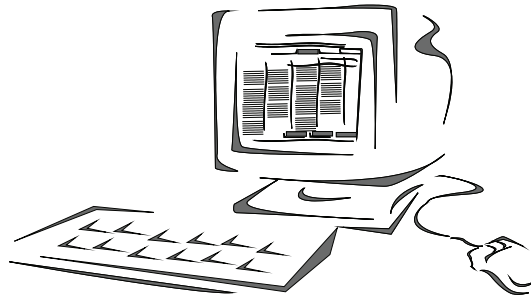


**Московский авиационный институт  
(государственный ТЕХнический университет)**

**Факультет прикладной математики**

**Кафедра вычислительной математики и программирования**

**Лабораторные работы по курсу «Базы данных»**



Студент: И. К. Никитин  
Преподаватели: А. М. Марасанов  
Е. С. Гаврилов

**Москва, 2010**

# Содержание

<b>Введение</b>	<b>4</b>
<b>JDBC</b>	<b>5</b>
<b>1 Теоретическая часть</b>	<b>5</b>
1.1 JDBC драйвера	6
1.2 Интерфейсы и классы	7
1.3 Подключение к БД	9
1.3.1 Подключение к MS SQL Server	9
1.3.2 Подключение к Oracle	10
1.3.3 Реализация	11
1.4 Транзакции	12
1.5 Бизнес логика	13
1.5.1 Логика в базе данных	14
1.5.2 Логика в приложении	16
1.6 Дополнительные функции	17
<b>2 Решение</b>	<b>18</b>
2.1 Концепция	18
2.2 Средства	18
2.3 Исходный код	18
2.4 Трудности	21
2.4.1 Среда выполнения	21
2.4.2 Среда разработки	21
<b>3 Пример</b>	<b>22</b>
3.1 Окно входа	22
3.2 Рабочее окно	22
<b>4 Выводы</b>	<b>24</b>

<b>ADO.NET</b>	<b>25</b>
<b>5 Теоретическая часть</b>	<b>25</b>
5.1 Типы объектов . . . . .	25
5.2 Присоединенные объекты . . . . .	26
5.2.1 Используемые паттерны . . . . .	27
5.3 Отсоединенные объекты . . . . .	28
5.3.1 Используемые паттерны . . . . .	29
<b>6 Решение</b>	<b>30</b>
6.1 Концепция . . . . .	30
6.1.1 Было в проекте . . . . .	31
6.2 Средства . . . . .	32
6.3 Окна . . . . .	32
6.4 Исходный код . . . . .	33
6.4.1 C# . . . . .	33
6.4.2 PL/SQL . . . . .	39
<b>7 Пример</b>	<b>40</b>
7.1 Окно входа . . . . .	40
7.2 О программе . . . . .	40
7.3 Рабочее окно . . . . .	41
<b>8 Выводы</b>	<b>45</b>

<b>ASP.NET MVC и Nhibernate</b>	<b>46</b>
<b>9 Теоретическая часть</b>	<b>46</b>
9.1 ORM . . . . .	46
9.1.1 Возможности ORM . . . . .	46
9.1.2 Преимущества ORM . . . . .	47
9.1.3 Недостатки ORM . . . . .	47
9.2 MVC . . . . .	48
9.2.1 Классический MVC . . . . .	48
9.2.2 MVC в веб-приложении . . . . .	49
9.3 Архитектура слоев приложения . . . . .	50
<b>10 Решение</b>	<b>51</b>
10.1 Концепция . . . . .	51
10.1.1 Оптимистическая блокировка . . . . .	51
10.2 Средства . . . . .	52
10.3 Исходный код . . . . .	53
10.3.1 Конфигурация Nhibernate . . . . .	53
10.3.2 Описание модели . . . . .	54
10.3.3 Mapping модели . . . . .	54
10.3.4 Описание DAO . . . . .	55
10.3.5 Описание класса для взаимодействия с базой данных. . . . .	57
10.3.6 SQL . . . . .	57
10.4 Возникшие трудности . . . . .	57
<b>11 Пример</b>	<b>58</b>
11.1 Список пользователей . . . . .	58
11.2 Оптимистическая блокировка . . . . .	59
<b>12 Выводы</b>	<b>61</b>

## Введение

**Задача:** Написать программу организующую взаимодействие с базой данных по одной из предложенных моделей:

- «Толстый» клиент.
- «Тонкий» клиент (с использованием хранимых процедур).
- Веб-ориентированное приложение с «легким» клиентом.

В итоге нужно реализовать три приложения, каждое из которых использует тот или иной подход. Клиенты вместе должны реализовывать все или большую часть Use-Case'ов, спроектированных в прошлом семестре. Взаимодействие должно производиться с базой данных, созданной в прошлом семестре. Очень рекомендуется в ходе выполнения лабораторных работ рассмотреть различные технологии доступа к данным, используя различные среды программирования.

- 1) В качестве первой лабораторной работы мы решили реализовать «толстого» клиента с использованием технологии JDBC.
- 2) В качестве второй лабораторной работы мы решили реализовать «тонкого» клиента с использованием технологии ADO.NET. В рамках того же клиента мы рассмотрели технологию «отсоединенной модели».
- 3) В качестве веб-ориентированного приложения было реализовано ASP.NET MVC приложение на C#. Для взаимодействия с базой данных мы использовали ORM-технологию NHibernate.

---

Текущая версия документа собрана специально для [twirpx.com](http://twirpx.com). Любые замечания и поправки желательно оставлять именно на указанном сайте. Если это не возможно, пишите на наш адрес электронной почты [w@w-495.ru](mailto:w@w-495.ru).

# JDBC

## 1. Теоретическая часть

**JDBC** — это универсальное средство связи с Базой данных. По существу — всего лишь Java API для доступа к данным. Диспетчер драйверов позволяет подключать к БД драйвера независимых производителей. Такой подход позволяет производителям СУБД создавать собственные драйвера. Клиентское приложение использует интерфейсы JDBC API.



JDBC API состоит из интерфейсов и классов, используемых для доступа к данным, независимо от их источника.

Основные интерфейсы JDBC API :

- Connection
- Statement
- ResultSet

Конкретный драйвер БД реализует все JDBC интерфейсы и набор их функций. Например: реализации драйверов от поставщиков

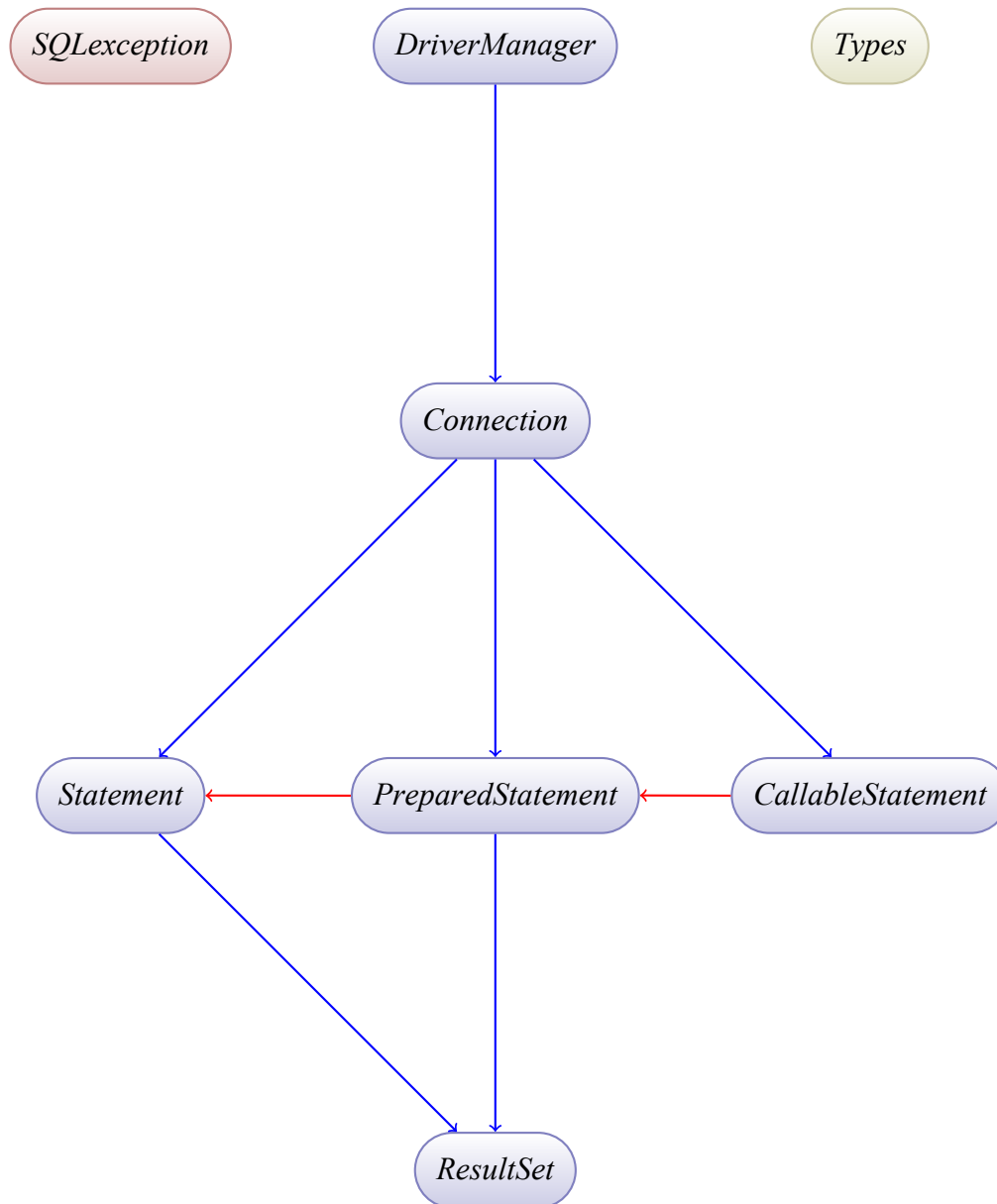
- OracleConnection
- OracleStatement
- OracleResultSet

Для одной и тех же СУБД может быть несколько драйверов.

### 1.1. Типы JDBC драйверов

- Мост JDBC-ODBC — устарело
  - + Плохая производительность
  - + Плохая переносимость
  - + Есть старая база данных — единственный вариант
- native — **используется часто**
  - + Быстрая реализация
  - + Но надо ставить базу на машину клиента
- Pure Java (универсальный)
  - + Плохая производительность
  - + Надо ставить базу на машину клиента
  - + Хорошая переносимость
- Pure Java (проприетарный) — **используется часто**
  - + Плохая производительность
  - + Надо ставить базу на машину клиента
  - + Хорошая переносимость

Большинство современных драйверов 4 типа написаны целиком на Java, не зависят от платформы и не требуют установки дополнительных библиотек (например клиента Oracle)

**1.2. Основные интерфейсы и классы JDBC**



- **DriverManager**
  - + getConnection : *Connection*
- **Connection**
  - + createStatement : *Statement*
  - + preparedStatement : *PreparedStatement*
  - + preparedCall : *CallableStatement*
  - + commit : *void*
  - + rollback : *void*
  - + close : *void*
- **Statement**
  - + executeQuery : *ResultSet*
  - + executeUpdate : *int*
  - + close : *void*
- **PreparedStatement**
  - + executeQuery : *ResultSet*
  - + executeUpdate : *int*
  - + setLong : *void*
  - + setNull : *void*
  - + setString : *void*
  - + setDate : *void*
- **CallableStatement**
  - + registerOutParametr : *void*
  - + getNull : *Null*
  - + getString : *String*
  - + getLong : *Long*
  - + getDate : *java.sql.Date*
- **ResultSet**
  - + next : *boolean*
  - + close : *void*
  - + getString : *String*
  - + getLong : *Long*
  - + getDate : *java.sql.Date*

### 1.3. Подключение к БД

Использование источника данных (конкретного и драйвера можно указывать в настройках приложения, чтобы исходный код не зависел от типа, имени и расположения базы данных. Независимость от СУБД кончается там, где используется специфический для СУБД SQL-запрос, или нестандартная функция JDBC.

Параметры соединения:

- Класс JDBC драйвера для СУБД:  
`com.mysql.jdbc.Driver`
- URL – содержит протокол, имя сервера, порт и имя экземпляра БД:  
`jdbc: драйвер: //сервер: порт; dbName=экз. БД`
- Имя:  
`root`
- Пароль:  
`123`

#### 1.3.1. Подключение к MS SQL Server

- Класс драйвера:  
`com.mysql.jdbc.Driver`
- URL:  
`jdbc: mysql: //localhost: 3306/sample?characterEncoding=UTF- 8`
- Имя:  
`root`
- Пароль:  
`123`

---

```
666
667 /// Загрузка драйвера
668 Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
669 /// Соединение с базой данных
670 Connection connection = DriverManager.getConnection(
671 "jdbc:sqlserver://localhost:1433;dbName=o01";, "sa", "123");
```

---

### 1.3.2. Подключение к Oracle

- Класс драйвера:  
`oracle.jdbc.OracleDriver`
- URL:  
`jdbc:oracle:thin:@localhost:1521:stud`
- Имя:  
`o1`
- Пароль:  
`o1`

---

```
666 / Загрузка драйвера
667 Class.forName("oracle.jdbc.OracleDriver");
668 // Соединение с базой данных
669 Connection connection = DriverManager.getConnection(
670 "jdbc:oracle:thin:@localhost:1521:orcl", "o01", "o01");
```

---

### 1.3.3. Реализация

Рассмотрим конкретную реализацию.

