Кафедра цифровой экономики

Лабораторный практикум по дисциплине

ЭЛЕКТРОННЫЙ БИЗНЕС

Методические указания

к лабораторным работам для студентов

направлений подготовки:

38.03.05 «Бизнес-информатика» (степень - бакалавр)

Сведения о разработчиках:

ФИО	Кафедра	Должность, ученая степень, звание
Сковиков Анатолий Геннадьевич	Цифровой экономики	К.т.н., доцент

Ульяновск
2017

Лабораторная работа №1. «Электронная коммерция в Интернет»

Цель работы: Рассмотрение и анализ способов организации сайтов Интернет-магазинов, формирования покупательской корзины, различных возможностей оплаты и доставки товаров и услуг в Интернет.

1. Приведите примеры (3-4) электронных магазинов в зонах .ru и .com
Дайте краткое описание интернет-магазинов (заполните таблицу)

№	Адрес	Название	Продукция услуги	Достоинства интернет-магазина	Недостатки интернет-магазина	Способы доставки	Дополнительно
	http://www.eldorado.ru/	*Эльдорадо*	*Бытовая техника*	*Удобный каталог, расположение данных о контактах, реклама*	*Просмотр фото товара, описание продукции*	*адресная доставка, самовывоз из магазина*	*Рекламные акции Распродажа Газета Эльдорадо Клуб Эльдорадо*
1							
2							
3							
4							

2. Дополните таблицу своими примерами и проведите краткий анализ предложенных электронных торговых площадок в Рунете.

Отрасль	Электронная торговая площадка	Тип рынка	Модель организации рынка
Энергетика	www.b2b-energo.ru		
	Свой пример		
Сельское хозяйство	www.idk.ru		
	Свой пример		
Продукты питания	http://foodruss.ru/		
	Свой пример		
Машиностроение	www.promnavigator.ru		
	Свой пример		
Упаковка	www.unipack.ru		
	Свой пример		
Многоотраслевая	www.usetender.com		
	Свой пример		
	Свой пример		

3. Ознакомьтесь с предложенным электронные торговых площадок, обеспечивающих госзакупки в России. Дополните каждую группу сайтов 1-2 примерами и дайте краткое описание структуры и контента сайтов (заполните таблицу).

Сайт	Электронный адрес	Структура (основные блоки)	Описание контента (новости, документы, каталоги и т.д.)
Официальный сайт по госзакупкам РФ	www.zakupki.gov.ru		
	Свой пример		
Официальные сайты субъектов РФ для размещения информации о размещении заказов	www.tender.mos.ru www.altai-republic.ru www.mineconom.ru www.gz-spb.ru		
	Свой пример		
Официальные сайты муниципальных образований	www.kurgan-city.ru www.budgorod.ru www.vlc.ru/links.htm		
	Свой пример		

Самостоятельные сайты по госзакупкам	www.bob.ru , www.gostorgi.ru		
	Свой пример		
Сайты, на которых проводятся электронные аукционы	www.goszskaznso.ru www.agzrt.ru		
	Свой пример		
Сайты, на которых проводятся запросы котировок	www.gz-spb.ru www.set.cuban.ru		
	Свой пример		

Контрольные вопросы

1. Какие есть основные теги XHTML?
2. В чем заключается ограниченность технологий web первого поко-ления? Почему эти ограничения сдерживают дальнейшее развитие web?
3. В чем заключается основное преимущество каскадный таблиц стилей?
4. Опишите основные элементы пролога документа XHTML
5. Какие существуют способы создания комментариев документа XHTML?
6. Чем блочный тег отличается от линейного?
7. Перечислите исключенные теги и атрибуты.
8. Почему язык HTML можно назвать закрытым языком?

