**Софийски университет „Св. Климент Охридски“**

*Факултет по математика и информатика*

*Специалност: “Информационни системи”*

*Курс: 2, Група: 3*

**КУРСОВ ПРОЕКТ**

**по**

**Бази от данни**

***Изготвил:***

Никола Петров Кирилов(ФН: 71986)

***Преподавател:***

Доц. Радослава Христова

 София 2021

**1.Обхват на модела. Дефиниране на задачата**

Базата от данни за метрото ще съхранява информация за пътниците на метрото. Разработената база от данни ще обслужва вътрешната система на метрото(т.е ще е за служителите) и няма да се показва на сайт. Метрополитен разполага с гишета за вадене на карти на десетки места в София, а на всяка метростанция се продават електронни билети. Всеки офис на метрото се определя от името на дадената метростанция, в която се намира.

,,Метрополитен“ АД работи с клиенти(пътници), чиято информация се пази в системата. В системата биват записани e-mail, телефонен номер, ЕГН, имена на клиента. Пътниците се определят по ЕГН и телефонен номер. В офисите на метростанциите работят служители. Служителите се определят еднозначно по служебен номер. За служителите се пази следната информация в системата: три имена, телефонен номер и служебен номер. Всеки служител отговаря за офиса, в който работи.

Офисите на ,,Метрополитен“ се занимават с изваждане или подновяване на карти за градски транспорт и продаване на електронни билети, а гишетата само продават електронни билети.

В системата се пази информация за картите: имена на притежател, валидност(брой дни оставащи), уникален номер на карта, вид карта според това дали е преференциална или не.

В системата се пази информация за билетите: уникален номер на билет, дата на

закупуване, дали е използван за превоз в метрото.

**1.2.Множества от същности и техните атрибути**

Офиси – име на офис, адрес

Клиенти(пътници) – име, телефонен номер, e – mail, ЕГН, тип

клиент(непреференциран, преференциран)

Служители – име, телефонен номер, служебен номер

Карти - имена на притежател, валидност(брой дни оставащи), уникален

номер на карта, вид карта

Билети – уникален номер на билет, дата на закупуване, валидност

**1.3.Домейн на атрибутите**

Офиси – име на офис: низ, адрес: низ

Клиенти(пътници) – име: низ, телефонен номер: низ, e – mail:низ, ЕГН: низ,

Служители – име: низ, телефонен номер: низ, служебен номер: низ

Карти - валидност: инт, уникален номер на карта: инт,

Билети – уникален номер на билет: инт, дата на закупуване: низ, валидност: булева стойност

**1.4.Връзки**

В един Офис работят много Служители. Служителите работят точно в един Офис.

Карта може да се извади от точно един офис

Картата може да се подновява в много офиси

Един клиент може да си извади точно една карта

Една карта се притежава точно от един пътник

Един клиент може да си закупи неограничен брой билети

Един билет може да бъде закупен от точно един пътник

**1.5. Ограничения по единствена стойност, референтна цялостност и друг тип ограничения**

Офиси – име: еднозначно определя офиса

Клиенти – ЕГН и телефонен номер: еднозначно определят клиента, e-mail: проверка за валиден e-mail

Служители – сл. номер: еднозначно определя служителя,

Карти – уникален номер на карта: еднозначно определя картата

Билети – уникален номер на билет: еднозначно определя билета

**1.6. Правила и проверки**

Да се проверяват валидността на картите и билетите.

Да се проверява и валидността на имейлите.

**1.7. Източници**

https://www.metropolitan.bg/

**2. Представяне на базата от данни чрез E/R модел**



**3. Представяне на базата от данни чрез релационен модел**

Първо ще преобразуваме множествата и същностите, заедно с техните атрибути, а после и isa йерархиите.

Staff (staffID, name, telephone, branchName)

Passengers (EGN, name, e-mail, telephone )

Cards( cardID, EGN, PassengerName, validity, branchName)

Tickets (ticketID, validity, date, branchName)

Servs (EGN, name, staffID)

Goes (EGN, name, branchName)

Renew (EGN, ownerName, cardID, branchName)

Тъй като в модела не може едновременно да съществуват Branches, които да са и Cardbranches и Ticketbranches най-удачно е да ползваме или ООП подход, или NULL. Аз избирам NULL подход, тъй като смятам, че е най-подходящият за моят проект.

Branches (name, address)

**Финално моделът ни изглежда така:**

Staff (staffID, name, telephone, branchName)

Passengers (EGN, name, e-mail, telephone )

Cards (cardID, EGN, validity, branchName)

Tickets (ticketID, validity, date, branchName)

Servs (EGN, staffID)

Goes (EGN, branchName)

Renew (EGN, cardID, branchName)

Branches (name, address)

**4. Нормализация**

За схемата на БД по-горе са в сила следните ФЗ:

**ФЗ-1:** staffID -> name, telephone, branchName (Staff)

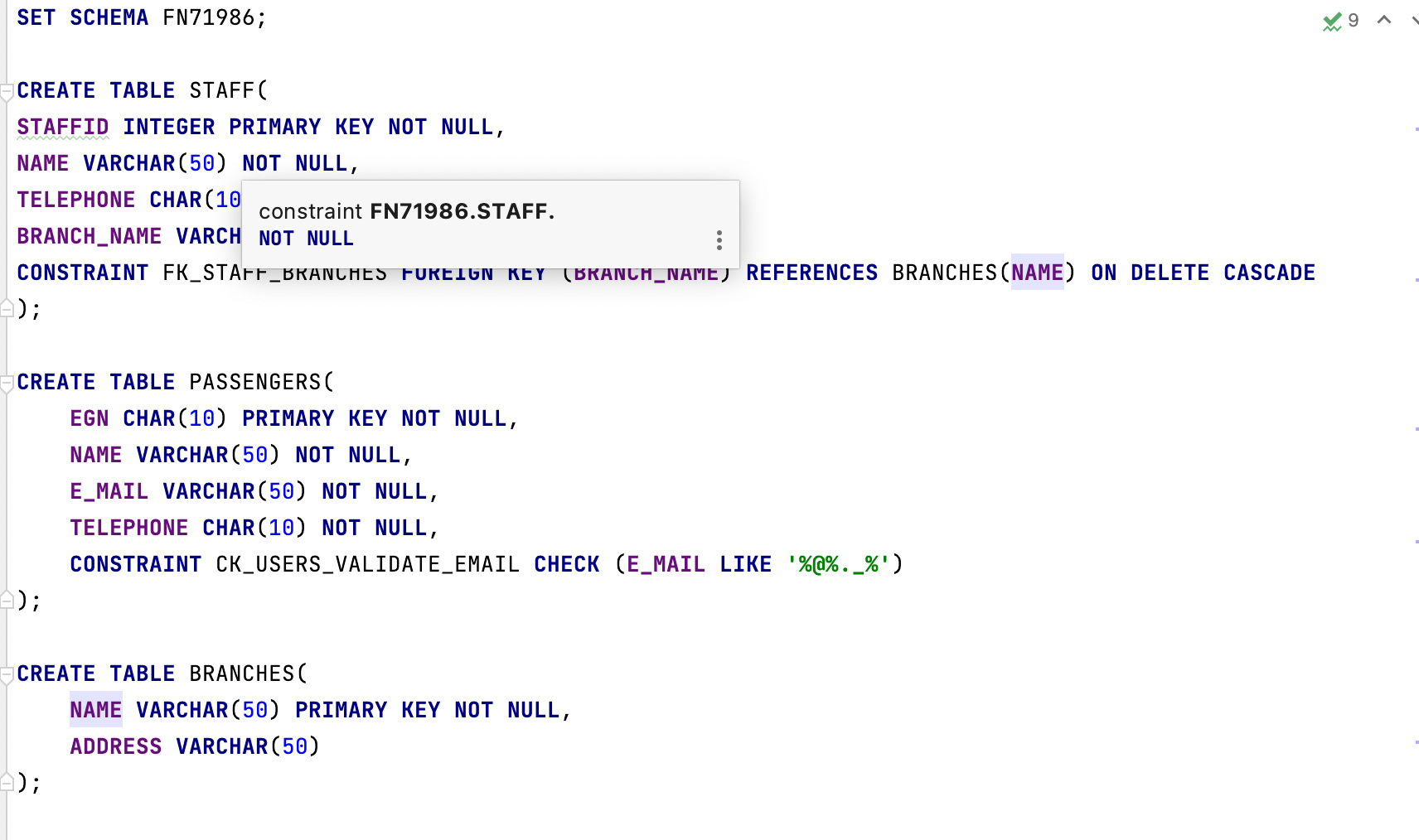
**ФЗ-2:** EGN -> name, e-mail, telephone (Passengers)

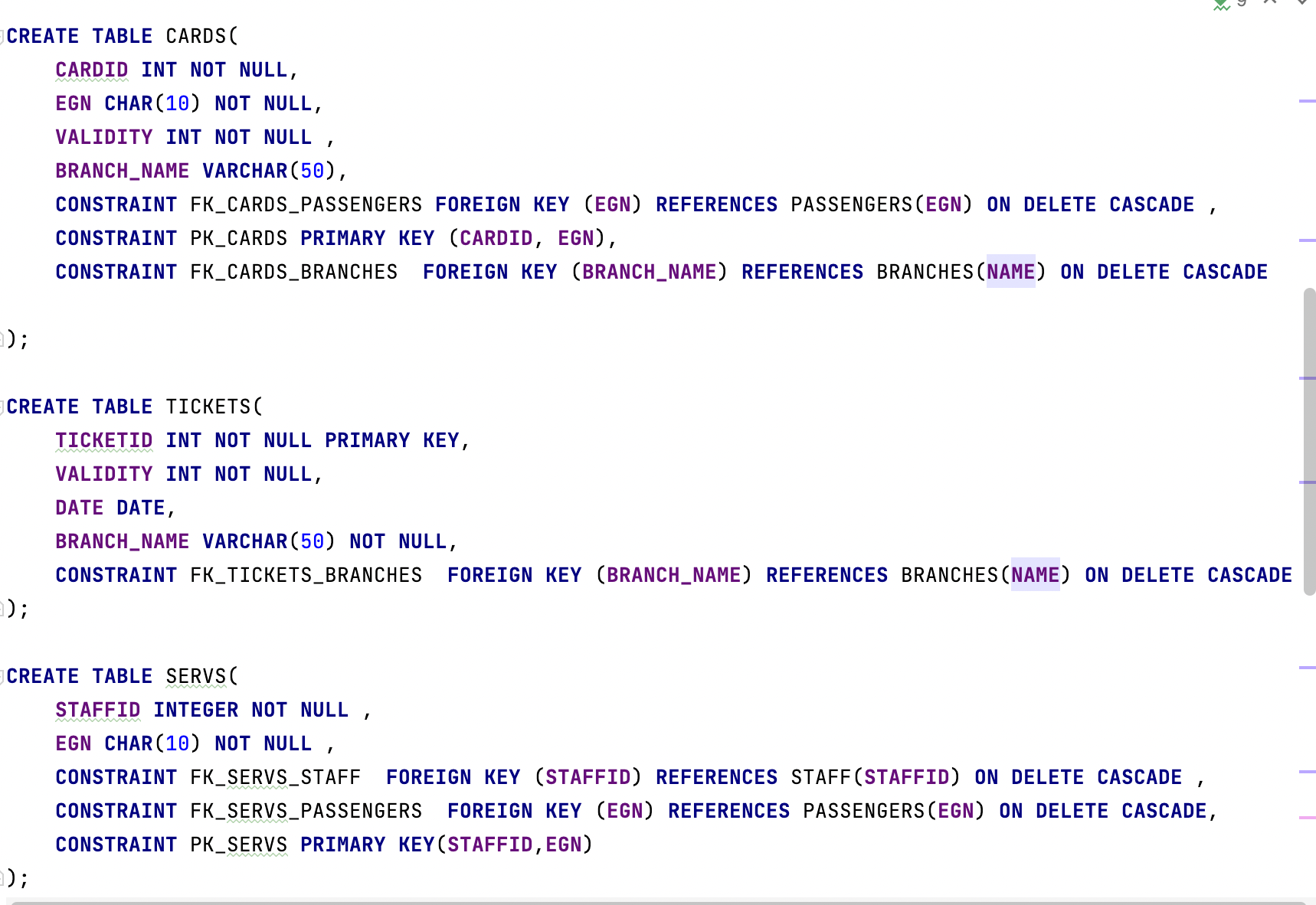
**ФЗ-3:** cardID, EGN -> PassengerName, validity, branchName (Cards)

