ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

« КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет

по курсовому проекту по

Базам Данных

Выполнил:

Баймухаметова Гузель Аскатовна  
группа 09-215

Казань 2014

**Цели и задачи БД.**

Целью данной работы является автоматизирование работы туристического агентства ,т.е. упрощение процесса работы персонала с клиентом, документами, а также уменьшить степень его участия в процессах обработки информации и облегчение управлением базами данных. В результате был получен легкий в использовании программный продукт, обладающий спектром возможностей необходимым для работы туристического агентства и имеющий понятный интерфейс.

**Описание реализации проекта**

**Описание схемы БД**

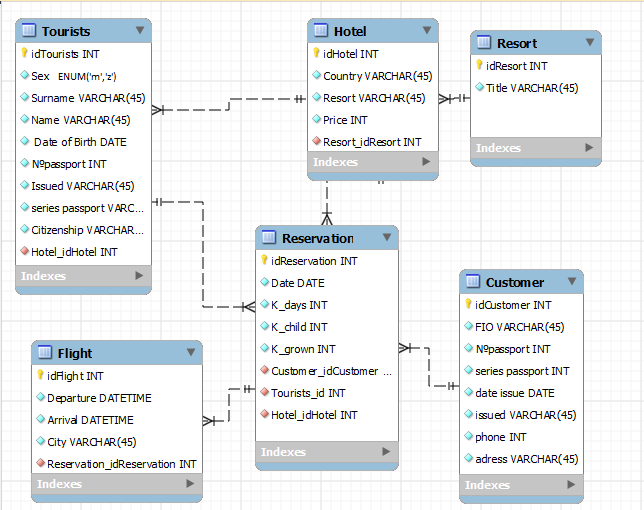


Таблица **Сustomer** :

Сustomer\_id-первичный ключ таблицы

FIO – ФИО заказчика

Num\_Passport- номер паспорта

Series\_passport-серия паспорта

Date\_issue-дата выдачи

Issued-кем выдан

Phone-телефон

Adress-адрес

Таблица **Resort**

id –первичный ключ таблицы

Title - Курорт

Таблица **Hotel** :

Id- первичный ключ таблицы

Country - Страна

Resort –Количестов звезд

Price –Стоимость путевки

ResortId внешний ключ таблицы

Таблица **Tourists**:

id - первичный ключ таблицы

Sex - пол

Surname -фамилия

Name -Имя

Date\_of\_birth –Дата рождения

Num\_Passport – Номер паспорта

Series\_passport –серия паспорта

Issued – Кем выдан

Citizenship -Гражданство

HotelId внешний ключ таблицы

Таблица **Reservation**:

id - первичный ключ таблицы

date\_p – дата вылета

K\_days количество дней

K\_child количество детей

K\_grown количество взрослых

CustomerId внешний ключ таблицы

TouristsId внешний ключ таблицы

Таблица **Flight**:

id -первичный ключ таблицы

Departure –дата вылета

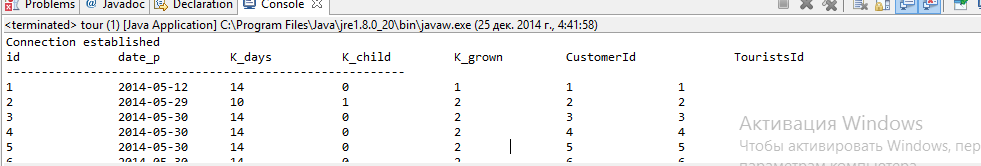
Arrival –дата прилета

City – города вылета

ReservationId внешний ключ таблицы

**Консольное приложение.**

Подключается к базе данных и выводит информацию хранящуюся таблицах в консоль. Так же есть возможность добавить новую запись в таблицу.



