Белгородский Государственный Технологический Университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

## Лабораторная работа №4 по теме: «Разработка веб-приложения на PHP и ASP.NET»

**Выполнил:**  
студент группы ПВ-41  
Адаменко И. И.

**Проверил:**  
старший преподаватель  
Гарибов А. И.

Белгород  
2015

**Цель работы:** создать БД из 3-4 связных таблиц и два веб-приложения на PHP и ASP.NET, которые будут работать с этой базой и добавлять, удалять и изменять записи в этих таблицах.

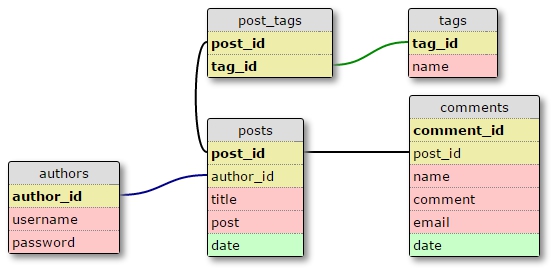
# Предметная область

Для этой лабораторной работы на PHP и ASP.NET я выбрал блог в качестве предметной области. Таким образом, веб-приложения должны позволять добавлять, удалять и изменять записи в нём, а также оставлять комментарии к каждой записи.

# База данных

В качестве СУБД для выполнения этой лабораторной работы была выбрана MySQL, поскольку PHP имеет встроенную поддержку этой СУБД, а для ASP.NET есть специальный модуль.

Общая структура базы данных:



Для работы с MySQL на PHP использовались следующие функции:

* mysqli\_connect — устанавливает новое соединение с сервером MySQL;
* mysqli\_error — возвращает строку с описанием последней ошибки;
* mysqli\_select\_db — устанавливает базу данных для выполняемых запросов;
* mysqli\_real\_escape\_string — экранирует специальные символы в строке для использования в SQL выражении, используя текущий набор символов соединения;
* mysqli\_query — выполняет запрос к базе данных;
* mysqli\_num\_rows — получает число рядов в результирующей выборке;
* mysqli\_affected\_rows — получает число строк, затронутых предыдущей операцией MySQL;
* mysqli\_set\_charset — задает набор символов по умолчанию;
* mysqli\_fetch\_assoc — извлекает результирующий ряд в виде ассоциативного массива;
* mysqli\_insert\_id — возвращает автоматически генерируемый ID, используя последний запрос;
* mysqli\_close — закрывает ранее открытое соединение с базой данных.

Пример запроса к MySQL на PHP:

1. $result = $mysqli->query("SELECT Name FROM City LIMIT 10")

Для работы с MySQL на ASP.NET использовались следующие методы, свойства и классы:

* MySqlConnection — класс для создания и хранения соединения с сервером MySQL;
  + Open — метод для открытия соединения;
  + Close — метод для закрытия соединения;
* MySqlCommand — класс для построения и выполнения MySQL-запросов;
  + ExecuteNonQuery — метод, выполняющий инструкцию для установленного соединения и возвращающий количество задействованных в инструкции строк;
  + ExecuteReader — метод, выполняющий инструкцию для установленного соединения и возвращающий объект MySqlDataReader;
  + LastInsertedId — свойство, хранящее в себе ID последнего вставленного в БД объекта;
* MySqlDataReader — класс для чтения данных из БД;
  + Read — метод для чтения данных из потока;
  + Close — метод для завершения потока данных;
* MySqlException — класс для работы с исключениями;
  + Message — свойство, хранящее в себе сообщение об ошибке.

Пример запроса к MySQL на ASP.NET:

1. var res = (**new** MySqlCommand("COMMIT", **this**.databaseLink)).ExecuteNonQuery();

# PHP

Веб-приложение на PHP в общем случае состоит из включённых друг в друга файлов, которые вместе генерируют разметку, необходимую пользователю, и выполняют операции над базой данных.