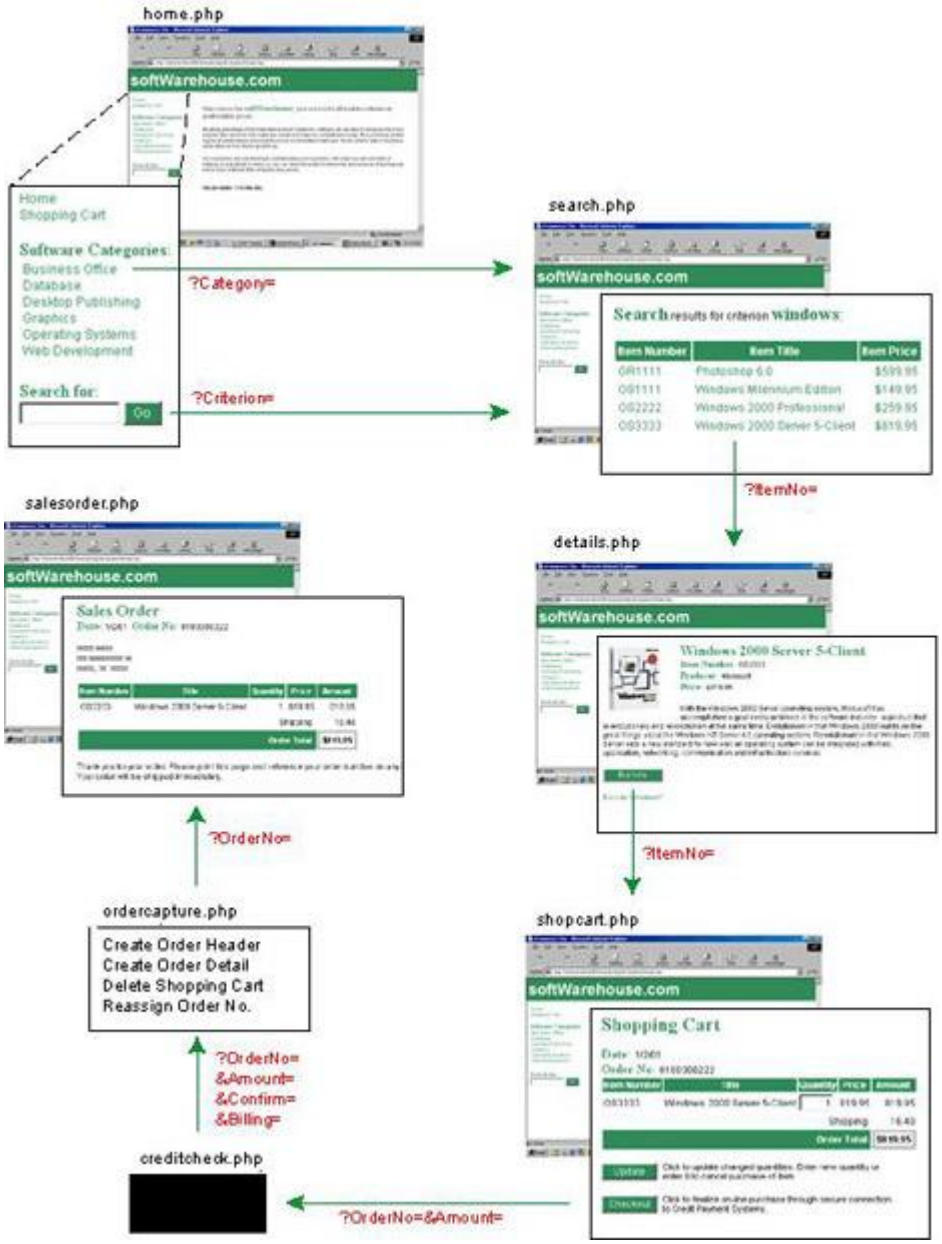
Лабораторная работа №2. «Электронная коммерция в Интернет»

Цель работы: Получение практических навыков разработки web-портала электронной коммерции.

Общий проект сайта

Сайт e-Commerce

В данном разделе последовательно рассматривается процесс разработки коммерческого сайта Web. В качестве примера используется база данных товаров `eCommerce.mdb`. Товары будут предложены для онлайн торговли. С помощью этого примера будут рассмотрены все основные вопросы разработки онлайн магазина. Следующая иллюстрация представляет страницы и механизмы связей, которые составляют типичный сайт е-коммерции. Основные части этой системы описаны ниже.



увеличить изображение

Страница home.php

Средство для поиска товара является частью всех страниц в качестве основного метода навигации. Здесь используется два способа поиска товаров: просмотр списка предложений в определенной категории товаров и поиск товаров по ключевым словам. В то же самое время мы отслеживаем посетителей, когда они

перемещаются со страницы на страницу. Задается механизм для идентификации пользователя и для поддержания этой информации в течение всего посещения. Для этой цели заказу присваивается уникальный номер.

Страница search.php

Процедуры поиска создают страницу, перечисляющую все товары в категории или все товары, которые соответствуют критериям поиска. Когда пользователь выбирает определенную категорию или задает слово для поиска, эта информация передается на страницу `search.php`, используемую для поиска товаров.

