

Частное учреждение образования
«Колледж бизнеса и права»

УТВЕРЖДАЮ
Ведущий методист
колледжа
_____ Е.В. Паскал
«___» _____ 2022

Специальность: «Программное обеспечение информационных технологий»	Учебная дисциплина: «Базы данных и системы управления базами данных»
--	--

ЛАБОРАТОРНАЯ РАБОТА № 34

Инструкционно-технологическая карта

Тема: Создание веб-форм для отображения данных из базы данных.

Цель работы: научиться создавать веб-формы для отображения данных из базы данных.

Время выполнения: 2 часа

Содержание работы

1. Порядок выполнения работы
2. Пример выполнения работы
3. Контрольные вопросы
4. Литература

1. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Рассмотреть способ создания веб-формы для отображения данных из базы данных, описанные в разделе «Пример выполнения работы» настоящей инструкционно-технологической карты.

2. Выполнить все пункты примера, представленные в настоящей инструкционно-технологической карте, и проанализировать полученные результаты.

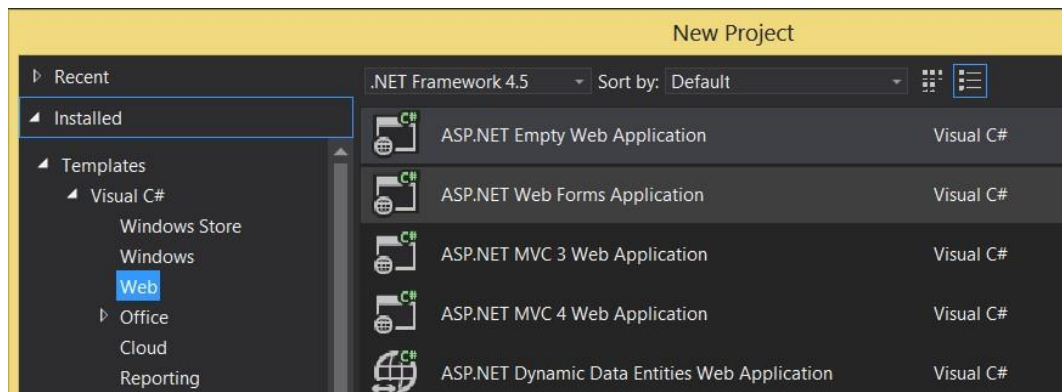
3. Получить у преподавателя индивидуальное задание и выполнить лабораторную работу.

4. Ответить на контрольные вопросы.

2. ПРИМЕР ВЫПОЛНЕНИЯ РАБОТЫ

Создание базовой формы записи

Создайте страницу с именем AddMovie.cshtml.



Замените содержимое файла следующей разметкой. Перезапись всего; Вы добавите блок кода в начало в ближайшее время.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>Add a Movie</title>
</head>
<body>
<h1>Add a Movie</h1>
<form method="post">
  <fieldset>
    <legend>Movie Information</legend>
    <p><label for="title">Title:</label>
      <input type="text" name="title" value="@Request.Form["title"]" />
    </p>

    <p><label for="genre">Genre:</label>
      <input type="text" name="genre" value="@Request.Form["genre"]" />
    </p>

    <p><label for="year">Year:</label>
      <input type="text" name="year" value="@Request.Form["year"]" />
    </p>