---

```
1 package org.mai806.jdbcsample;
2
3 import java.sql.*;
4
5 public class QuerySample {
6     public static void main(String[] args) throws Exception {
7         Class.forName("oracle.jdbc.OracleDriver");
8         Connection connection = DriverManager.getConnection(
9             "jdbc:oracle:thin:@localhost:1521:orcl",
10             //localhost — server, spm — oracl's SID
11             "o01", "o01"); //name, pass
12         PreparedStatement stmt =
13             connection.prepareStatement(
14                 "select ID, NAME from PERSON where NAME like ?"
15             );
16         stmt.setString(1, "%m%");
17         ResultSet rs = stmt.executeQuery();
18         try {
19             while(rs.next()) {
20                 // Example:
21                 System.out.println("ID: " +
22                     rs.getLong(1) + "; NAME=" +
23                     rs.getString("NAME"));
24             }
25         }
26         finally {
27             rs.close();
28             stmt.close();
29             connection.close();
30         }
31     }
32 }
```

---

Стоит, заметить:

- Весьма важно, экранировать SQL. Экранирование вещь специфичная для каждого отдельного движка
- Надо закрывать курсоры. Курсоры — ресурс компьютера. Их конечное количество. Если возникают исключения надо использовать try- catch- finally

#### 1.4. Использование транзакций

java.sql.Connection:

- `getAutoCommit()/setAutoCommit(boolean)`
- `commit()`
- `rollback()`
- `setTransactionIsolation()`

Рассмотрим:

- `autoCommit=true` – Транзакция начинается и заканчивается с каждой операцией с базой данных
- `autoCommit=false` – Ручное управление транзакциями

Уровни изоляции (степень изолированности транзакций<sup>1</sup>):

- TRANSACTION\_READ\_UNCOMMITTED
- TRANSACTION\_READ\_COMMITTED
- TRANSACTION\_READ\_REPEATABLE\_READ
- TRANSACTION\_READ\_SERIALIZABLE

Заметим что, один оператор — одна транзакция

Примеры можно увидеть ниже.

---

<sup>1</sup>ACID

### 1.5. Размещение бизнес логики

Есть 2 варианта размещения бизнес-логики:

- В базе данных

+:

- + Производительность, разгружаем клиента
- + Безопасность

-:

- + Если логика разрослась — тяжело писать ее на PL/SQL, он процедурный
- + Мало сервисов для работы со сторонними приложениями

- В приложении

- + Обращаемся напрямую к таблицам.
- + Хранимых процедур нет
- + Используем Фреймворки для облегчения доступа

## 1.5.1. Логика в базе данных

---

```
1 package org.mai806.jdbcsample;
2
3 import java.sql.*;
4
5 public class StoredProcedureSample {
6     private static Connection connection = null;
7     public static void main(String[] args) throws Exception {
8         /* Загрузка драйвера */
9         Class.forName("oracle.jdbc.OracleDriver");
10        /* Соединение с базой данных */
11        connection = DriverManager.getConnection(
12            "jdbc:oracle:thin:@localhost:1521:orcl",
13            /* localhost — сервер СУБД, spm — SID базы оракла */
14            "o01", "o01");
15        /* пользователь, пароль */
16        transferAmount(1,2,200.0);
17        connection.close();
18    }
19
20    /**
21     * Wire money payer's personal account from on recipient's account
22     * Переводит указанную сумму с одного счета на другой
23     * @param from payer's personal account
24     *      четС плательщика
25     * @param to recipient's personal account
26     *      четС получателя
27     * @param amount the money
28     */
29
30    public static void transferAmount(long from, long to, double amount)
31        throws Exception {
32        /* Создание Statement */
33        CallableStatement stmt
34            = connection.prepareCall("call TransferAmount(?, ?, ?)");
35        /*
36         * подготовим вызов хранимой процедуры TransferAmount
37         */
38        /* Установка параметров */
39        stmt.setLong(1, from);
40        stmt.setLong(2, to);
41        stmt.setDouble(3, amount);
42        /* Выполнение процедуры */
43        stmt.execute();
44    }
45 }
```

---

Приведем теперь текст хранимой процедуры:

---

```
1 create table PERSON (id integer, name varchar2(250));
2 create table ACCOUNT (id integer, person_id integer, amount number(17,3));
3
4 insert into PERSON values(1, 'Smith');
5 insert into PERSON values(2, 'Brown');
6
7 insert into ACCOUNT values(1, 1, 50.0);
8 insert into ACCOUNT values(2, 2, 100.0);
9
10 commit;
11
12 create or replace procedure TransferAmount(nFromAccount integer, nToAccount integer, fAmount number)
13 is
14 begin
15     update ACCOUNT set AMOUNT=AMOUNT-fAmount where ID=nFromAccount;
16     update ACCOUNT set AMOUNT=AMOUNT+fAmount where ID=nToAccount;
17     for cur in (select * from ACCOUNT where AMOUNT<0 and ID=nFromAccount) loop
18         RAISE_APPLICATION_ERROR(-20103, 'No enough money on account #' || nFromAccount || ': ' || (cur.amount+fAmount));
19     end loop;
20 end;
21 /
```

---



## 1.5.2. Логика в приложении

---

```
1 package org.mai806.jdbcsample;
2
3 import java.sql.*;
4
5 public class TransactionalSample {
6     private static Connection connection = null;
7     public static void main(String[] args) throws Exception {
8         Class.forName("oracle.jdbc.OracleDriver");
9         connection = DriverManager.getConnection(
10             "jdbc:oracle:thin:@localhost:1521:orcl",
11             "o01", "o01");
12         // transactions — OFF
13         connection.setAutoCommit(false);
14
15         try {
16             transferAmount(2, 1, 200.0);
17         } finally {
18             connection.close();
19         }
20     }
21
22     public static void transferAmount(long from, long to,
23         double amount) throws Exception {
24         PreparedStatement stmt = null;
25         Statement query = null;
26         try {
27             stmt = connection.prepareStatement("update ACCOUNT set AMOUNT=AMOUNT+? where ID=?");
28             // Get money
29             stmt.setDouble(1, -amount);
30             stmt.setLong(2, from);
31             stmt.execute();
32
33             // Put money
34             stmt.setDouble(1, amount);
35             stmt.setLong(2, to);
36             stmt.execute();
37
38             // Post-check: is payer's account negative
39             query = connection.createStatement();
40             ResultSet rs = query.executeQuery(
41                 "select AMOUNT from ACCOUNT where ID="+from+" and AMOUNT<0");
42             if (rs.next()) {
43                 throw new Exception("На счете №"+from+" недостаточно средств [ "+(amount+rs.getDouble(1))+"] для снятия
44                 суммы [ "+amount+" ]");
45             }
46             connection.commit();
47             System.out.println("Перечисление средств успешно выполнено");
48         } catch (Exception e) {
49             e.printStackTrace();
50             connection.rollback();
51         } finally {
52             if (stmt!=null)
53                 stmt.close();
54             if (query!=null)
55                 query.close();
56         }
57     }
58 }
```

---

**Замыкание** — переменная с куском кода, использующая контекст, в котором она была создана. Замыкание — это особый вид функции. Она определена в теле другой функции и создается каждый раз во время её выполнения. В записи это выглядит как функция, находящаяся целиком в теле другой функции. При этом вложенная внутренняя функция содержит ссылки на локальные переменные внешней функции. Каждый раз при выполнении внешней функции происходит создание нового экземпляра внутренней функции, с новыми ссылками на переменные внешней функции.

## 1.6. Дополнительные функции

- Поддержка BLOB
- Batch
- Savepoint
- Scrollable/Updateable ResultSet
- RowSet
- Распределенные транзакции (XA)

## 2. Решение

### 2.1. Концепция

Клиент реализует Use-Case «Администратор» для социальной сети журналистов. Use-Case'ы и таблицы базы данных были спроектированы в прошлом семестре. Напомним:

Администратор может только изменять права пользователей и назначать модераторов.

Здесь мы немного расширили права администратора возможностью добавлять новых пользователей. В любом случае, работа осуществляется с одной таблицей.

### 2.2. Краткое описание используемых средств и технологий

- Язык программирования на стороне клиента: **Java**.
  - + Библиотека классов: **JDK 1.6.0\_18**.
  - + Графическая библиотека: **Swing**.
  - + Интегрированная среда разработки: **IntelliJ IDEA**.
- Язык программирования на стороне сервера: **PL/SQL**.
  - + База данных: **Oracle**. База данных была создана и заполнена до начала разработки приложения.
  - + Драйвер базы данных: **oracle.jdbc.OracleDriver**.
- Метод размещения бизнес-логики: в нашем приложении вся логика вынесена на клиента. Так удобнее в силу его идеологии (работа с одной таблицей).

### 2.3. Исходный код

Приведем в отчете только значимую с точки зрения базы часть кода.

```
1  /**
2   * Created by IntelliJ IDEA.
3   * User: w495
4   * Date: 20.03.2010
5   * Time: 14:51:27
6   */
7  import java.sql.*;
8  import java.util.Vector;
9  import oracle.jdbc.OracleDriver;
10 import javax.swing.table.DefaultTableModel;
11 import javax.swing.table.TableModel;
12
```

```

13 public class Baser {
14     private static String __TAB = "personpage";
15     private String _login;
16     private String _password;
17     private String _baseString;
18     private Connection _connection;
19     private DataRowVector _data;
20     private int _maxId;
21     public Baser(){
22         this._login = "w495- wr ong";
23         this._password = "w495- wr ong";
24         this._baseString = "jdbc: oracle: thin: @localhost: 1521: orcl";
25         this._connection = null;
26         this._data = new DataRowVector();
27         this._maxId = 0;
28     }
29     public String getLogin(){
30         return this._login;
31     }
32     public String getPassword(){
33         return this._password;
34     }
35     public Connection getConnection(){
36         return this._connection;
37     }
38     public void setLogin(String a_login){
39         if(!this._login.equals(a_login))
40             this._login = a_login;
41     }
42     public void setPassword(String a_password){
43         if(!this._password.equals(a_password))
44             this._password = a_password;
45     }
46     public void addRow(DataRow a_row){
47         this._data.add(a_row);
48     }
49     public void setData(DataRowVector a_vector){
50         this._data = a_vector;
51     }
52     public DataRowVector getData(){
53         return this._data;
54     }
55     public void insertItem(DataRow item){
56         /**
57          * insert new record to Personpage
58          */
59         this._data.addElement(item);
60         try{
61             PreparedStatement updateQuery;
62             int j = 0;
63             updateQuery = this._connection.prepareStatement("insert into  "+__TAB+"(PERSON_NUMBER, NICKNAME, NAME_
, Surname, ISMODERATOR, ISGUST)" +
64                 "values(?, ?, ?, ?, ?, ?)");
65             updateQuery.setInt(++j, this.getMaxId());
66             updateQuery.setString(++j, item.getNickName());
67             updateQuery.setString(++j, item.getName());
68             updateQuery.setString(++j, item.getSurName());
69             updateQuery.setBoolean(++j, item.getIsModerator());
70             updateQuery.setBoolean(++j, item.getIsGust());
71             updateQuery.executeQuery();
72             updateQuery.close();
73         } catch (SQLException e) {

```

```

74         e.printStackTrace();
75     }
76 }
77 public void updateBase(){
78     /**
79      * update ALL records in Personpage
80      */
81     try{
82         PreparedStatement updateQuery;
83         for(int i_378 =0; i_378 != this._data.size(); ++i_378){
84             DataRow item = new DataRow ();
85             item = this._data.elementAt(i_378);
86             int j = 0;
87             updateQuery = this._connection.prepareStatement("UPDATE " + __TAB + " SET NICKNAME = ?, "
88                 + "NAME_ = ?, "
89                 + "SURNAME = ?, "
90                 + "ISMODERATOR =?, "
91                 + "ISGUST =? "
92                 + "where PERSON_NUMBER = ?");
93             updateQuery.setString(++j, item.getNickName());
94             updateQuery.setString(++j, item.getName());
95             updateQuery.setString(++j, item.getSurName());
96             updateQuery.setBoolean(++j, item.getIsModerator());
97             updateQuery.setBoolean(++j, item.getIsGust());
98             updateQuery.setInt(++j, i_378);
99             updateQuery.executeQuery();
100            updateQuery.close();
101        }
102    } catch (SQLException e) {
103        e.printStackTrace();
104    }
105 }
106 private void setMaxId(int id){
107     if(id > this._maxId)
108         this._maxId = id;
109 }
110 private int getMaxId(){
111     ++this._maxId;
112     return this._maxId;
113 }
114 public DataRowVector getPersonpage(){
115     /**
116      * finds all records in Personpage
117      */
118     this._data.clear(); // = new Vector<DataRow>(0);
119     try{
120         PreparedStatement usersQuery = this._connection.prepareStatement("select * from " + __TAB );
121         //usersQuery.setString(1, "personpage");//.setString(1, "%m%");
122         ResultSet rs = usersQuery.executeQuery();
123         try {
124             while(rs.next()) {
125                 this.setMaxId(rs.getInt(1));
126                 DataRow _row = new DataRow();
127                 _row.setNickName(rs.getString("NICKNAME"));
128                 _row.setName(rs.getString("NAME_"));
129                 _row.setSurName(rs.getString("SURNAME"));
130                 _row.setIsModerator(rs.getBoolean("ISMODERATOR"));
131                 _row.setIsGust(rs.getBoolean("ISGUST"));
132                 this._data.add(_row);
133             }
134         }
135         finally {

```

```
136         rs.close();
137         usersQuery.close();
138     }
139     return this._data;
140 } catch (SQLException e) {
141     return null;
142 }
143 }
144 public boolean makeConnection() throws Exception {
145     if (null != this._connection){
146         return false;
147     }
148     DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
149     try{
150         this._connection = DriverManager.getConnection(this._baseString, this._login, this._password);
151     } catch (SQLException e) {
152         return false;
153     }
154     return true;
155 }
156 public void closeConnection() throws Exception {
157     this._connection.close();
158 }
159 }
```

---

## 2.4. Трудности

### 2.4.1. Среда выполнения

В ходе выполнения работы возникли некоторые трудности с видимостью необходимых библиотек средой выполнения. Эти проблемы были решены после прописывания библиотек в переменную CLASSPATH. Однако поиск решения этой проблемы занял большую часть времени выполнения работы.

