**2 ЛАБОРАТОРНАЯ РАБОТА №2**

ДОСТУП К СУРБД С ИСПОЛЬЗОВАНИЕМ ПАКЕТА JDBC

**2.1 Цель работы**

Изучить основные приемы подключения к реляционным базам данных из Java приложения, научится выполнять запросы на выборку информации из базы, а также выполнять модифицирующие запросы на добавление, изменение и удаление данных.

**2.2 Краткие теоретические сведения**

Специалистами фирмы Sun разработан специализированный пакет java.sql, который позволяет Java-программе осуществлять доступ к системе управления реляционными базами данных (СУРБД, далее просто БД).

Благодаря пакету JDBC (Java Database Connectivity) можно подключаться к реляционной базе данных и взаимодействовать с ней, используя популярный язык управления базами данных – SQL.

JDBC является не протоколом, а интерфейсом и основан на спецификациях SAG CLI (SQL Access Group Call Level Interface — интерфейс уровня вызова группы доступа SQL). Существует 4 различных вида драйверов.

Тип 1 – драйверы этого типа используют технологию bridge для доступа к базе данных. Мост JDBC-ODBC, поставляемый с JDK - хороший пример этого типа драйвера. Он представляет собой шлюз к ODBC API. Использование технологии моста требует установки программного обеспечения на клиентских системах, что является неприемлемым для приложений, которые не позволяют такую установку.

Тип 2 – это драйверы ”родного” API. Это означает, что драйверы содержат Java-код, который вызывает методы на C или C++, предоставляемые разработчиками баз данных, обеспечивающие доступ к базам данных. Это решение также требует установку программного обеспечения на клиентской системе.

Тип 3 – драйверы этого типа обеспечивают клиента общим сетевым API, который транслирует специфический для базы данных доступ на серверный уровень. Драйвер JDBC клиента использует сокет чтобы обратится к промежуточному программному обеспечению на сервере, которое транслирует клиентские запросы в API требуемого драйвера. Это решение не требует установки дополнительного программного обеспечения на клиентской системе, и фактически с помощью одного драйвера можно осуществить работу с различными серверами баз данных.

Тип 4 – драйверы этого типа используют сетевой протокол, встроенный в механизм базы данных. Эти драйверы обращаются непосредственно к базе данных, используя механизм Java - сокетов. Это непосредственное решение исключительно на языке Java. Поскольку эти протоколы практически не документированы, драйверы этого типа практически всегда поставляются разработчиками баз данных.

В Java приложении используется интерфейс JDBC, как фасад для непосредственного доступа к базе данных, а реализация JDBC выполняет фактический доступ к базе данных.

JDBC Driver Manager - это основа JDBC-архитектуры. Его первичные функции очень просты - соединить Java-программу и соответствующий JDBC драйвер, а затем отстраниться. Дальнейшая работа проходит в три этапа:

1. Подключится к БД;

2. Подготовить и передать SQL запрос;

3. Чтение результатов и использование их в приложении.

**2.3 Выполнение работы**

**2.3.1 Packege ua.cn.stu.database**

**package** ua.cn.stu.database;

**import** java.sql.ResultSet;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Properties;

**import** javax.swing.table.DefaultTableModel;

**import** ua.cn.stu.model.Test;

**public** **class** DataController {

**private** DBSession ds;

**public** DataController() {

String url = "jdbc:postgresql://localhost:5432/MobileMarket";

Properties prop = **new** Properties();

prop.setProperty("user", "postgres");

prop.setProperty("password", "1108");

ds = **new** DBSession(url, prop);

}

**public** List<Test> getTestList() {

List<Test> listTest = **new** ArrayList<Test>();

**try** {

ResultSet set = ds.runQuery("SELECT \* FROM \"Test\"");

**while** (set.next()) {

**int** id = set.getInt(1);

String name = set.getString(2);

String sName = set.getString(3);

Boolean boolVariable = set.getBoolean(4);

listTest.add(**new** Test(name, id, sName, boolVariable));

}

} **catch** (Exception exp) {

exp.printStackTrace();

}

**return** listTest;

}

**public** **void** insertTest(Test obj) {

ds.runUpdate("insert into \"Test\" (id, \"Name\", \"SecondName\", \"BoolVariable\") values ("

+ String.*valueOf*(obj.getId()) + ", '" + obj.getName() + "', '" + obj.getsName().toString() + "', "