    <p><input type="submit" name="buttonSubmit" value="Add Movie" /></p>
  </fieldset>
</form>
</body>
</html>
```

В этом примере показан типичный HTML-код для создания формы. Он использует `<input>` элементы для текстовых полей и кнопки отправки. Заголовки

для текстовых полей создаются с помощью стандартных `<label>` элементов. `<legend>` Элементы `<fieldset>` помещают в форму хорошую коробку.

Обратите внимание, что `<form>` на этой странице элемент используется `post` в качестве значения атрибута `method`. На этой странице вы внесете изменения — вы добавите новые записи базы данных. Таким образом, эта форма должна использовать `post` метод.

Обратите внимание, что каждое `name` текстовое поле имеет элемент (`title`, `genre`). `year` Как вы видели в предыдущем руководстве, эти имена важны, так как у вас должны быть эти имена, чтобы позже можно было получить входные данные пользователя. Можно использовать любые имена. Полезно использовать значимые имена, которые помогают запоминать данные, с которыми вы работаете.

Атрибут `value` каждого `<input>` элемента содержит немного кода Razor (например, `Request.Form["title"]`). Вы узнали версию этого трюка в предыдущем руководстве, чтобы сохранить значение, введенное в текстовое поле (если таковые имеются) после отправки формы.

Получение значений формы

Затем добавьте код, обрабатывающий форму. В структуре вы выполните следующие действия:

1. Проверьте, публикуется ли страница (была отправлена). Код должен выполняться только в том случае, если пользователи нажали кнопку, а не при первом запуске страницы.
2. Получите значения, введенные пользователем в текстовые поля. В этом случае, так как форма использует `POST` команду, вы получите значения формы из `Request.Form` коллекции.
3. Вставьте значения в качестве новой записи в таблицу базы данных `Movies`.

В верхней части файла добавьте следующий код:

```
@{
    var title = "";
    var genre = "";
    var year = "";

    if(IsPost){
        title = Request.Form["title"];
        genre = Request.Form["genre"];
        year = Request.Form["year"];
    }
}
```

Первые несколько строк создают переменные (`title`, `genre` и `year`) для хранения значений из текстовых полей. `if(IsPost)` Строка гарантирует, что переменные задаются только при нажатии кнопки "Добавить фильм", то есть при публикации формы.

Вы получите значение текстового поля с помощью выражения, например `Request.Form["name"]`, где имя — имя `<input>` элемента.

Имена переменных (`title`, `genre` `year`) являются произвольными. Как и имена, назначенные `<input>` элементам, вы можете вызывать их как угодно. (Имена переменных не должны соответствовать атрибутам `<input>` имен элементов в форме.) Но как и `<input>` в случае с элементами, рекомендуется использовать имена переменных, которые отражают содержащиеся в них данные. При написании кода согласованные имена упрощают запоминать данные, с которыми вы работаете.

Добавление данных в базу данных

В только что добавленном блоке кода просто внутри закрывающей скобки `if (}` блока (не только внутри блока кода) добавьте следующий код (с#):

```
var db = Database.Open("WebPagesMovies");
var insertCommand = "INSERT INTO Movies (Title, Genre, Year) VALUES(@0,
@1, @2)";
db.Execute(insertCommand, title, genre, year);
```

Этот пример аналогичен коду, используемому в предыдущем руководстве для получения и отображения данных. Строка, которая начинается с `db =` открытия базы данных, как и раньше, и следующая строка определяет оператор SQL еще раз, как вы видели ранее. Однако на этот раз он определяет оператор `SQLInsert Into`. В следующем примере показан общий синтаксис инструкции `Insert Into` :

```
INSERT INTO table (column1, column2, column3, ...) VALUES (value1, value2,
value3, ...)
```

Другими словами, вы указываете таблицу для вставки, а затем перечисляете столбцы для вставки, а затем перечисляете значения для вставки. (Как отмечалось ранее, SQL не учитывает регистр, но некоторые люди прописывают ключевые слова, чтобы упростить чтение команды.)

Столбцы, в которые вы вставляете, уже перечислены в команде (`Title`, `Genre`, `Year`). Интересно, как получить значения из текстовых полей в `VALUES` часть команды. Вместо фактических значений отображаются `@0@1` и `@2`, которые, конечно, являются заполнителями. При выполнении команды (в строке `db.Execute`) передаются значения, полученные из текстовых полей.

Важно! Помните, что единственным способом включения данных, введенных пользователем в режиме "в сети", в инструкции SQL является использование заполнителей, как показано здесь (`VALUES(@0, @1, @2)`). Если вы объединяете данные пользователя в инструкцию SQL, вы открываете себя для атаки путем внедрения SQL.

Тем не менее внутри `if` блока добавьте следующую строку после `db.Execute` строки (css):

```
Response.Redirect("~/Movies");
```

После вставки нового фильма в базу данных эта строка перенаправляет вас (перенаправляет) на страницу "Фильмы" , чтобы увидеть только что введенный фильм. Оператор ~ означает "корень веб-сайта". (Оператор ~ работает только на ASP.NET страницах, а не в HTML в целом.)

Полный блок кода выглядит следующим образом (CSHTML):

