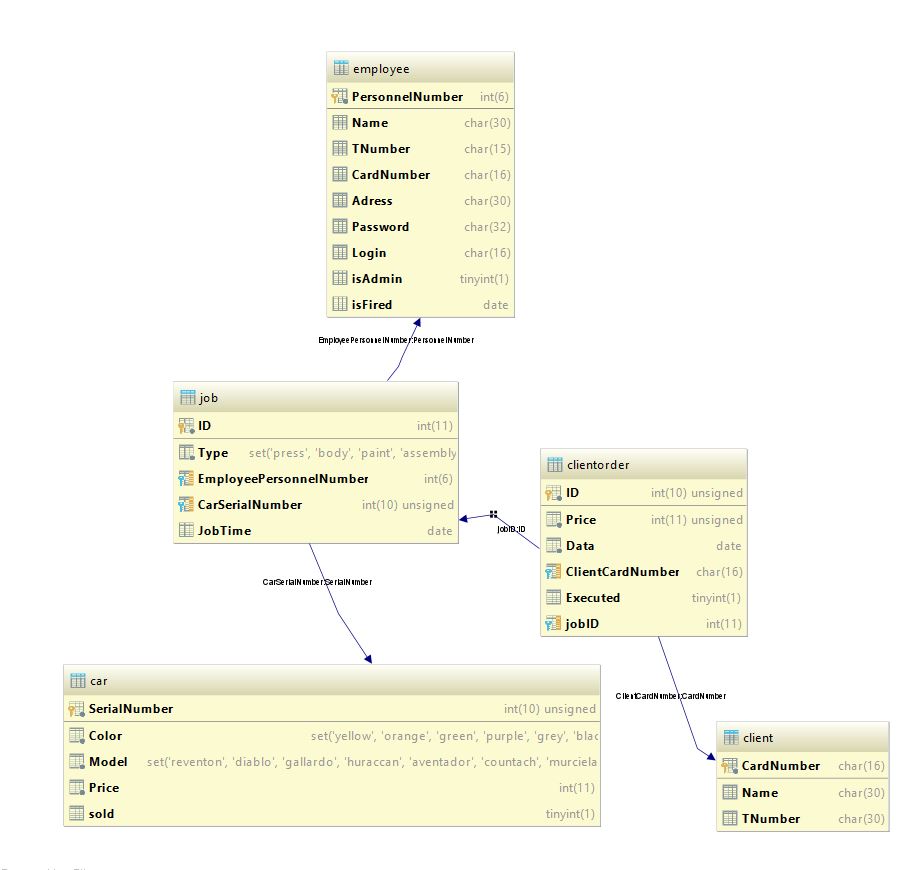


Рисунок 5.1 – Фізична модель даних

**3.5. Аналіз нормалізації схеми бази даних**

Відношення знаходиться в 1НФ, якщо всі його атрибути є простими, всі використовувані домени повинні містити тільки скалярні значення. Не повинно бути повторень рядків в таблиці. Як ми бачимо на фізичній моделі на рис 5.1 всі відношення мають прості атрибути.

Відношення знаходиться в 2НФ, якщо воно знаходиться в 1НФ і кожен не ключовий атрибут неприводимо залежить від Первинного Ключа (ПК). Неприводимість означає, що в складі потенційного ключа відсутня менша підмножина атрибутів, від якої можна також вивести цю функціональну залежність. На нашому прикладі ніякої неповної залежності не видно.

Відношення знаходиться в 3НФ, коли знаходиться у 2НФ і кожен не ключовий атрибут нетранзитивно залежить від первинного ключа. Простіше кажучи, друге правило вимагає виносити всі неключові поля, вміст яких може стосуватися кількох записів таблиці в окремі таблиці. На нашому прикладі також не видно транзитивних відношень.

Відношення знаходиться в НФБК, коли кожна нетривіальна і неприводима зліва функціональна залежність має потенційний ключ як детермінант. Наша база даних не знаходиться в НФБК, так як існує функціональна залежність car -> job в якій ліва частина не є потенційним ключем відношення. З чого можна зробити висновок що наша база даних знаходиться у третій нормальній формі.