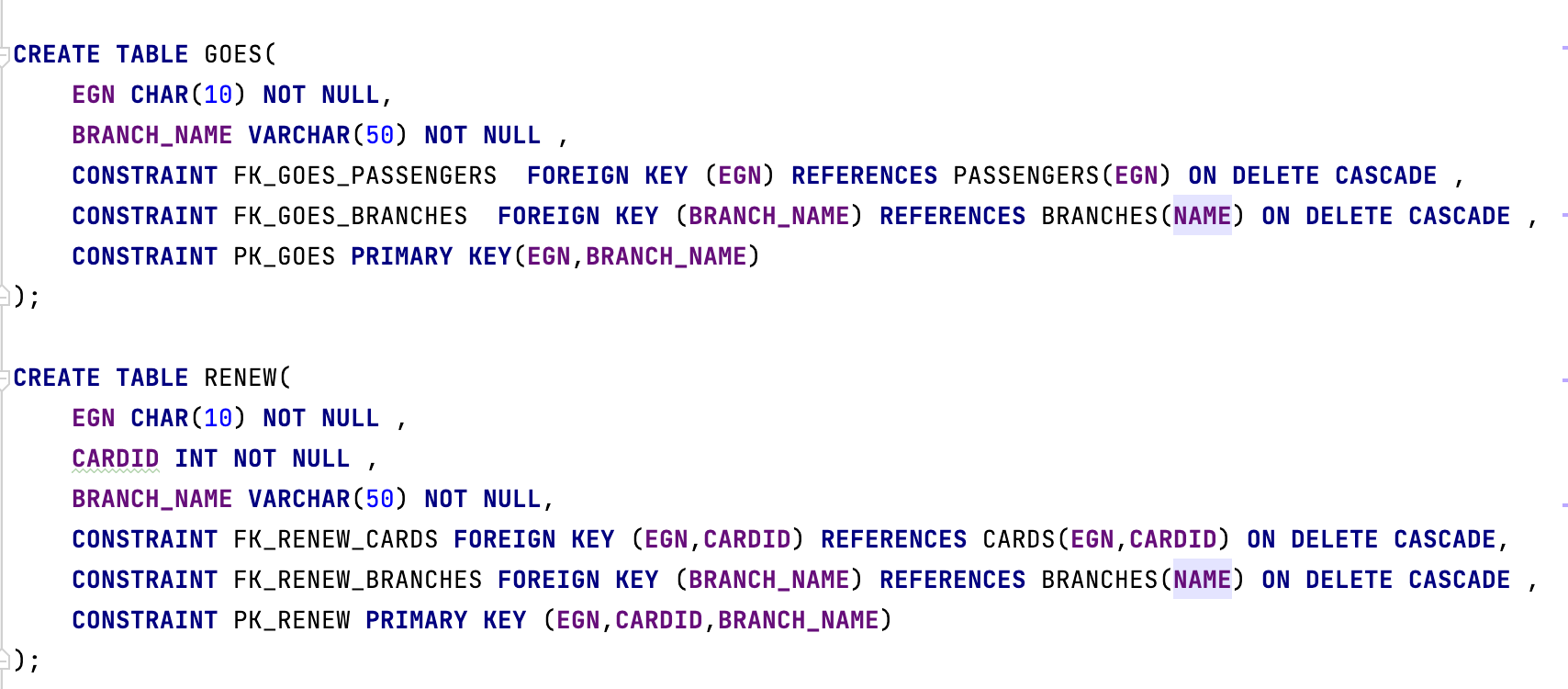
**ФЗ-4:** ticketID -> validity, date, branchName (Tickets)

**Вижда се, че релациите много към много двата ключа образуват ключа за връзката, тоест са НФБК**

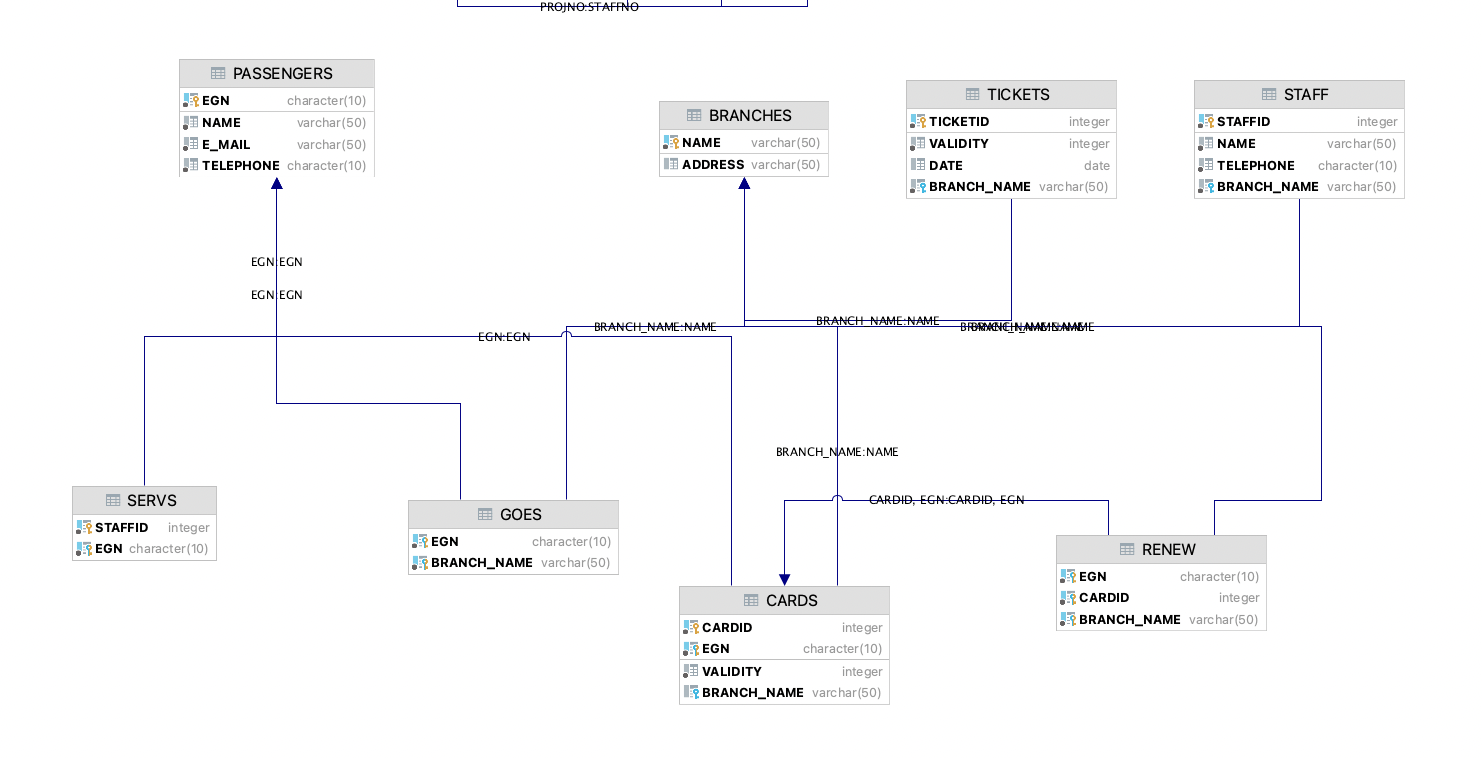
**5.Създаване на таблици чрез DataGrip и SQL (DDL)**