Страница detail.php

Когда на странице `search.php` выбирается определенный товар, это действие соединяется со страницей `detail.php`, где выводится вся информация о продукте, включая изображение.
Страница `detail.php` должна получить идентификацию товара, чтобы извлечь информацию. На этой странице пользователь может искать другие товары или купить этот товар. Для добавления товара в корзину покупателя создается кнопка `"Buy"`. Товар помещается в корзину покупателя на этой странице, а затем происходит соединение со страницей `shopcart.php`, чтобы посетитель мог просмотреть содержимое своей корзины покупателя. Также доступно меню поиска, чтобы можно было вернуться к процессу выбора товара.

Страница shopcart.php

Страница корзины покупателя перечисляет объекты в корзине покупателя и предоставляет посетителю возможность изменить количество единиц товара или удалить товары. Когда посетитель готов купить товар, происходит соединение со страницей `creditcheck.php`.

Страница creditcheck.php

Эту страницу создавать не требуется. Необходимо только соединиться с ней и предоставить идентификацию заказа, объем заказа и имя страницы возврата. С заказчиком через защищенное соединение будет взаимодействовать компания по обслуживанию кредитных карт. Компания по обслуживанию кредитных карт возвращает код подтверждения, а информация для выставления счета к оплате собирается на странице формирования заказа.

Страница ordercapture.php

После получения кода возврата от компании кредитных карт остаются завершающие задачи оформления — они происходят скрытно на этой странице, и одной из них является создание нового идентификационного номера заказа на тот случай, если посетитель захочет продолжить покупки. Товары в корзине покупателя заказчика удаляются, и создаются заголовок заказа и другие записи для создания постоянной записи заказа покупателя. Эти записи можно использовать для восстановления заказа в будущем. Наконец создается подтверждение e-mail, если заказчик предоставил адрес e-mail.

Страница salesorder.php

Эта страница выводит окончательный торговый заказ, чтобы заказчик подтвердил покупку и напечатал копию. Отвергнутые заказы выводят подтверждающее сообщение.

Когда мы перейдем к проектированию и кодированию этих страниц, возникнут другие вопросы, которые потребуется решать. Мы можем также придумать новые свойства, которые необходимо добавить на страницы. Тем не менее, имеется достаточно хорошее представление, что мы будем делать и как все это должно работать в совокупности.

Общая структура страниц

Здесь представлена общая модель компоновки всех страниц
сайта *Web* eCommerce. *Разделы Content* (Содержимое) страниц будут различаться в зависимости от функции страницы. Однако *разделы* `Header` (Заголовок) и `Menu` (*Меню*) одинаковы для всех страниц. Они создают общий внешний вид страниц, плюс раздел *Menu* предоставляет общее *меню* ссылок для перемещения и другие функции, которые доступны на всех страницах.

Этот *сайт* использует свойства каскадных таблиц стилей (*CSS*) для форматирования страниц и управления их представлением. Здесь будут даны некоторые общие рекомендации без излишних подробностей. Существует много специальных руководств по *CSS*, где можно найти дополнительную информацию об их применении.

Код *XHTML*, задающий структуру всех страниц, имеет следующий вид:

```
<html>
<head>
  <title>Сайт eCommerce </title>
  <link href="stylesheetEC.css" rel="stylesheet">
  require("jscript.inc")
</head>
<body>

<div style="position:absolute; top:0px; left:0px; width:780px;
  background-color:seagreen; color:white; padding:5px">

  require("header.inc")

</div>

<div style="position:absolute; top:75px; left:10px; width:175px">

  require("menu.inc")

</div>

<div style="position:absolute; top:75px; left:200px; width:550px">

  — основное содержание страницы (контент) —

</div>

</body>
</html>
```

Пример I.1.

Теги `<div>` используются для создания общей компоновки страницы и применяют спецификации таблиц стилей для точного размещения разделов на странице. Имеется три отдельных раздела, содержащие заголовок, *меню* и *контент*, которые будут выводиться на странице. Такая *компоновка* соблюдается для всех страниц сайта, единственное различие будет в разделе, содержащем основной *контент* этой страницы.

Использование файлов INCLUDE

Разделы Header и Menu не кодируются непосредственно на всех страницах. Используемый код содержится в документах `header.inc` и `menu.inc`, которые импортируются (включаются) на все страницы, когда к ним обращается браузер.