```
@{
    var title = "";
    var genre = "";
    var year = "";

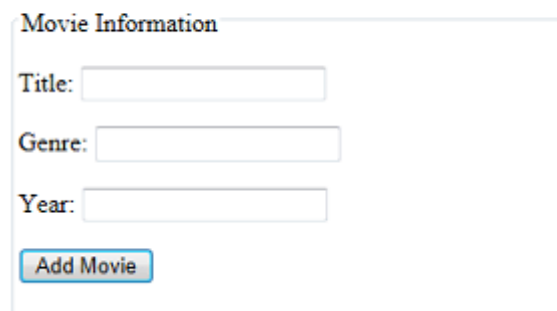
    if(IsPost){
        title = Request.Form["title"];
        genre = Request.Form["genre"];
        year = Request.Form["year"];

        var db = Database.Open("WebPagesMovies");
        var insertCommand = "INSERT INTO Movies (Title, Genre, Year) Values(@0, @1, @2)";
        db.Execute(insertCommand, title, genre, year);
        Response.Redirect("~/Movies");
    }
}
```

Тестирование команды insert

В представлении дерева файлов в WebMatrix щелкните правой кнопкой мыши страницу AddMovie.cshtml и выберите команду "Запустить в браузере".

Add a Movie



(Если вы в конечном итоге на другой странице в браузере, убедитесь, что URL-адрес <http://localhost:nnnnn/AddMovie>), где nnnnn — номер порта, который вы используете.)

Вы получили страницу ошибки? Если да, внимательно прочтите его и убедитесь, что код выглядит точно, что было указано ранее.

Введите фильм в форме, например "Гражданин Кейн", "Драма" и "1941". (Или что угодно.) Затем нажмите кнопку "Добавить фильм".

Если все пойдет хорошо, вы будете перенаправлены на страницу "Фильмы ". Убедитесь, что ваш новый фильм указан.

Movies

Genre to look for:

(Leave blank to list all movies.)

Movie title contains the following:

<u>Title</u>	<u>Genre</u>	<u>Year</u>
Citizen Kane	Drama	1941

Проверка введенного пользователем

Перейдите назад на страницу AddMovie или снова запустите ее. Введите другой фильм, но на этот раз введите только заголовок, например введите "Singin" в дожде. Затем нажмите кнопку "Добавить фильм".

Вы снова перейдете на страницу "Фильмы ". Вы можете найти новый фильм, но он неполный.

Movies

Genre to look for:

(Leave blank to list all movies.)

Movie title contains the following:

<u>Title</u>	<u>Genre</u>	<u>Year</u>
Singin' in the Rain		

1 2 3 4 >

При создании таблицы Movies вы явно указали, что ни одно из полей не может быть пустым. Здесь у вас есть форма записи для новых фильмов, и вы оставляете поля пустыми. Это ошибка.

В этом случае база данных не вызвала (или не вызвала) ошибку. Вы не указали жанр или год, поэтому код на странице AddMovie рассматривал эти значения как так называемые пустые строки. Когда команда SQL Insert Into выполнена, поля жанра и года не имели полезных данных в них, но они не были пустыми.

Очевидно, что вы не хотите, чтобы пользователи вводили в базу данных полупустые сведения о фильмах. Решение заключается в проверке входных данных пользователя. Изначально проверка будет просто убедиться, что пользователь ввел значение для всех полей (то есть, что ни один из них не содержит пустую строку).

Пустые строки

В программировании существует различие между различными понятиями "нет значения". Как правило, значение равно NULL, если оно никогда не было задано или инициализировано каким-либо образом. В отличие от этого, переменная, которая ожидает символьные данные (строки), может быть задана как пустая строка. В этом случае значение не равно NULL; Его просто явно задали в

строку символов, длина которой равна нулю. Эти два оператора показывают разницу (C#):