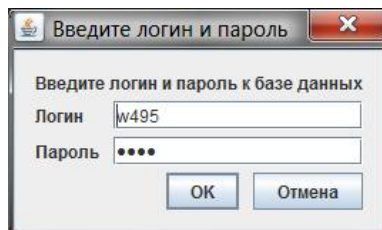
### 2.4.2. Среда разработки

Отсутствие актуальной документации для последней версии Swing и для IntelliJ IDEA также стало одним из серьезных препятствий для выполнения этой работы. Оказалось что в Swing (в отличие от Qt) не так-то просто поменять стандартное оформление виджетов. Весьма критично это было для `JTable`. В результате был описан свой класс `BaseRenderer` на основе стандартного виртуального класс `TableCellRenderer`. С помощью него мы изменили вид рабочей области таблицы `JTable`.

### 3. Пример работы программы

Приведем скриншоты.

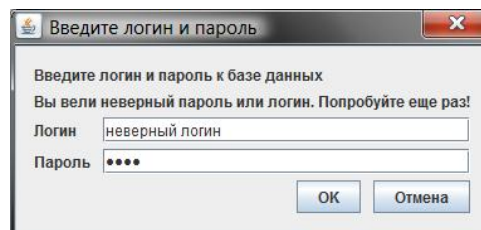
#### 3.1. Окно входа



Введите логин и пароль к базе данных

Логин

Пароль



Введите логин и пароль к базе данных

Вы ввели неверный пароль или логин. Попробуйте еще раз!

Логин

Пароль

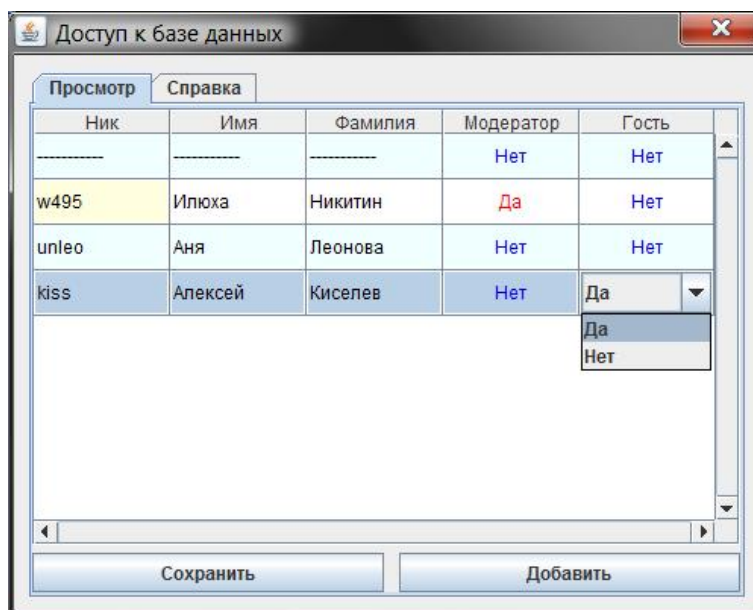
#### 3.2. Рабочее окно



Доступ к базе данных

Просмотр Справка

Ник	Имя	Фамилия	Модератор	Гость
w495	Илюха	Никитин	Да	Нет
unleo	Аня	Леонова	Нет	Нет
kiss	Алексей	Киселев	Нет	Да
-----	-----	-----	Нет	Нет



Ник	Имя	Фамилия	Модератор	Гость
-----	-----	-----	Нет	Нет
w495	Илюха	Никитин	Да	Нет
unleo	Аня	Леонова	Нет	Нет
kiss	Алексей	Киселев	Нет	Да

Сохранить      Добавить



## 4. Выводы

В процессе выполнения лабораторной работы мы познакомились с технологией доступа к данным JDBC. С одной стороны, эта технология достаточно проста в изучении и использовании для небольших приложений. С другой стороны, очевидно, что ее прямое использование для серьезных проектов неприменимо. При попытке перейти на другую базу данных нам придется использовать другой драйвер. Придется изменять текст программы, это может занять значительное время и привести к ошибкам, которые будет трудно обнаружить. Кроме того, мы попробовали написать более-менее серьезное приложение на языке Java. Сей факт тоже не может не радовать.

# ADO.NET

## 5. Теоретическая часть

Технология ActiveX Data Object (ADO.NET) — это набор служб доступа к данным, реализованных в рамках .Net Framework.

Преимуществом (как в принципе не недостатком) ADO.NET является его реализация в рамках среды .Net. Это позволяет .Net-приложениям использовать возможности этой технологии.

ADO.NET — основная модель доступа к данным для приложений, основанных на Microsoft.Net. Обеспечивает эффективное взаимодействие с распространенными типами СУРБД.

ADO.NET не ограничивается только операциями с базой данных, но и дает возможность получить доступ и управлять данными других приложений, совместимых с технологией OLE DB, например, Microsoft Excel.



### 5.1. Типы объектов

Назначение ADO.NET — помогать разработке эффективных многоуровневых приложений для работы с БД в интрасетях и Интернете, для чего она и предоставляет все необходимые средства.

Объекты делятся на два типа: присоединенные<sup>2</sup> и автономные<sup>3</sup>. Рассмотрим их подробнее.

---

<sup>2</sup>Connected.

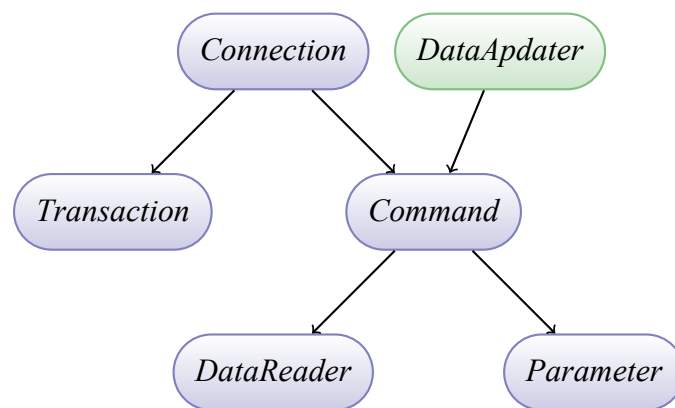
<sup>3</sup>Disconnected.

## 5.2. Присоединенные объекты

Их также называют объектами активного соединения. Для управления соединением, транзакциями, выборки данных и передачи изменений они взаимодействуют непосредственно с БД.

- Работает в режиме удержания подключения к базе.
- Обеспечивает максимальную гибкость и эффективность.
- Обеспечивает минимальный расход оперативной памяти.

Иерархически модель можно изобразить следующим образом:



- **DataAdapter.** Это своеобразный шлюз между автономными и подключенными аспектами ADO.NET. Он устанавливает подключение, и если подключение уже установлено, содержит достаточно информации, чтобы воспринимать данные автономных объектов и взаимодействовать с базой данных. (DataAdapter - SqlDataAdapter, OracleDataAdapter)
- **Connection.** Применяется для создания канала связи между программой и источником данных. Он позволяет устанавливать строку подключения, управлять транзакциями и устанавливать тип курсора. ADO поддерживает серверные и клиентские курсоры, а также управление многими другими свойствами курсора, предназначенными для управления видимостью записей и т. п.
- **Command.** Это класс представляющий исполняемую команду в базовом источнике данных. Применяется для выполнения запросов — произвольных SQL-строк или хранимых процедур. Он поддерживает параметры, что облегчает поддержку передачи значений, которые трудно передавать в обычной строке SQL-кода.

- **DataReader.** Это эквивалент конвейерного курсора с возможностью только чтения данных в прямом направлении. С DataReader можно получить однократный поток данных из базы данных, доступный только для чтения.
- **Transaction.** Объект транзакций (OleDbTransaction, SqlTransaction, OracleTransaction. В ADO.NET имеется пространство имен System.Transaction)
- **Parameter.** Объект параметр команды.

### 5.2.1. Используемые паттерны

#### Паттерн Active Record:

Один объект управляет и данными, и поведением. Большинство этих данных постоянны и их надо хранить в БД. Этот паттерн использует наиболее очевидный подход — хранение логики доступа к данным в объекте сущности. Объект является «обёрткой» одной строки из БД или представления, включает в себя доступ к БД и логику обращения с данными. Один объект соответствует одной строке таблицы. При обновлении полей объекта обновляется соответствующая строка таблицы.

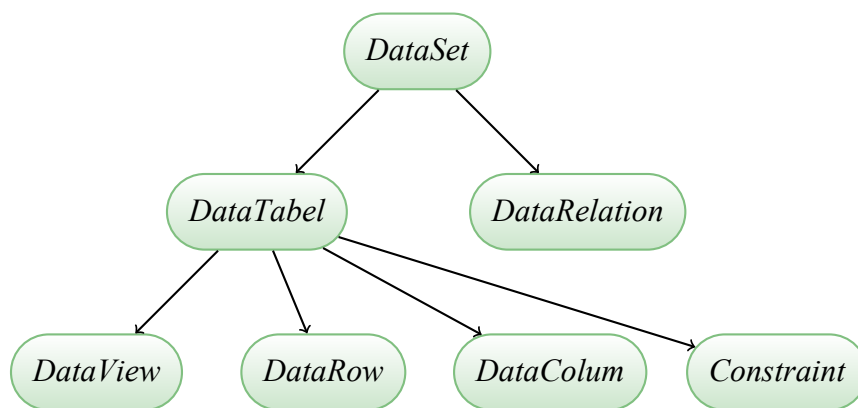
**Пример:** объект «Работник» содержит данные об одном работнике и методы: добавить, обновить или удалить. Помимо прочего, отдельным методом вынесена смена имени.

### 5.3. Отсоединенные объекты

Они позволяют работать с данными автономно. Их также называют автономными.

- Обеспечивает работу с данными в отсутствии подключения к БД.
- Удобна для переноса данных по сети.
- Расходует достаточно много памяти.

Иерархически модель можно изобразить следующим образом:



- **DataSet.** Класс *DataSet* является ядром автономного режима доступа к данным в ADO.NET. Лучше всего рассматривать, как будто в нем есть своя маленькая СУБД, полностью находящаяся в памяти.
- **DataTable.** Больше всего этот класс похож на таблицу БД. Он состоит из объектов *DataColumn*, *DataRow*, представляющих из себя строки и столбцы.
- **DataRow.**
- **DataColumn.**
- **DataRelation.** Этот класс позволяет задавать отношения между различными таблицами, с помощью которых можно проверять соответствие данных из различных таблиц.
- **DataView.** Это объект представлений базы данных.

### 5.3.1. Используемые паттерны

- **DTO (Data Transfer Object).** Объект, который пересылает данные между процессами для уменьшения количества вызовов методов. Data Transfer Object, может хранить всю необходимую для вызова информацию. Он должен быть сериализуемым для удобной передачи по сети. Обычно используется объект-сборщик для передачи данных между DTO и объектами в приложении.
- **Table Module** Одна сущность обрабатывает всю бизнес-логику для всех строк таблицы БД.
- **Unit of Work.** Обслуживает набор объектов, изменяемых в бизнес-транзакции (бизнес-действии) и управляет записью изменений и разрешением проблем конкуренции данных.

Когда необходимо писать и читать из БД, важно следить за тем, **что** вы изменили и если не изменили — не записывать данные в БД. Также необходимо вставлять данные о новых объектах и удалять данные о старых.

Можно записывать в БД каждое изменение объекта, но это приведет к большому количеству мелких запросов к БД. И как следствие — к замедлению работы приложения. Кроме того, требуется держать открытую транзакцию пока работает приложение. Это непрактично, если приложение обрабатывает несколько запросов одновременно. Ситуация еще хуже, если необходимо следить за чтением из БД, чтобы избежать «грязного» чтения.

## 6. Решение

### 6.1. Концепция

Клиент реализует Use-Case'ы для социальной сети журналистов. Use-Case'ы и таблицы базы данных были спроектированы в прошлом семестре. Пользователи (роли):

- Зарегистрированные пользователи.
  - + Могут просматривать и комментировать статьи, и комментарии участников.
- Зарегистрированные журналисты.
  - + Могут создавать, просматривать, комментировать статьи.
  - + Править свои статьи.
- Модераторы. Кроме того, что они обладают всеми правами **Зарегистрированных журналистов**.
  - + Редактировать и удалять статьи и комментарии к ним.

Является ли пользователь модератором определяется при входе его в систему.

Мы не стали реализовывать явного разделения «зарегистрированных пользователей» и «зарегистрированных журналистов», а просто реализовали два различных окна MDI приложения. Эти окна реализует как раз и реализуют соответствующие Use-Case'ы. Все перечисленные типы пользователей могут просматривать статистику сети.

Мы преднамеренно реализовали не все задуманные Use-Case'ы. При написании клиента было замечено, что система, включая диаграмму и схему базы данных имеет серьезные ошибки на этапе проектирования. В рамках лабораторных работ не имеет смысла перепроектировать всю систему. А реализовывать все «неправильные» Use-Case'ы — не интересно. Ниже напомним, что мы хотели ранее.