Код, содержащийся во внешних документах, копируется или включается на страницу PHP с помощью одного из следующих операторов:

```
require('FileName')
include('FileName')
```

Оба оператора выполняют одно и то же действие, однако оператор `require()` возвращает неисправимую ошибку при неудаче, в то время как оператор `include()` возвращает только предупреждение.

В обеих конструкциях `FileName` является именем файла, содержащим код, который копируется в это место документа. *Включаемые файлы* могут содержать код XHTML, код PHP, код JavaScript, спецификации таблиц стилей или обычный текст.

При использовании такой техники нет необходимости дублировать общий код на каждой странице. Код можно поместить в отдельный файл, а затем копировать на страницы по мере необходимости.

Эта техника работает только для страниц `.php` (а не страниц `.htm`). Возможно, имеет смысл задавать имена всех страниц с расширением `.php`, независимо от того, содержат ли они сценарий или нет.

Кодирование документов INCLUDE

Документы, включаемые на страницу, являются простыми текстовыми документами, содержащими вставляемый код. Имя файла и расширение этого документа могут быть произвольными. Расширение `".inc"` используется здесь для идентификации документов `header.inc` и `menu.inc` как файлов, которые вставляются на страницу. Такие имена, как `header.txt` и `menu.txt` или `header.php` и `menu.php`, также будут работать. Мы рассмотрим кодирование этих документов немного позже. Существует еще один *включаемый файл* с именем `jscript.inc`, который копируется в раздел `<head>` каждой страницы. Этот документ содержит код JavaScript для управления поведением кнопок. Мы рассмотрим этот код при обсуждении отдельных страниц.

Когда общая компоновка создана, остается побеспокоиться только о реальном "содержимом" отдельных страниц. Все остальное будет делаться автоматически с помощью файлов `INCLUDE`.

Использование таблиц стилей

Как отмечалось выше, таблицы стилей интенсивно используются для управления форматированием страниц. Для этой цели для сайта подготовлен внешний документ `stylesheetEC.css` и сохранен как стандартный текстовый документ. Каждая страница сайта применяет стили, соединяясь с этим документом с помощью тега `<link>` раздела `<head>`:

```
<html>
<head>
  <title>Page Title</title>
  <link type="text/css" href="stylesheetEC.css" rel=stylesheet">
</head>
```

Общая таблица стилей, используемая сайтом `eCommerce`, показана ниже, а остальные стили оформления применяются индивидуально к элементам страницы по мере необходимости.

`stylesheetEC.css`

```
body
  {margin:0px; background-color:white; font-family:arial; font-size:9pt}
td
  {font-family:arial; font-size:9pt}
th
  {font-family:arial; font-size:9pt; text-align:center;
   background-color:seagreen; color:white}

.head1
  {font-family:times new roman; font-size:18pt; font-weight:bold;
   color:seagreen}
.head2
  {font-family:times new roman; font-size:16pt; font-weight:bold;
   color:seagreen}
```

```
.head3
  {font-family:times new roman; font-size:14pt; font-weight:bold;
   color:seagreen}
.head4
  {font-family:times new roman; font-size:12pt; font-weight:bold;
   color:seagreen}

a:link, a:active, a:visited
  {text-decoration:none; color:seagreen}
a:hover
  {text-decoration:none; color:darkgreen; background-color:lightgreen}

.buttonS
  {width:35px; text-align:center; font-family:arial; font-size:9pt;
   background-color:seagreen; color:white}
.buttonM
  {width:70px; text-align:center; font-family:arial; font-size:9pt;
   background-color:seagreen; color:white}
.buttonL
  {width:100px; text-align:center; font-family:arial; font-size:9pt;
   background-color:seagreen; color:white}

.textbox
  {font-family:arial; font-size:10pt}
.qtybox
  {font-family:arial; font-size:10pt; text-align:right}

.small
  {font-family:arial; font-size:8pt}
```

Пример I.2.