В этой лабораторной работе вся логика содержится в файлах для работы с БД, в которые включены:

* перед началом работы с БД:
  + blob.php — файл со вспомогательными функциями;
  + config.php — файл конфигураций;
  + db.php — файл с реализацией класса DB, упрощающего работу с MySQL;
* после работы с БД:
  + один из файлов шаблона, лежащих в поддиректории templs.

Исходный код класса DB, а также примеры работы с шаблонами и БД представлены в разделе «Исходный код».

# ASP.NET

Веб-приложение на ASP.NET состоит из набора страниц с компонентами, к каждой из которых прикреплён вспомогательный код, который выполняет необходимую этой странице логику (в нашем случае — работает с БД).

В этой лабораторной работе логика ASP.NET сохранена. Таким образом, создана одна мастер-страница, на которой расположено несколько включаемых областей, каждая из которых специфицируется уже на конкретной странице.

Помимо этого, в этой версии приложения также реализован вспомогательный класс DB для работы с БД. Исходный код этого класса, а также примеры работы с шаблонами и БД представлены в разделе «Исходный код».

# Скриншоты



Рисунок 1. Главная страница без постов



Рисунок 2. Главная страница с постами

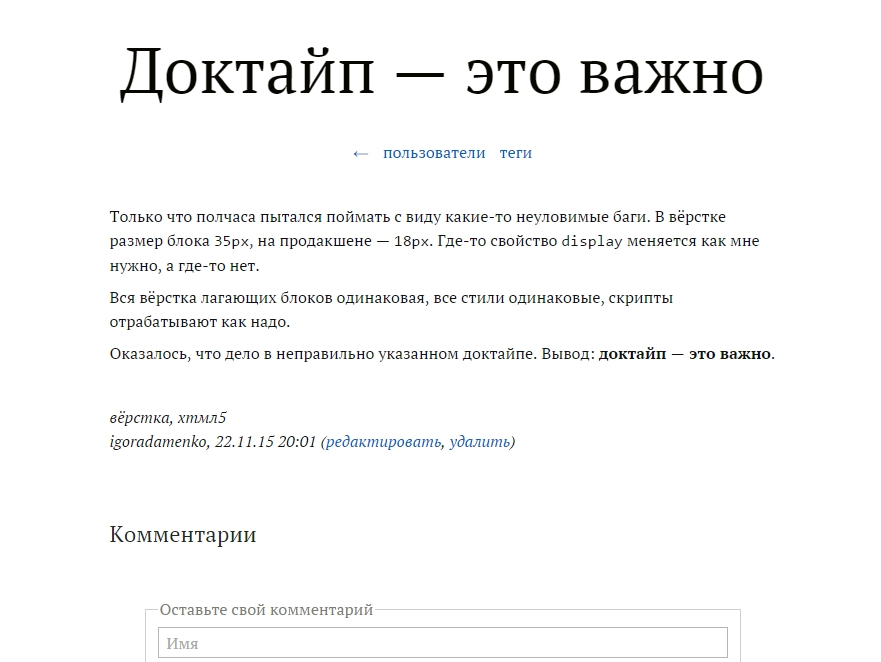


Рисунок 3. Страница поста

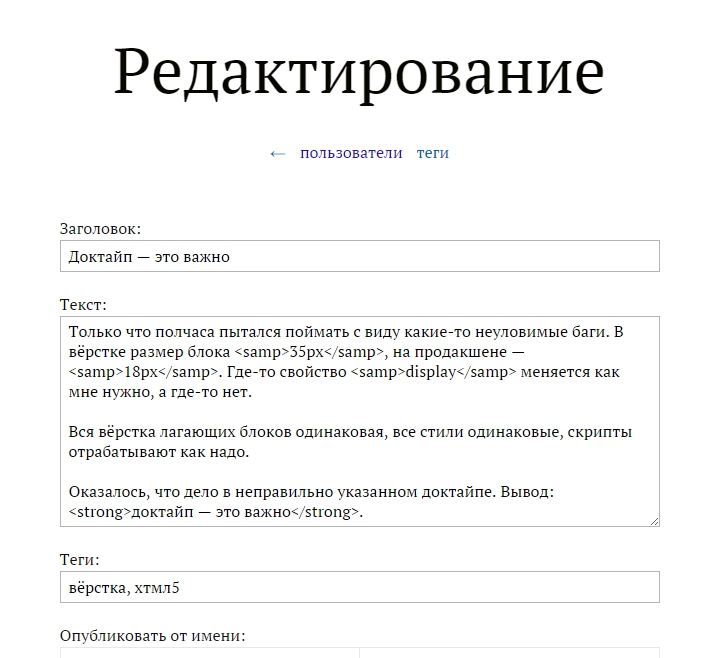


Рисунок 4. Страница редактирования поста

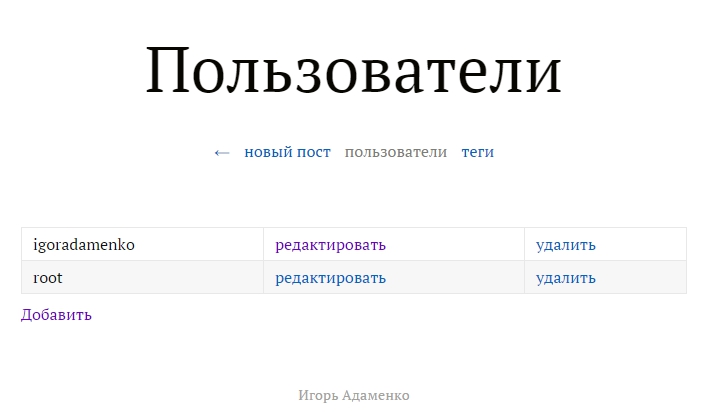


Рисунок 5. Страница со списком пользователей

# Исходные коды

## PHP

Реализация класса для работы с БД:

1. <?
2. /\* DB \*/
3. **class** DB {
5. **public**  $lastError;
6. **public**  $lastQuery;
7. **public**  $result;
8. **public**  $records;
9. **public**  $affected;
10. **public**  $rawResults;
11. **public**  $arrayedResult;
13. **private** $hostname;
14. **private** $username;
15. **private** $password;
16. **private** $database;
18. **private** $databaseLink;
20. **function** \_\_construct($database, $username, $password, $hostname = 'localhost') {
21. $this->database = $database;
22. $this->username = $username;
23. $this->password = $password;
24. $this->hostname = $hostname;
26. $this->Connect();
27. }
29. **function** \_\_destruct() {
30. $this->CloseConnection();
31. }
33. // Create connection
34. **private** **function** Connect() {
35. $this->CloseConnection();
37. $this->databaseLink = mysqli\_connect($this->hostname, $this->username,  
     $this->password);
39. **if** (!$this->databaseLink) {
40. $this->lastError = 'Can`t connect to server: ' .  
     mysqli\_error($this->databaseLink);
42. **return** false;
43. }
45. **if** (!$this->SelectDB()) {
46. $this->lastError = 'Can`t connect to database: ' .  
     mysqli\_error($this->databaseLink);
48. **return** false;
49. }
51. $this->SetCharset();
53. **return** true;
54. }
56. // Select database to use
57. **private** **function** SelectDB() {
58. **if** (!mysqli\_select\_db($this->databaseLink, $this->database)) {
59. $this->lastError = 'Can`t select database: ' .  
     mysqli\_error($this->databaseLink);
61. **return** false;
62. } **else** {
63. **return** true;
64. }
65. }
67. // Performs a 'mysqli\_real\_escape\_string' on the entire array/string
68. **private** **function** SecureData($data) {
69. **if** (is\_array($data)) {
70. $i = 0;
72. **foreach** ($data **as** $key => $val) {
73. **if** (!is\_array($data[$key])) {
74. $data[$key] = mysqli\_real\_escape\_string($this->databaseLink,  
     $data[$key]);
75. $i++;
76. }
77. }
78. } **else** {
79. $data = mysqli\_real\_escape\_string($this->databaseLink, $data);
80. }
82. **return** $data;
83. }
85. // Executes MySQL query
86. **public** **function** Execute($query) {
87. $this->lastQuery = $query;
89. **if** ($this->result = mysqli\_query($this->databaseLink, $query)) {
90. **if** (gettype($this->result) === 'object') {
91. $this->records = @mysqli\_num\_rows($this->result);
92. } **else** {
93. $this->records = 0;
94. }
96. $this->affected = @mysqli\_affected\_rows($this->databaseLink);
98. **if** ($this->records > 0) {
99. $this->ArrayResults();
101. **return** $this->arrayedResult;
102. } **else** {
103. **return** true;
104. }
106. } **else** {
107. $this->lastError = mysqli\_error($this->databaseLink);
109. **return** false;
110. }
111. }
113. **public** **function** Commit() {
114. **return** mysqli\_query($this->databaseLink, "COMMIT");
115. }
117. **public** **function** Rollback() {
118. **return** mysqli\_query("ROLLBACK", $this->databaseLink);
119. }
121. **public** **function** SetCharset($charset = 'UTF8') {
122. **return** mysqli\_set\_charset($this->databaseLink, $this->SecureData($charset));
123. }
125. // Adds a record to the database based on the array key names
126. **public** **function** Insert($table, $vars) {
127. $vars = $this->SecureData($vars);
129. $query = "INSERT INTO `{$table}` SET ";
131. **foreach** ($vars **as** $key => $value) {
132. $query .= "`{$key}` = '{$value}', ";
133. }
135. $query = trim($query, ', ');
137. **return** $this->Execute($query);
138. }
140. // Deletes a record from the database
141. **public** **function** Delete($table, $where = '', $limit = '') {
142. $query = "DELETE FROM `{$table}` WHERE ";