* 1. **. Форми для вводу даних**

Вибір режиму роботи (також головне вікно) зображено на рисунку 7.1



Рисунок 7.1 – вікно вибору режиму роботи

Вікно авторизації зображено на рисунку 7.2

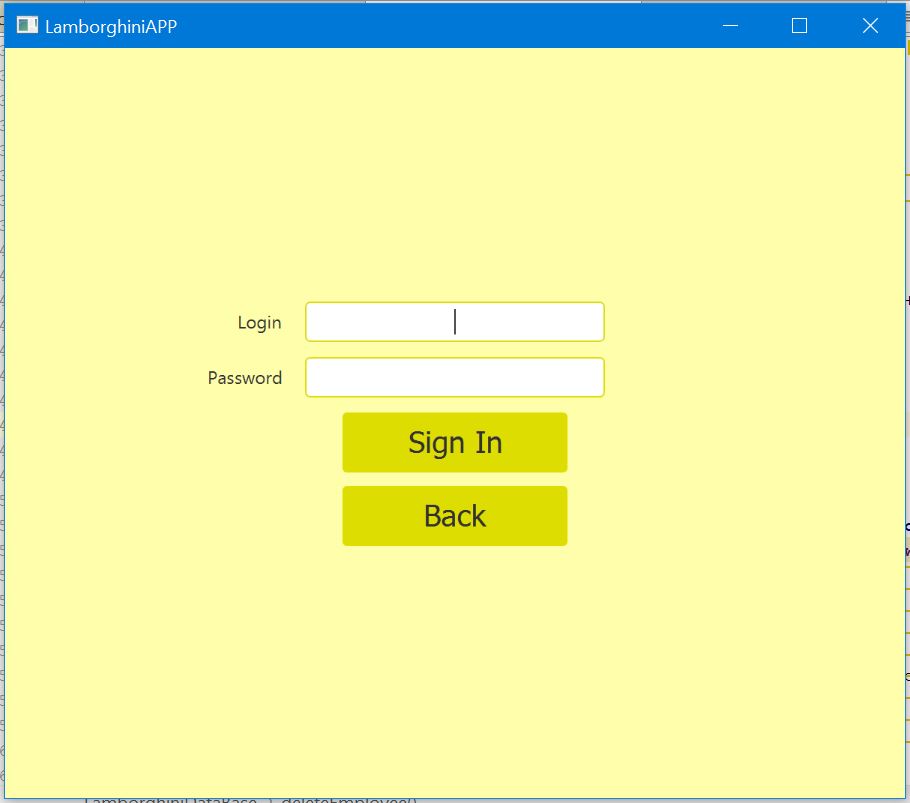


Рисунок 7.2 – вікно авторизації

Вікно вводу виконаної роботи зображено на рисунку 7.3

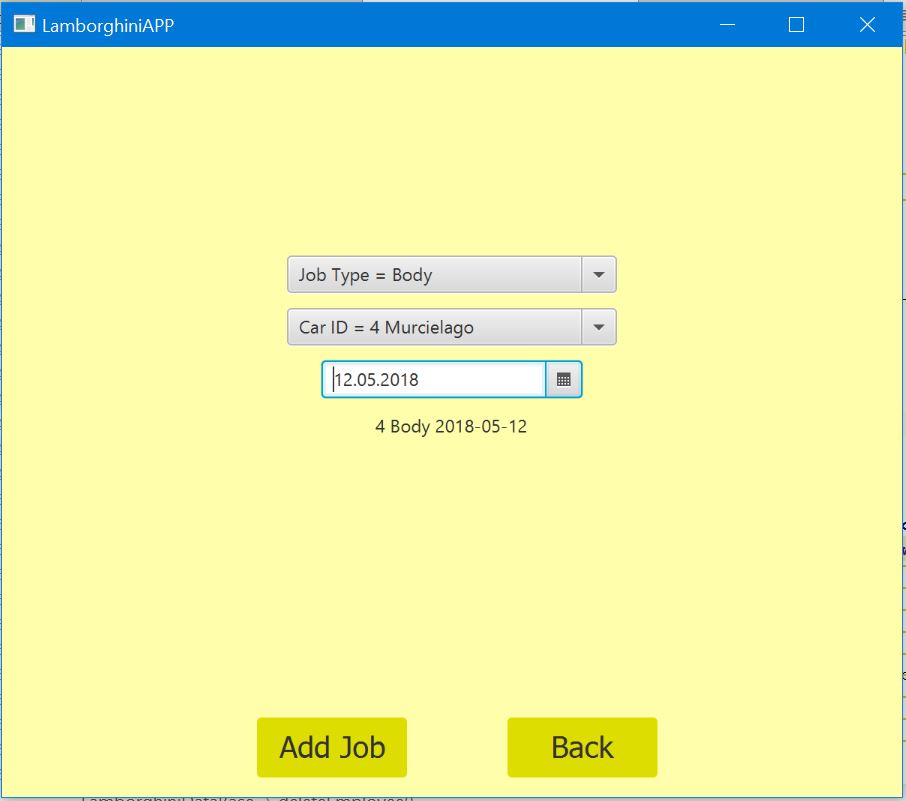


Рисунок 7.3 – вікно вводу виконаної роботи

Вікно замовлення зображено на рисунку 7.4

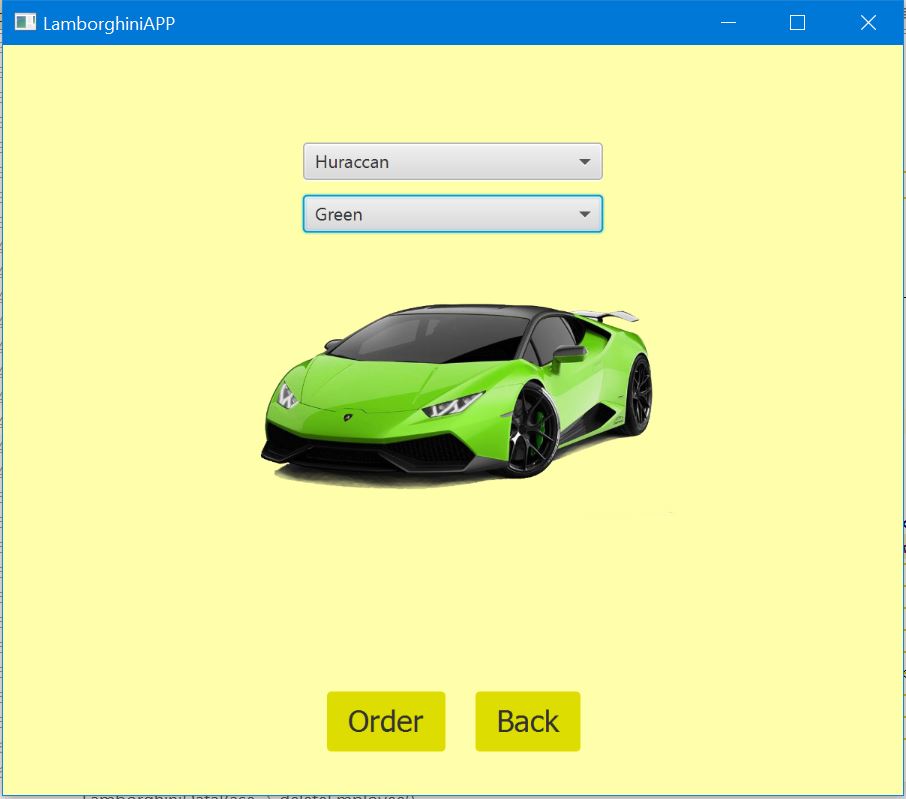


Рисунок 7.4 – Вікно замовлення

Вікно вводу інформації о клієнті зображено на рисунку 7.5

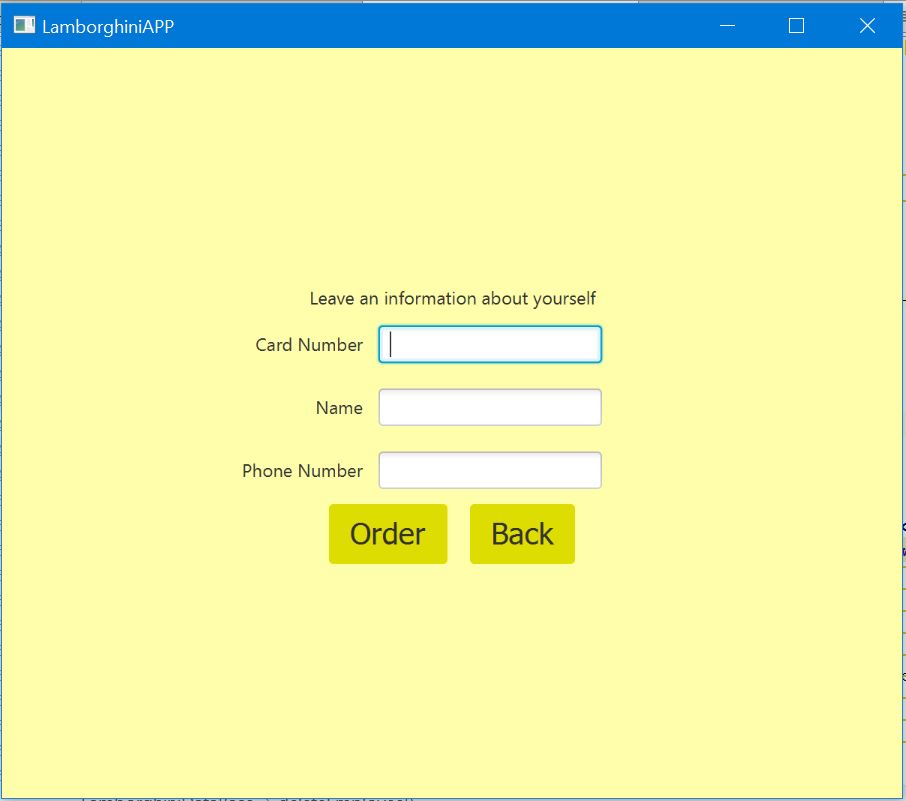


Рисунок 7.5 – Вікно вводу інформації о клієнті

Вікно додавання нової машини зображено на рисунку 7.6

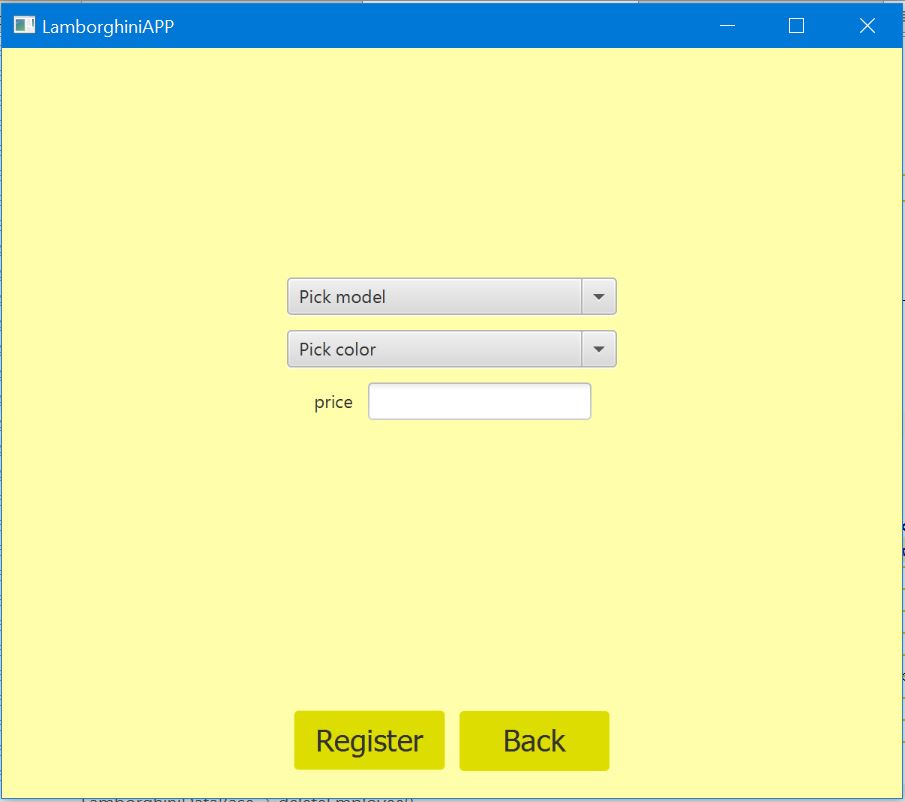


Рисунок 7.6 – Вікно додавання нової машини

Вікно звільнення персоналу зображено на рисунку 7.7

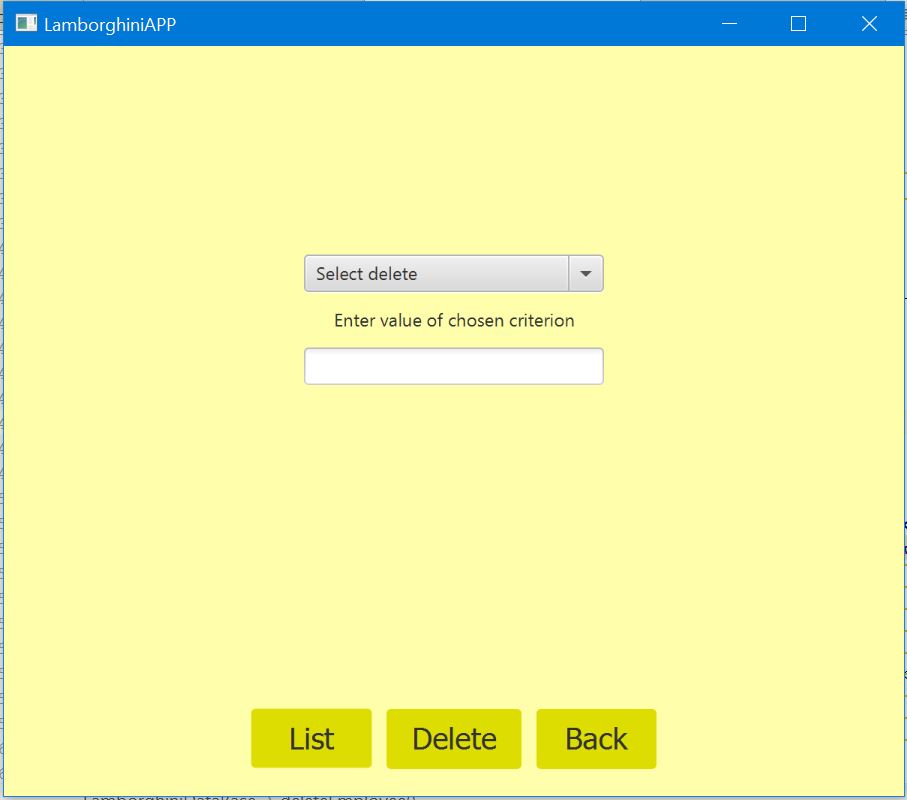


Рисунок 7.7 – Вікно звільнення персоналу

Вікно реєстрації зображено на рисунку 7.8

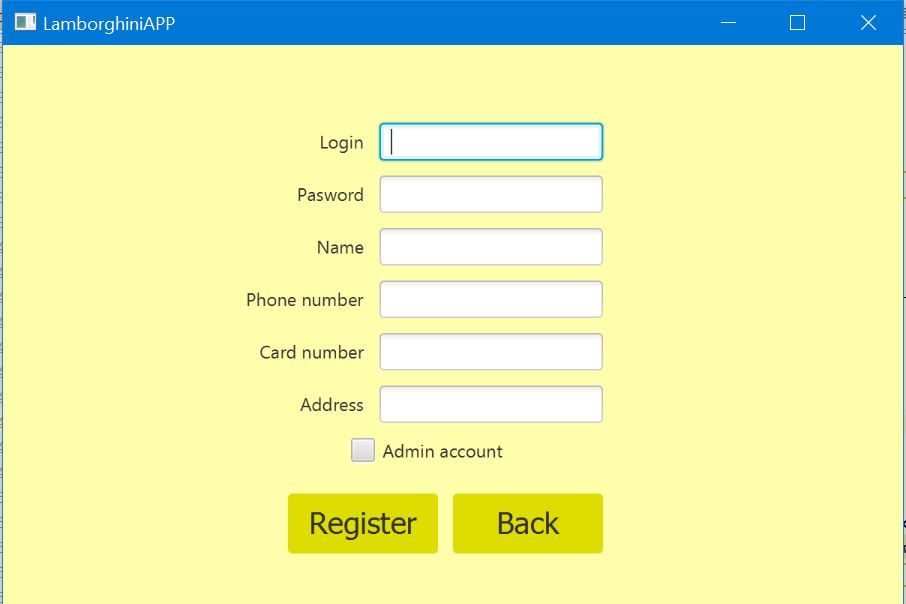


Рисунок 7.8 – Вікно реєстрації

Вікно вводу статусу заказу зображено на рисунку 7.9

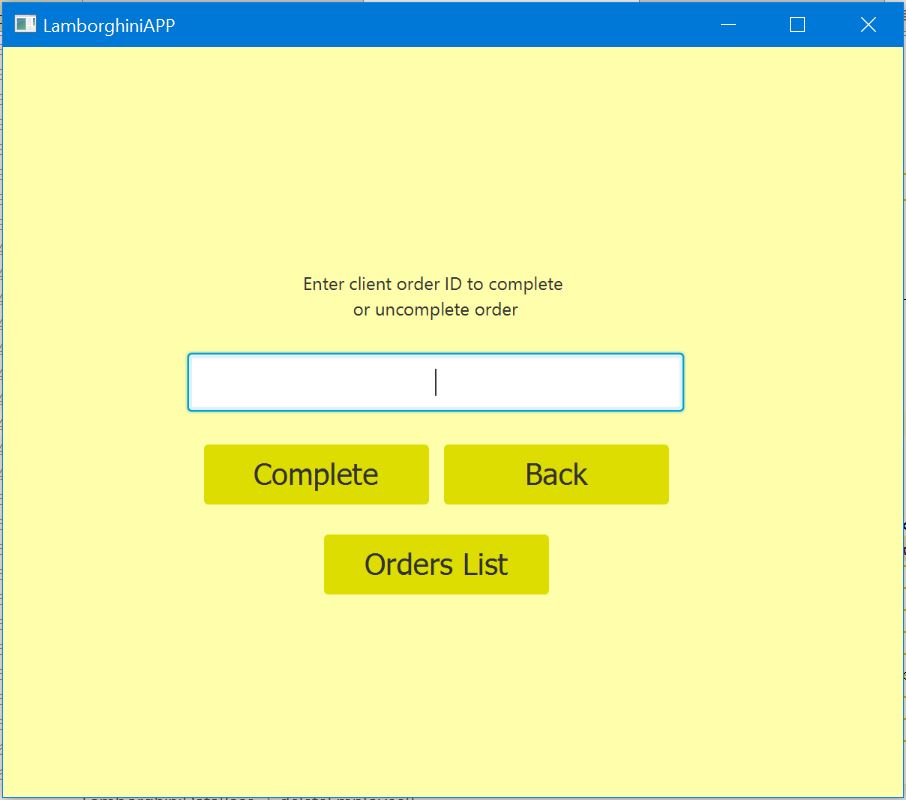


Рисунок 7.9 – Вікно вводу статусу заказу

**4.2. Запити до бази даних**

Для вибору даних було реалізовано такі запити:

**"SELECT** *\** **from** Car”

**"SELECT** *\** **from** Client”

**"SELECT** *\** **from** ClientOrder”

**"SELECT** *\** **from** Job”

**"SELECT** *\** **from** Employee”

**“SELECT ClientOrder.\*, Job.ID, Employee.Name FROM ClientOrder, Job, Employee WHERE ClientOrder.jobID = Job.ID AND Job.EmployeePersonnelNumber = Employee.PersonnelNumber”**

**“SELECT Job.\*, Employee.PersonnelNumber, Car.SerialNumber FROM Job, Employee, Car WHERE Job.EmployeePersonnelNumber = Employee.PersonnelNumber AND Job.CarSerialNumber = Car.SerialNumber”**

Для додавання даних було реалізовано такі запити:

**"INSERT INTO job (ID, Type, EmployeePersonnelNumber, CarSerialNumber, JobTime) VALUES (NULL, '"**+type+**"', "**+employeeN+**", "**+carSerial+**", '"**+date+**"')"**

**"INSERT INTO employee (PersonnelNumber, Name, TNumber, CardNumber, Adress, Password, Login, isAdmin, isFired) VALUES (NULL, '"**+name+**"', '"**+tNumber+**"', '"**+cardNumber+**"', '"**+address+**"', '"**+password+**"', '"**+login+**"', "**+isAdmin+**", NULL)"**

**"INSERT INTO client (Name, TNumber, CardNumber) VALUES ('"**+name+**"', '"**+tNumber+**"', '"**+cardNumber+**"')"**

**"INSERT INTO car (SerialNumber, Color, Model, Price) VALUES (NULL, '"**+color+**"', '"**+model+**"', "**+price+**")"**

**"INSERT INTO clientorder (ID, Price, Data, ClientCardNumber, jobID) VALUES (NULL, "**+price+**", '"**+data+**"', '"**+clientCardNumber+**"', "**+jobID+**")"**

Для видалення даних було реалізовано один запит, для робітників:

**"DELETE FROM employee WHERE PersonnelNumber='"**+ID+**"'"**

Для оновлення даних було реалізовано такі запити:

**"UPDATE employee SET Name = '"**+name+**"', TNumber = '"**+tNumber+**"', CardNumber = '"**+cardNumber+**"', Adress = '"**+address+**"', Password = '"**+password+**"', Login = '"**+login+**"', isAdmin= "**+isAdmin+**", isFired = '"**+isFired+**"' WHERE PersonnelNumber = "**+personnelNumber

**"UPDATE clientorder SET Price = "**+price+**", Data = '"**+data+**"', ClientCardNumber = '"**+clientCardNumber+**"', Executed = "**+executed+**", jobID = "**+jobID+**" WHERE ID = "**+id

**"UPDATE car SET Color = '"**+color+**"', Model = '"**+model+**"', Price = "**+price+**", sold = "**+sold+**" WHERE SerialNumber = "**+serial

**4.3. Звіт, вихідні форми**

Вивід клієнтів зображено на рисунку 8.1

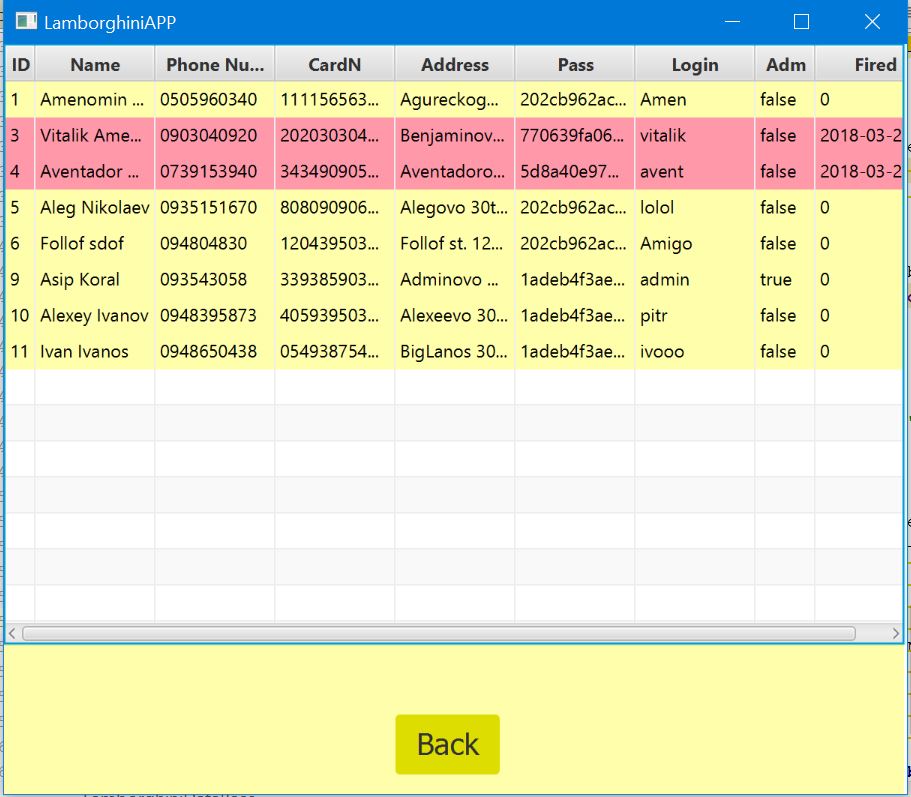


Рисунок 8.1 – Вивід клієнтів

Вивід заказів зображено на рисунку 8.2

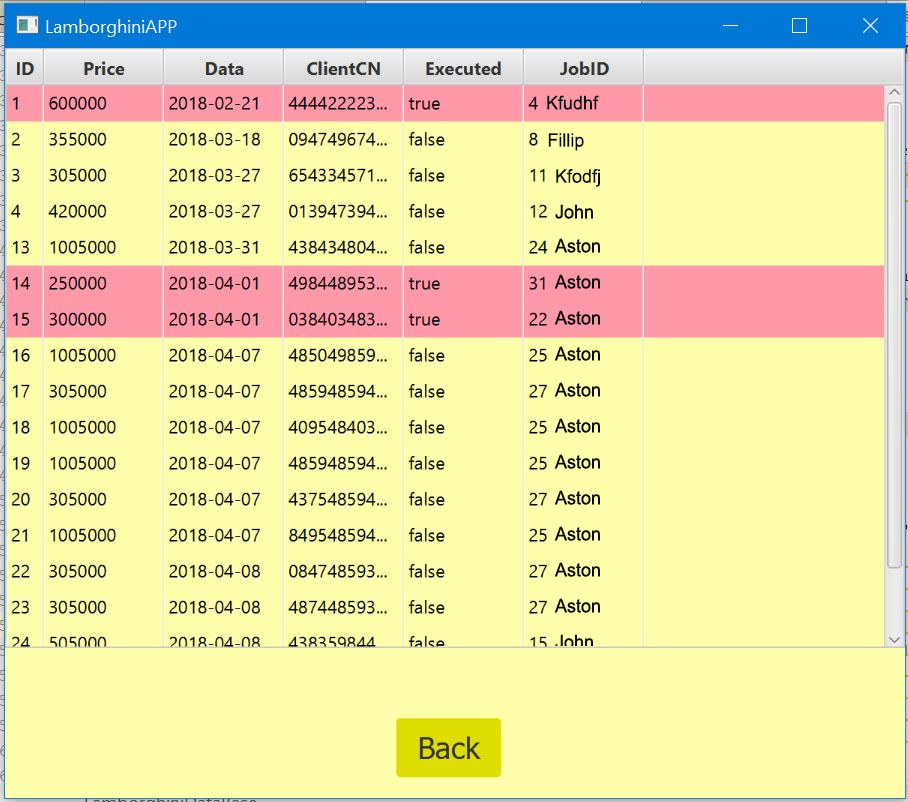


Рисунок 8.2 – Вивід заказів

Вивід робітників зображено на рисунку 8.3

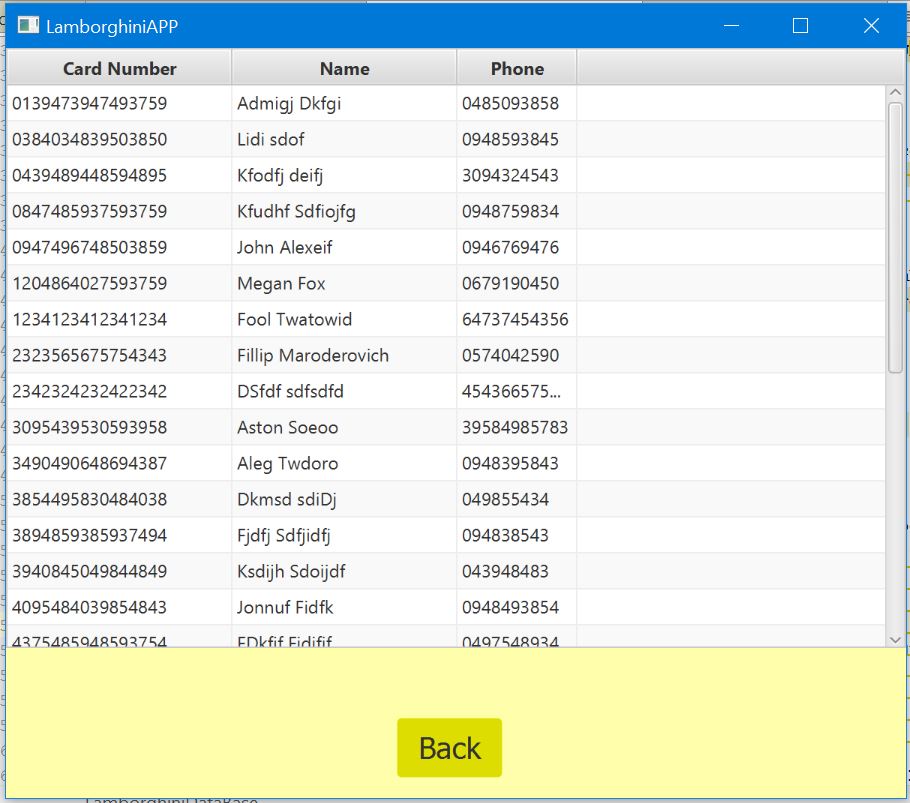


Рисунок 8.3 – Вивід робітників

Вивід робот зображено на рисунку 8.4

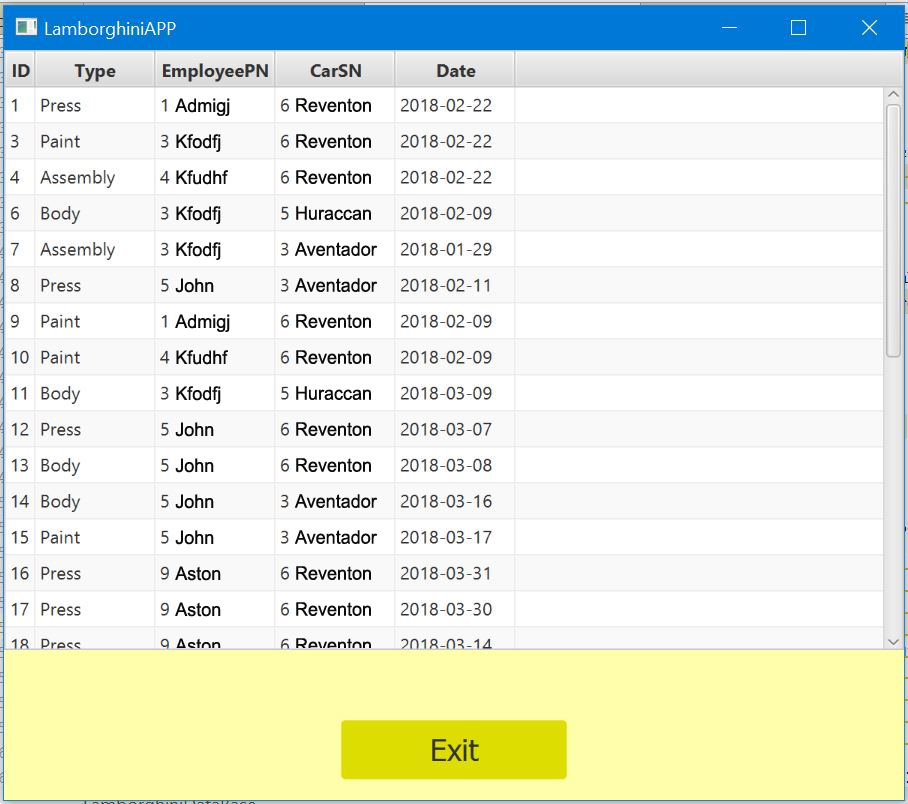


Рисунок 8.4 – Вивід робот

Вивід машин зображено на рисунку 8.5

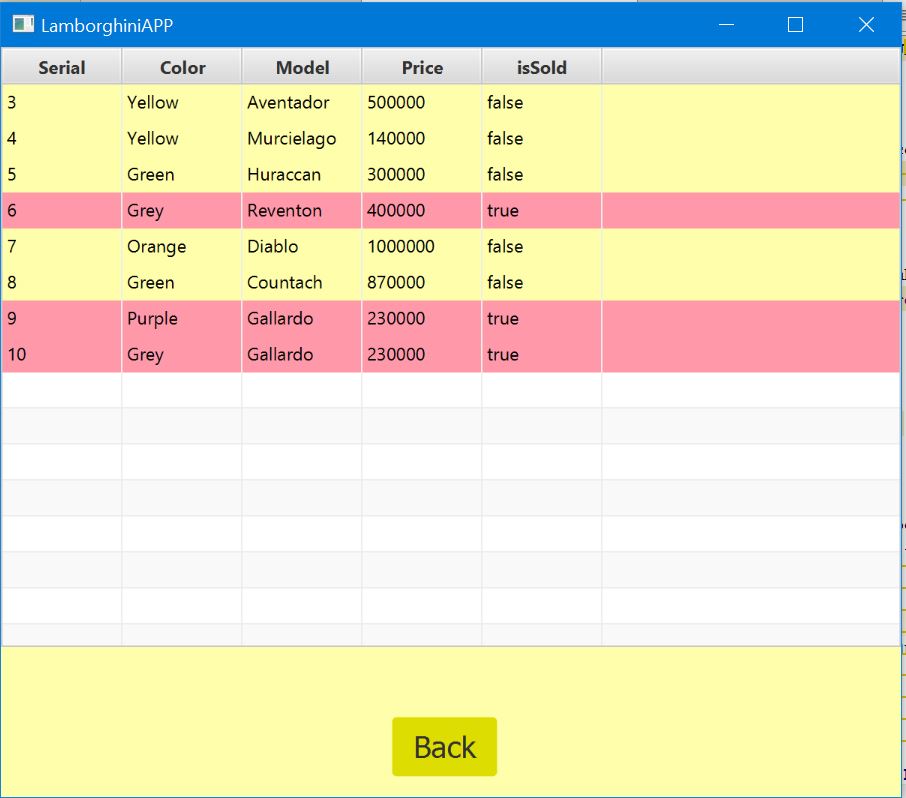


Рисунок 8.5 – Вивід машин

Після успішного заказу авто, автоматично генерується чек, який зображено на рисунку 8.6.

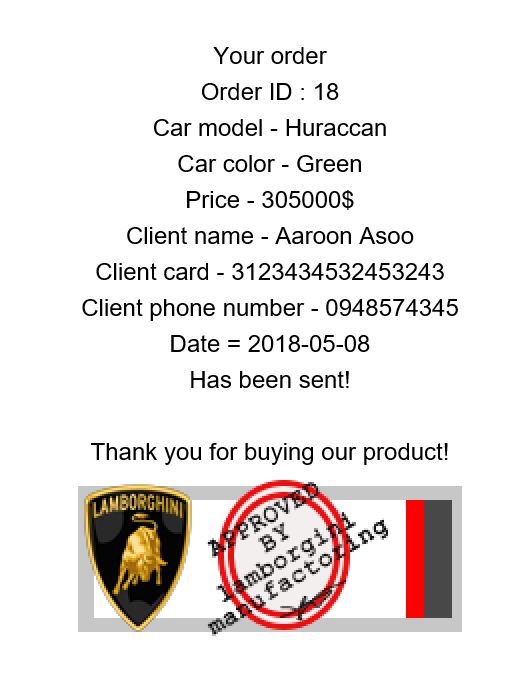


Рисунок 8.6 – Чек заказу авто

## Список використаної літератури

1. Методичні вказівки до виконання курсових робіт з дисципліни "Мова програмування Java" / Є. Ю. Беркунський. – Миколаїв : НУК, 2015. – 29 c.