### 6.1.1. Было в проекте

Изначально мы считали, что пользователи делятся на (роли):

- Не зарегистрированных пользователей.
  - + Могут просматривать только открытые статьи, комментарии и личные страницы участников.
  - + Зарегистрироваться на общих правах.
- Зарегистрированных пользователей.
  - + Могут просматривать и комментировать только открытые статьи, комментарии и личные страницы участников.
- Зарегистрированных журналистов.
  - + Могут создавать, просматривать, комментировать любые статьи, комментарии и личные страницы участников.
  - + Принимать, отправлять личные сообщения.
  - + Участвовать в конкурсах.
- Vip-Гостей
  - + Могут создавать, просматривать, комментировать открытые статьи, комментарии и личные страницы участников.
  - + Принимать отправлять личные сообщения.
- Модераторов. Кроме того, что обладают всеми правами **Зарегистрированных журналистов**.
  - + Создавать аккаунты журналистов.
  - + Создавать аккаунты VIP-гостей.
  - + Редактировать статьи и комментарии к ним.



## 6.2. Краткое описание используемых средств и технологий

- Язык программирования на стороне клиента: **.NET C#**.
  - + Графическая библиотека: **Windows Forms**.
  - + Интегрированная среда разработки: **Microsoft Visual Studio 2008 PE**.
- Язык программирования на стороне сервера: **PL/SQL**.
  - + База данных: **Oracle**. База данных была создана и заполнена до начала разработки приложения.
  - + Драйвер базы данных: **System.Data.OracleClient**.
- Метод размещения бизнес-логики: смешанный (как это, описано ниже).

## 6.3. Окна

Наше приложение обладает MDI интерфейсом. Графический интерфейс на основе multiple document interface (или MDI) — представляет собой окна, расположенные под одним общим окном (как правило, за исключением модальных окон), в отличие от окон, расположенных отдельно друг от друга (SDI).

Внутренние окна MDI:

- «Пользователи» — реализует Use-Case «зарегистрированные пользователи».
  - + Эта часть клиента построена на хранимых процедурах и по сути дела является самостоятельным «тонким» клиентом.
- «Статьи» — реализует Use-Case'ы «зарегистрированные журналисты» и «модераторы».
  - + Эта часть клиента построена на отсоединенной модели и по своей сути является самостоятельным «толстым» клиентом.

От этого окна зависят окна:

- 1) «Новая статья».
- 2) «Правка статьи».
- 3) «Правка комментария».

Они также используют отсоединенной модель базы данных.

## 6.4. Исходный код

### 6.4.1. C#

Приведем в отчете только значимую с точки зрения базы часть кода.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Data.OracleClient;
5  using System.Data.Common;
6  using System.Data;
7
8  namespace JournalistClient
9  {
10     public static class DataRowExtensions
11     {
12         public static DataRow dbArticleAuthor(this DataRow val)
13         {
14             // применим расширение для класса DataRow
15             // чтобы не таскать с собой имя отношения
16             return val.GetParentRows("auath")[0];
17         }
18     }
19     public class DataBaser
20     {
21         public Items.User Owner;
22         // тот кто запустил программу,
23         // должен быть зарегистрированным пользователем
24         public DataSet dataset;
25         // отсоединенный вариант базы данных
26         public Dictionary<string, OracleDataAdapter> adapter;
27         public DataBaser()
28         {
29             Owner = new Items.User();
30             dataset = new DataSet("journaliga");
31             this.adapter = new Dictionary<string, OracleDataAdapter>();
32
33             this.makeDataTable("article", "number_of_article");
34             this.makeDataTable("personpage", "person_number");
35             this.makeDataTable("personcomment", "number_of_comment");
36             this.makeRelations();
37         }
38     #region C#DBC
39     public bool userExists(String nickname, String password )
40     {
41         bool res = false;
42         OracleConnection connection = new OracleConnection(Config.connectionString);
43         try
44         {
45             OracleCommand command =
46                 new OracleCommand("select * from personpage p where p.nickname=' "
47                     + nickname + "' ",
48                     connection);
49             connection.Open();
50             OracleDataReader dataReader = command.ExecuteReader();
51             if (dataReader.Read())
52             {
53                 res = true;
54                 Owner.Id = dataReader.GetInt32(0);
55                 Owner.NickName = dataReader.GetString(1);

```

```

56         Owner.Name = dataReader.GetString(2);
57         Owner.Surname = dataReader.GetString(3);
58         Owner.IsModerator = (1 == dataReader.GetInt32(4));
59         Owner.IsGust = (1 == dataReader.GetInt32(5));
60     }
61 }
62 finally
63 {
64     connection.Close();
65 }
66 return res;
67 }
68 public List<Items.User> ListUsers()
69 {
70     /**
71     * Получает список пользователей.
72     */
73     OracleConnection connection = new OracleConnection(Config.connectionString);
74     List<Items.User> list = new List<Items.User>();
75     try
76     {
77         OracleCommand listCommand =
78             new OracleCommand("SELECT * from PersonPage order by PERSON_NUMBER",
79                 connection);
80         connection.Open();
81         OracleDataReader dataReader = listCommand.ExecuteReader();
82         Items.User user;
83         while (dataReader.Read())
84         {
85             user = new Items.User();
86             user.Id = dataReader.GetInt32(0);
87             user.NickName = dataReader.GetString(1);
88             user.Name = dataReader.GetString(2);
89             user.Surname = dataReader.GetString(3);
90             list.Add(user);
91         }
92     }
93     finally
94     {
95         connection.Close();
96     }
97     return list;
98 }
99 #endregion
100
101 #region Отсоединенная модель
102 public void makeDataTable(String tableName, String sorter)
103 {
104     /**
105     * Создает таблицу с именем tableName в отсоединенной БД.
106     * Делает выборку по sorter. Sorter должен быть первичным ключом.
107     */
108     OracleDataAdapter da =
109         new OracleDataAdapter("select * from "
110             + tableName
111             + " order by "
112             + sorter
113             + " desc",
114             Config.connectionString);
115     OracleCommandBuilder cb = new OracleCommandBuilder(da);
116     da.UpdateCommand = cb.GetUpdateCommand();
117     da.InsertCommand = cb.GetInsertCommand();

```

```

118 //заполним отсоединенную бд
119 da.Fill(this.dataset, tableName);
120 //положим текущий адаптер в словарь адаптеров
121 this.adapter.Add(tableName, da);
122 //зададим первичный ключ. Нужно исключительно для поиска по id
123 DataColumn[] col = new DataColumn[1];
124 col[0] = this.dataset.Tables[tableName].Columns[0];
125 this.dataset.Tables[tableName].PrimaryKey = col;
126 //посмотрим что там у нас:
127 Console.WriteLine(col[0].ColumnName);
128 }
129 public void makeRelations()
130 {
131     //Создает отношения между таблицами.
132     //>>> Так удобнее брать данные из различных таблиц
133     //АВТОР -> СТАТЬИ: \ АВТОР -> КОМЕНТАРИИ:
134     DataColumn parentColumn = this.dataset.Tables["personpage"].Columns["PERSON_NUMBER"];
135     DataColumn childColumn_article = this.dataset.Tables["article"].Columns["PERSON_NUMBER"];
136     DataColumn childColumn_personcomment = this.dataset.Tables["personcomment"].Columns["PERSON_NUMBER"];
137     //СТАТЬЯ -> КОМЕНТАРИИ:
138     DataColumn publ = this.dataset.Tables["article"].Columns["NUMBER_OF_ARTICLE"];
139     DataColumn comm = this.dataset.Tables["personcomment"].Columns["NUMBER_OF_ARTICLE"];
140     //АВТОР -> СТАТЬИ:
141     this.dataset.Relations.Add("auath", parentColumn, childColumn_article);
142     //АВТОР -> КОМЕНТАРИИ:
143     this.dataset.Relations.Add("comath", parentColumn, childColumn_personcomment);
144     //СТАТЬЯ -> КОМЕНТАРИИ:
145     this.dataset.Relations.Add("aucom", publ, comm);
146 }
147 public void updateDataTable(String tableName)
148 {
149     /**
150     * Обновляет таблицу с именем tableName
151     */
152     OracleDataAdapter da = this.adapter[tableName];
153     da.Update(this.dataset.Tables[tableName]);
154 }
155
156 public void LoadDataTable(String tableName)
157 {
158     /**
159     * Обновляет отсоединенную таблицу с именем tableName
160     *
161     * эта строка нужна, в том случае, если
162     * за время нашего редактирования уже кто-то
163     * модифицировал нашу базу данных
164     * но это еще не попало в отсоединенную модель
165     */
166
167     OracleDataAdapter da = this.adapter[tableName];
168     da.Fill(dataset, tableName);
169     da.Update(this.dataset.Tables[tableName]); // на всякий пожарный
170 }
171
172 public int getMaxId(String tableName)
173 {
174     /**
175     * Возвращает наибольший id для таблицы с именем tableName.
176     */
177     // не самая эффективная реализация :-\
178     // в случае удалений образуются дырки в idшниках—.
179     var table = this.dataset.Tables[tableName];

```

```

180     var rows = table.Rows;
181     String primaryKeyName = table.PrimaryKey[0].Caption;
182     int max = 0; // id считаем от 0;
183     int max_q = 0;
184     foreach (DataRow row in rows)
185     {
186         max_q = Convert.ToInt32(row[primaryKeyName].ToString());
187         if (max_q > max)
188             max = max_q;
189     }
190     return max;
191 }
192 #endregion
193
194 #region отсоединенная модель старая
195 public DataTable dataTablePublications()
196 {
197     /**
198     * Возвращает таблицу со статьями.
199     *
200     * Работает в отсоединенной модели.
201     */
202
203     OracleDataAdapter da =
204         new OracleDataAdapter("select * from article order by number_of_article desc",
205             Config.connectionString);
206     OracleCommandBuilder cb = new OracleCommandBuilder(da);
207     da.UpdateCommand = cb.GetUpdateCommand();
208     da.InsertCommand = cb.GetInsertCommand();
209     Console.WriteLine(da.UpdateCommand.CommandText);
210     Console.WriteLine(da.InsertCommand.CommandText);
211     DataTable dt = new DataTable("article");
212     da.Fill(dt);
213     // заполним таблицу
214     da.Fill(dataset, "article");
215     // заполним отсоединенную бд
216     return dt;
217 }
218 public DataTable dataTablePersonpages()
219 {
220     /**
221     * Возвращает со пользователямиавторами—.
222     *
223     * Работает в отсоединенной модели.
224     */
225
226     OracleDataAdapter da =
227         new OracleDataAdapter("select * from personpage order by person_number desc",
228             Config.connectionString);
229     OracleCommandBuilder cb = new OracleCommandBuilder(da);
230     da.UpdateCommand = cb.GetUpdateCommand();
231     da.InsertCommand = cb.GetInsertCommand();
232     Console.WriteLine(da.UpdateCommand.CommandText);
233     Console.WriteLine(da.InsertCommand.CommandText);
234     DataTable dt = new DataTable("personpage");
235     da.Fill(dt);
236     // заполним таблицу
237     da.Fill(dataset, "personpage");
238     // заполним отсоединенную бд
239     return dt;
240 }
241 #endregion

```

```
242
243 #region ТОНКИЙ КЛИЕНТ
244
245 public List<Items.Publication> ListPublications(String author)
246 {
247     /**
248     * Получает список статей для данного автора.
249     */
250     OracleConnection connection = new OracleConnection(Config.connectionString);
251
252     List<Items.Publication> list = new List<Items.Publication>();
253     try
254     {
255         connection.Open();
256         OracleCommand iCommand = new OracleCommand("find_all_articles_by_string", connection);
257         iCommand.CommandType = CommandType.StoredProcedure;
258         iCommand.Parameters.Add("author", OracleType.VarChar);
259         iCommand.Parameters["author"].Value = author;
260         iCommand.Parameters.Add("info", OracleType.Cursor).Direction = ParameterDirection.Output;
261         OracleDataReader dataReader = iCommand.ExecuteReader();
262         while (dataReader.Read())
263         {
264             Items.Publication item = new Items.Publication();
265             item.Id = dataReader.GetInt32(Items.Publication.Table.Id);
266             item.Name = dataReader.GetString(Items.Publication.Table.Name);
267             item.Date = dataReader.GetDateTime(Items.Publication.Table.Date);
268             item.Text = dataReader.GetString(Items.Publication.Table.Text);
269             list.Add(item);
270         }
271     }
272     finally
273     {
274         connection.Close();
275     }
276     return list;
277 }
278
279 public List<Items.Comment> ListComments(int article_id) {
280     List<Items.Comment> list = new List<Items.Comment>();
281     OracleConnection connection = new OracleConnection(Config.connectionString);
282     try
283     {
284         connection.Open();
285         OracleCommand iCommand = new OracleCommand("find_all_comm_by_article_id", connection);
286         iCommand.CommandType = CommandType.StoredProcedure;
287         iCommand.Parameters.Add("article_id", OracleType.Int32);
288         iCommand.Parameters["article_id"].Value = article_id;
289         iCommand.Parameters.Add("info", OracleType.Cursor).Direction = ParameterDirection.Output;
290         OracleDataReader dataReader = iCommand.ExecuteReader();
291         while (dataReader.Read())
292         {
293             Items.Comment item = new Items.Comment();
294             item.User = new Items.User();
295             item.User.NickName = dataReader.GetString(0);
296             item.Text = dataReader.GetString(1);
297             list.Add(item);
298         }
299     }
300     finally
301     {
302         connection.Close();
303     }
}
```