**Листинг кода.**

package myapp;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.sql.ResultSetMetaData;

import java.sql.Date;

import java.sql.DatabaseMetaData;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Scanner;

///////////////////////////////////////////////////////////////////////////////////////////

public class tour{

String url = "jdbc:mysql://localhost:3306/";// драйвер, 3306-порт подключения

String dbName = "Tour";

String driver = "com.mysql.jdbc.Driver";

String userName = "root";

String password = "";

private static String dbURL = "jdbc:mysql://localhost:3306/tour?user=root&password=root";//user-ключ значение

private static String tableName = "reservation";

// jdbc Connection

private static Connection conn = null;

private static Statement stmt = null;

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

private static void createConnection()

{try

{Class.forName("com.mysql.jdbc.Driver").newInstance();

//Get a connection

conn = DriverManager.getConnection(dbURL); // подключение с БД

System.out.println("Connection established");

}

catch (Exception except)

{except.printStackTrace();

}

}

////// метод для вывода в консоль таблицы Reservation/////////////////////////////////////////////////////////////////

private static void selectReservation()

{

try

{stmt = conn.createStatement();//Выражение с sql запросами

ResultSet results = stmt.executeQuery("select \* from " + tableName);

ResultSetMetaData rsmd = results.getMetaData();

int numberCols = rsmd.getColumnCount();

for (int i=1; i<=numberCols; i++)

{ //print Column Names

System.out.print(rsmd.getColumnLabel(i)+"\t\t");

}

String format = "%-16s%-16s%-16s%-16s%-16s%-16s%-16s%n";

while(results.next())

{ String str = "";

for (int i = 1; i <= numberCols; i++) {

str += results.getString(i) + " "; }

System.out.println(str);

int id = results.getInt(1);

Date dat = results.getDate(2);

int K\_days = results.getInt(3);

int K\_child = results.getInt(4);

int K\_grown = results.getInt(5);

int CustomerId = results.getInt(6);

int TouristsId = results.getInt(7);

System.out.printf(format, id, dat,K\_days,K\_child, K\_grown,CustomerId,TouristsId );

}

results.close();

stmt.close();

}

catch (SQLException sqlExcept)

{sqlExcept.printStackTrace();

}}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

private static void insertReservation( int id, Date dat, int K\_days,int K\_child, int K\_grown,int CustomerId, int TouristsId, int HotelId)

{ try {

stmt = conn.createStatement();

stmt.execute("insert into Reservation(" + id + ",'" + dat+ "','" + K\_days + "'," + K\_child + "','" + CustomerId + "','" + TouristsId+ "','" + HotelId+")");

} catch (SQLException e) {

e.printStackTrace();}}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

private static void shutdown()

{try

{if (stmt != null)

{stmt.close();

}

if (conn != null)

{

DriverManager.getConnection(dbURL + ";shutdown=true");

conn.close();

}

}

catch (SQLException sqlExcept)

{ }

}

// метод для получения списка таблиц БД

private static ArrayList<String> getListOfTables() {

ArrayList<String> list = new ArrayList<>();

try {

DatabaseMetaData dbmd =conn.getMetaData(); // хранение метаданных о БД

ResultSet rs = dbmd.getTables(null, null, null, null); // получить инфу о таблицах в виде строк

while (rs.next()) {

String str = rs.getString(1);

list.add(str);

}

} catch (SQLException e) {

e.printStackTrace();

}

return list;

}

// метод для вывода в консоль таблицы по названию

private static void selectTable() {

try {

stmt = conn.createStatement();

ResultSet results = stmt.executeQuery("select \* from " + tableName);

ResultSetMetaData rsmd = results.getMetaData(); // получить метаданные о строках

int numberCols = rsmd.getColumnCount();

for (int i = 1; i <= numberCols; i++) {

// print Column Names

System.out.print(rsmd.getColumnLabel(i) + "\t\t");

}

System.out.println("\n---------------------------------------------------------");

while (results.next()) {

for (int i = 1; i <= numberCols; i++) {

String str = results.getString(i);

System.out.print(str + "\t\t\t");

}

System.out.println();

}

results.close();

stmt.close();

} catch (SQLException sqlExcept) {

sqlExcept.printStackTrace();

}

}

public static void main(String[] args) {

SimpleDateFormat f = new SimpleDateFormat("dd.MM.yyyy hh:mm");

createConnection();

//insertReservation(11, new Date(01,12,2014),1,1,1,1,1);

ArrayList<String> list = getListOfTables();

System.out.println(list);

Scanner scan = new Scanner(System.in);

int number = scan.nextInt();