2. Система управління базами даних Access. Навчальний посібник з курсу «Організація баз даних і баз знань»/ Укладач В.О. Нелюбов. – Ужгород: Редакційно-видавничій відділ ЗакДУ, 2011. – 62 с.

3. Лабораторний практикум з курсу «організація баз даних і баз знань». Розділ «керування застосунками баз даних»/ укладач В.О. Нелюбов. – Ужгород: видавничий центр ЗакДУ, 2009. – 51 с.

4. Методичні вказівки до лабораторних робіт з курсу «Організація баз даних і баз знань». Розділ «Мова запитів SQL»/ Укладач В.О. Нелюбов. – Ужгород: Видавничий центр ЗакДУ, 2010. – 28 с.

5. Методичні вказівки до лабораторних робіт з курсу ″Організація баз даних і баз знань″. Розділ ″Мова запитів QBE″/ Укладач В.О. Нелюбов. – Ужгород: Видавничий центр ЗакДУ, 2010. – 28 с.

6. Лекції з курсу ″Організація баз даних і баз знань″ в електронному вигляді / Укладач В.О. Нелюбов.

7. Презентація навчальних і наукових матеріалів: Електронний навчальний посібник / Нелюбов В. О., Ващук О. М. - Ужгород: ЗакДУ, 2011.- 156 с.: іл.

8. Bведення в системи баз даних/ пер. с англ.- 8-е изд. / К. Дж. Кейт. - М.: Издательский дом «Bильямс», 2006.- 1328 С.

9. Bведение в системи управления базами данных. Часть 1. Реляционная модель данньх: учебное пособие/ Изд-е Башкирского ун-та / Пушников А.Ю. - Уфа, 1999. - 108 с.

10. Bведение в системи управления базами данных. Часть 2

Нормальніше форми отношений и транзакции: учебное пособие/ Изд-е Башкирского ун-та. / Пушников А.Ю. - Уфа, 1999. - 138 с.