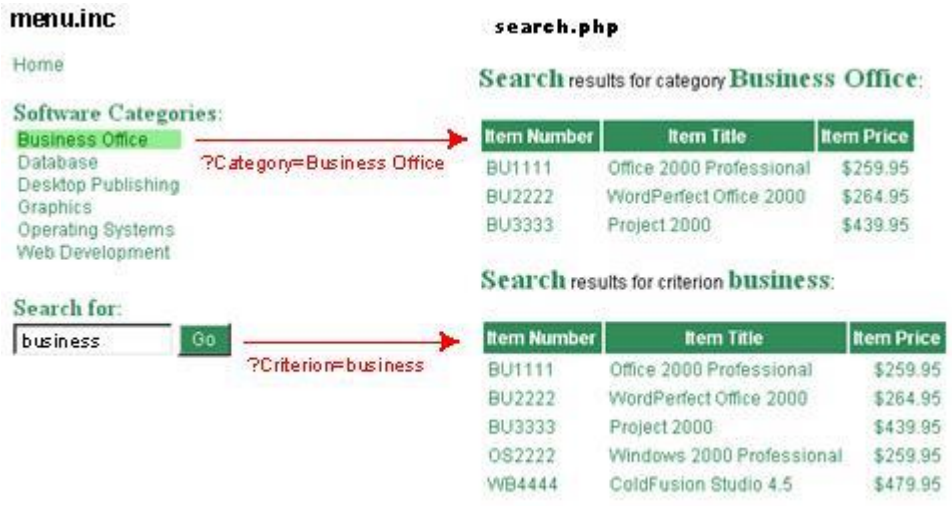
Выбор категорий продуктов

Все страницы сайта eCommerce содержат раздел ссылок для поиска информации о программном обеспечении, содержащейся в таблице Products *базы данных* eCommerce.mdb.

Посетители сайта могут просматривать названия программных продуктов двумя способами. Через множество ссылок на категории продуктов посетитель может вывести все продукты, которые попадают в эту категорию, а через *поиск* по ключевому слову *пользователь* может вывести все продукты, которые содержат *ключевое слово* в одном из полей в их записях в базе данных. Результат любого поиска создает *список* подходящих кодов продуктов, названий и цен.

Файл menu.inc

Меню поиска сделано доступным на всех страницах с помощью файла `INCLUDE`, который можно импортировать и поместить в разделе, расположенном вдоль левого края страницы. Этот файл `menu.inc` содержит код, позволяющий посетителю выбрать категорию продукта или ввести ключевое слово для поиска. Реальный поиск и вывод подходящих продуктов происходит, однако, на странице `search.php`. Поэтому выбор категории или ключевого слова включает ссылку на страницу `search.php`, и искомое значение передается на эту страницу в строке запроса.



Сначала мы рассмотрим, как создать варианты выбора категорий продуктов для поиска, а затем рассмотрим поиск по ключевым словам. Оба эти метода содержатся в файле `menu.inc`, который импортируется на все страницы сайта eCommerce. После отдельного рассмотрения кодирования этих методов поиска будет представлена вся страница `menu.inc`.

Создание ссылок на категории

Множество ссылок на страницу `search.php`, через которые передается выбранная категория, основывается на типе продукта. `ItemType` является одним из полей таблицы `Products`. Поэтому эти типы необходимо извлечь из таблицы, а затем форматировать как ссылки на страницу `search.php`. Следующий код в файле `menu.inc` делает варианты выбора доступными на всех страницах.

```
<span class="head4">Категории программ:</span>
<table>
<?php

//Создание соединения с данными

  $conn = odbc_connect
  ('Driver={Microsoft Access Driver (*.mdb)};
DBQ=c:\inetpub\wwwroot\PHPTutorial\Ecommerce\databases\ecommerce.mdb','','');

//Формирование оператора SQL SELECT

  $sql = "SELECT DISTINCT ItemType FROM Products ORDER BY ItemType";

//Выполнение оператора SQL и создание множества записей

  $rs = odbc_exec($conn, $sql);

//Цикл по множеству записей и вывод необходимых записей

 while($row = odbc_fetch_array($rs))

        {
```

```
echo "<tr style=\"color:seagreen; line-height:8pt; font-size:9pt\"
                        onMouseOver=\"this.style.backgroundColor='lightgreen';
                        this.style.color='darkgreen';
this.style.cursor='hand'\"
                        onMouseOut=\"this.style.backgroundColor='white';
                        this.style.color='seagreen'\"


        onClick=\"location.href='search.php?Category=$row[ItemType]'\">

                        <td>$row[ItemType]</td>
                </tr>";

                }

//Закрытие соединения с базой данных

odbc_close($conn);

?>
</table>
```

Пример I.3.