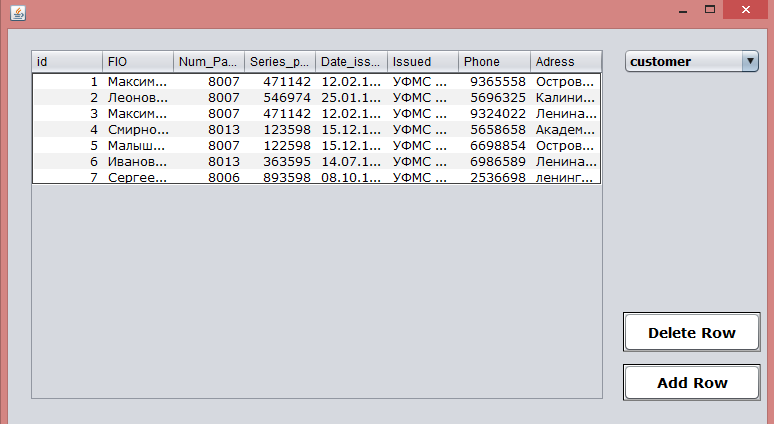
selectTable();

shutdown();

}}

**Приложение с GUI.**

Просмотр данных с выбором таблиц из выпадающего списка, полный CRUD:

добавление, удаление и изменение данных.

Реализован класс MyTableModel унаследованный от абстрактного класса AbstractTableModel. Данный класс нужен для того чтобы изменять модель JTable.

**Листинг кода.**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

**package** myapp;

**import** java.sql.Connection;

**import** java.sql.DatabaseMetaData;

**import** java.sql.Date;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.ResultSetMetaData;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**import** java.util.ArrayList;

**import** java.util.logging.Level;

**import** java.util.logging.Logger;

**import** javax.swing.table.AbstractTableModel;

/\*\*

\*

\* **@author** N

\*/

**public** **class** CRUDApp **extends** javax.swing.JFrame {

**static** Connection *conn*;

**static** Statement *stmt*;

**static** String *dbURL* = "jdbc:mysql://localhost:3306/tour?user=root&password=root";

**static** ArrayList<String> *tableNames* = **new** ArrayList<String>();

MyTableModel dbm = **new** MyTableModel(**true**);

**public** **static** **void** tableNamesBoxContentChange(ArrayList<String> str)

{

**for**(**int** i = 0; i < str.size(); i++)

*tableNamesBox*.addItem(str.get(i));

}

**public** **static** **void** getDataBaseTableNames() **throws** SQLException

{

DatabaseMetaData dbmd = *conn*.getMetaData();

String[] types = {"TABLE"};

ResultSet rs = dbmd.getTables(**null**, **null**, "%", types);

**while** (rs.next()) {

*tableNames*.add(rs.getString(3));

}

}

**public** **static** **void** connectDB()

{

**try** {

Class.*forName*("com.mysql.jdbc.Driver").newInstance();

} **catch** (Exception ex) {

Logger.*getLogger*(CRUDApp.**class**.getName()).log(Level.***SEVERE***, **null**, ex);

}

**try** {

// объект-соединение с БД

*conn* = DriverManager.*getConnection*(*dbURL*);

} **catch** (SQLException ex) {

Logger.*getLogger*(CRUDApp.**class**.getName()).log(Level.***SEVERE***, **null**, ex);

}

}

**public** **void** tableContentShow(String tableName)

{

**try** {

*stmt* = *conn*.createStatement();

} **catch** (SQLException ex) {

Logger.*getLogger*(CRUDApp.**class**.getName()).log(Level.***SEVERE***, **null**, ex);

}

ResultSet rs;

**try** {

rs = *stmt*.executeQuery("select \* from "+ tableName);

**try** {

dbm.setDataSource(rs);

} **catch** (Exception ex) {

Logger.*getLogger*(CRUDApp.**class**.getName()).log(Level.***SEVERE***, **null**, ex);

}

myTable.setModel(dbm);

} **catch** (SQLException ex) {

Logger.*getLogger*(CRUDApp.**class**.getName()).log(Level.***SEVERE***, **null**, ex);

}

}

/\*\*

\* Creates new form CRUDApp

\*/

**public** CRUDApp() {

initComponents();

}

/\*\*

\* This method is called from within the constructor to initialize the form.