143. **if** (is\_array($where) && $where != '') {
144. $where = $this->SecureData($where);
146. **foreach** ($where **as** $key => $value) {
147. $query .= "`{$key}` = '{$value}' AND ";
148. }
150. $query = substr($query, 0, -5);
151. }
153. **if** ($limit != '') {
154. $query .= ' LIMIT ' . $limit;
155. }
157. **return** $this->Execute($query);
158. }
160. // Gets a single row from $from where $where is true
161. **public** **function** Select($from, $where = '', $orderBy = '', $limit = '', $cols = '\*',  
      $operand = 'AND') {
162. **if** (trim($from) == '') {
163. **return** false;
164. }
166. $query = "SELECT {$cols} FROM `{$from}` WHERE ";
168. **if** (is\_array($where) && $where != '') {
169. $where = $this->SecureData($where);
171. **foreach** ($where **as** $key => $value) {
172. $query .= "`{$key}` = '{$value}' {$operand} ";
173. }
175. $query = substr($query, 0, -(strlen($operand)+2));
177. } **else** {
178. $query = substr($query, 0, -6);
179. }
181. **if** ($orderBy != '') {
182. $query .= ' ORDER BY ' . $orderBy;
183. }
185. **if** ($limit != '') {
186. $query .= ' LIMIT ' . $limit;
187. }
189. $result = $this->Execute($query);
191. **if** (is\_array($result)) **return** $result;
193. **return** **array**();
194. }
196. // Updates a record in the database based on WHERE
197. **public** **function** Update($table, $set, $where) {
198. **if** (trim($table) == '' || !is\_array($set) || !is\_array($where)) {
199. **return** false;
200. }
202. $set    = $this->SecureData($set);
203. $where  = $this->SecureData($where);
205. $query = "UPDATE `{$table}` SET ";
207. **foreach** ($set **as** $key => $value) {
208. $query .= "`{$key}` = '{$value}', ";
209. }
211. $query = substr($query, 0, -2);
213. $query .= ' WHERE ';
215. **foreach** ($where **as** $key => $value) {
216. $query .= "`{$key}` = '{$value}' AND ";
217. }
219. $query = substr($query, 0, -5);
221. **return** $this->Execute($query);
222. }
224. // 'Arrays' a single result
225. **public** **function** ArrayResult() {
226. $this->arrayedResult = mysqli\_fetch\_assoc($this->result) **or**  
      **die** (mysqli\_error($this->databaseLink));
227. **return** $this->arrayedResult;
228. }
230. // 'Arrays' multiple result
231. **public** **function** ArrayResults() {
232. **if** ($this->records == 1) {
233. **return** $this->ArrayResult();
234. }
236. $this->arrayedResult = **array**();
238. **while** ($data = mysqli\_fetch\_assoc($this->result)) {
239. $this->arrayedResult[] = $data;
240. }
242. **return** $this->arrayedResult;
243. }
245. // 'Arrays' multiple results with a key
246. **public** **function** ArrayResultsWithKey($key = 'id') {
247. **if** (isset($this->arrayedResult)) {
248. unset($this->arrayedResult);
249. }
251. $this->arrayedResult = **array**();
253. **while**($row = mysqli\_fetch\_assoc($this->result)) {
254. **foreach** ($row **as** $theKey => $theValue) {
255. $this->arrayedResult[$row[$key]][$theKey] = $theValue;
256. }
257. }
259. **return** $this->arrayedResult;
260. }
262. // Returns last insert ID
263. **public** **function** GetLastInsertID() {
264. **return** mysqli\_insert\_id($this->databaseLink);
265. }
267. // Returns number of rows
268. **public** **function** CountRows($from, $where = '') {
269. $result = $this->Select($from, $where, '', '', 'count(\*)');
271. **return** $result["count(\*)"];
272. }
274. // Closes the connections
275. **public** **function** CloseConnection() {
276. **if** ($this->databaseLink) {
278. // you never know, man :3
279. $this->Commit();
281. mysqli\_close($this->databaseLink);
282. }
283. }
284. }
285. ?>