+ String.*valueOf*(obj.isBoolVariable()) + ");");

}

**public** DefaultTableModel getTableModel(List<Test> list) {

**if** (list.size() == 0)

**return** **new** DefaultTableModel();

String[] headrs = list.get(0).getHeadrs();

Object[][] arrData = **new** Object[list.size()][headrs.length];

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

arrData[i] = list.get(i).getData();

}

**return** **new** DefaultTableModel(arrData, headrs);

}

**public** **void** updateTest(Test obj) {

ds.runUpdate("update \"Test\" set \"Name\" = '" + obj.getName() + "', " + "\"SecondName\" = '"

+ obj.getsName().toString() + "', " + "\"BoolVariable\" = " + String.*valueOf*(obj.isBoolVariable())

+ " where id = " + String.*valueOf*(obj.getId()));

}

**public** **void** deleteTest(Test obj) {

ds.runUpdate("delete from \"Test\" where id = " + String.*valueOf*(obj.getId()));

}

}

**package** ua.cn.stu.database;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**import** java.util.Properties;

**public** **class** DBSession {

**private** Connection conn = **null**;

**private** Statement stmt = **null**;

**public** DBSession(String url, Properties prop) {

**try** {

conn = DriverManager.*getConnection*(url, prop);

} **catch** (SQLException e) {

System.***err***.println("Проблеми із підключенням до " + url);

e.printStackTrace();

}

}

// Оновлення бази даних (INSERT, UPDATE, CREATE TABLE і т.і.)

**public** **int** runUpdate(String sql) {

**int** count = 0;

**try** {

stmt = conn.createStatement();

count = stmt.executeUpdate(sql);

} **catch** (SQLException e) {

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

e.printStackTrace();

}

**return** count;

}

// Запит до бази (Select)

**public** ResultSet runQuery(String sql) {

ResultSet resultSet = **null**;

**try** {

stmt = conn.createStatement();

resultSet = stmt.executeQuery(sql);

} **catch** (SQLException e) {

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

e.printStackTrace();

}

**return** resultSet;

}

@Override

**protected** **void** finalize() **throws** Throwable {

**if** (conn != **null**)

conn.close();

**super**.finalize();

}

}

**2.3.2 Packege ua.cn.stu.model**

**package** ua.cn.stu.model;

**public** **interface** IDBModel {

**public** String[] getHeadrs();

**public** Object[] getData();

**public** String getSelectedRequest();

}

**package** ua.cn.stu.model;

**import** javax.xml.bind.annotation.XmlAttribute;

**public** **class** Test **implements** IDBModel {

**private** String name;

**private** **int** id;

**private** String sName;

**private** **boolean** boolVariable;

**public** Test() {

**super**();

}

@XmlAttribute(name = "Name")

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

@XmlAttribute(name = "ID")

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

@XmlAttribute(name = "SecondName")

**public** String getsName() {

**return** sName;

}

**public** **void** setsName(String sName) {

**this**.sName = sName;

}

@XmlAttribute(name = "BoolVariable")

**public** **boolean** isBoolVariable() {

**return** boolVariable;

}

**public** **void** setBoolVariable(**boolean** boolVariable) {

**this**.boolVariable = boolVariable;

}

@Override

**public** String[] getHeadrs() {

**return** **new** String[] { "id", "Name", "Second Name", "BoolVariable" };

}

@Override

**public** Object[] getData() {

**return** **new** String[] { String.*valueOf*(id), name, sName, String.*valueOf*(boolVariable) };

}

@Override

**public** String getSelectedRequest() {

**return** "SELECT \* FROM Test";

}

**public** Test(String name, **int** id, String sName, **boolean** boolVariable) {

**super**();

**this**.name = name;

**this**.id = id;

**this**.sName = sName;

**this**.boolVariable = boolVariable;

}

@Override

**public** String toString() {

**return** "Test [name=" + name + ", id=" + id + ", second name=" + sName + ", boolVariable=" + boolVariable + "]";

}

}

**2.3.3 Packege ua.cn.stu.factory**

**package** ua.cn.stu.factory;

**import** ua.cn.stu.GUI.TestDialog;

**import** ua.cn.stu.database.DataController;

**import** ua.cn.stu.model.Test;

**public** **class** DialogTestFactory {

**public** **static** TestDialog createDialog(**int** mode, Test object, DataController ctrl) {

TestDialog dlg = **new** TestDialog();

dlg.setModal(**true**);

