

# «Дежурный» редактор SQL-запросов на Java



**Алексей Вторников**

**Замечательно, когда все программное обеспечение унифицировано и находится в пределах досягаемости. Но часто об этом остается только мечтать, особенно когда приходится иметь дело с унаследованными системами, написанными разными разработчиками, в разное время и к тому же «разбросанными» по территории с площадью средней европейской страны.**

**З**адача, решение которой описано в настоящей статье, возникла на практике: необходимо было сопровождать базы данных, локально работавшие на нескольких десятках компьютеров. Задача была нестандартной: многие из компьютеров не имели выхода в Интернет, и удаленное управление ими исключалось. Кроме того, часть компьютеров работала под управлением Windows, часть – под управлением Linux. Наконец, ситуация осложнялась еще и тем,

что БД на местах отличались: основную часть составляла MySQL различных версий, несколько меньше встречалась PostgreSQL и было даже несколько экземпляров Derby.

Вот этот «зоопарк» и нужно было сопровождать. Разумеется, для каждой из перечисленных БД имелись мощные и развитые средства администрирования, однако это создавало неоправданно высокие нагрузки на сотрудников группы сопровождения, которым приходилось переключаться с одного инструмента администрирования на другие и помнить массу не относящихся к собственно сопровождению деталей. Решение оказалось довольно простым и элегантным: был разработан простой редактор SQL-запросов, с помощью которого сотрудники группы сопровождения могли оперативно просматривать информацию в локальной БД и при необходимости исправлять ее.

Конечно, возможности такого редактора значительно скромнее воз-

возможностей каждого из специализированных инструментов. С другой стороны, повседневные задачи, решаемые группой сопровождения были весьма простыми, и поэтому функциональность редактора SQL-запросов оказалась вполне достаточной для решения большинства проблем. Читатели, кому такой подход показался заслуживающим внимания, могут, разумеется, самостоятельно доработать его под свои потребности.

Редактор реализован на языке программирования Java версий 1.5 и старше [1]. Рассмотрим теперь некоторые особенности исходного кода.

Исходный код редактора SQL-запросов состоит из двух небольших классов: первый (SqlEditor) отвечает за формирование простого пользовательского интерфейса, а второй (DataTableModel) – за модель таблицы.

Приведу (с некоторыми сокращениями) исходный текст класса SqlEditor:

```
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;

public class SqlEditor extends JFrame {
    // Объект-соединение
    private Connection conn = null;
    // Модель таблицы и сама таблица результатов SQL-запроса
    private DataTableModel dtm = new DataTableModel ();
    private JTable table = new JTable (dtm);
    // Элементы интерфейса
    private JTextArea sqlArea = new JTextArea (8, 50);
    private JButton runButton = new JButton ("Выполнить");

    public SqlEditor (Connection conn) {
        super ("Редактор SQL-запросов");
        this.conn = conn;
        table.setAutoResizeMode (JTable.AUTO_RESIZE_OFF);
        table.setSelectionMode (
            JListSelectionModel.SINGLE_SELECTION);
        // Сформировать и добавить панель
        // пользовательского интерфейса
        getContentPane ().add (createGUI ());

        // Обработчик событий мыши
        runButton.addActionListener (new ActionListener () {
            public void actionPerformed (ActionEvent e) {
                String sql = sqlArea.getText ().trim ();
                if (sql.length () != 0) {
                    // Передать SQL-запрос (как строку)
                    dtm.setQuery (sql);
                    // Обработать SQL-запрос
                    dtm.setData (conn);
                }
            }
        });
        ...
    }

    // Сформировать интерфейс пользователя
    private JPanel createGUI () {
        JPanel topPane = new JPanel (new BorderLayout ());
        botPane = new JPanel (new FlowLayout (
            FlowLayout.RIGHT));
        // Области SQL-запросов (sqlPane) и таблицы
        // результатов (tblPane)
        JScrollPane sqlPane = new JScrollPane (sqlArea);
        tblPane = new JScrollPane (table);
        // Разделительная (горизонтальная) панель
        JSplitPane splPane = new JSplitPane (
            JSplitPane.VERTICAL_SPLIT, sqlPane, tblPane);

        setSize (new Dimension (500, 400));
        ...
        tblPane.setBorder (
            (BorderFactory.createEtchedBorder ());
        // Каждой области - половину пространства
        splPane.setResizeWeight (0.50);
    }
}
```