****

****

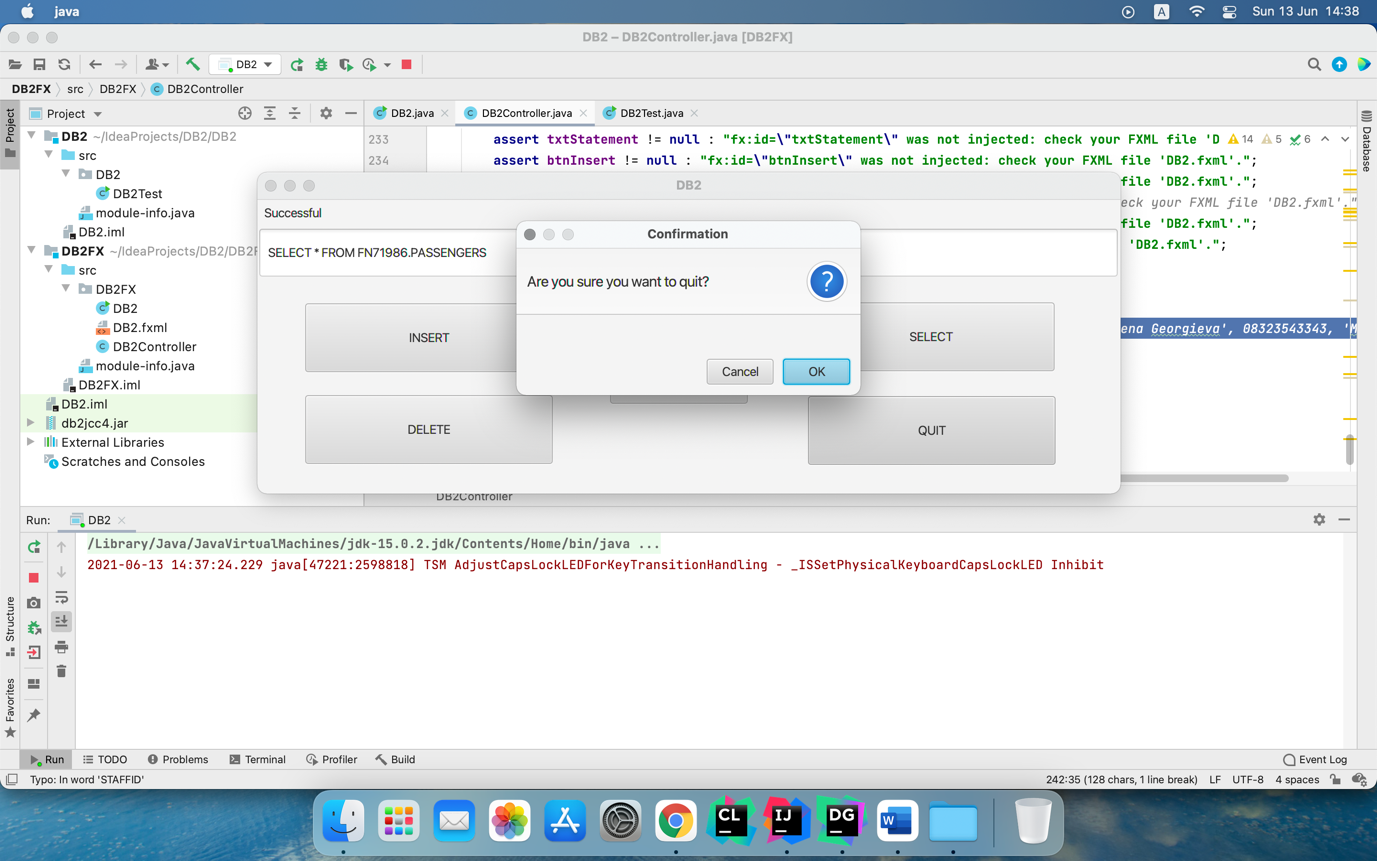
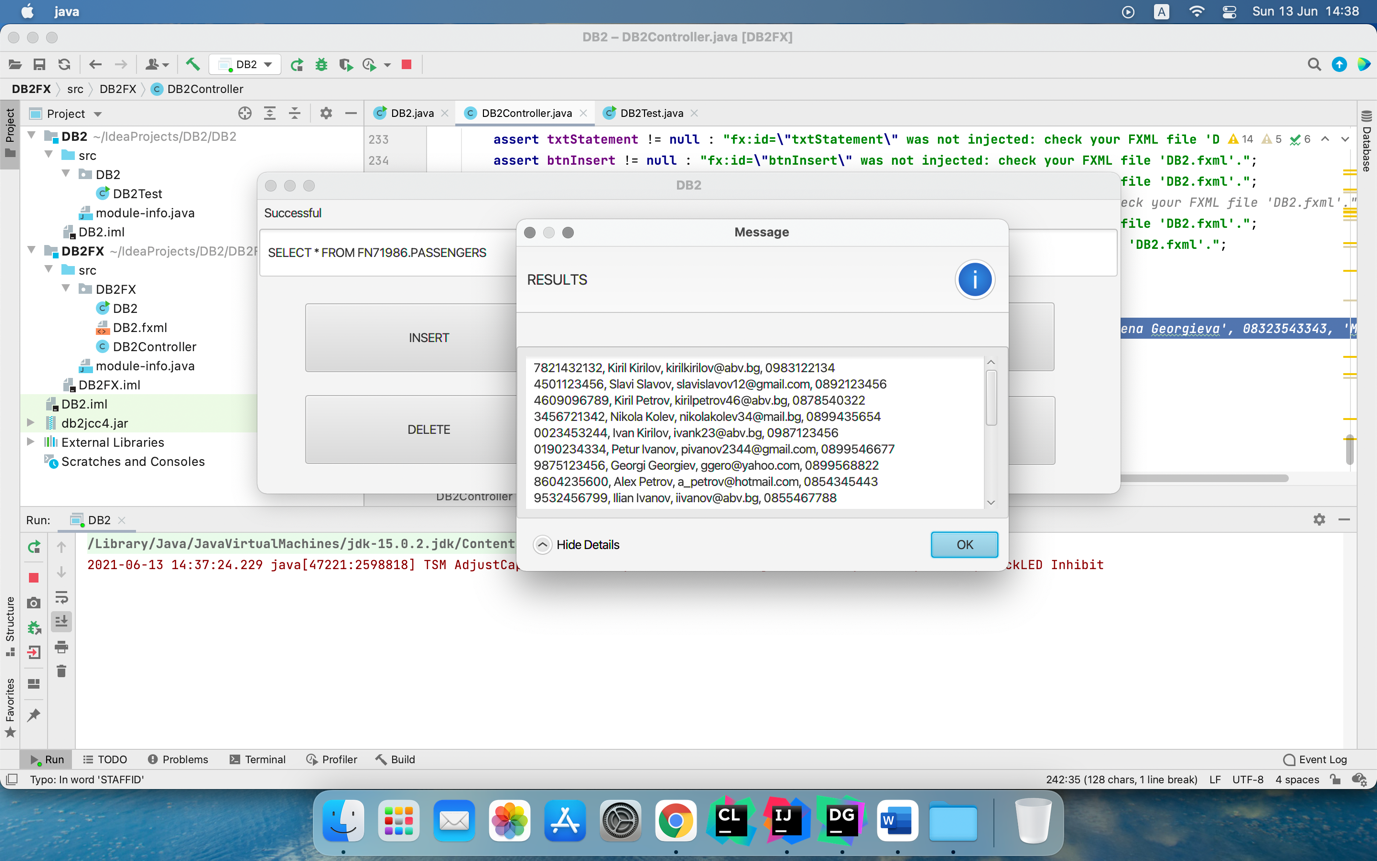