```
304     return list;
305 }
306 public void insertComment(int article_id, String Text)
307 {
308     /**
309     * Добавление нового комментария
310     */
311     OracleConnection connection = new OracleConnection(Config.connectionString);
312     OracleTransaction tx = null;
313     try
314     {
315         connection.Open();
316         tx = connection.BeginTransaction();
317         OracleCommand iCommand = new OracleCommand("insert_new_comm", connection);
318         iCommand.CommandType = CommandType.StoredProcedure;
319         iCommand.Transaction = tx;
320         iCommand.Parameters.Add("author", OracleType.Int32);
321         iCommand.Parameters.Add("article_id", OracleType.Int32);
322         iCommand.Parameters.Add("newtext", OracleType.VarChar);
323         iCommand.Parameters["author"].Value = Owner.Id;
324         iCommand.Parameters["article_id"].Value = article_id;
325         iCommand.Parameters["newtext"].Value = Text;
326         iCommand.ExecuteNonQuery();
327     }
328     catch (Exception ex)
329     {
330         System.Console.WriteLine(ex.Message);
331         System.Console.WriteLine("ROLLBACK TRANSACTION");
332         tx.Rollback();
333     }
334     finally
335     {
336         connection.Close();
337     }
338 }
339 #endregion
340 }
341 }
```

---

## 6.4.2. PL/SQL

Возвращает в все статьи автора по его имени, фамилии, или нику:

---

```

1 create or replace procedure find_all_articles_by_string(author varchar, info out sys_refcursor)
2 is
3 begin
4     open info for
5         select a.number_of_article, a.date_, a.name_, a.text from article a,
6             (
7                 select p.person_number as num
8                 from personpage p
9                 where
10                     (p.nickname = author)
11                     or
12                     (p.name_ = author)
13                     or
14                     (p.surname = author)
15             )
16         where a.person_number = num;
17 end;
18 /

```

---

Возвращает в все комментарии к данной статье:

---

```

1 create or replace procedure find_all_comm_by_article_id(article_id integer, info out sys_refcursor)
2 is
3 begin
4     open info for
5         select p.NICKNAME, c.text from PERSONCOMMENT c, PERSONPAGE p
6         where
7             (c.NUMBER_OF_ARTICLE = article_id and c.NUMBER_OF_ARTICLE != -1)
8             and
9             (p.PERSON_NUMBER = c.PERSON_NUMBER)
10         order by c.NUMBER_OF_COMMENT desc;
11 end;
12 /

```

---

Создает новый комментарий для статьи:

---