Account_Payer	Account_Receiver	Sum	Date_Value	Numb_Document
60308810600000000065	20202810700000000001	140.7900	27.05.2008	015
70606810100002640602	60308810600000000065	140.7900	27.05.2008	051
40702810300000000499	40802810500000000306	367.1800	27.05.2008	48
40911810000000000001	40817810716000000211	2675.0000	27.05.2008	3323
40911810100000000011	40702810600000000041	3550.0000	27.05.2008	3322
40702810500000000024	407028103000050002050	4500.0000	27.05.2008	137
40911810400000000012	407038106000000004243	5050.0000	27.05.2008	3320
40702810500000000325	40702810209580000090	6500.0000	26.05.2008	108
40909810800000000046	20202810700000000001	7000.0000	27.05.2008	808
20202810700000000001	40817810016000000898	7432.0000	27.05.2008	019

Образец SQL-запроса и результат его выполнения

```
botPane.add (runButton);
topPane.add (splPane, BorderLayout.CENTER);
topPane.add (botPane, BorderLayout.SOUTH);
return (topPane);
}
```

Графический интерфейс представляет собой обычное окно (JFrame), внутреннее пространство которого разделено по горизонтали (JSplitPane) на две области (JScrollPane): в верхней вводятся SQL-запросы; в нижней выводятся результаты их выполнения (см. **рисунок**).

Обратите внимание, что в конструктор класса SqlEditor передается параметр Connection, т.е. соединение с БД; иными словами – перед вызовом редактора SQL-запросов соединение с БД должно быть установлено. Этим и достигается универсальность решения: какова бы ни была БД, редактор одинаково успешно будет работать с любой (кроме БД, перечисленных в начале статьи, редактор SQL-запросов был проверен в работе с Oracle, MS-SQL и Pervasive). Задача установления соединения с БД решается тривиально (см., например, [2]): достаточно загрузить так называемый jdbc-драйвер для БД.