Реализация главной страницы блога:

1. <?
2. **require\_once**('blob.php');
3. **require\_once**('db.php');
4. **require\_once**('config.php');
6. $db = **new** DB(
7. $config['dbname'],
8. $config['dbuser'],
9. $config['dbpassword'],
10. $config['dbhost']
11. );
13. **function** getPost($rec, $authors) {
14. **return** **array**(
15. 'link' => $config['index'] . 'post?id=' . $rec['post\_id'],
16. 'title' => $rec['title'],
17. 'author' => $authors[$rec['author\_id']],
18. 'excerpt' => nl2br(strip\_tags(mb\_substr($rec['post'], 0, 200, 'UTF-8')) . '..'),
19. 'date' => date("d.m.y H:i", strtotime($rec['date']))
20. );
21. }
23. $posts = **array**();
24. $records = $db->Select('posts', false, 'post\_id DESC', '10');
26. $author\_ids = **array**();
28. **if** (is\_array($records[0])) {
29. $author\_ids = array\_unique(array\_map(**function**($o) {
30. **return** $o['author\_id'];
31. }, $records));
32. } **elseif** ($records) {
33. $author\_ids[] = $records['author\_id'];
34. }
36. $authors = **array**();
38. **foreach** ($author\_ids **as** $author\_id) {
39. $rec = $db->Select('authors', **array**(
40. 'author\_id' => $author\_id
41. ));
43. $authors[$author\_id] = $rec['username'];
44. }
46. **if** (is\_array($records[0])) {
47. **foreach** ($records **as** $rec) {
48. $posts[] = getPost($rec, $authors);
49. }
50. } **elseif**($records) {
51. $posts[] = getPost($records, $authors);
52. }
54. **require\_once**('templs/index.php');
55. ?>

Реализация шаблона главной страницы:

1. <!DOCTYPE html>
2. <html lang="ru">
3. <head>
4. <meta charset="UTF-8">
5. <meta name="viewport" content="width=device-width, initial-scale=1.0">
6. <title>Блог</title>
7. <link rel="stylesheet" href="awsm.min.css">
8. </head>
9. <body>
10. <header>
11. <h1>Блог</h1>
12. <nav>
13. <ul>
14. <li><a href="<?=$config['index']?>new">новый пост</a></li>
15. <li><a href="<?=$config['index']?>users">пользователи</a></li>
16. <li><a href="<?=$config['index']?>tags">теги</a></li>
17. </ul>
18. </nav>
19. </header>
20. <main>
22. <? **if** ($posts): ?>
24. <? **foreach** ($posts **as** $post): ?>
26. <section>
27. <h2><a href="<?=$post['link']?>"><?=$post['title']?></a></h2>
28. <aside>
29. <?=$post['author']?>
30. <br>
31. <?=$post['date']?>
32. </aside>
33. <p><?=$post['excerpt']?></p>
34. </section>
36. <? **endforeach** ?>
38. <? **else**: ?>
40. <p><em>Увы, но постов пока нет.</em></p>
42. <? **endif** ?>
44. </main>
45. <footer>
46. <p>Игорь Адаменко</p>
47. </footer>
48. </body>
49. </html>