Здесь делается соединение с базой данных и извлекается множество записей из таблицы `Products`. Это множество записей содержит уникальные значения, которые находятся в поле `ItemType`, получаемые при использовании оператора `SQL SELECT` с параметром `DISTINCT`. Каждая запись в множестве записей `$rs` содержит одно значение - один из типов объектов в поле. Теперь необходимо просто выполнить итерации по множеству записей, форматируя ссылки для значений категорий.

Выделение строки таблицы и соединение

Значения `ItemType` выводятся в таблице, чтобы воспользоваться средствами таблицы стилей, которые предоставляют интересный метод присвоения ссылок ячейкам таблицы, а не самим значениям. Значения `ItemType` появляются в таблице в следующем виде и выделяются, когда указатель мыши перемещается по записям (вокруг таблицы показана граница, чтобы эффект был более заметен):

Ссылки делаются из ячеек таблицы - на самом деле из строк таблицы - а не из текстовых строк, появляющихся внутри ячеек. Это позволяет присвоить выделение и визуальные индикаторы строкам, когда указатель мыши перемещается по ним, а ссылки активируются при щелчке на строках таблицы, содержащих данные, а не на самих данных. Такой метод соединения из строк таблицы более четко иллюстрируется списком продуктов, появляющимся на странице `search.php`. Такая же техника используется здесь, хотя эффект не такой впечатляющий, как выделение и соединение из нескольких ячеек в строке.

Эти действия по выделению и соединению создаются следующим кодом для строк таблицы. Особенно отметим, что код находится внутри тега `<tr>`, так что результат проявляется во всех ячейках строки (хотя в данном конкретном случае в строке имеется только одна ячейка).

```
<tr style="color:seagreen; line-height:8pt; font-size:9pt"
```

```
onMouseOver="this.style.backgroundColor='lightgreen';this.style.color='darkgr
een';
    this.style.cursor='hand'"
  onMouseOut="this.style.backgroundColor='white';this.style.color='seagreen'"
  onClick="location.href='search.php?Category=$row[ItemType]'"
  >
  <td>$row[ItemType]</td>
</tr>
```

Свойства таблицы стилей используются для создания начального вида строк в ячейках. Свойство color задает цвет текста, выводимого в ячейках, свойство `line-height` задает высоту текста (и соответственно высоту самой ячейки), а свойство `font-size` задает размер шрифта текста.

Кроме этих статических свойств необходимо добавить свойства, которые изменяются, когда посетитель взаимодействует с ячейками. Мы хотим, чтобы "что-то происходило", когда посетитель перемещает курсор над ячейками, смещается с ячеек и щелкает в ячейке. Описанные события являются фактически событиями сценариев, которые в языке JavaScript называются `onMouseOver`, `onMouseOut` и `onClick`. Поэтому эти события можно использовать для включения изменений в стилях оформления ячеек таблицы.

Теги XHTML со сценарием

Процедуры JavaScript добавляются в теги `<tr>` для задания различных стилей и для включения ссылок, когда посетитель выполняет одно из трех указанных выше действий. Эти события и действия перечислены в следующей таблице.

Событие	Действие
onmouseover	this.style.backgroundColor='lightgreen' this.style.color='darkgreen' this.style.cursor='hand'"
onmouseout	this.style.backgroundColor='white' this.style.color='seagreen'
onclick	location.href='search.php...'

Когда мышь перемещается поверх строки таблицы, включается событие `onmouseover` и выполняются три оператора JavaScript. Цвет фона (свойство `backgroundColor`) строки задается как светло-зеленый (`lightgreen`), цвет текста (свойство `color`) задается как темно-зеленый (`darkgreen`), а форма курсора (свойство `cursor`) задается в форме ладони. Когда курсор мыши смещается со строки таблицы, включается событие onmouseout и свойства возвращаются к своим исходным значениям.

Общий формат выполнения команд JavaScript внутри тега HTML имеет вид

```
EventName = "statement1 [; statement2] [; statement3]..."
```

То есть за именем `onmouseover`, `onmouseout`, `onclick` или другим таким именем события, для которого будет применяться действие, следует знак равенства и в кавычках список операторов, которые будут выполнены, когда произойдет событие. Несколько операторов разделяются точкой с запятой.