## Додатки

Додаток А

**public class** Main **extends** Application {  
  
 **private static final** LamborghiniDataBase ***ldb*** = **new** LamborghiniDataBase();  
 **private static final** Coder ***coder*** = **new** Coder();  
 **private static** TextField *deleteField* = **new** TextField();  
 **private static** String *enteredEmployee*;  
 **private static final** PDFApprover ***pdfApprover*** = **new** PDFApprover();  
  
 @Override  
 **public void** start(Stage primaryStage) **throws** IOException {  
 joinDriver();  
 ***ldb***.setUser(**"admin"**);  
 ***ldb***.setPassword(**"123lol"**);  
 ***ldb***.database();  
 Parent root = FXMLLoader.*load*(getClass().getResource(**"fxml/lamborghiniApp.fxml"**));  
 primaryStage.setTitle(**"LamborghiniAPP"**);  
 primaryStage.setScene(**new** Scene(root, 600, 500));  
 primaryStage.show();  
 }  
  
 **private void** joinDriver(){  
 **try** {  
 Class.*forName*(**"com.mysql.jdbc.Driver"**);  
 System.***out***.println(**"Driver loaded!"**);  
 } **catch** (ClassNotFoundException e) {  
 **throw new** IllegalStateException(**"Cannot find the driver in the classpath!"**, e);  
 }  
 }

**public class** LoadingController **extends** Controller{  
  
 @FXML TextField **login**;  
 @FXML PasswordField **password**;  
 **private** Coder **coder** = Main.*getCoder*();  
 **private static int** *mode*;  
  
 **public void** passwordInit(ActionEvent actionEvent) **throws** IOException, SQLException, InterruptedException {  
 refreshBD();  
 tryToLogin(actionEvent);  
 }  
  
 **private void** tryToLogin(ActionEvent actionEvent) **throws** IOException, SQLException, InterruptedException {  
 Employee current = **null**;  
 **int** pass = 0;  
 String loginText = **login**.getText();  
 String passwordText = **coder**.md5Custom(**password**.getText());  
 **for** (Employee employee : **employees**) {  
 **if** (employee.getLogin().equals(loginText)) {  
 **if** (passwordText.equals(employee.getPassword())) {  
 current = employee;  
 **if**(employee.isIsFired().equals(**"0"**))  
 pass=1;  
 **else** System.***out***.println(**"You have been fired :\*(((("**);  
 **break**;  
 } **else** {  
 System.***out***.println(**"Wrong Password"**);  
 *sleep*(1000);  
 }  
 }  
 }  
  
 **if**(pass == 1){  
 *sleep*(200);  
 **if**(*getMode*() == 1) {  
 EmployeeModeController emc = **new** EmployeeModeController();  
 emc.toEmployeeMode(actionEvent);  
 Main.*setEnteredEmployee*(loginText);  
 }  
 **else if**(*getMode*() == 3)  
 **if**(current.isAdmin()){  
 AdminModeController amc = **new** AdminModeController();  
 amc.toAdminMode(actionEvent);  
 Main.*setEnteredEmployee*(loginText);  
 System.***out***.println(**"Welcome"**);  
 }  
 **else** {  
 System.***out***.println(**"This user is not admin"**);  
 }  
 }  
 }  
  
 **public void** employeeEnter(ActionEvent actionEvent) **throws** IOException {  
 *mode* = 1;  
 enterLoginMenu(actionEvent);  
 }  
  
 **public void** adminEnter(ActionEvent actionEvent) **throws** IOException {  
 *mode* = 3;  
 enterLoginMenu(actionEvent);  
 }  
  
 **public void** userEnter(ActionEvent actionEvent) **throws** IOException, SQLException {  
 **ldb**.setUser(**"JustUser"**);  
 **ldb**.setPassword(**"123"**);  
 **ldb**.database();  
 **new** UserModeController().toUserMode(actionEvent);  
 }  
  
  
 **private static int** getMode(){  
 **return** *mode*;  
 }  
  
  
 **public void** exitProgram(ActionEvent actionEvent) {  
 Stage stage = (Stage) ((Node) actionEvent.getSource()).getScene().getWindow();  
 stage.close();  
 }  
}

**public class** AdminModeController **extends** Controller {  
  
 **private** TableView<Employee> **employeeTable** = **new** TableView<>();  
 **private** TableView<Car> **carTable** = **new** TableView<>();  
 **private** TableView<Client> **clientTable** = **new** TableView<>();  
 **private** TableView<ClientOrder> **clientOrdersTable** = **new** TableView<>();  
 @FXML CheckBox **isAdminReg**;  
 @FXML TextField **loginReg**;  
 @FXML TextField **passwordReg**;  
 @FXML TextField **nameReg**;  
 @FXML TextField **pNumberReg**;  
 @FXML TextField **cardNumberReg**;  
 @FXML TextField **adressReg**;  
 @FXML Label **error**;  
 @FXML TextField **orderIdT**;  
 **private static** ArrayList *find* = **new** ArrayList<>();  
 **private static** String *entered*;  
 **private** TextField **deleteField** = Main.*getDeleteField*();  
 **private static** TextField *textField* = **new** TextField();  
 **private** JavaFXFakerAdmin **javaFX** = **new** JavaFXFakerAdmin();  
 **private** Coder **coder** = Main.*getCoder*();  
  
 **public void** toAdminMode(ActionEvent actionEvent) **throws** IOException, SQLException {  
 String source;  
 source = **"adminMode/adminMode.fxml"**;  
 loading(source, actionEvent);  
 refreshBD();  
 }  
  
 **public void** toEmployeeDelete(ActionEvent actionEvent) **throws** SQLException, IOException {  
 String source = **"adminMode/deletePage.fxml"**;  
 refreshBD();  
 **javaFX**.createDeletePage(actionEvent, source, **deleteField**, 1);  
 }  
  
 **public void** toSubmitPage(ActionEvent actionEvent) **throws** IOException, SQLException {  
 String source = **"adminMode/submitPage.fxml"**;  
 refreshBD();  
 *entered* = **deleteField**.getText();  
 *find* = **javaFX**.createSubmitPage(actionEvent, source, *entered*, **employees**);  
 System.***out***.println(**employees**.get(1).getName());  
 }  
  
 **public void** toEmployeeList(ActionEvent actionEvent) **throws** IOException, SQLException {  
 String source = **"adminMode/listPage.fxml"**;  
 refreshBD();  
 ObservableList<Employee> obs = **javaFX**.javaFXEmployeeList(actionEvent, source, **employeeTable**);  
 obs.addAll(**employees**);  
 }  
  
 **public void** toAdminEmployeeEdit(ActionEvent actionEvent) **throws** IOException, SQLException {  
 String source;  
 source = **"adminMode/adminEmployeeEditPage.fxml"**;  
 loading(source, actionEvent);  
 refreshBD();  
 }  
  
 **public void** registerConf(ActionEvent actionEvent) **throws** SQLException, IOException {  
 refreshBD();  
 String loginText = **loginReg**.getText();  
 String passText = **passwordReg**.getText();  
 String nameText = **nameReg**.getText();  
 String pNumberText = **pNumberReg**.getText();  
 String cardNumberText = **cardNumberReg**.getText();  
 String adressText = **adressReg**.getText();  
 **error**.setText(**""**);  
 **boolean** pass = **true**;  
 **if**(loginText.isEmpty() || loginText.length()<4 || loginText.length()>17) {  
 **error**.setText(**error**.getText() + **"Please enter login field correctly (4-16 characters)\n"**);  
 pass = **false**;  
 }  
 **else** {  
 **for** (Employee employee : **employees**) {  
 **if** (loginText.equals(employee.getLogin())) {  
 pass = **false**;  
 **error**.setText(**error**.getText() + **"This login has already registered\n"**);  
 **break**;  
 }  
 }  
 }  
 **if**(passText.isEmpty() || passText.length()<3 || passText.length()>17){  
 **error**.setText(**error**.getText()+**"Please enter password field correctly (4-16 characters)\n"**);  
 pass = **false**;  
 }  
 passText = **coder**.md5Custom(passText);  
 **if**(nameText.isEmpty() || nameText.length()<3 || nameText.length()>33){  
 **error**.setText(**error**.getText()+**"Please enter name field correctly (4-32 characters)\n"**);  
 pass = **false**;  
 }  
 **if**(pNumberText.isEmpty() || pNumberText.length()<3 || pNumberText.length()>16){  
 **error**.setText(**error**.getText()+**"Please enter phone number field correctly (max 15 characters)\n"**);  
 pass = **false**;  
 }  
 cardNumberText = cardNumberText.replaceAll(**" "**, **""**);  
 **if**(cardNumberText.isEmpty() || cardNumberText.length()!=16){  
 **error**.setText(**error**.getText()+**"Please enter card number field correctly (XXXX XXXX XXXX XXXX)\n"**);  
 pass = **false**;  
 }  
 **if**(adressText.isEmpty()){  
 **error**.setText(**error**.getText()+**"Please enter address field correctly (max 30 characters)\n"**);  
 pass = **false**;  
 }  
 **if**(pass){  
 addEmployeeSubmit(loginText, passText, nameText, pNumberText, cardNumberText, adressText, **isAdminReg**.isSelected());  
 toAdminMode(actionEvent);  
 }  
 }  
  