## ASP.NET

Реализация класса для работы с БД:

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** MySql.Data.MySqlClient;
5. **using** System.Data;
7. **namespace** WebApplication
8. {
9. **public** **class** DB
10. {
11. #region public fields
12. **public** **string** lastError;
13. **public** **string** lastQuery;
14. **public** **long** lastInsertedID;
15. **public** **int** affected;
16. #endregion
18. #region private fields
19. **private** **string** hostname;
20. **private** **string** username;
21. **private** **string** password;
22. **private** **string** database;
23. **private** MySqlConnection databaseLink = **null**;
24. #endregion
26. **public** DB(**string** database, **string** username, **string** password,  
     **string** hostname = "localhost")
27. {
28. **this**.database = database;
29. **this**.username = username;
30. **this**.password = password;
31. **this**.hostname = hostname;
33. Connect();
34. }
36. ~DB()
37. {
38. CloseConnection();
39. }
41. **private** **bool** Connect()
42. {
43. CloseConnection();
44. **string** conString =  "SERVER=" + **this**.hostname + ";" +
45. "DATABASE=" + **this**.database + ";" +
46. "UID=" + **this**.username + ";" +
47. "PASSWORD=" + **this**.password + ";";
48. conString = SetCharset(conString);
50. **this**.databaseLink = **new** MySqlConnection(conString);
51. **this**.databaseLink.Open();
53. **if** (**this**.databaseLink == **null**)
54. {
55. **this**.lastError = "Can't connect to server";
57. **return** **false**;
58. }
60. **return** **true**;
61. }
63. **private** **void** CloseConnection()
64. {
65. **if** (**this**.databaseLink != **null**)
66. {
67. Commit();
69. **this**.databaseLink.Close();
70. }
71. }
73. **private** **bool** Commit()
74. {
75. **return** (**new** MySqlCommand("COMMIT", **this**.databaseLink)).ExecuteNonQuery() > 0;
76. }
78. **private** **bool** Rollback()
79. {
80. **return** (**new** MySqlCommand("ROLLBACK", **this**.databaseLink)).ExecuteNonQuery() > 0;
81. }
83. **private** **string** SetCharset(**string** conString, **string** charset = "utf8")
84. {
85. **return** conString + "CharSet=" + charset + ";";
86. }
88. **private** **string** MySQLRealEscapeString(**string** str)
89. {
90. **return** System.Text.RegularExpressions.Regex.Replace(str,  
     @"[\000\010\011\012\015\032\042\047\134\140]", "\\$0");
91. }
93. **private** **string** SecureData(**string** data)
94. {
95. **return** MySQLRealEscapeString(data);
96. }
98. **private** Dictionary<**string**, **string**> SecureData(Dictionary<**string**, **string**> data)
99. {
100. **return** data.ToDictionary(x => x.Key, x => MySQLRealEscapeString(x.Value));
101. }
103. **private** **bool** ExecuteNonQuery(**string** query)
104. {
105. **this**.lastQuery = query;
106. **this**.affected = 0;
108. **try**
109. {
110. **this**.affected = (**new** MySqlCommand(query,  
      **this**.databaseLink)).ExecuteNonQuery();
111. }
112. **catch** (Exception e)
113. {
114. **this**.lastError = e.Message;
115. }
117. **return** **this**.affected > 0;
118. }
120. **private** **bool** ExecuteScalar(**string** query)
121. {
122. **this**.lastQuery = query;
124. **long** id = 0;
126. **try** {
127. var cmd = (**new** MySqlCommand(query, **this**.databaseLink));
128. **this**.affected = cmd.ExecuteNonQuery();
130. id = cmd.LastInsertedId;
131. } **catch** (MySqlException e)
132. {
133. **this**.lastError = e.Message;
134. }
136. **this**.lastInsertedID = id;
138. **return** id > 0;
139. }
141. **private** DataTable ExecuteQuery(**string** query)
142. {