Основная работа по отображению результатов SQL-запросов возложена на класс модели таблицы DataTableModel:

```
import java.sql.*;
import java.util.*;

public class DataTableModel extends AbstractTableModel {
    // Коллекции имен столбцов, их типов и данных
    private ArrayList <String> colNames = new ArrayList <String> ();
    private ArrayList <Class> colTypes = new ArrayList <Class> ();
    private ArrayList <ArrayList <Object>> data = new ArrayList <ArrayList <Object>> ();
    private String sql = "";

    // SQL-запрос для выполнения
    public void setQuery (String sql) {this.sql = sql;}
    // Методы доступа к элементам модели таблицы
    public int getRowCount () {synchronized (data) {
        return data.size ();}}
    public int getColumnCount () {return colNames.size ();}
    public Class getColumnClass (int column) {
        return (Class)colTypes.get (column);
    }
}
```

```

public String getColumnName (int column) {
    return (String)colNames.get (column);
}
public Object getValueAt (int row, int column) {
    synchronized (data) {
        return ((ArrayList)data.get (row)).get (column);
    }
}
// Запись данных (value) в строку (row) и столбец (column)
public void setValueAt (Object value, int row, int column) {
    synchronized (data) {
        data.get (row).set (column, value);
    }
}

// Подготовка данных для отображения
public void setData (Connection conn) {
    // Подготовка коллекций (см.выше)
    colNames.clear ();
    colTypes.clear ();
    data.clear ();
    try {
        // Подготовка и исполнение SQL-запроса
        Statement st = conn.createStatement ();
        if (st.execute (sql)) {
            // Результирующий набор исполнения SQL-запроса
            ResultSet rs = st.executeQuery (sql);
            // Метаданные результирующего набора
            ResultSetMetaData rsmd = rs.getMetaData ();
            // Количество столбцов
            int colCount = rsmd.getColumnCount ();
            for (int i = 0; i < colCount; i++) {
                // Имена столбцов
                colNames.add (rsmd.getColumnLabel (i + 1));
                // Типы столбцов
                Class type = Class.forName (rsmd.getColumnClassName (i + 1));
                colTypes.add (type);
            }
            fireTableStructureChanged ();
            // Перебор результирующего набора
            while (rs.next ()) {
                ArrayList <Object> row = new ArrayList <Object> ();
                for (int i = 0; i < colCount; i++) {
                    // Определить тип столбца
                    if (colTypes.get (i) == Boolean.class)
                        {row.add (rs.getBoolean (i + 1)); continue;}
                    if (colTypes.get (i) == Byte.class)
                        {row.add (rs.getByte (i + 1)); continue;}
                    if (colTypes.get (i) == java.util.Date.class)
                        {row.add (rs.getDate (i + 1)); continue;}
                    if (colTypes.get (i) == Double.class)
                        {row.add (rs.getDouble (i + 1)); continue;}
                    if (colTypes.get (i) == Float.class)
                        {row.add (rs.getFloat (i + 1)); continue;}
                    if (colTypes.get (i) == Integer.class)
                        {row.add (rs.getInt (i + 1)); continue;}
                    if (colTypes.get (i) == Long.class)
                        {row.add (rs.getLong (i + 1)); continue;}
                    if (colTypes.get (i) == Short.class)
                        {row.add (rs.getShort (i + 1)); continue;}
                    if (colTypes.get (i) == String.class)
                        {row.add (rs.getString (i + 1)); continue;}
                    row.add (rs.getObject (i + 1));
                }
                synchronized (data) {
                    data.add (row);
                    fireTableRowsInserted (data.size () - 1, data.size () - 1);
                }
            }
            // Результирующий набор обработан; закрыть его
            if (rs != null) {
                try {rs.close ();} catch (SQLException sqle) {}
                finally {rs = null;}
            }
        }
        st.close ();
    }
    // Перехват и обработка исключений
    catch (ClassNotFoundException cnfe) {
        System.out.println ("Class.forName loading error");
    }
    catch (SQLException sqle) {
        System.out.println ("SQLException: " + sqle.getMessage ());
        System.out.println ("SQLState : " + sqle.getSQLState ());
        System.out.println ("VendorError : " + sqle.getErrorCode ());
    }
}
}
}

```

Небольшой комментарий к исходному коду класса DataTableModel для программистов, впервые столкнувшихся с программированием таблиц в Java.


Класс модели результирующей таблицы DataTableModel расширяет (extends) базовый класс AbstractTableModel, в котором определены методы доступа к элементам модели таблицы, подробнее смотрите в руководстве [3].

Далее, в классе DataTableModel широко используются шаблоны («generics») – своего рода аналоги «template» в C++ [4]. Шаблоны позволяют программисту точно специфицировать типы данных коллекций [5] и методы их обработки. Коллекции в исходном коде представлены классом массива-списка ArrayList.

Наконец, обратите внимание на то, как автоматически определяются типы данных столбцов (column) при переборе результирующего набора. После того как тип данных столбца определен, данные из столбца добавляются в строку (row), которая добавляется в модель данных (data).

Вот, собственно, и все. Я надеюсь, что статья послужит полезным примером и подвигнет кого-либо на разработку инструментария с более широкими возможностями (например, подсветку синтаксиса SQL-инструкций, возможность загружать SQL-скрипты из файлов, экспорт/импорт данных и т. п.).

В заключение я, пользуясь случаем, хочу выразить искреннюю благодарность своему коллеге И.Олейникову за ценные советы и большую работу по тестированию редактора SQL-запросов.

Удачи! 

1. Java-технологии Sun Microsystems – <http://java.sun.com>.
2. JDBC и доступ к БД – <http://java.sun.com/docs/books/tutorial/jdbc/index.html>.
3. Таблицы в Java – <http://java.sun.com/docs/books/tutorial/uiswing/components/table.html>.
4. Шаблоны в Java – <http://java.sun.com/developer/technicalArticles/J2SE/generics/index.html>.
5. Коллекции в Java – <http://java.sun.com/docs/books/tutorial/collections/index.html>.