Операторы, используемые для задания свойств, имеют следующий формат,

```
this.style.property='value'
```

где `style` является указанием на задание таблицы стилей, `property` является формальным именем задаваемого свойства, а значение `value` (заключенное в кавычки) является конкретным значением для задания этого свойства. Специальное ключевое слово `self` является ссылкой на себя объекта XHTML, содержащего это задание свойства, в данном случае тег `<tr>`.

Когда происходит событие `onClick`, создается ссылка на страницу `search.php`. Соединение происходит при задании свойству `location.href` (не тега `<tr>`, а текущего окна) имени страницы, на которую указывает ссылка. Это задание свойства приводит к замене URL в поле адреса браузера другим URL, в данном случае адресом страницы `search.php`.

Передача строк запроса

Когда делается ссылка из категории продуктов (в действительности из строки таблицы) на страницу `search.php`, то принимающая страница должна знать, какая категория продуктов была выбрана, чтобы создать список подходящих продуктов этой категории. Эта информация передается на страницу `search.php` через строку запроса, присоединенную к URL ссылки. Для каждой категории в списке меню (для каждой строки таблицы), значением строки запроса будет то же самое значение `ItemType`, которое выводится в ячейке таблицы:

```
onClick="location.href='search.php?Category=$row[ItemType]'"
```

Когда происходит щелчок на строке таблицы, создается URL в виде

```
search.php?Category=ItemType
```

Все строки таблицы содержат различные типы объектов, и это конкретное значение `ItemValue` присоединяется к URL этой строки. Одна из пар имя/значение

```
?Category=Business Office
?Category=Database
?Category=Desktop Publishing
?Category=Graphics
?Category=Operating Systems
?Category=Web Development
```

посылается на страницу `search.php`, информируя эту страницу о разыскиваемом типе для выполнения поиска программных продуктов, попадающих в эту категорию.

Определение критериев продукта

Кроме выбора категорий продуктов для поиска названия программного продукта, посетители сайта `eCommerce` могут использовать общий *поиск* по ключевым словам.

Посетители могут ввести *слово* или фразу в текстовое *поле*, и процедура поиска найдет все записи в таблице `Product`, которые содержат эту текстовую строку. *Файл* `menu.inc` поэтому содержит код для представления этого текстового поля ввода и для передачи введенного текста на страницу `search.php`, где происходит реальный *поиск* подходящих продуктов.

Текстовое *поле* ввода кодируется в форме, находящейся в файле `menu.inc`. Форма выводит текстовое *поле* вместе с кнопкой для отправки информации формы на страницу `search.php`.

```
<form action="search.php" method="get">
  <span class="head4">Search for:</span><br>
  <input type="text" name="Criterion" class="textbox" size="12"
    value="<?php echo $_GET[Criterion]?>">
  <input type="submit" class="buttonS" name="Submit" value="Go"
    onMouseOver="OverMouse(this)"; onMouseOut="OutMouse(this)">
</form>
```

Передача формы в виде строки запроса

В этой форме необходимо обратить внимание, прежде всего, на атрибут `method="get"`. До сих пор для передачи информации формы использовался метод `POST`. Этот метод передает пары имя/значение из формы на сервер как отдельный поток данных в URL запроса атрибута `ACTION`. В данном случае метод `GET` передает пары имя/значение как строку запроса, присоединенную к URL в параметре `ACTION`.

Причина применения метода `GET` в этом приложении состоит в желании получить согласованность с предыдущими ссылками на категории объектов. Эти ссылки передают выбранную категорию продуктов на страницу `search.php` как строку запроса. Эта форма делает то же самое для поиска по ключевым словам. Когда посетитель вводит слово или фразу в текстовое поле и нажимает кнопку отправки, в форме создается URL

```
search.php?Criterion=value
```

и значение в текстовом поле соединяется с именем *`Criterion`* для передачи на сервер. Таким образом, когда страница `search.php` получает имя категории или ключевое слово, они будут получены в форме строки запроса, различаясь только именами и значениями:

```
search.php?Category=value   из ссылки
search.php?Criterion=value   из формы
```

Отметим также, что текстовое поле имеет значение строки запроса *`Criterion`*, которая передается на сервер, когда посылается форма:

```
<input type="text" name="Criterion" class="textbox" size="12"
  value="<?php echo $_GET[Criterion]?>"
```

Когда посылается эта форма, создается URL для страницы `search.php` и в него добавляется строка запроса, содержащая введенное в это поле значение. Помните также, что файл `menu.inc`, частью которого является эта форма, появляется на всех страницах, включая страницу `search.php`. Поэтому, когда загружается страница `serach.php`, вместе с этой формой, содержащейся в файле `menu.inc`, текстовое поле отражает переданное на страницу значение *`Criterion`*. Поэтому значение присутствует в текстовом поле, хотя оно находится в другой форме на другой странице.