143. **this**.lastQuery = query;
144. **try**
145. {
146. var reader = (**new** MySqlCommand(query,  
      **this**.databaseLink)).ExecuteReader();
148. var dt = **new** DataTable();
149. dt.Load(reader);
150. reader.Close();
152. **return** dt;
153. }
154. **catch** (Exception e)
155. {
156. **this**.lastError = e.Message;
158. **return** **new** DataTable();
159. }
160. }
162. **public** **bool** Insert(**string** table, Dictionary<**string**, **string**> vars)
163. {
164. vars = SecureData(vars);
166. var query = String.Format("INSERT INTO `{0}` SET ", table);
167. **foreach** (var item **in** vars)
168. {
169. query += String.Format("`{0}` = '{1}', ", item.Key, item.Value);
170. }
172. **return** ExecuteScalar(query.Substring(0, query.Length - 2));
173. }
175. **public** **bool** Delete(**string** table, Dictionary<**string**, **string**> where = **null**,  
      **string** limit = **null**)
176. {
177. var query = String.Format("DELETE FROM `{0}` WHERE", table);
179. **if** (where != **null**)
180. {
181. where = SecureData(where);
183. **foreach** (var item **in** where)
184. {
185. query += String.Format("`{0}` = '{1}' AND ", item.Key, item.Value);
186. }
188. query = query.Substring(0, query.Length - 5);
189. }
191. **if** (limit != **null**)
192. {
193. query += " LIMIT " + limit;
194. }
196. **return** ExecuteNonQuery(query);
197. }
199. **public** DataTable Select(**string** from, Dictionary<**string**, **string**> where = **null**,
200. **string** orderBy = **null**, **string** limit = **null**,  
       **string** cols = **null**,
201. **string** operand = "AND")
202. {
203. **if** (from.Trim().Length == 0)
204. {
205. **return** **null**;
206. }
208. var numCols = (cols == **null**) ? "\*" : cols;
209. var query = String.Format("SELECT {0} FROM `{1}` WHERE ", numCols, from);
211. **if** (where != **null**)
212. {
213. where = SecureData(where);
215. **foreach** (var item **in** where)
216. {
217. query += String.Format("`{0}` = '{1}' {2} ", item.Key, item.Value,  
      operand);
218. }
220. query = query.Substring(0, query.Length - (operand.Length + 2));
221. }
222. **else**
223. {
224. query = query.Substring(0, query.Length - 7);
225. }
227. **if** (orderBy != **null**)
228. {
229. query += String.Format(" ORDER BY {0}", orderBy);
230. }
232. **if** (limit != **null**)
233. {
234. query += String.Format(" LIMIT {0}", limit);
235. }
237. **return** ExecuteQuery(query);
238. }
240. **public** **bool** Update(**string** table, Dictionary<**string**, **string**> **set**,  
      Dictionary<**string**, **string**> where)
241. {
242. **if** (table.Length == 0 || **set** == **null** || where == **null**)
243. {
244. **return** **false**;
245. }
247. **set** = SecureData(**set**);
248. where = SecureData(where);
250. var query = String.Format("UPDATE `{0}` SET ", table);
252. **foreach** (var item **in** **set**)
253. {
254. query += String.Format("`{0}` = '{1}', ", item.Key, item.Value);
255. }
257. query = String.Format("{0} WHERE ", query.Substring(0, query.Length - 2));
259. **foreach** (var item **in** where)
260. {
261. query += String.Format("`{0}` = '{1}' AND ", item.Key, item.Value);
262. }
264. query = query.Substring(0, query.Length - 5);
266. **return** ExecuteNonQuery(query);
267. }
269. **public** DataTable CountRows(**string** from, Dictionary<**string**, **string**> where = **null**)
270. {
271. **if** (from.Trim().Length == 0)
272. {
273. **return** **null**;
274. }
276. var result = Select(from, where, **null**, **null**, "count(\*)");
278. **return** result;
279. }
280. }
281. }