 **private synchronized void** addEmployeeSubmit(String loginText, String passText, String nameText, String pNumberText, String cardNumberText, String adressText, **boolean** isAdmin) **throws** SQLException {  
 **ldb**.addEmployee(nameText, pNumberText, cardNumberText, adressText, passText, loginText, isAdmin);  
 }  
  
 **public void** submitOffered(ActionEvent actionEvent) **throws** SQLException {  
 refreshBD();  
 **boolean** hesCool = **false**;  
 **for** (Employee employee: (ArrayList<Employee>)*find*) {  
 **for** (**int** i = 0; i < **jobs**.size(); i++) {  
 **if**(**jobs**.get(i).getEmployeePersonnelNumber()==employee.getPersonnelNumber()) {  
 hesCool = **true**;  
 **break**;  
 }  
 }  
 **if**(!hesCool)  
 **ldb**.deleteEmployee(employee.getPersonnelNumber());  
 **else  
 ldb**.editEmployee(employee.getPersonnelNumber(),  
 employee.getName(),employee.getNumber(),  
 employee.getCardNumber(),employee.getAdress(),  
 employee.getPassword(),employee.getLogin(),  
 employee.isAdmin(),  
 Date.*valueOf*(LocalDate.*now*()).toString());  
 }  
 }  
  
 **public void** toAdminCarEdit(ActionEvent actionEvent) **throws** IOException, SQLException {  
 String source = **"adminMode/adminCarEditPage.fxml"**;  
 loading(source, actionEvent);  
 refreshBD();  
 }  
  
 **public void** toAdminCarRegister(ActionEvent actionEvent) **throws** IOException, SQLException {  
 String source = **"adminMode/adminCarRegisterPage.fxml"**;  
 refreshBD();  
 **javaFX**.registerCarFX(actionEvent, source, *textField*);  
 }  
  
 **public void** registerCar(ActionEvent actionEvent) **throws** SQLException {  
 refreshBD();  
 **boolean** isCool = **true**;  
 **for** (**int** i = 0; i < *textField*.getText().length(); i++) {  
 **if**(!Character.*isDigit*(*textField*.getText().charAt(i)))  
 {  
 isCool = **false**;  
 }  
 }  
 **if**(isCool)  
 **ldb**.addCar(**javaFX**.getColor(), **javaFX**.getModel(), Integer.*parseInt*(*textField*.getText()));  
 **else** System.***out***.println(**"Enter digits only"**);  
 }  
  
 **public void** toCarListPage(ActionEvent actionEvent) **throws** SQLException, IOException {  
 String source = **"adminMode/listPage.fxml"**;  
 refreshBD();  
 ObservableList<Car> obs = **javaFX**.javaFXCarList(actionEvent, source, **carTable**);  
 obs.addAll(**cars**);  
 }  
  
 **public void** toAdminClientEdit(ActionEvent actionEvent) **throws** IOException, SQLException {  
 String source = **"adminMode/adminClientEditPage.fxml"**;  
 loading(source, actionEvent);  
 }  
  
 **public void** toClientListPage(ActionEvent actionEvent) **throws** SQLException, IOException {  
 String source = **"adminMode/listPage.fxml"**;  
 refreshBD();  
 ObservableList<Client> obs = **javaFX**.javaFXClientList(actionEvent, source, **clientTable**);  
 obs.addAll(**clients**);  
 }  
  
 **public void** toAdminJobEdit(ActionEvent actionEvent) **throws** IOException {  
 String source = **"adminMode/adminJobEditPage.fxml"**;  
 loading(source, actionEvent);  
 }  
  
 **public void** toAdminClientOrderEdit(ActionEvent actionEvent) **throws** IOException {  
 String source = **"adminMode/adminClientOrderEditPage.fxml"**;  
 loading(source, actionEvent);  
 }  
  
 **public void** toJobListPage(ActionEvent actionEvent) **throws** SQLException, IOException {  
 refreshBD();  
 enterJobsView(actionEvent);  
 }  
  
 **public void** toClientOrdersListPage(ActionEvent actionEvent) **throws** SQLException, IOException {  
 String source = **"adminMode/listPage.fxml"**;  
 refreshBD();  
 ObservableList<ClientOrder> obs = **javaFX**.javaFXClientOrderList(actionEvent, source, **clientOrdersTable**);  
 obs.addAll(**clientOrders**);  
 }  
  
 **public void** enterAddJobView(ActionEvent actionEvent) **throws** IOException, SQLException {  
 String source = **"addJobView.fxml"**;  
 Parent root = FXMLLoader.*load*(getClass().getResource(**"fxml/"**+source));  
 refreshBD();  
 **jfx**.addJobFX(actionEvent, root,**cars**,**jobs**,**clients**,**clientOrders**,**employees**);  
 }  
  
 **public void** toOrderComplete(ActionEvent actionEvent) **throws** IOException {  
 String source = **"adminMode/completeOrder.fxml"**;  
 loading(source, actionEvent);  
 }  
  
 **public void** completeOrder(ActionEvent actionEvent) **throws** SQLException, IOException {  
 refreshBD();  
 String orderId = **orderIdT**.getText();  
 **boolean** isCool = **true**;  
 **for** (**int** i = 0; i < orderId.length(); i++) {  
 **if**(!Character.*isDigit*(orderId.charAt(i))) {  
 isCool = **false**;  
 **break**;  
 }  
 }  
 **if**(isCool) {  
 **int** orderIdInt = Integer.*parseInt*(orderId);  
 ClientOrder clientOrderIns = **null**;  
 **boolean** isFound = **false**;  
 **for** (ClientOrder clientOrder1 : **clientOrders**) {  
 **if** (clientOrder1.getId() == orderIdInt) {  
 clientOrderIns = clientOrder1;  
 isFound = **true**;  
 **break**;  
 }  
 }  
 **if** (isFound) {  
 **ldb**.editOrder(clientOrderIns.getId(), clientOrderIns.getPrice(), clientOrderIns.getDate(), clientOrderIns.getClientCardNumber(), !clientOrderIns.isExecuted(), clientOrderIns.getJobID());  
 **for** (**int** i = 0; i < **jobs**.size(); i++) {  
 **if**(**jobs**.get(i).getId() == clientOrderIns.getJobID()){  
 **for** (**int** j = 0; j < **cars**.size(); j++) {  
 **if**(**cars**.get(j).getSerialNumber()==**jobs**.get(i).getCarSerialNumber())  
 {  
 Car car = **cars**.get(j);  
 System.***out***.println(car.getSerialNumber());  
 **ldb**.editCar(car.getSerialNumber(), car.getColor(), car.getModel(), car.getPrice(), !car.isSold());  
 }  
 }  
  
 }  
 }  
 toAdminClientOrderEdit(actionEvent);  
 }  
 }  
 }  
}

**public class** LamborghiniDataBase {  
 **private static final** String ***url*** = **"jdbc:mysql://localhost:3306/lamborghiniDB"**;  
 **private** String **user**;  
 **private** String **password**;  
  
 **private static** Connection *con*;  
 **private static** Statement *stmt*;  
 **private static** ResultSet *rs*;  
  
 **public void** setUser(String user){  
 **this**.**user** = user;  
 }  
  
 **public void** setPassword(String password){  
 **this**.**password** = password;  
 }  
  
 **public void** database(){  
 **try** {  
 *con* = DriverManager.*getConnection*(***url***,**user**,**password**);  
 *stmt* = *con*.createStatement();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **public** ResultSet getTable(String table) **throws** SQLException {  
 String query = **"SELECT** *\** **from "**+table;  
 *rs* = *stmt*.executeQuery(query);  
 **return** *rs*;  
 }  
  
 **public void** finish() **throws** SQLException {  
 *con*.close();  
 *stmt*.close();  
 *rs*.close();  
 }  
  
 **public** ArrayList<Job> getJobs() **throws** SQLException {  
 ResultSet rs = getTable(**"job"**);  
 ArrayList<Job> jobs = **new** ArrayList<>();  
 **while** (rs.next()) {  
 **int** index = rs.getInt(1);  
 String type = rs.getString(2);  
 **int** employeeN = rs.getInt(3);  
 **int** carSerial = rs.getInt(4);  
 String date = rs.getDate(5).toString();  
 jobs.add(**new** Job(index, type, employeeN, carSerial, date));  
 }  
 **return** jobs;  
 }  
  