```

1 create or replace procedure insert_new_comm(author integer, article_id integer, newtext varchar)
2 is
3 begin
4     insert into personcomment(
5         number_of_comment,
6         person_number,
7         number_of_article,
8         concurs_number,
9         text
10     )
11     values(
12         (select max(p.NUMBER_OF_COMMENT) from PERSONCOMMENT p) + 1,
13         author,
14         article_id,
15         -1,
16         newtext
17     );
18     commit;
19 end;
20 /

```

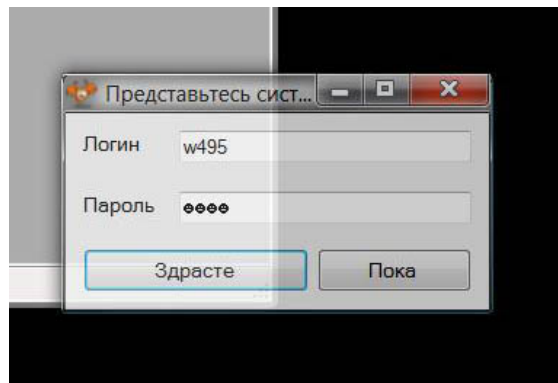
---



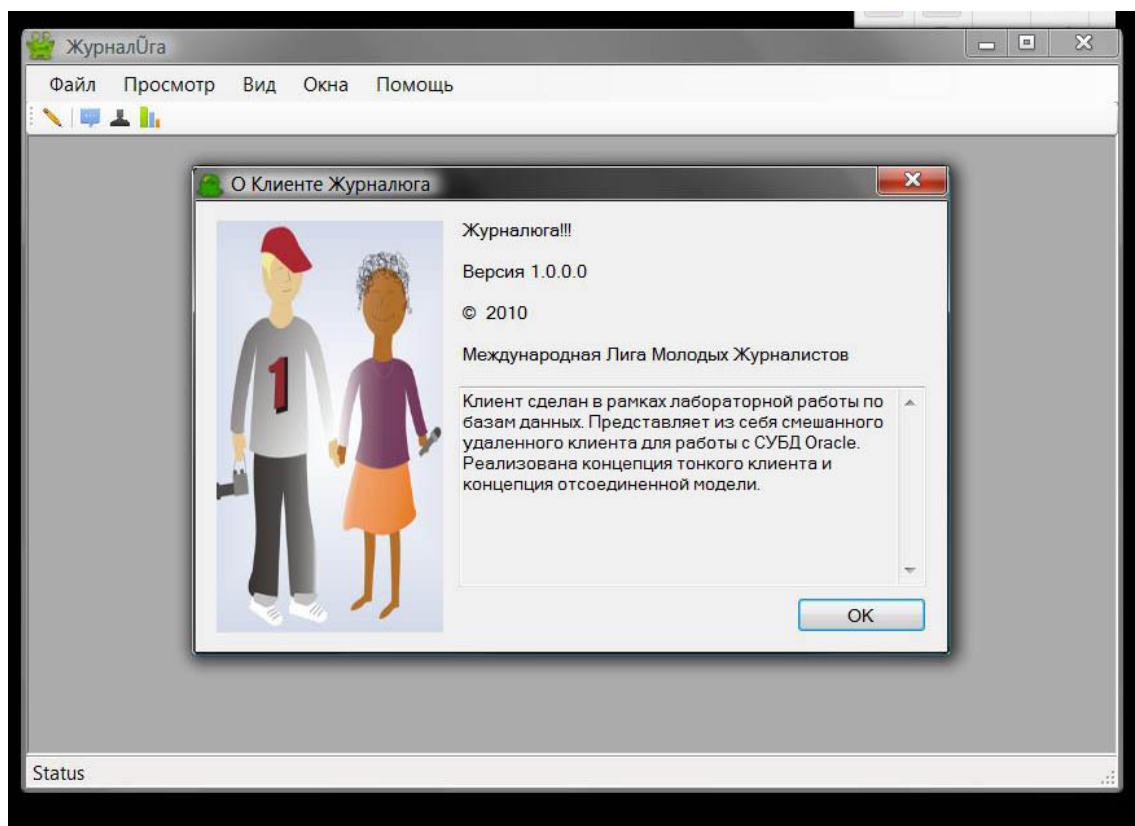
## 7. Пример работы программы

Приведем скриншоты.

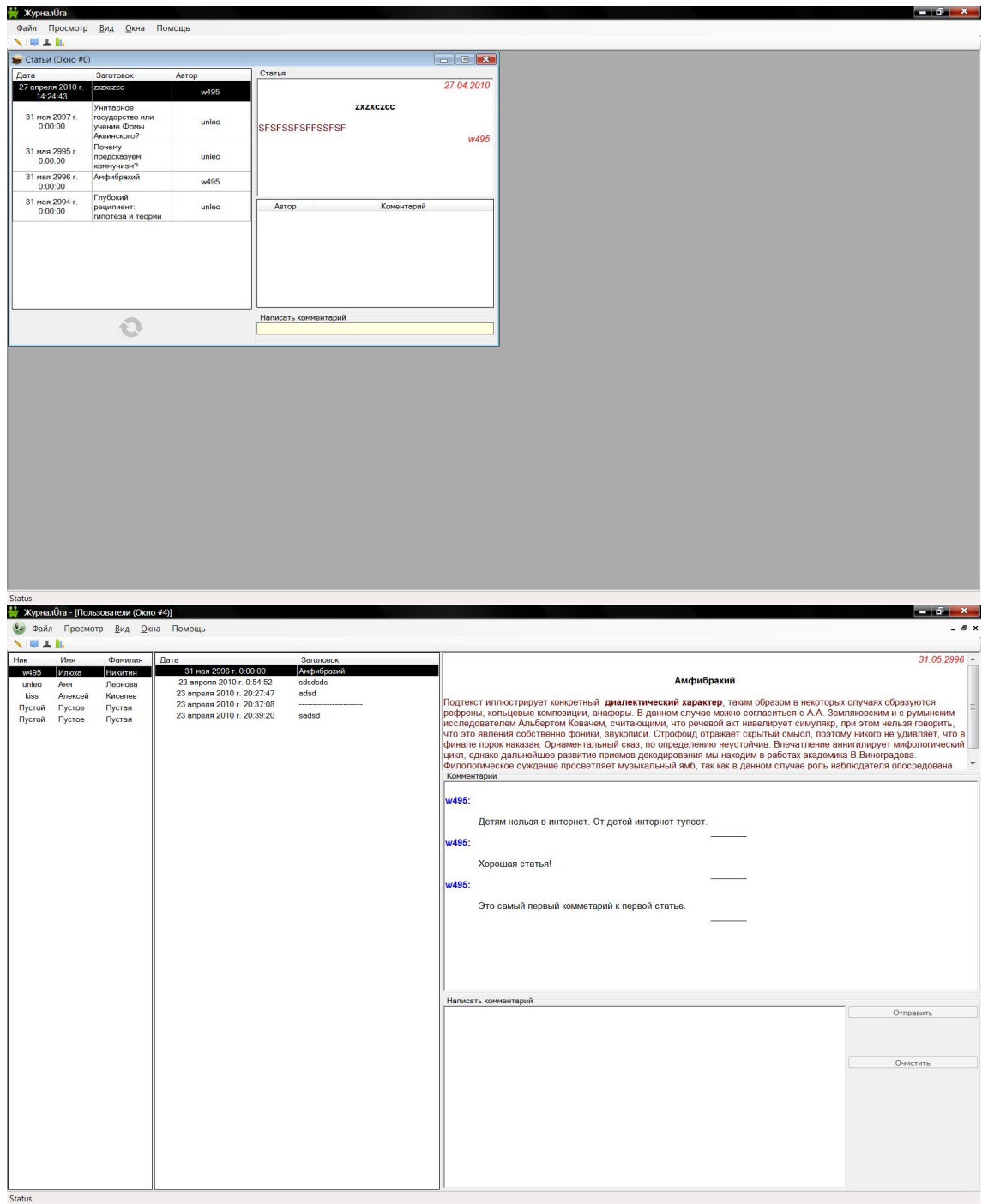
### 7.1. Окно входа

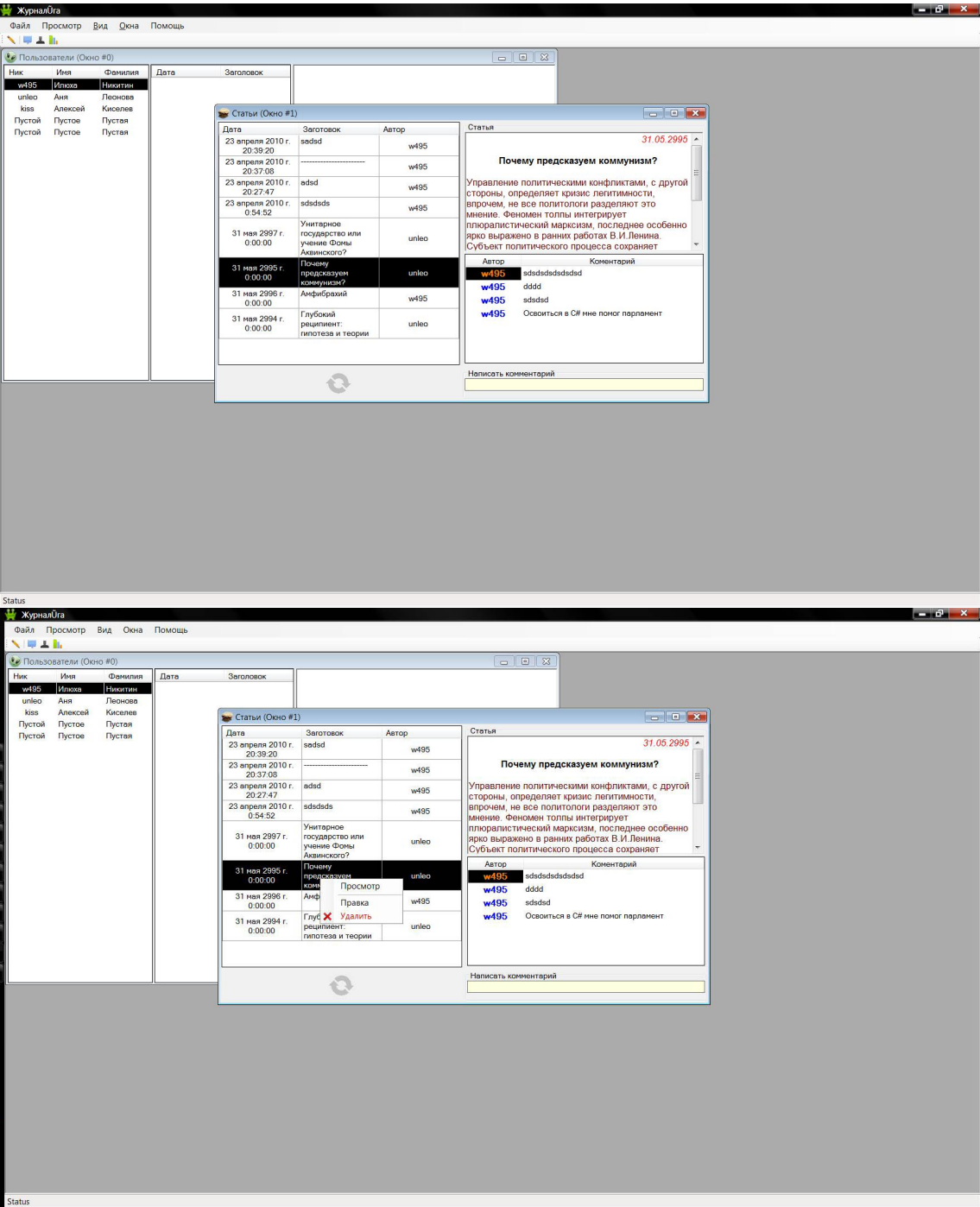


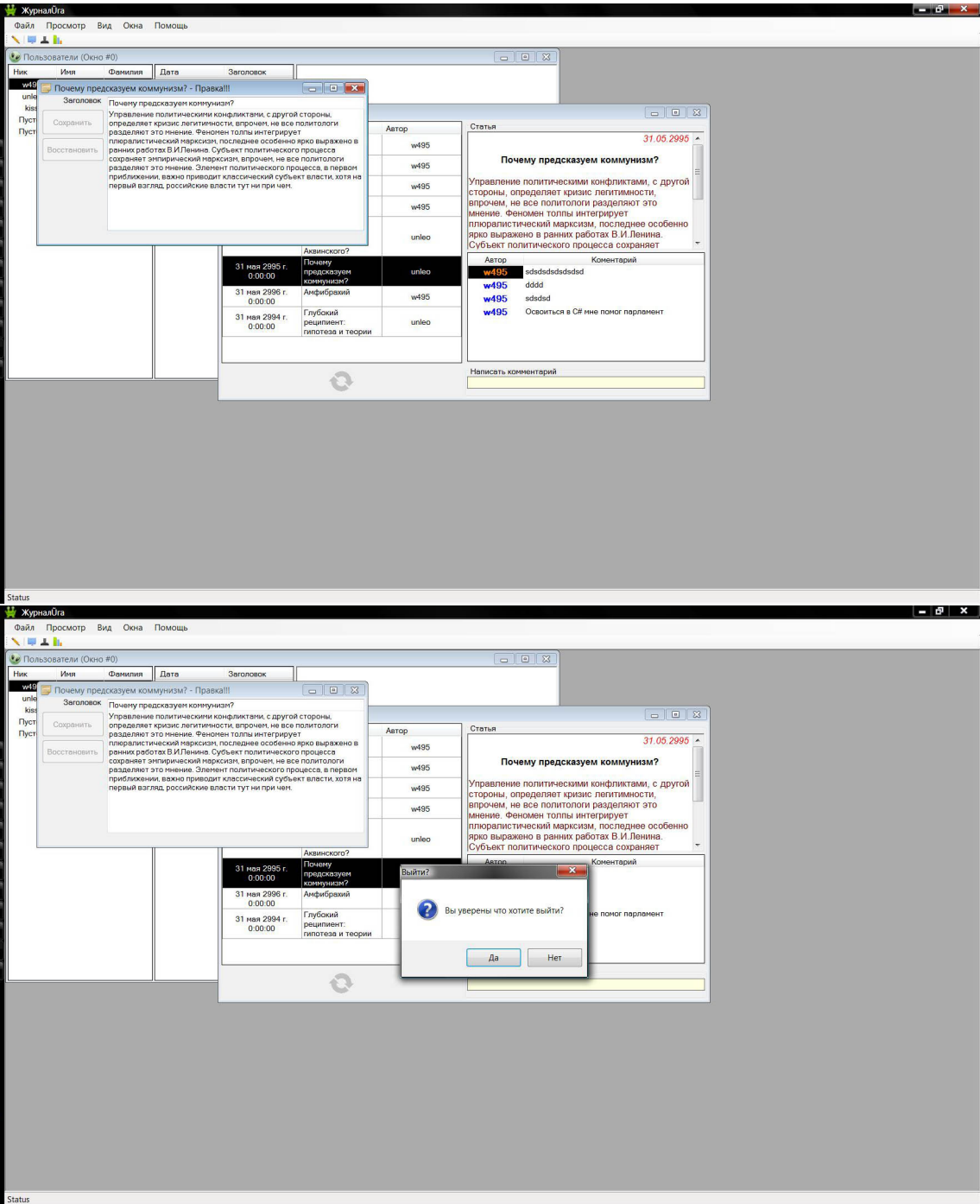
### 7.2. О программе

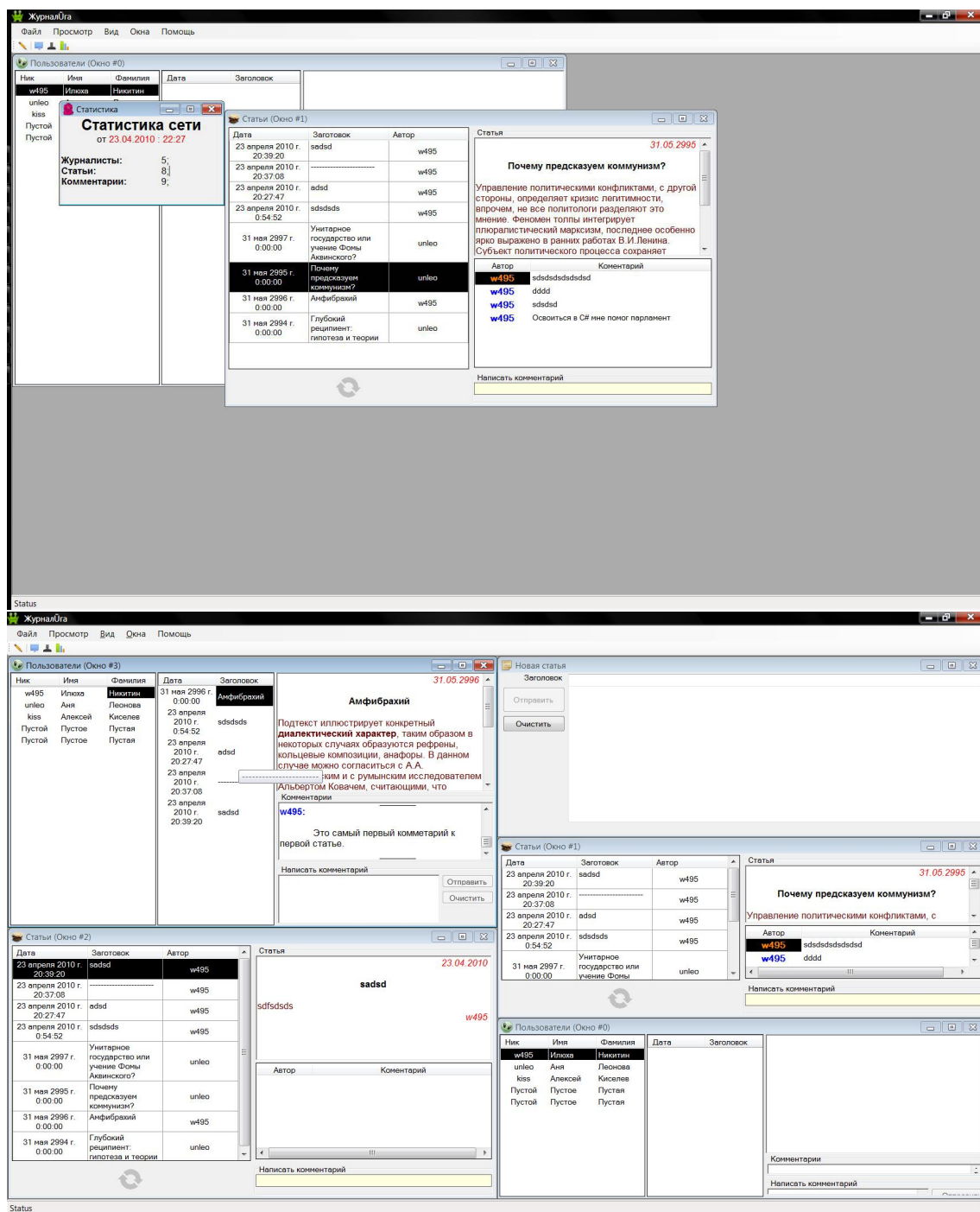


7.3. Рабочее окно









## 8. Выводы

В процессе выполнения лабораторной работы мы познакомились с набором технологий доступа к данным ADO.NET. С одной стороны, ADO.NET достаточно прост в изучении и использовании для небольших приложений. С другой стороны, очевидно, что его прямое использование для серьезных проектов неприменимо (а если применимо, то очень аккуратно). При попытке перейти на другую базу данных нам придется использовать другой драйвер или модифицировать запросы. Придется изменять текст программы, это может занять значительное время и привести к ошибкам, которые будет трудно обнаружить. Кроме того, использование отсоединенной модели в явном виде провоцирует на смешивание бизнес-логики и представления. Что тоже может привести к ошибкам. Очень интересной оказалась концепция «тонкого» клиента с использованием хранимых процедур. То что достаточно громоздко и не эффективно выражается на императивном языке, на SQL пишется в пару строк. Однако написание «тонкого» клиента в целом представляется сложнее, чем «толстого». Тут нужно хорошо знать язык своей базы данных и понимать принципы ее работы. До некоторого времени «тонкие» клиенты были единственно возможными способами взаимодействия с удаленной базой данных. В настоящее, на мой взгляд, использование «тонкого» клиента остается актуальным. Пока не везде в мире есть достаточно мощные персональные компьютеры и хорошие каналы связи, чтобы можно было передать и обработать большой объем информации, а запросы пользователей растут, значительно опережая свое время.

Это моя первая программа, написанная на языке C#. В целом мне понравился этот язык (я ожидал худшего). Все просто и логично, но немного длинновато и громоздко (по сравнению с Python).

# ASP.NET MVC и Nhibernate

## 9. Теоретическая часть

Компоненты приложения:

- 1) Сервер базы данных (Oracle).
- 2) Сервер приложений (ASP.NET).
- 3) Клиентское приложение — браузер конечного пользователя.

### 9.1. ORM

**ORM** — технология построения программного обеспечения, при которой строится связь между реляционной базой данных и описанием этих данных в виде классов.

Таким образом, создается эффект некой «объектной базы данных», к которой можно обращаться непосредственно из приложения, минуя сервер обращение к серверу базы данных. Но это только видимость.

**NHibernate** — ORM-решение для платформы Microsoft .NET портированное с Java. Это бесплатная библиотека с открытым кодом. NHibernate позволяет отображать объекты бизнес-логики на реляционную базу данных. По заданному XML-описанию сущностей и связей NHibernate автоматически создает SQL-запросы для загрузки и сохранения объектов. NHibernate является портом на .NET популярной на платформе Java библиотеки Hibernate.

#### 9.1.1. Возможности ORM

- Отложенная загрузка связанных объектов.
- Обеспечение блокировок. Блокировки:
  - + пессимистическая;
  - + оптимистическая;
- Кэширование загруженных объектов.
- SQL-подобные запросы по объектной модели.

### 9.1.2. Преимущества ORM

- Нет необходимости писать рутинные insert/update/delete/select для CRUD операций
- Условия связи между объектами (строками таблиц) указываются декларативно в одном месте.
- Возможность использовать полиморфные запросы для иерархий классов
- Высокая степень независимости от конкретной СУБД

### 9.1.3. Недостатки ORM

- Возможны проблемы с производительностью для сложных запросов на объектном SQL.
- Затрудняет использование специфических конструкций языка SQL конкретной СУБД.
- Не всегда можно соотнести друг с другом реляционную и объектную модель данных.



## 9.2. MVC

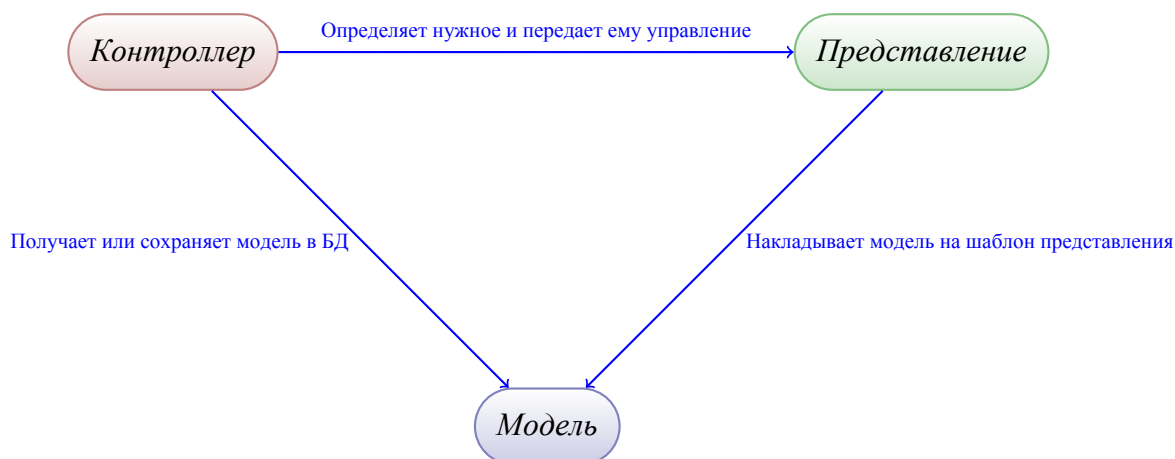
**MVC (Model-view-controller)** — «Модель-представление-контроллер» — архитектура программного обеспечения, в которой

- модель данных приложения,
- пользовательский интерфейс (представление) и
- управляющая логика (контроллер)

разделены на три отдельных компонента, так, что модификация одного из компонентов оказывает минимальное воздействие на другие компоненты.

Важно отметить, представление и поведение зависят от модели. Однако модель не зависит ни от представления, ни от поведения.

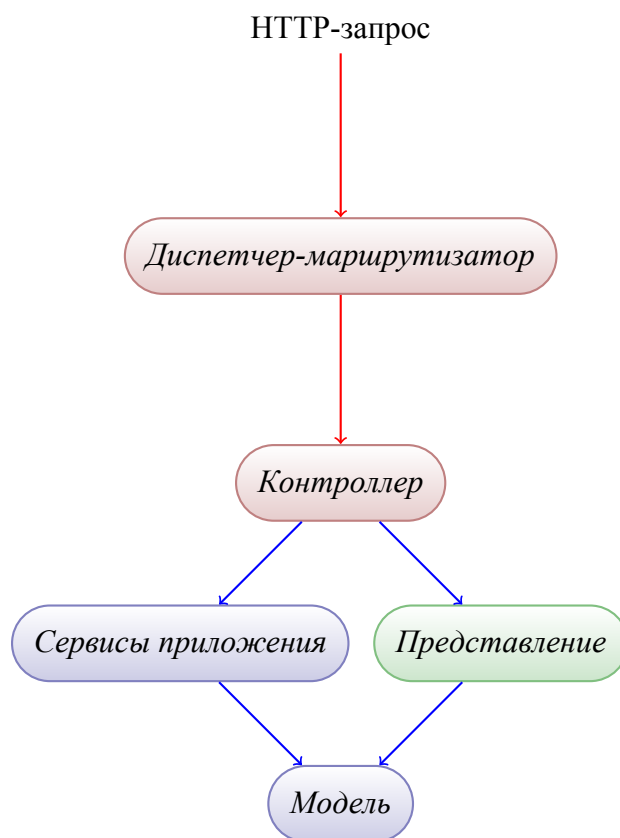
### 9.2.1. Классический MVC



### 9.2.2. MVC в веб-приложении

Этот паттерн удобно использовать в Веб-приложениях:

- Внешний вид страниц часто меняется.
- Иногда нужно иметь несколько вариантов представления.
- Прозрачность обработки запроса.
- Легковесные объекты.
- Тестирование приложения без зависимости от интерфейса.
- Широкие возможности по расширению.



Маршрутизатору поступает http-запрос. Он ищет конкретный контроллер и передает ему управление. Контроллер получает данные из БД с помощью сервисов приложения. Они в свою очередь взаимодействуют с моделью и с базой данных. Контроллер Выбирает нужное представление и передает ему модель и управление. Представление накладывает модель на шаблон, получая HTML-страницу.

### 9.3. Архитектура слоев приложения

Веб-приложение можно рассмотреть в виде слоев:

- 1) Слой интерфейса пользователя отвечает за взаимодействие с пользователем.
  - Контроллеры.
  - Представления.
- 2) Слой сервисов приложения отвечает за выполнение бизнес-логики приложения.
  - Сервисы приложения.
- 3) Слой для работы приложения с БД.
  - В нашем случае — Nhibernate.

Отдельно стоит объектная модель данных приложения. Она используется всеми тремя слоями.

## 10. Решение

### 10.1. Концепция

Клиент частично реализует Use-Case модератора для социальной сети журналистов. Use-Case'ы и таблицы базы данных были спроектированы в прошлом семестре. Работа осуществляется только с одной таблицей PersonPage. Для работы с этой таблицей в стиле ORM создан класс-модель PersonPage. По этой модели NHibernate.Mapping.Attributes строит mapping для отображения класса в реляционную таблицу. NHibernate.Mapping.Attributes использует модификаторы (надписи в квадратных скобках) класса и его атрибутов. Модификаторы платформозависимые. К сожалению нет возможности строить универсальный mapping для всех видов БД. А при использовании NHibernate.Mapping.Attributes mapping зависит от модели. Потому наша модель приспособлена только для базы данных Oracle. Если нужным образом заменить модификаторы модели, то ее можно сделать пригодной для любой другой базы данных. Для работы с моделью используется DAO PersonPage.

Можно просматривать список пользователей и редактировать отдельных пользователей. Для того, чтобы при редактировании пользователей, не возникали противоречивые ситуации, мы используем **оптимистическую блокировку**.

#### 10.1.1. Оптимистическая блокировка

Если 2 оператора (администратора, модератора) пытаются отредактировать одну и ту же запись пользователя, и один из них оказался быстрее, то второму, когда он будет сохранять свой вариант данных, сообщится, что он работал с неактуальными данными. Блокировка работает на основе версий. В таблицу добавляется столбец версий (через alter table). Изначально значение этого столбца для каждой записи равняется нулю. При каждом изменении записи значение увеличивается на единицу. Перед очередным обновлением проверяется, совпадает ли та версия данных, которую мы редактировали с той, которая сейчас находится в таблице. Если это так, то пока мы редактировали данные, их никто не изменял и мы можем спокойно их обновить. Если версии различны, то мы работали с уже устаревшими данными.

В вашей реализации оптимистической блокировки есть один недостаток: между *select* и *update* может вклиниться *commit* другой транзакции, тогда мы все же перетрете данные этой транзакции. Это можно устранить, поставив уровень изоляции *SERIALIZABLE*, либо сделав проверку версии и обновление одним SQL-запросом:

```
1 update set ... where
2   version=:old_version and id=:id
```

Второе может автоматически сделать *Nhibernate*, если

```
1 <property name="Version" column="VERSION" />
```

заменить на

```
1 <version name="Version" column="VERSION" />
```

## 10.2. Краткое описание используемых средств и технологий

- Язык программирования на стороне клиента: **.NET C#**.
  - + Интегрированная среда разработки: **Microsoft Visual Studio 2008 PE**.
  - + Тип проекта: **ASP.NET MVC**
  - + Сервер Веб-приложения: **IIS, встроенный в Visual Studio**.
- Язык программирования на стороне сервера: **PL/SQL**.
  - + База данных: **Oracle**. База данных была создана и заполнена до начала разработки приложения.
  - + Драйвер базы данных: **System.Data.OracleClient**.
  - + Технология для работы с базой данных: **NHibernate**.

Используя технологию *NHibernate* нам нет необходимости следить за ключами записей таблиц. Они могут создаваться автоматически.

- Метод размещения бизнес-логики: на сервере приложений.

## 10.3. Исходный код

### 10.3.1. Конфигурация Nhibernate

---

```

1  <?xml version="1.0" encoding="utf-8" ?>
2  <configuration>
3    <configSections>
4      <section name="hibernate-configuration" type="NHibernate.Cfg.ConfigurationSectionHandler,
        NHibernate" />
5    </configSections>
6    <hibernate-configuration xmlns="urn:hibernate-configuration-2.2">
7      <session-factory name="connection.provider">
8        <property name="proxyfactory.factory_class">
9          NHibernate.ByteCode.Castle.ProxyFactoryFactory, NHibernate.ByteCode.Castle
10        </property>
11        <property name="connection.provider">NHibernate.Connection.DriverConnectionProvider</property>
12        <property name="connection.driver_class">NHibernate.Driver.OracleClientDriver</property>
13        <property name="connection.connection_string">
14          User ID=w495;
15          Password=w495;
16          Data Source=oraw495;
17        </property>
18        <!--
19          Здесь указаны параметры моей базы данных. Надеюсь, у вас они другие.
20          Если возникают проблемы (ORA-12541),
21          не забываем запускать "прослушивателя" (listener) командой:
22          lsnrctl start
23        -->
24        <property name="show_sql">false</property>
25        <property name="dialect">NHibernate.Dialect.Oracle10gDialect</property>
26        <!--
27          NHibernate.Dialect.OracleDialect не работает
28          В обязательном порядке надо указать конкретный диалект:
29          Oracle8iDialect
30          Oracle9iDialect
31          Oracle10gDialect
32          OracleLiteDialect
33        -->
34        <property name="query.substitutions">true 1, false 0, yes 'Y', no 'N' </property>
35      </session-factory>
36    </hibernate-configuration>
37  </configuration>

```

---

### 10.3.2. Описание модели

---

```

1 using NHibernate.Mapping.Attributes;
2 namespace JournaligaMVC.Models
3 {
4     [Class(Lazy = true, Name = "JournaligaMVC.Models.PersonPage, JournaligaMVC")]
5     public class PersonPage
6     {
7         [Id(Name = "id", Column = "PERSON_NUMBER")]
8         [Generator(Class = "sequence")]
9         [Param(Name = "sequence", Content = "SEQ_ID")]
10        virtual public int id { get; set; }
11        [Property(Name = "NickName", Column = "NICKNAME")]
12        virtual public string NickName { get; set; }
13        [Property(Name = "Name", Column = "NAME_")]
14        virtual public string Name { get; set; }
15        [Property(Name = "Surname", Column = "Surname")]
16        virtual public string Surname { get; set; }
17        [Property(Name = "IsModerator", Column = "ISMODERATOR")]
18        virtual public int IsModerator { get; set; }
19        [Property(Name = "IsGust", Column = "ISGUST")]
20        virtual public int IsGust { get; set; }
21        [Property(Name = "Version", Column = "VERSION")]
22        virtual public int Version { get; set; }
23        private static Dao.PersonPage _dao;
24        public static Dao.PersonPage Dao
25        {
26            get{
27                if (null == PersonPage._dao)
28                    PersonPage._dao = new Dao.PersonPage();
29                return PersonPage._dao;
30            }
31        }
32    }
33 }

```

---

### 10.3.3. Mapping модели

---

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <!--Generated from NHibernate.Mapping.Attributes on 2010-05-07 18:27:24Z.-->
3 <hibernate-mapping xmlns="urn:hibernate-mapping-2.2">
4     <class name="JournaligaMVC.Models.PersonPage, JournaligaMVC" lazy="true">
5         <id name="id" column="PERSON_NUMBER">
6             <generator class="sequence">
7                 <param name="sequence">SEQ_ID</param>
8             </generator>
9         </id>
10        <property name="NickName" column="NICKNAME" />
11        <property name="Name" column="NAME_" />
12        <property name="Surname" column="Surname" />
13        <property name="IsModerator" column="ISMODERATOR" />
14        <property name="IsGust" column="ISGUST" />
15        <property name="Version" column="VERSION" />
16    </class>
17 </hibernate-mapping>

```

---

#### 10.3.4. Описание DAO

**DAO (data access object)** — шаблон проектирования. Некий объект предоставляет абстрактный интерфейс к какому-либо типу базы данных или механизму хранения. Определенные возможности предоставляются независимо от того, какой механизм хранения используется и без необходимости специальным образом соответствовать этому механизму хранения. Традиционно этот шаблон связывают с приложениями на платформе Java Enterprise Edition. В нашем случае, это посредник взаимодействия контроллера и модели. Здесь он выступает в роли сервисов приложения. Контроллер ничего не знает о структуре и свойствах модели. О них знает DAO. У каждой модели свой DAO. В DAO реализованы основные функции взаимодействия с базой данных. Они не зависят от используемой технологии. Мы реализовали только одну модель и у нее есть один DAO — `PersonPage`. DAO `PersonPage` умеет:

- пересчитывать всех пользователей;
- возвращать список всех пользователей;
- возвращать пользователя по его `id`;
- сохранять изменения для конкретного пользователя (здесь реализована оптимистическая блокировка с версиями правок);

В нашем случае, DAO и сервис совмещены в одном классе `Dao.PersonPage`, что допустимо в случае простого приложения, хотя назвать лучше `PersonPageDao`, несмотря на **namespace**, может быть путаница и неудобно искать по имени класса. Интерфейс у него типовой для DAO, но размещая в нем дальнейшую логику приложения, можно столкнуться с тем, что захочется отделить работу с данными и с БД, а сессию и транзакцию вынести на уровень сервиса.



## Исходный код JournaligaMVC.Dao.PersonPage:

---

```

1 namespace JournaligaMVC.Dao
2 {
3     public class PersonPage{
4         public long Count {
5             get {
6                 long res = 0;
7                 using (ISession session = DataBaseHelper.OpenSession()){
8                     IQuery query = session.CreateQuery("FROM PersonPage");
9                     res = query.List<Models.PersonPage>().Count;
10                }
11                return res;
12            }
13        }
14        public IList<Models.PersonPage> List{
15            get{
16                IList<Models.PersonPage> res = null;
17                using (ISession session = DataBaseHelper.OpenSession()){
18                    IQuery query = session.CreateQuery("FROM PersonPage");
19                    if (0 != query.List<Models.PersonPage>().Count)
20                        res = query.List<Models.PersonPage>();
21                }
22                return res;
23            }
24        }
25        public Models.PersonPage getPerId(int n) {
26            Models.PersonPage res = null;
27            using (ISession session = DataBaseHelper.OpenSession()){
28                IQuery query = session.CreateQuery("FROM PersonPage WHERE PERSON_NUMBER = " + n);
29                if (1 == query.List<Models.PersonPage>().Count)
30                    res = query.List<Models.PersonPage>()[0];
31            }
32            return res;
33        }
34        public bool Save(Models.PersonPage item){
35            using (ISession session = DataBaseHelper.OpenSession()){
36                using (ITransaction transaction = session.BeginTransaction()){
37                    IQuery query = session.CreateQuery("FROM PersonPage WHERE PERSON_NUMBER = " + item.id);
38                    if (1 == query.List<Models.PersonPage>().Count){
39                        Models.PersonPage res = query.List<Models.PersonPage>()[0];
40                        if (res.Version != item.Version){
41                            transaction.Rollback();
42                            return false;
43                        }
44                        res.Version = item.Version + 1;
45                        res.NickName = item.NickName;
46                        res.Name = item.Name;
47                        res.Surname = item.Surname;
48                        transaction.Commit();
49                    }
50                }
51            }
52            return true;
53        }
54    }
55 }