```
var firstName;    // Not set, so its value is null
var firstName = ""; // Explicitly set to an empty string -- not null
```

Это немного сложнее, чем это, но важно то, что null представляет своего рода неопределенное состояние.

Важно понимать, когда значение равно NULL и когда это просто пустая строка. В коде страницы AddMovie вы получите значения текстовых полей с помощью Request.Form["title"] и т. д. При первом запуске страницы (перед нажатием кнопки) значение Request.Form["title"] null. Но при отправке формы Request.Form["title"] получает значение текстового title поля. Это не очевидно, но пустое текстовое поле не равно NULL; В нем просто есть пустая строка. Поэтому, когда код выполняется в ответ на нажатие кнопки, Request.Form["title"] в ней есть пустая строка.

Почему это различие важно? При создании таблицы Movies вы явно указали, что ни одно из полей не может быть пустым. Но здесь у вас есть форма входа для новых фильмов, и вы оставляете поля пустыми. Вы бы разумно ожидали, что база данных будет жаловаться, когда вы попытаетесь сохранить новые фильмы, которые не имели значений для жанра или года. Но это точка, даже если оставить эти текстовые поля пустыми, значения не являются пустыми; Они пустые строки. В результате вы сможете сохранять новые фильмы в базе данных с пустыми столбцами, но не пустыми! — значения. Поэтому необходимо убедиться, что пользователи не будут отправлять пустую строку, которую можно сделать, проверяя входные данные пользователя.

Валидатор проверки

Веб-страницы ASP.NET включает в себя валидатор (вспомогательный Validation), который можно использовать, чтобы убедиться, что пользователи вводят данные, соответствующие вашим требованиям. Вспомогательный Validation компонент является одним из вспомогательных средств, встроенных в веб-страницы ASP.NET, поэтому вам не нужно устанавливать его как пакет с помощью NuGet, как вы установили валидатор Gravatar в предыдущем руководстве.

Чтобы проверить входные данные пользователя, выполните следующие действия.

Используйте код, чтобы указать, что необходимо требовать значения в текстовых полях на странице.

Поместите тест в код, чтобы сведения о фильме добавлялись в базу данных только в том случае, если все проверяется правильно.

Добавьте код в разметку для отображения сообщений об ошибках.

В блоке кода на странице AddMovie прямо вверху перед объявлениями переменных добавьте следующий код (c#):

```
Validation.RequireField("title", "You must enter a title");
Validation.RequireField("genre", "Genre is required");
Validation.RequireField("year", "You haven't entered a year");
```

Вы вызываете `Validation.RequireField` один раз для каждого поля (`<input>` элемента), в котором требуется запись. Вы также можете добавить пользовательское сообщение об ошибке для каждого вызова, как показано здесь.

Если возникла проблема, необходимо запретить вставку новых сведений о фильме в базу данных. В блоке `if(IsPost)` используйте `&&` (логическое И) для добавления другого условия, которое проверяет `Validation.IsValid()`. По завершении весь `if(IsPost)` блок выглядит следующим образом (с#):

```
if(IsPost && Validation.IsValid()){
    title = Request.Form["title"];
    genre = Request.Form["genre"];
    year = Request.Form["year"];

    var db = Database.Open("WebPagesMovies");
    var insertCommand = "INSERT INTO Movies (Title, Genre, Year) Values(@0,
@1, @2)";
    db.Execute(insertCommand, title, genre, year);
    Response.Redirect("~/Movies");
}
```

Если возникает ошибка проверки с любым из полей, зарегистрированных с помощью вспомогательного `Validation` элемента, `Validation.IsValid` метод возвращает значение `false`. В этом случае ни один из кодов в этом блоке не будет выполняться, поэтому в базу данных не будут вставляться недопустимые записи фильмов. И, конечно, вы не перенаправляетесь на страницу "Фильмы".

Полный блок кода, включая код проверки, теперь выглядит следующим образом (CSHTML):

```
@{
    Validation.RequireField("title", "You must enter a title");
    Validation.RequireField("genre", "Genre is required");
    Validation.RequireField("year", "You haven't entered a year");

    var title = "";
    var genre = "";
    var year = "";

    if(IsPost && Validation.IsValid()){
        title = Request.Form["title"];
        genre = Request.Form["genre"];
        year = Request.Form["year"];
```