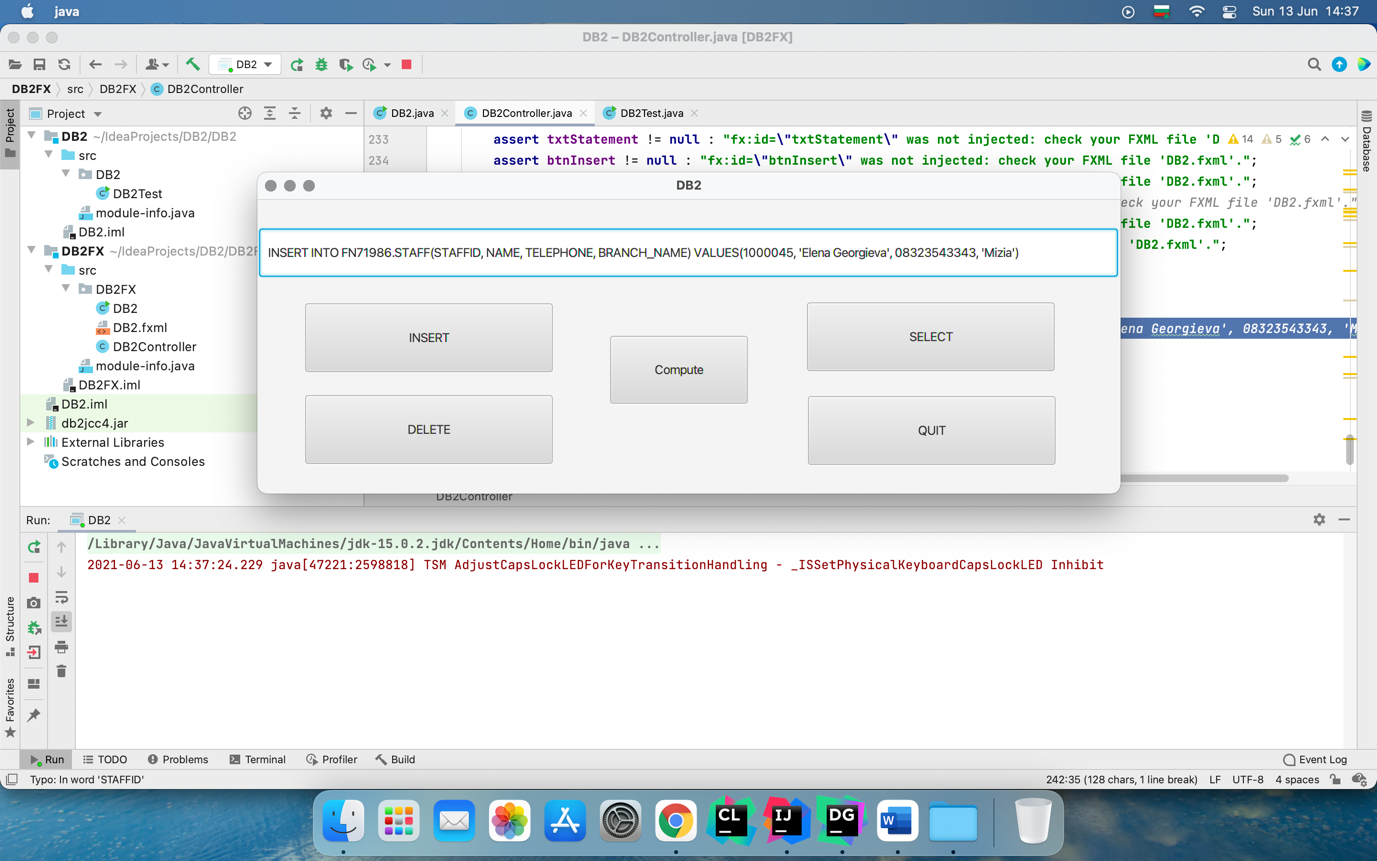
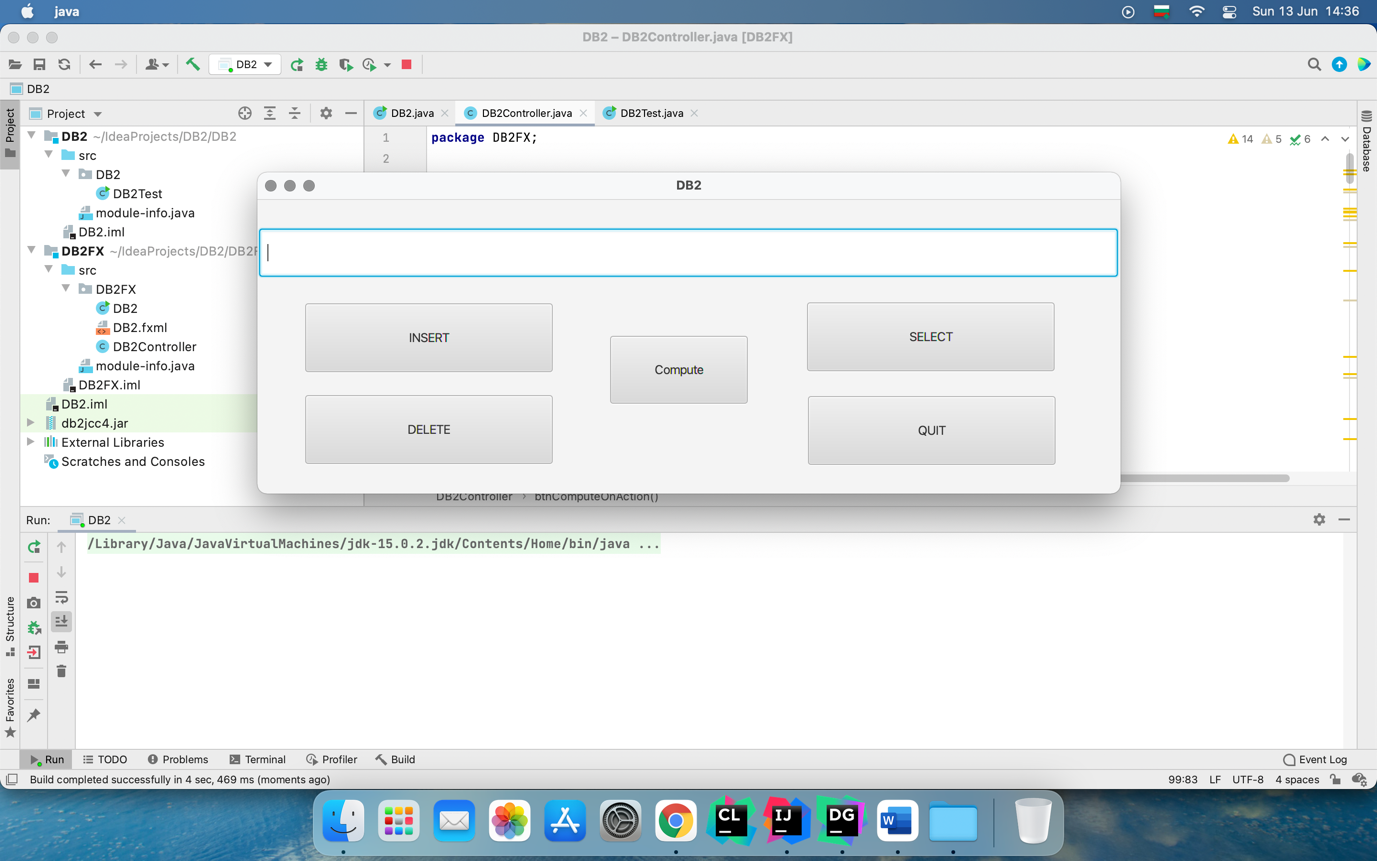
****