 **public** ArrayList<Employee> getEmployees() **throws** SQLException {  
 ResultSet rs = getTable(**"employee"**);  
 ArrayList<Employee> employee = **new** ArrayList<>();  
 **while** (rs.next()) {  
 **int** index = rs.getInt(1);  
 String name = rs.getString(2);  
 String tNumber = rs.getString(3);  
 String cardNumber = rs.getString(4);  
 String adress = rs.getString(5);  
 String password = rs.getString(6);  
 String login = rs.getString(7);  
 **boolean** isAdmin = rs.getBoolean(8);  
 String isFired;  
 **if**(rs.getDate(9)==**null**)  
 isFired = **"0"**;  
 **else** isFired= rs.getDate(9).toString();  
 employee.add(**new** Employee(index, name, tNumber, cardNumber, adress, password, login, isAdmin, isFired));  
 }  
 **return** employee;  
 }  
  
 **public** ArrayList<Car> getCars() **throws** SQLException {  
 ResultSet rs = getTable(**"car"**);  
 ArrayList<Car> car = **new** ArrayList<>();  
 **while** (rs.next()) {  
 **int** serialNumber = rs.getInt(1);  
 String color = rs.getString(2);  
 String model = rs.getString(3);  
 **int** price = rs.getInt(4);  
 **boolean** sold = rs.getBoolean(5);  
 car.add(**new** Car(serialNumber, color, model, price, sold));  
 }  
 **return** car;  
 }  
  
 **public** ArrayList<Client> getClients() **throws** SQLException {  
 ResultSet rs = getTable(**"client"**);  
 ArrayList<Client> clients = **new** ArrayList<>();  
 **while** (rs.next()) {  
 String cardNumber = rs.getString(1);  
 String name = rs.getString(2);  
 String tNumber = rs.getString(3);  
 clients.add(**new** Client(cardNumber, name, tNumber));  
 }  
 **return** clients;  
 }  
  
 **public** ArrayList<ClientOrder> getClientOrders() **throws** SQLException {  
 ResultSet rs = getTable(**"clientorder"**);  
 ArrayList<ClientOrder> clientOrders = **new** ArrayList<>();  
 **while** (rs.next()) {  
 **int** id = rs.getInt(1);  
 **int** price = rs.getInt(2);  
 String date = rs.getDate(3).toString();  
 String clientCardNumber = rs.getString(4);  
 **int** jobID = rs.getInt(6);  
 **boolean** executed = rs.getBoolean(5);  
 clientOrders.add(**new** ClientOrder(id,price,date,clientCardNumber, jobID, executed));  
 }  
 **return** clientOrders;  
 }  
  
 **public void** addJob(String type, **int** employeeN, **int** carSerial, String date) **throws** SQLException {  
 String query = **"INSERT INTO job (ID, Type, EmployeePersonnelNumber, CarSerialNumber, JobTime) VALUES (NULL, '"**+type+**"', "**+employeeN+**", "**+carSerial+**", '"**+date+**"')"**;  
 *stmt*.executeUpdate(query);  
 }  
  
 **public void** addEmployee(String name, String tNumber, String cardNumber, String address, String password, String login, **boolean** isAdmin) **throws** SQLException {  
 String query = **"INSERT INTO employee (PersonnelNumber, Name, TNumber, CardNumber, Adress, Password, Login, isAdmin, isFired) VALUES (NULL, '"**+name+**"', '"**+tNumber+**"', '"**+cardNumber+**"', '"**+address+**"', '"**+password+**"', '"**+login+**"', "**+isAdmin+**", NULL)"**;  
 *stmt*.executeUpdate(query);  
 }  
  
 **public void** addClient(String name, String tNumber, String cardNumber) **throws** SQLException {  
 String query = **"INSERT INTO client (Name, TNumber, CardNumber) VALUES ('"**+name+**"', '"**+tNumber+**"', '"**+cardNumber+**"')"**;  
 *stmt*.executeUpdate(query);  
 }  
  
 **public void** addCar(String color, String model, **int** price) **throws** SQLException {  
 String query = **"INSERT INTO car (SerialNumber, Color, Model, Price) VALUES (NULL, '"**+color+**"', '"**+model+**"', "**+price+**")"**;  
 *stmt*.executeUpdate(query);  
 }  
  
 **public void** addClientOrder(**int** price, String data, String clientCardNumber, **int** jobID) **throws** SQLException {  
 String query = **"INSERT INTO clientorder (ID, Price, Data, ClientCardNumber, jobID) VALUES (NULL, "**+price+**", '"**+data+**"', '"**+clientCardNumber+**"', "**+jobID+**")"**;  
 *stmt*.executeUpdate(query);  
 }  
  
 **public void** deleteEmployee(**int** ID) **throws** SQLException {  
 String query = **"DELETE FROM employee WHERE PersonnelNumber='"**+ID+**"'"**;  
 *stmt*.executeUpdate(query);  
 }  
  
 **public void** editEmployee(**int** personnelNumber, String name, String tNumber, String cardNumber, String address, String password, String login, **boolean** isAdmin, String isFired) **throws** SQLException {  
 String query = **"UPDATE employee SET Name = '"**+name+**"', TNumber = '"**+tNumber+**"', CardNumber = '"**+cardNumber+**"', Adress = '"**+address+**"', Password = '"**+password+**"', Login = '"**+login+**"', isAdmin= "**+isAdmin+**", isFired = '"**+isFired+**"' WHERE PersonnelNumber = "**+personnelNumber;  
 *stmt*.executeUpdate(query);  
 }  
  
 **public void** editOrder(**int** id, **int** price, String data, String clientCardNumber, **boolean** executed, **int** jobID) **throws** SQLException {  
 String query = **"UPDATE clientorder SET Price = "**+price+**", Data = '"**+data+**"', ClientCardNumber = '"**+clientCardNumber+**"', Executed = "**+executed+**", jobID = "**+jobID+**" WHERE ID = "**+id;  
 *stmt*.executeUpdate(query);  
 }  
  
 **public void** editCar(**int** serial, String color, String model, **int** price, **boolean** sold) **throws** SQLException {  
 String query = **"UPDATE car SET Color = '"**+color+**"', Model = '"**+model+**"', Price = "**+price+**", sold = "**+sold+**" WHERE SerialNumber = "**+serial;  
 *stmt*.executeUpdate(query);  
 }  
}

**public class** Controller {  
  
 JavaFXFaker **jfx** = **new** JavaFXFaker();  
 LamborghiniDataBase **ldb** = Main.*getLdb*();  
 ArrayList<Job> **jobs**;  
 ArrayList<Car> **cars**;  
 ArrayList<Employee> **employees**;  
 ArrayList<Client> **clients**;  
 ArrayList<ClientOrder> **clientOrders**;  
 **private static** String *prevPage*;  
  
 **public void** setPrevPage(String prevPage){  
 Controller.*prevPage* = prevPage;  
 }  
  
 **void** loading(String source, ActionEvent actionEvent) **throws** IOException {  
 Parent root = FXMLLoader.*load*(getClass().getResource(**"fxml/"**+source));  
 Scene scene = **new** Scene(root);  
 Stage stage = (Stage) ((Node) actionEvent.getSource()).getScene().getWindow();  
 stage.setScene(scene);  
 stage.show();  
 *prevPage* = source;  
 }  
  
 **public void** toMain(ActionEvent actionEvent) **throws** IOException {  
 String source = **"LamborghiniApp.fxml"**;  
 loading(source, actionEvent);  
 }  
  
 **void** enterLoginMenu(ActionEvent actionEvent) **throws** IOException{  
 String source = **"login.fxml"**;  
 loading(source, actionEvent);  
 }  
  
 **public void** enterRegisterMenu(ActionEvent actionEvent) **throws** IOException{  
 String source = **"adminMode/adminEmployeeRegisterPage.fxml"**;  
 loading(source, actionEvent);  
 }  
  
 **void** refreshBD() **throws** SQLException {  
 **clientOrders** = **ldb**.getClientOrders();  
 **jobs** = **ldb**.getJobs();  
 **cars** = **ldb**.getCars();  
 **clients** = **ldb**.getClients();  
 **employees** = **ldb**.getEmployees();  
 }  
  
 **public void** enterJobsView(ActionEvent actionEvent) **throws** IOException, SQLException {  
 refreshBD();  
 TableView<Job> jobTable = **new** TableView<>();  
 ObservableList<Job> obs = **jfx**.jobViewFX(actionEvent, jobTable);  
 obs.addAll(**jobs**);  
 }  
  
 **public void** back(ActionEvent actionEvent) **throws** IOException {  
 loading(*prevPage*, actionEvent);  
 }  
  
}