**if** (object != **null**) {

dlg.setObject(object);

}

dlg.setMode(mode);

dlg.setControllers(ctrl);

**return** dlg;

}

}

**2.3.4 Packege ua.cn.stu.GUI**

package ua.cn.stu.GUI;

import java.awt.BorderLayout;

import java.awt.EventQueue;

import javax.swing.JFileChooser;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.border.EmptyBorder;

import javax.xml.bind.JAXBContext;

import javax.xml.bind.JAXBException;

import javax.xml.bind.Marshaller;

import ua.cn.stu.database.DataController;

import ua.cn.stu.factory.DialogTestFactory;

import ua.cn.stu.model.Test;

import ua.cn.stu.model.TestData;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import java.awt.event.ActionListener;

import java.io.File;

import java.util.List;

import java.awt.event.ActionEvent;

import javax.swing.JMenuBar;

import javax.swing.JMenu;

import javax.swing.JMenuItem;

public class MainWindow extends JFrame {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private JPanel contentPane;

private DataController dataCntr = new DataController();

private JMenuBar menuBar;

private JMenu mnNewMenu;

private JMenuItem mntmInsert;

private JMenuItem mntmUpdate;

private JMenuItem mntmDelete;

private JMenuItem mntmShowAllTable;

private JScrollPane scrollPane;

private JTable table;

private JMenu mnExport;

private JMenuItem mntmExportToXml;

/\*\*

\* Launch the application.

\*/

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

public void run() {

try {

MainWindow frame = new MainWindow();

frame.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

});

}

/\*\*

\* Create the frame.

\*/

public MainWindow() {

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setBounds(100, 100, 450, 300);

menuBar = new JMenuBar();

setJMenuBar(menuBar);

mnNewMenu = new JMenu("Action");

menuBar.add(mnNewMenu);

mntmInsert = new JMenuItem("Insert");

mntmInsert.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

TestDialog dlg = DialogTestFactory.createDialog(0, null, dataCntr);

dlg.setVisible(true);

updateTable();

}

});

mnNewMenu.add(mntmInsert);

mntmUpdate = new JMenuItem("Update");

mntmUpdate.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

Test selected = dataCntr.getTestList().get(table.getSelectedRow());

TestDialog dlg = DialogTestFactory.createDialog(1, selected, dataCntr);

dlg.setVisible(true);

updateTable();

}

});

mnNewMenu.add(mntmUpdate);

mntmDelete = new JMenuItem("Delete");

mntmDelete.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

Test selected = dataCntr.getTestList().get(table.getSelectedRow());

dataCntr.deleteTest(selected);

updateTable();

}

});

mnNewMenu.add(mntmDelete);

mntmShowAllTable = new JMenuItem("Show all table");

mntmShowAllTable.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

updateTable();

}

});

mnNewMenu.add(mntmShowAllTable);

mnExport = new JMenu("Export");

menuBar.add(mnExport);

mntmExportToXml = new JMenuItem("export to XML");

mntmExportToXml.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

saveToXML();

}

});

mnExport.add(mntmExportToXml);

contentPane = new JPanel();

contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

setContentPane(contentPane);

contentPane.setLayout(new BorderLayout(0, 0));

scrollPane = new JScrollPane();

contentPane.add(scrollPane, BorderLayout.CENTER);

table = new JTable();

scrollPane.setViewportView(table);

table.setModel(dataCntr.getTableModel(dataCntr.getTestList()));

}

protected void updateTable() {

table.setModel(dataCntr.getTableModel(dataCntr.getTestList()));

}

protected void saveToXML() {

JFileChooser fileCh = new JFileChooser("Save xml");

fileCh.showSaveDialog(this);

File f = fileCh.getSelectedFile();

List<Test> list = dataCntr.getTestList();

TestData tdata = new TestData();

for (int i = 0; i < list.size(); i++) {

tdata.add(list.get(i));

}

JAXBContext jaxbContext;

try {

jaxbContext = JAXBContext.newInstance(TestData.class);

Marshaller jaxbMarshaller = jaxbContext.createMarshaller();

jaxbMarshaller.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT, true);

jaxbMarshaller.marshal(tdata, f);

jaxbMarshaller.marshal(tdata, System.out);