Така изглежда диаграмата в **DataGrip** на вече създадените таблици с техните релации.



**6.Приложения**

Направих избор да ползвам за връзката Java, като използвам JavaFX за създаване на графичен UI, който да е по-лесно използваем от потребителя. Използвам модули за свързването на отделните елементи между основния клас и графичната част. Програмата сама познава какво е действието с натискането на бутона, а бутоните **SELECT**, **INSERT**, **DELETE** просто слагат думичката в текстовото поле, за да се предпази потребителя от неволното слагане на знак или празно пространство пред ключовата дума, което може да доведе то счупване на програмата. Също така е добавена защита, която пази от изписването на думите **ALTER** и **DROP**, които могат да направят нежелани от нас въздействия върху базата от данни.



**КОДЪТ НА ПРИЛОЖЕНИЕТО:**

**DB2Test(от Мудъл, но модифициран с цел поправяне на грешки и по-добър error handling)**

**package** DB2;  
**import** java.sql.\*;  
  
  
**public class** DB2Test {  
  
 **private** Connection **connection**;  
 **private** Statement **statement**;  
 **private** ResultSet **resultSet**;  
  
  
 **public void** openConnection(){  
  
*// Step 1: Load IBM DB2 JDBC driver* **try** {  
  
 DriverManager.*registerDriver*(**new** com.ibm.db2.jcc.DB2Driver());  
  
 }  
  
 **catch**(Exception cnfex) {  
  
 System.***out***.println(**"Problem in loading or registering IBM DB2 JDBC driver"**);  
  
 cnfex.printStackTrace();  
 }  
  
*// Step 2: Opening database connection* **try** {  
  
 **connection** = DriverManager.*getConnection*(**"jdbc:db2://62.44.108.24:50000/SAMPLE"**, **"db2admin"**, **"db2admin"**);  
  
 **statement** = **connection**.createStatement();  
  
 }  
  
 **catch**(SQLException s){  
  
 s.printStackTrace();  
  
 }  
  
 }  
  
 **public void** closeConnection(){  
  
 **try** {  
  
 **if**(**null** != **connection**) {  
  
 *// cleanup resources, once after processing* **resultSet**.close();  
  
 **statement**.close();  
  
  
 *// and then finally close connection* **connection**.close();  
  
 }  
  
 }  
  
 **catch** (SQLException s) {  
  
 s.printStackTrace();  
  