\* WARNING: Do NOT modify this code. The content of this method is always

\* regenerated by the Form Editor.

\*/

@SuppressWarnings("unchecked")

// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents

**private** **void** initComponents() {

jScrollPane1 = **new** javax.swing.JScrollPane();

myTable = **new** javax.swing.JTable();

*tableNamesBox* = **new** javax.swing.JComboBox();

addRowBtn = **new** javax.swing.JButton();

deleteRowBtn = **new** javax.swing.JButton();

setDefaultCloseOperation(javax.swing.WindowConstants.***EXIT\_ON\_CLOSE***);

myTable.setBorder(javax.swing.BorderFactory.*createEtchedBorder*(**null**, java.awt.Color.***darkGray***));

myTable.setFont(**new** java.awt.Font("Verdana", 0, 12)); // NOI18N

myTable.setModel(**new** javax.swing.table.DefaultTableModel(

**new** Object [][] {

},

**new** String [] {

}

));

myTable.setToolTipText("");

myTable.setGridColor(**new** java.awt.Color(51, 0, 51));

myTable.setRequestFocusEnabled(**false**);

jScrollPane1.setViewportView(myTable);

*tableNamesBox*.setFont(**new** java.awt.Font("Verdana", 1, 12)); // NOI18N

*tableNamesBox*.addActionListener(**new** java.awt.event.ActionListener() {

**public** **void** actionPerformed(java.awt.event.ActionEvent evt) {

tableNamesBoxActionPerformed(evt);

}

});

addRowBtn.setBackground(**new** java.awt.Color(255, 255, 255));

addRowBtn.setFont(**new** java.awt.Font("Verdana", 1, 14)); // NOI18N

addRowBtn.setText("Add Row");

addRowBtn.setBorder(javax.swing.BorderFactory.*createBevelBorder*(javax.swing.border.BevelBorder.***RAISED***, java.awt.Color.***black***, java.awt.Color.***white***, **null**, **null**));

addRowBtn.addActionListener(**new** java.awt.event.ActionListener() {

**public** **void** actionPerformed(java.awt.event.ActionEvent evt) {

addRowBtnActionPerformed(evt);

}

});

deleteRowBtn.setBackground(**new** java.awt.Color(255, 255, 255));

deleteRowBtn.setFont(**new** java.awt.Font("Verdana", 1, 14)); // NOI18N

deleteRowBtn.setText("Delete Row");

deleteRowBtn.setToolTipText("");

deleteRowBtn.setBorder(javax.swing.BorderFactory.*createBevelBorder*(javax.swing.border.BevelBorder.***RAISED***, java.awt.Color.***black***, java.awt.Color.***white***, **null**, **null**));

deleteRowBtn.addActionListener(**new** java.awt.event.ActionListener() {

**public** **void** actionPerformed(java.awt.event.ActionEvent evt) {

deleteRowBtnActionPerformed(evt);

}

});

javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());

getContentPane().setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.***LEADING***)

.addGroup(layout.createSequentialGroup()

.addGap(21, 21, 21)

.addComponent(jScrollPane1, javax.swing.GroupLayout.***DEFAULT\_SIZE***, 576, Short.***MAX\_VALUE***)

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.***TRAILING***, **false**)

.addComponent(*tableNamesBox*, 0, javax.swing.GroupLayout.***DEFAULT\_SIZE***, Short.***MAX\_VALUE***)

.addComponent(deleteRowBtn, javax.swing.GroupLayout.***DEFAULT\_SIZE***, 138, Short.***MAX\_VALUE***)

.addComponent(addRowBtn, javax.swing.GroupLayout.***DEFAULT\_SIZE***, javax.swing.GroupLayout.***DEFAULT\_SIZE***, Short.***MAX\_VALUE***))

.addContainerGap())

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.***LEADING***)

.addGroup(layout.createSequentialGroup()

.addGap(19, 19, 19)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.***LEADING***)

.addGroup(layout.createSequentialGroup()