```

var db = Database.Open("WebPagesMovies");
var insertCommand = "INSERT INTO Movies (Title, Genre, Year) Values(@0, @1, @2)";
db.Execute(insertCommand, title, genre, year);
Response.Redirect("~/Movies");
}
}

```

Отображение ошибок проверки

Последним шагом является отображение сообщений об ошибках. Вы можете отобразить отдельные сообщения для каждой ошибки проверки или отобразить сводку или оба.

Рядом с каждым `<input>` проверяющим элементом вызовите `Html.ValidationMessage` метод и передайте имя проверяемого `<input>` элемента. Вы поместите `Html.ValidationMessage` метод в нужное место, где будет отображаться сообщение об ошибке. При запуске страницы метод отрисовывает `` элемент, `Html.ValidationMessage` в котором будет отображаться ошибка проверки. (Если ошибка отсутствует, элемент отображается, `` но в нем нет текста.)

Измените разметку на странице так, чтобы она включает `Html.ValidationMessage` метод для каждого из трех `<input>` элементов на странице, как показано в следующем примере (CSHTML):

```

<p><label for="title">Title:</label>
  <input type="text" name="title" value="@Request.Form["title"]" />
  @Html.ValidationMessage("title")
</p>

<p><label for="genre">Genre:</label>
  <input type="text" name="genre" value="@Request.Form["genre"]" />
  @Html.ValidationMessage("genre")
</p>

<p><label for="year">Year:</label>
  <input type="text" name="year" value="@Request.Form["year"]" />
  @Html.ValidationMessage("year")
</p>

```

Чтобы узнать, как работает сводка, добавьте следующую разметку и код сразу после `<h1>Add a Movie</h1>` элемента на странице (CSHTML):

```
@Html.ValidationSummary()
```

По умолчанию `Html.ValidationSummary` метод отображает все сообщения проверки в списке (`` элемент, который находится внутри `<div>` элемента). Как и в случае с методом `Html.ValidationMessage`, разметка для сводки проверки всегда отображается; если ошибки отсутствуют, элементы списка не отображаются.

Сводка может быть альтернативным способом отображения сообщений проверки вместо использования `Html.ValidationMessage` метода для отображения каждой ошибки, относящейся к конкретному полю. Кроме того, можно использовать сводку и сведения. Кроме того, можно использовать `Html.ValidationSummary` этот метод для отображения универсальной ошибки, а затем использовать отдельные `Html.ValidationMessage` вызовы для отображения сведений.

Теперь полная страница выглядит следующим образом (CSHTML):

```
@{
    Validation.RequireField("title", "You must enter a title");
    Validation.RequireField("genre", "Genre is required");
    Validation.RequireField("year", "You haven't entered a year");

    var title = "";
    var genre = "";
    var year = "";

    if(IsPost && Validation.IsValid()){
        title = Request.Form["title"];
        genre = Request.Form["genre"];
        year = Request.Form["year"];

        var db = Database.Open("WebPagesMovies");
        var insertCommand = "INSERT INTO Movies (Title, Genre, Year) Values(@0, @1, @2)";
        db.Execute(insertCommand, title, genre, year);
        Response.Redirect("~/Movies");
    }
}

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Add a Movie</title>
</head>
<body>
    <h1>Add a Movie</h1>
    @Html.ValidationSummary()
```

```

<form method="post">
  <fieldset>
    <legend>Movie Information</legend>
    <p><label for="title">Title:</label>
      <input type="text" name="title" value="@Request.Form["title"]" />
      @Html.ValidationMessage("title")
    </p>

    <p><label for="genre">Genre:</label>
      <input type="text" name="genre" value="@Request.Form["genre"]" />
      @Html.ValidationMessage("genre")
    </p>


    <p><label for="year">Year:</label>
      <input type="text" name="year" value="@Request.Form["year"]" />
      @Html.ValidationMessage("year")
    </p>