 }  
  
 }  
  
 **public** String select(String stmnt) {  
  
 **try**{  
 **resultSet** = **statement**.executeQuery(stmnt);  
 StringBuilder result = **new** StringBuilder();  
 ResultSetMetaData resultSetMetaData = **resultSet**.getMetaData();  
 **int** column = resultSetMetaData.getColumnCount();  
  
 **while**(**resultSet**.next()) {  
  
 **for** (**int** i = 1; i <= column; i++) {  
  
 result.append(**resultSet**.getString(i));  
  
 **if** (i == column)  
 {  
 result.append(**" \n"**);  
 }  
  
 **else** {  
 result.append( **", "**);  
 }  
 }  
 }  
 **return** result.toString();  
 }  
 **catch** (SQLException s)  
 {  
 s.printStackTrace();  
 **return "Error occurred"**;  
 }  
 }  
  
 **public boolean** insert(String stmnt) {  
  
 **try**{  
 **if**(checkWord(stmnt, **"INSERT"**))  
 {  
 **statement**.executeUpdate(stmnt);  
 **return true**;  
 }  
 **else** {  
 **return false**;  
 }  
  
 }  
  
 **catch** (SQLException s)  
 {  
  
 s.printStackTrace();  
 **return false**;  
  
 }  
  
 }  
  
  
 **public boolean** delete(String stmnt) {  
  
 **try**{  
  
 **if**(checkWord(stmnt, **"DELETE"**))  
 {  
 **statement**.executeUpdate(stmnt);  
 **return true**;  
 }  
 **else** {  
 **return false**;  
 }  
  
 }  
  
 **catch** (SQLException s){  
  
 s.printStackTrace();  
 **return false**;  
  
 }  
 }  
 **private boolean** checkWord(String stringToSearch, String wordToMatch)  
 {  
 **int** intIndex = stringToSearch.indexOf(wordToMatch);  
 **if** (intIndex == -1)  
 {  
 **return false**;  
 } **else** {  
 **return true**;  
 }  
 }  
  
}

**DB2.fxml**

*<?***xml version="1.0" encoding="UTF-8"***?>  
  
<?***import javafx.scene.control.Button***?>  
<?***import javafx.scene.control.Label***?>  
<?***import javafx.scene.control.TextField***?>  
<?***import javafx.scene.layout.AnchorPane***?>*<**AnchorPane prefHeight="308.0" prefWidth="903.0" xmlns="http://javafx.com/javafx/15.0.1" xmlns:fx="http://javafx.com/fxml/1" fx:controller="DB2FX.DB2Controller"**>  
 <**children**>  
 <**TextField fx:id="txtStatement" layoutX="2.0" layoutY="31.0" onAction="#txtStatementOnAction" prefHeight="50.0" prefWidth="898.0" promptText="Please intert the statement for DB2"** />  
 <**Button fx:id="btnInsert" layoutX="50.0" layoutY="109.0" mnemonicParsing="false" onAction="#btnInsertOnAction" prefHeight="72.0" prefWidth="259.0" text="INSERT"** />  
 <**Button fx:id="btnSelect" layoutX="575.0" layoutY="108.0" mnemonicParsing="false" onAction="#btnSelectOnAction" prefHeight="72.0" prefWidth="259.0" text="SELECT"** />  
 <**Button fx:id="btnDelete" layoutX="50.0" layoutY="205.0" mnemonicParsing="false" onAction="#btnDeleteOnAction" prefHeight="72.0" prefWidth="259.0" text="DELETE"** />  
 <**Button fx:id="btnQuit" layoutX="576.0" layoutY="206.0" mnemonicParsing="false" onAction="#btnQuitOnAction" prefHeight="72.0" prefWidth="259.0" text="QUIT"** />  
 <**Label fx:id="lblError" layoutX="7.0" layoutY="6.0" prefHeight="17.0" prefWidth="203.0"** />  
 <**Button fx:id="btnCompute" layoutX="369.0" layoutY="143.0" mnemonicParsing="false" onAction="#btnComputeOnAction" prefHeight="71.0" prefWidth="144.0" text="Compute"** />  
 </**children**>  
</**AnchorPane**>

**DB2Controller**

**package** DB2FX;  
  
**import** java.net.URL;  
**import** java.util.Locale;  
**import** java.util.Optional;  
**import** java.util.ResourceBundle;  
  
**import** javafx.event.ActionEvent;  
**import** javafx.fxml.FXML;  
**import** javafx.scene.control.\*;  
**import** DB2.\*;  
  
**public class** DB2Controller {  
  
 @FXML  
 **private** ResourceBundle **resources**;  
  
 @FXML  
 **private** URL **location**;  
  
 @FXML  
 **private** TextField **txtStatement**;  
  
 @FXML  
 **private** Button **btnInsert**;  
  
 @FXML  
 **private** Button **btnSelect**;  
  
 @FXML  
 **private** Button **btnCompute**;  
  
 *//@FXML  
 //private TextField txtColumnNumber;* @FXML  
 **private** Button **btnDelete**;  
  
 @FXML  
 **private** Button **btnQuit**;  
  
 @FXML  
 **private** Label **lblError**;  
  
  
 **private** DB2Test **db2Object** = **new** DB2Test();  
  
 **private** String **statement** = **new** String();  
 **private int column**;  
  
 @FXML  
 **void** btnDeleteOnAction(ActionEvent event)  
 {  
 **txtStatement**.setText(**"DELETE"**);  
 */\*statement = "";  
 lblError.setText("");  
 if(txtStatement.getText() != null)  
 {  
 statement = txtStatement.getText();  
 if(db2Object.delete(statement))  
 {  
 lblError.setText("Successful");  
 }  
 else  
 {  
 lblError.setText("Error occurred");  
 }  
 }\*/* }  
  
 @FXML  
 **void** btnInsertOnAction(ActionEvent event)  
 {  
 **txtStatement**.setText(**"INSERT"**);  
 */\*statement = "";  
 lblError.setText("");  
 if(txtStatement.getText()!=null)  
 {  
 statement = txtStatement.getText();  
 if(db2Object.insert(statement))  
 {  
 lblError.setText("Successful");  
 }  
 else  
 {  
 lblError.setText("Error occurred");  
 }  
 }  
 \*/* }  
  