Реализация главной страницы блога:

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Data;
5. **namespace** WebApplication
6. {
7. **public** partial **class** \_Default : System.Web.UI.Page
8. {
9. **protected** List<Dictionary<**string**, **string**>> posts;
11. **protected** **void** Page\_Load(**object** sender, EventArgs e)
12. {
13. var records = Global.DB.Select("posts", **null**, "post\_id DESC", "10");
14. var author\_ids = **new** List<**string**>();
16. **if** (records.Rows.Count > 0)
17. {
18. **foreach** (DataRow row **in** records.Rows)
19. {
20. **if** (! author\_ids.Contains(row["author\_id"].ToString()))
21. {
22. author\_ids.Add(row["author\_id"].ToString());
23. }
24. }
25. }
27. var authors = **new** Dictionary<**string**, **string**>();
29. **foreach** (var author\_id **in** author\_ids)
30. {
31. var rec = Global.DB.Select("authors", **new** Dictionary<**string**, **string**>()
32. {
33. { "author\_id", author\_id }
34. });
36. authors.Add(author\_id, rec.Rows[0]["username"].ToString());
37. }
39. posts = **new** List<Dictionary<**string**, **string**>>();
41. **foreach** (DataRow rec **in** records.Rows)
42. {
43. posts.Add(GetPost(rec, authors));
44. }
45. }
47. **private** Dictionary<**string**, **string**> GetPost(DataRow rec,  
     Dictionary<**string**, **string**> authors)
48. {
49. var post = rec["post"].ToString();
50. **if** (post.Length > 200)
51. {
52. post.Substring(0, 200);
53. }
54. post += "..";
55. post = post.Replace("\r\n", "<br>\r\n");
57. **return** **new** Dictionary<**string**, **string**> {
58. { "link", "/Post.aspx?id=" + rec["post\_id"].ToString() },
59. { "title", rec["title"].ToString() },
60. { "author", authors[rec["author\_id"].ToString()] },
61. { "excerpt", post },
62. {   
     "date",  
     (DateTime.Parse(rec["date"].ToString())).ToString("dd.MM.yy HH:mm")  
     }
63. };
64. }
65. }
66. }

Реализация шаблона главной страницы:

1. **<**%@ Page Title="Блог" Language="C#" MasterPageFile="~/Site.master"
2. CodeBehind="Default.aspx.cs" Inherits="WebApplication.\_Default" %**>**
4. **<asp:Content** ID="HeaderContent" runat="server" ContentPlaceHolderID="HeaderContent"**>**
5. **<h1>**Блог**</h1>**
6. **<nav>**
7. **<ul>**
8. **<li><a** href="~/New.aspx" runat="server"**>**новый пост**</a></li>**
9. **<li><a** href="~/Users.aspx" runat="server"**>**пользователи**</a></li>**
10. **<li><a** href="~/Tags.aspx" runat="server"**>**теги**</a></li>**
11. **</ul>**
12. **</nav>**
13. **</asp:Content>**
15. **<asp:Content** ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent"**>**
16. **<**% if (posts.Count **>** 0)
17. { %**>**
18. **<**% foreach (var post in posts)
19. { %**>**
20. **<section>**
21. **<h2><a** href="**<**%=post["link"]%>"**><**%=post["title"]%**></a></h2>**
22. **<aside>**
23. **<**%=post["author"]%**>**
24. **<br>**
25. **<**%=post["date"]%**>**
26. **</aside>**
27. **<p><**%=post["excerpt"]%**></p>**
28. **</section>**
29. **<**% } %**>**
30. **<**% }
31. else
32. { %**>**
33. **<p><em>**Увы, но постов пока нет.**</em></p>**
34. **<**% } %**>**
35. **</asp:Content>**