    <p><input type="submit" name="buttonSubmit" value="Add Movie" /></p>
  </fieldset>
</form>
</body>
</html>

```

Вот и все. Теперь вы можете протестировать страницу, добавив фильм, но не выходя из одного или нескольких полей. При этом отображается следующее сообщение об ошибке:

Add a Movie



• Genre is required
• You haven't entered a year

Movie Information

Title:

Genre: Genre is required

Year: You haven't entered a year

Один из последних шагов — сделать его удобным для получения страницы AddMovie из исходного списка фильмов.

Снова откройте страницу "Фильмы ". После закрывающего </div> тега, следующего за вспомогательным элементом WebGrid , добавьте следующую разметку (CSHTML):

```
<p>
  <a href="~/AddMovie">Add a movie</a>
</p>
```

Как вы видели ранее, ASP.NET интерпретирует ~ оператор как корень веб-сайта. Оператору не нужно использовать. Вы можете использовать ~ разметку `Add a movie` или другой способ определить путь, который распознает HTML. ~ Но оператор является хорошим общим подходом при создании ссылок для страниц Razor, так как он делает сайт более гибким — если переместить текущую страницу в вложенную папку, ссылка по-прежнему перейдет на страницу AddMovie. (Помните, что ~ оператор работает только на страницах CSHTML. ASP.NET понимает его, но это не стандартный HTML.)

По завершении запустите страницу "Фильмы". Она будет выглядеть следующим образом:

Movies

Genre to look for:

(Leave blank to list all movies.)

Movie title contains the following:

Title	Genre	Year
Ronin	Action	1998
The Three Musketeers	Adventure	1973
Fantasia	Children	1939
1 2 3 ≥		

[Add a movie](#)

Щелкните ссылку **"Добавить фильм"**, чтобы убедиться, что она перейдет на страницу *AddMovie*.

Полное описание страницы AddMovie:

```
@{
```

```
Validation.RequireField("title", "You must enter a title");
Validation.RequireField("genre", "Genre is required");
Validation.RequireField("year", "You haven't entered a year");
```

```
var title = "";
var genre = "";
var year = "";
```

```
if(IsPost && Validation.IsValid()){
    title = Request.Form["title"];
```

```

genre = Request.Form["genre"];
year = Request.Form["year"];

var db = Database.Open("WebPagesMovies");
var insertCommand = "INSERT INTO Movies (Title, Genre, Year) Val-
ues(@0, @1, @2)";
db.Execute(insertCommand, title, genre, year);
Response.Redirect("~/Movies");
}
}

```

```

<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>Add a Movie</title>
<style type="text/css">
.field-validation-error {
font-weight:bold;
color:red;
background-color:yellow;
}
.validation-summary-errors{
border:2px dashed red;
color:red;
background-color:yellow;
font-weight:bold;
margin:12px;
}
</style>
</head>
<body>
<h1>Add a Movie</h1>
@Html.ValidationSummary()
<form method="post">
<fieldset>
<legend>Movie Information</legend>
<p><label for="title">Title:</label>
<input type="text" name="title" value="@Request.Form["title"]" />
@Html.ValidationMessage("title")
</p>

<p><label for="genre">Genre:</label>
<input type="text" name="genre" value="@Request.Form["genre"]" />
@Html.ValidationMessage("genre")

```

```

</p>

<p><label for="year">Year:</label>
  <input type="text" name="year" value="@Request.Form["year"]" />
  @Html.ValidationMessage("year")
</p>

<p><input type="submit" name="buttonSubmit" value="Add Movie" /></p>
</fieldset>
</form>
</body>
</html>

```

3. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. С каким расширением хранятся веб-формы?
2. Опишите добавление данных в базу данных.
3. Для чего используются пустые строки?
4. Охарактеризуйте валидатор проверки.

4. ЛИТЕРАТУРА

1. Постолиит, А. Visual Studio .NET: разработка приложений баз данных / А. В. Постолиит. – СПб.: БХВ-Петербург, 2003. – 544 с.: ил.
2. Сеть разработчиков Microsoft [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/aspnet/web-pages/overview/getting-started/introducing-aspnet-web-pages-2/entering-data>

Преподаватель

В.Ю.Купцова

Рассмотрено на заседании цикловой
комиссии программного обеспечения
информационных технологий №10
Протокол № __ от «__»_____202_
Председатель ЦК В.Ю.Михалевич