} catch (JAXBException e) {

e.printStackTrace();

}

}

}

package ua.cn.stu.GUI;

import java.awt.BorderLayout;

import java.awt.FlowLayout;

import javax.swing.JButton;

import javax.swing.JDialog;

import javax.swing.JPanel;

import javax.swing.border.EmptyBorder;

import ua.cn.stu.database.DataController;

import ua.cn.stu.model.Test;

import javax.swing.JLabel;

import javax.swing.JTextField;

import javax.swing.JCheckBox;

import javax.swing.BoxLayout;

import javax.swing.SwingConstants;

import java.awt.event.ActionListener;

import java.awt.event.ActionEvent;

public class TestDialog extends JDialog {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private final JPanel contentPanel = new JPanel();

private JTextField textField;

private JTextField textField\_1;

private JTextField textField\_2;

private int mode = -1;

private DataController ctrl;

private JCheckBox chckbxTrue;

/\*\*

\* Launch the application.

\*/

public static void main(String[] args) {

try {

TestDialog dialog = new TestDialog();

dialog.setDefaultCloseOperation(JDialog.DISPOSE\_ON\_CLOSE);

dialog.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* Create the dialog.

\*/

public TestDialog() {

setResizable(false);

setBounds(100, 100, 294, 213);

getContentPane().setLayout(new BorderLayout());

contentPanel.setBorder(new EmptyBorder(5, 5, 5, 5));

getContentPane().add(contentPanel, BorderLayout.CENTER);

JLabel lblNewId = new JLabel("ID");

lblNewId.setHorizontalAlignment(SwingConstants.CENTER);

JLabel lblNewLabel = new JLabel("NAME");

JLabel lblNewLabel\_1 = new JLabel("SECOND NAME");

contentPanel.setLayout(new BoxLayout(contentPanel, BoxLayout.Y\_AXIS));

contentPanel.add(lblNewId);

textField = new JTextField();

textField.setColumns(10);

contentPanel.add(textField);

contentPanel.add(lblNewLabel);

textField\_1 = new JTextField();

textField\_1.setColumns(10);

contentPanel.add(textField\_1);

contentPanel.add(lblNewLabel\_1);

textField\_2 = new JTextField();

textField\_2.setColumns(10);

contentPanel.add(textField\_2);

chckbxTrue = new JCheckBox("booleanvariable");

contentPanel.add(chckbxTrue);

{

JPanel buttonPane = new JPanel();

buttonPane.setLayout(new FlowLayout(FlowLayout.RIGHT));

getContentPane().add(buttonPane, BorderLayout.SOUTH);

{

JButton okButton = new JButton("OK");

okButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

submit();

setVisible(false);

}

});

okButton.setActionCommand("OK");

buttonPane.add(okButton);

getRootPane().setDefaultButton(okButton);

}

{

JButton cancelButton = new JButton("Cancel");

cancelButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

setVisible(false);

}

});

cancelButton.setActionCommand("Cancel");

buttonPane.add(cancelButton);

}

}

}

public void setMode (int mode) {

this.mode = mode;

}

public void setObject(Test test) {

this.textField.setText(String.valueOf(test.getId()));

this.textField\_1.setText(test.getName());

this.textField\_2.setText(test.getsName());

this.chckbxTrue.setSelected(test.isBoolVariable());

}

public void setControllers(DataController ctrl) {

this.ctrl = ctrl;

}

public Test getObject() {

int id = Integer.parseInt(textField.getText());

String name = textField\_1.getText();

String sName = textField\_2.getText();

boolean boolVariable = this.chckbxTrue.isSelected();

return new Test(name, id, sName, boolVariable);

}

protected void submit() {

if(mode == 0) {

this.ctrl.insertTest(getObject());

} else if(mode == 1) {

this.ctrl.updateTest(getObject());

}

}

public JCheckBox getChckbxTrue() {

return chckbxTrue;

}

}

**2.4 Выводы**

В ходе выполнения данной лабораторной работы были изучены основные способы подключения и работы с реляционными базами данных. Были рассмотрены все возможные способы подключения к базе данных, способы выполнения запросов. Также были рассмотрены различные варианты обновления данных. Платформа Java предоставляет широкие, и в то же время удобные инструменты для разработки приложений, взаимодействующих с базами данных.