.addComponent(*tableNamesBox*, javax.swing.GroupLayout.***PREFERRED\_SIZE***, javax.swing.GroupLayout.***DEFAULT\_SIZE***, javax.swing.GroupLayout.***PREFERRED\_SIZE***)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.***RELATED***, javax.swing.GroupLayout.***DEFAULT\_SIZE***, Short.***MAX\_VALUE***)

.addComponent(deleteRowBtn, javax.swing.GroupLayout.***PREFERRED\_SIZE***, 40, javax.swing.GroupLayout.***PREFERRED\_SIZE***)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.***UNRELATED***)

.addComponent(addRowBtn, javax.swing.GroupLayout.***PREFERRED\_SIZE***, 37, javax.swing.GroupLayout.***PREFERRED\_SIZE***))

.addComponent(jScrollPane1, javax.swing.GroupLayout.***DEFAULT\_SIZE***, 353, Short.***MAX\_VALUE***))

.addGap(26, 26, 26))

);

pack();

}// </editor-fold>//GEN-END:initComponents

**private** **void** tableNamesBoxActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_tableNamesBoxActionPerformed

// **TODO** add your handling code here:

tableContentShow(*tableNames*.get(*tableNamesBox*.getSelectedIndex()));

}//GEN-LAST:event\_tableNamesBoxActionPerformed

**private** **void** addRowBtnActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_addRowBtnActionPerformed

dbm.addRow();

dbm.fireTableStructureChanged();

myTable.setModel(dbm);

}//GEN-LAST:event\_addRowBtnActionPerformed

**private** **void** deleteRowBtnActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_deleteRowBtnActionPerformed

**int** row = myTable.getSelectedRow();

**try** {

*stmt* = *conn*.createStatement();

*stmt*.executeUpdate("delete from " + *tableNames*.get(*tableNamesBox*.getSelectedIndex()) + " where " + myTable.getColumnName(0)+ "= " + myTable.getValueAt(row, 0).toString());

dbm.data.remove(row);

dbm.fireTableStructureChanged();

myTable.setModel(dbm);

} **catch** (SQLException ex) {

Logger.*getLogger*(CRUDApp.**class**.getName()).log(Level.***SEVERE***, **null**, ex);

}

}//GEN-LAST:event\_deleteRowBtnActionPerformed

/\*\*

\* **@param** args the command line arguments

\*/

**public** **static** **void** main(String args[]) {

/\* Set the Nimbus look and feel \*/

//<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">

/\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.

\* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html

\*/

**try** {

**for** (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.*getInstalledLookAndFeels*()) {

**if** ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.*setLookAndFeel*(info.getClassName());

**break**;

}

}

} **catch** (ClassNotFoundException ex) {

java.util.logging.Logger.*getLogger*(CRUDApp.**class**.getName()).log(java.util.logging.Level.***SEVERE***, **null**, ex);

} **catch** (InstantiationException ex) {

java.util.logging.Logger.*getLogger*(CRUDApp.**class**.getName()).log(java.util.logging.Level.***SEVERE***, **null**, ex);

} **catch** (IllegalAccessException ex) {

java.util.logging.Logger.*getLogger*(CRUDApp.**class**.getName()).log(java.util.logging.Level.***SEVERE***, **null**, ex);

} **catch** (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.*getLogger*(CRUDApp.**class**.getName()).log(java.util.logging.Level.***SEVERE***, **null**, ex);

}

//</editor-fold>

/\* Create and display the form \*/

java.awt.EventQueue.*invokeLater*(**new** Runnable() {

**public** **void** run() {

**try** {

**new** CRUDApp().setVisible(**true**);

*connectDB*();

*getDataBaseTableNames*();

*tableNamesBoxContentChange*(*tableNames*);

} **catch** (SQLException ex) {

Logger.*getLogger*(CRUDApp.**class**.getName()).log(Level.***SEVERE***, **null**, ex);

}

}

});