 @FXML  
 **void** btnComputeOnAction(ActionEvent event)  
 {  
 **lblError**.setText(**""**);  
 **statement** = **""**;  
 **if** (**txtStatement**.getText() != **null** && isSafeString(**txtStatement**.getText()))  
 {  
 **statement** = **txtStatement**.getText();  
 **if** (**statement**.toUpperCase(Locale.***ROOT***).startsWith(**"INSERT"**))  
 {  
 **if** (**db2Object**.insert(**statement**))  
 {  
 **lblError**.setText(**"Successful"**);  
 } **else** {  
 **lblError**.setText(**"Error occurred"**);  
 }  
 }  
  
  
 **if** (**statement**.toUpperCase(Locale.***ROOT***).startsWith(**"SELECT"**))  
 {  
 Alert alert = **new** Alert(Alert.AlertType.***INFORMATION***);  
 *// statement = txtStatement.getText();* **if** (!**db2Object**.select(**statement**).equals(**"Error occurred"**))  
 {  
 **lblError**.setText(**"Successful"**);  
 String resultSelect = **db2Object**.select(**statement**);  
 alert.setHeaderText(**"RESULTS"**);  
 TextArea area = **new** TextArea(resultSelect);  
 area.setWrapText(**true**);  
 area.setEditable(**false**);  
 alert.getDialogPane().setExpandableContent(area);  
 alert.setResizable(**true**);  
 alert.setHeaderText(**"RESULTS"**);  
 alert.showAndWait();  
  
 } **else** {  
 **lblError**.setText(**"Error occurred"**);  
 alert.setHeaderText(**"Error occurred"**);  
 alert.showAndWait();  
 }  
 }  
  
 **if** (**statement**.toUpperCase(Locale.***ROOT***).startsWith(**"DELETE"**))  
 {  
 **if** (**db2Object**.delete(**statement**))  
 {  
 **lblError**.setText(**"Successful"**);  
 } **else** {  
 **lblError**.setText(**"Error occurred"**);  
 }  
 }  
  
  
 }  
 **else** {  
 Alert alertError = **new** Alert(Alert.AlertType.***ERROR***);  
 alertError.setHeaderText(**"You entered a forbidden word!!!"**);  
 alertError.showAndWait();  
 }  
 }  
  
  
 @FXML  
 **void** btnQuitOnAction(ActionEvent event)  
 {  
 Alert alert = **new** Alert(Alert.AlertType.***CONFIRMATION***);  
 alert.setHeaderText(**"Are you sure you want to quit?"**);  
  
 Optional<ButtonType> result = alert.showAndWait();  
 **if** (result.isPresent() && result.get() == ButtonType.***OK***)  
 {  
 System.*exit*(0);  
 **db2Object**.closeConnection();  
 }  
 }  
  
 @FXML  
 **void** btnSelectOnAction(ActionEvent event)  
 {  
 **txtStatement**.setText(**"SELECT"**);  
 */\*statement = "";  
 if(txtStatement.getText() != null)  
 {  
 Alert alert = new Alert(Alert.AlertType.INFORMATION);  
 statement = txtStatement.getText();  
 if(!db2Object.select(statement).equals("Error occurred"))  
 {  
 lblError.setText("Successful");  
 String resultSelect = db2Object.select(statement);  
 alert.setHeaderText("RESULTS");  
 TextArea area = new TextArea(resultSelect);  
 area.setWrapText(true);  
 area.setEditable(false);  
 alert.getDialogPane().setExpandableContent(area);  
 alert.setResizable(true);  
 alert.setHeaderText("RESULTS");  
 alert.showAndWait();  
  
 }  
 else  
 {  
 lblError.setText("Error occurred");  
 alert.setHeaderText("Error occurred");  
 alert.showAndWait();  
 }  
  
 }  
  
 column = 0;\*/* }  
  
  
 **private int** countCommas(String str)  
 {  
 **int** commas = 0;  
 **for**(**int** i = 0; i < str.length(); i++)  
 {  
 **if**(str.charAt(i) == **','**) commas++;  
 }  
 **return** commas+1;  
  
 }  
 **private boolean** isSafeString(String str)  
 {  
 **return** !str.toUpperCase(Locale.***ROOT***).contains(**"DROP"**) && !str.toUpperCase(Locale.***ROOT***).contains(**"ALTER"**);  
 }  
  
 @FXML  
 **void** txtStatementOnAction(ActionEvent event)  
 {  
 }  
  
 @FXML  
 **void** initialize() {  
 **assert txtStatement** != **null** : **"fx:id=\"txtStatement\" was not injected: check your FXML file 'DB2.fxml'."**;  
 **assert btnInsert** != **null** : **"fx:id=\"btnInsert\" was not injected: check your FXML file 'DB2.fxml'."**;  
 **assert btnSelect** != **null** : **"fx:id=\"btnSelect\" was not injected: check your FXML file 'DB2.fxml'."**;  
 *//assert txtColumnNumber != null : "fx:id=\"txtColumnNumber\" was not injected: check your FXML file 'DB2.fxml'.";* **assert btnDelete** != **null** : **"fx:id=\"btnDelete\" was not injected: check your FXML file 'DB2.fxml'."**;  
 **assert btnQuit** != **null** : **"fx:id=\"btnQuit\" was not injected: check your FXML file 'DB2.fxml'."**;  
 **db2Object**.openConnection();  
 }  
  
 *//INSERT INTO FN71986.STAFF(STAFFID, NAME, TELEPHONE, BRANCH\_NAME) VALUES(1000045, 'Elena Georgieva', 08323543343, 'Mizia')  
 //DELETE FROM FN71986.CARDS WHERE CARDID = 131254  
 //SELECT STAFFID, NAME, TELEPHONE, BRANCH\_NAME FROM FN71986.STAFF  
 //DELETE FROM FN71986.STAFF WHERE NAME = 'Alex Dimitrov'*}

**DB2(main class to start fxml)**

**package** DB2FX;  
  
**import** javafx.application.Application;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.Parent;  
**import** javafx.scene.Scene;  
**import** javafx.stage.Stage;  
  
**public class** DB2 **extends** Application {  
  
 @Override  
 **public void** start(Stage primaryStage) **throws** Exception{  
 Parent root = FXMLLoader.*load*(getClass().getResource(**"DB2.fxml"**));  
 primaryStage.setTitle(**"DB2"**);  
 primaryStage.setScene(**new** Scene(root));  
 primaryStage.show();  
 }  
  
  
 **public static void** main(String[] args) {  
 *launch*(args);  
 }

}