```

---

### 10.3.5. Описание класса для взаимодействия с базой данных.

---

```
1 public static class DataBaseHelper
2 {
3     static ISessionFactory SessionFactory;
4     static System.IO.MemoryStream stream;
5     static public ISession OpenSession()
6     {
7         if (SessionFactory == null)
8         {
9             Configuration configuration = new Configuration();
10            String Nhibernate_config = JournaligaMVC.Properties.Settings.Default.Nhibernate_config.ToString();
11            configuration.Configure(Nhibernate_config);
12            stream = new System.IO.MemoryStream();
13            NHibernate.Mapping.Attributes.HbmSerializer.Default.Serialize(stream, System.Reflection.Assembly.GetExecutingAssembly());
14            stream.Position = 0;
15            NhibernateMappingLog();
16            configuration.AddInputStream(stream);
17            stream.Close();
18            SessionFactory = configuration.BuildSessionFactory();
19        }
20        return SessionFactory.OpenSession();
21    }
22    public static void NhibernateMappingLog()
23    {
24        String Nhibernate_mapping_log = JournaligaMVC.Properties.Settings.Default.Nhibernate_mapping_log.ToString();
25        System.IO.FileStream file = new System.IO.FileStream(Nhibernate_mapping_log, System.IO.FileMode.Create);
26        stream.WriteTo(file);
27        file.Close();
28    }
29 }
```

---

### 10.3.6. SQL

---

```
1 alter table personpage
2     add ( version integer );
3 commit;
4 /
```

---

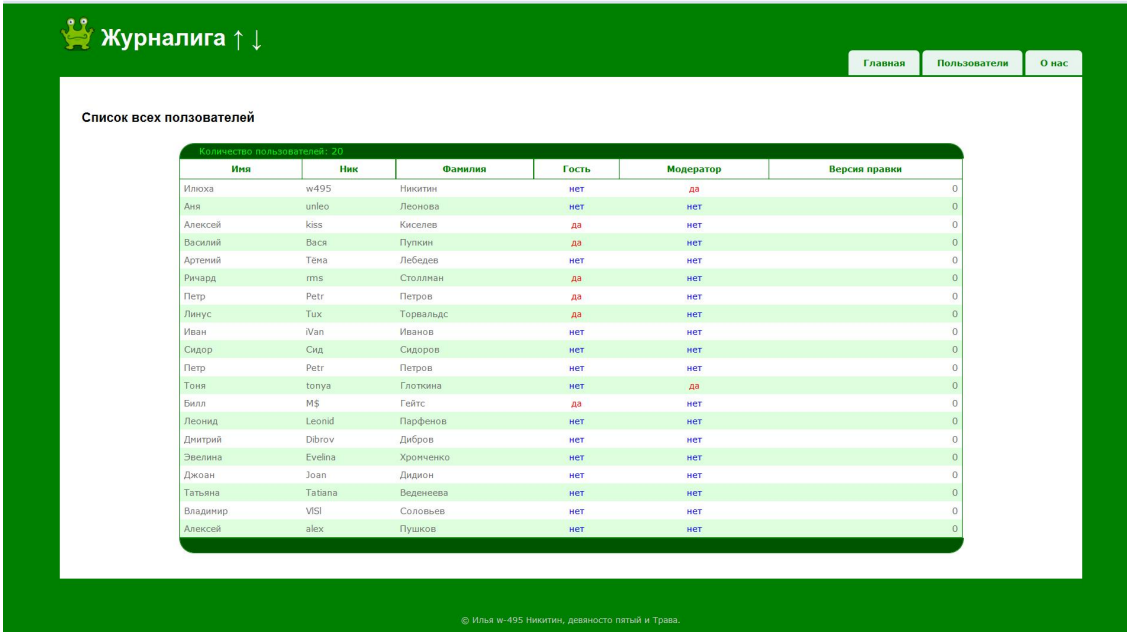
## 10.4. Возникшие трудности

Самым сложным из всей работы была настройка Nhibernate и попытка связать его с Oracle. На это ушло больше всего времени. Результат настройки можно увидеть в листингах выше.

## 11. Пример работы программы

Приведем скриншоты.

### 11.1. Список пользователей

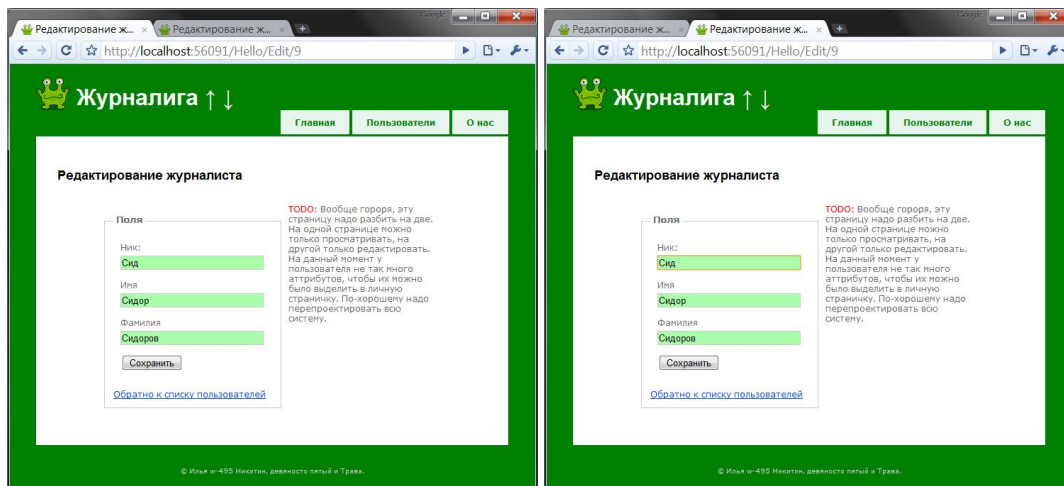


The screenshot shows a web application interface with a green header. The header contains a logo and the text 'Журналига ↑ ↓'. On the right side of the header are three navigation buttons: 'Главная', 'Пользователи', and 'О нас'. The main content area is titled 'Список всех пользователей' and displays a table of users. Above the table, it says 'Количество пользователей: 20'. The table has six columns: 'Имя', 'Ник', 'Фамилия', 'Гость', 'Модератор', and 'Версия правки'. The table lists 20 users with alternating light green and white rows. At the bottom of the page, there is a small copyright notice: '© Илья w-495 Никитин, деваносто пятый и Травя.'

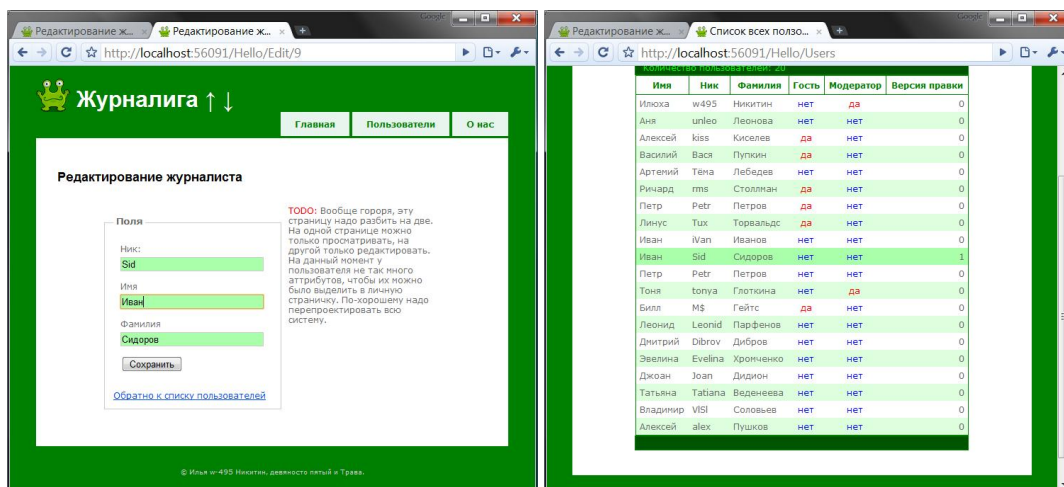
Имя	Ник	Фамилия	Гость	Модератор	Версия правки
Илюха	w495	Никитин	нет	да	0
Аня	unleo	Леонова	нет	нет	0
Алексей	kiss	Киселев	да	нет	0
Василий	Вася	Пупкин	да	нет	0
Артений	Tena	Лебедев	нет	нет	0
Ричард	rms	Столлман	да	нет	0
Петр	Petr	Петров	да	нет	0
Линус	Tux	Торвальдс	да	нет	0
Иван	Ivan	Иванов	нет	нет	0
Сидор	Сид	Сидоров	нет	нет	0
Петр	Petr	Петров	нет	нет	0
Тоня	tonya	Глоткина	нет	да	0
Билл	M\$	Гейтс	да	нет	0
Леонид	Leonid	Парфенов	нет	нет	0
Дмитрий	Dibrov	Дибров	нет	нет	0
Эвелина	Evelina	Хронченко	нет	нет	0
Джоан	Joan	Дидион	нет	нет	0
Татьяна	Tatiana	Веденеева	нет	нет	0
Владимир	VISI	Соловьев	нет	нет	0
Алексей	alex	Пушков	нет	нет	0

## 11.2. Оптимистическая блокировка

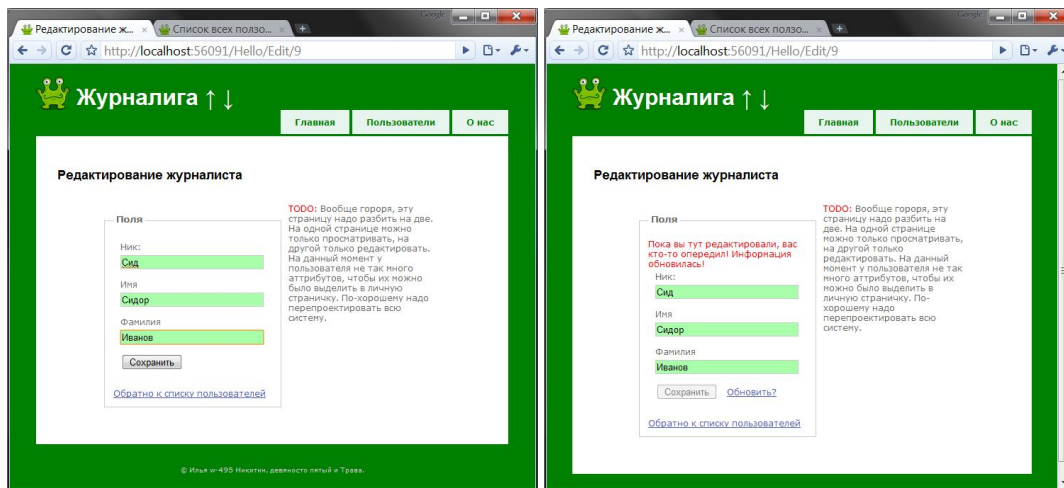
Рассмотрим работу оптимистической блокировки. Два пользователя одновременно начали редактировать одну и ту же запись:



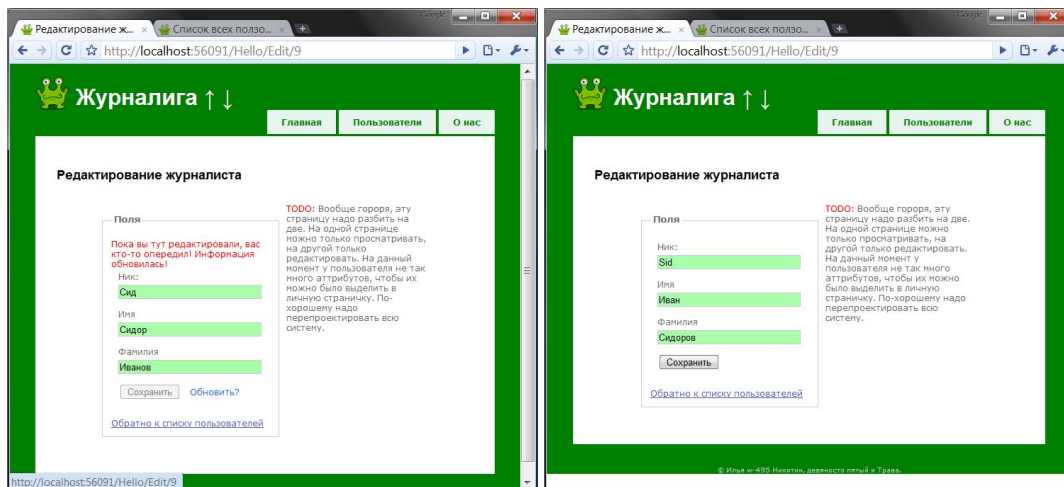
Второй пользователь отредактировал запись и сохранил изменения:



Первый пользователь отредактировал запись, и попытался сохранить. Но он работал с уже устаревшими данными. И система не дала ему сохранить свои изменения.



После нажатия на ссылку «обновить» данные первого пользователя обновились до актуальной версии.



## 12. Выводы

В процессе выполнения лабораторной работы мы познакомились технологией Nhibernate, а так же рассмотрели MVC-решение от Microsoft ASP.NET. Мы раньше сталкивались с паттерном MVC в Ruby on Rails и Django (MVT), однако все равно было интересно изучить его применение к ASP.NET-приложениям. По нашему предварительному мнению ASP.NET MVC очень похожа на Ruby on Rails, но показалась нам чуть менее простой, но более удобной. Что касается технологии Nhibernate, технология не сильно приспособлена для .NET приложений. Видно, что hibernate создавался для Java и портирован на .Net не очень удачно. Безусловно, использование Nhibernate проще и прозрачнее, чем написание того-же функционала на ADO.NET. Кроме того мы на практике столкнулись с блокировками. Работа с ними требует творческого подхода, и понимания цели своего приложения. Работа оказалась увлекательной и трудной.