}

**public** **class** MyTableModel **extends** AbstractTableModel {

// здесь мы будем хранить названия столбцов

**private** ArrayList<String> columnNames = **new** ArrayList<String>();

// список типов столбцов

**private** ArrayList<Class> columnTypes = **new** ArrayList<Class>();

// хранилище для полученных данных из базы данных

**private** ArrayList<ArrayList<Object>> data = **new** ArrayList<ArrayList<Object>>();

// признак редактирования таблицы

**private** **boolean** editable;

// конструктор позволяет задать возможность редактирования

**public** MyTableModel(**boolean** editable) {

**this**.editable = editable;

}

// количество строк

**public** **int** getRowCount() {

**return** data.size();

}

// количество столбцов

**public** **int** getColumnCount() {

**return** columnNames.size();

}

// тип данных столбца

**public** Class getColumnClass(**int** column) {

**return** columnTypes.get(column);

}

// название столбца

**public** String getColumnName(**int** column) {

**return** columnNames.get(column);

}

// данные в ячейке

**public** Object getValueAt(**int** row, **int** column) {

**return** data.get(row).get(column);

}

**public** ArrayList<Object> getRow(**int** row)

{

**return** data.get(row);

}

// возможность редактирования

**public** **boolean** isCellEditable(**int** row, **int** column) {

**return** editable;

}

// замена значения ячейки

**public** **void** setValueAt( Object value, **int** row, **int** column)

{

data.get(row).set(column, value);

**try**{

String columnName = getColumnName(column);

String val;

**if**(value **instanceof** String)

val = "'" + value.toString()+ "'";

**else**

val = value.toString();

String query =

"UPDATE " + *tableNames*.get(*tableNamesBox*.getSelectedIndex()) + " SET " + columnName + " = " + val +

" WHERE " + myTable.getColumnName(0) + " = " + myTable.getValueAt(row, 0).toString();

PreparedStatement pstmt =

*conn*.prepareStatement(query);

pstmt.executeUpdate();

}

**catch** (Exception ex)

{

System.***err***.println(ex);

}

}

// получение данных из объекта ResultSet

**public** **void** setDataSource(ResultSet rs) **throws** Exception {

// удаляем прежние данные

data.clear();

columnNames.clear();

columnTypes.clear();

// получаем вспомогательную информацию о столбцах

ResultSetMetaData rsmd = rs.getMetaData();

**int** columnCount = rsmd.getColumnCount();

**for** ( **int** i=0; i<columnCount; i++) {

// название столбца

columnNames.add(rsmd.getColumnName(i+1));

// тип столбца

Class type =

Class.*forName*(rsmd.getColumnClassName(i+1));

columnTypes.add(type);

}

// получаем данные

**while** ( rs.next() ) {

// здесь будем хранить ячейки одной строки

ArrayList<Object> row = **new** ArrayList<Object>();

**for** ( **int** i=0; i<columnCount; i++) {

row.add(rs.getObject(i+1));

}

data.add(row);

}

// сообщаем об изменениях в структуре данных

fireTableStructureChanged();

}

**public** **void** addRow()

{

**try** {

**int** rowCount = myTable.getRowCount();

**int** id = (**int**)myTable.getValueAt(rowCount-2, 0);

ArrayList<Object> Row = **new** ArrayList<Object>();

Row.add(id+2);

**for**(**int** i = 1; i < getColumnCount();i++)

{

Row.add(**null**);

}

rowCount++;

id++;

data.add(Row);

**this**.fireTableStructureChanged();

myTable.setModel(dbm);

*stmt* = *conn*.createStatement();

String query = "insert into " + *tableNames*.get(*tableNamesBox*.getSelectedIndex()) + " values("+ (id+1);

**for**(**int** i=1;i<getColumnCount();i++){

**if**(getColumnClass(i)==String.**class**)

query +=",'" + "NULL" + "'";

**else**

query +="," + "1" +"";

}

query += ")";

*stmt*.executeUpdate(query);

} **catch** (SQLException ex) {

Logger.*getLogger*(CRUDApp.**class**.getName()).log(Level.***SEVERE***, **null**, ex);

}

}

}

// Variables declaration - do not modify//GEN-BEGIN:variables

**private** javax.swing.JButton addRowBtn;

**private** javax.swing.JButton deleteRowBtn;

**private** javax.swing.JScrollPane jScrollPane1;

**private** javax.swing.JTable myTable;

**private** **static** javax.swing.JComboBox *tableNamesBox*;

// End of variables declaration//GEN-END:variables

}