Кнопки форматирования

Используемая на этой форме кнопка отправки отличается от используемых обычно кнопок. Она имеет другой размер и цвет и изменяет цвет, когда указатель мыши перемещается над ней.

Начальный вид кнопки задается спецификацией таблицы стилей. Определяется класс таблицы стилей для задания ширины кнопки, выравнивания текста, типа и размера шрифта, цвета фона и цвета текста:

```
.buttonS {
  width:35px;
  text-align:center;
  font-family:arial;
  font-size:9pt;
  background-color:seagreen;
  color:white;
  }
```

Эти настройки применяют, задавая атрибут `CLASS`, которому присваивается имя этого класса в теге кнопки:

```
<input type="submit" class="buttonS" name="Submit" value="Go"...>
```

Изменения в спецификациях стиля оформления происходят, когда указатель мыши перемещается над кнопкой и когда он смещается с кнопки. Эти изменения активируются операторами JavaScript, кодируемыми в теге кнопки:

```
<input type="submit" class="buttonS" name="Submit" value="Go"
  onMouseOver="OverMouse(this)" onMouseOut="OutMouse(this)">
```

События onMouseOver и onMouseOut активируют вызовы функций `OverMouse()` и `OutMouse()`, соответственно, передавая ссылку `self` на объект (эту кнопку), содержащий эти обработчики событий. Эти функции затем получают ссылку на эту кнопку отправки для применения сценария.

Кодирование сценария в функциях имеет две задачи. Первое: сценарии содержат несколько операторов, что делает их неудобными для кодирования внутри отдельных кнопок. Второе: функции можно вызывать из любых кнопок, на которые нужно оказать воздействие. Это означает, что функции должны быть доступны на всех страницах, чтобы можно было применить любую кнопку на этих страницах.

Как было сделано для заголовка и меню, которые присутствуют на всех страницах, код JavaScript для активации кнопок находится во *включаемом файле*, который можно импортировать на любую страницу. Обычный метод импортирования кода JavaScript состоит во включении его в раздел `<HEAD>` страницы, чтобы процедуры загружались и были доступны для активации до загрузки раздела `<BODY>`. Поэтому все страницы сайта `eCommerce` включают директиву

```
require(jscript.inc)
```

в своих разделах <HEAD>. Код файла jscript.inc показан ниже:

```
jscript.inc
<script language="javascript">
function OverMouse(button)
{
  button.style.backgroundColor="lightgreen"
  button.style.color='darkgreen'
  button.style.cursor='hand'
}
function OutMouse(button)
{
  button.style.backgroundColor="seagreen"
  button.style.color='white'
}
```

```
</script>
```

Вспомните, что обе функции получают ссылку на нажатую кнопку. Эта ссылка будет получена как переменная `button`, поэтому любая ссылка на `button` (кнопка) является ссылкой на реальную кнопку, которая в данный момент находится под курсором мыши или с которой курсор мыши был только что смещен. Две функции просто задают различные стили для кнопки в зависимости от того, какое действие совершил пользователь.

Кодирование файлов INCLUDE

Все страницы сайта `eCommerce` имеют одинаковый общий формат и содержат одинаковую информацию, за исключением специфической информации страницы. Это сходство достигается с помощью трех файлов `INCLUDE`, которые копируются на все страницы. Здесь мы рассмотрим эти файлы.

Форматы страниц

Следующий код воспроизводит общую компоновку всех страниц и показывает, где и как размещаются *включаемые файлы* в соответствующих местах страниц.

```
<html>
<head>
  <title>Сайт eCommerce</title>
  <link href="stylesheetEC.css" rel="stylesheet">
  <?php require('jscript.inc') ?>
</head>
<body>

<div style="position:absolute; top:0px; left:0px; width:780px;
  background-color:seagreen; color:white; padding:5px">

  <?php require('header.inc') ?>

</div>

<div style="position:absolute; top:75px; left:10px; width:175px">

  <?php require('menu.inc') ?>

</div>

<div style="position:absolute; top:75px; left:200px; width:550px">

  — основной контент страницы —

</div>


</body>
</html>
```

Пример I.4.

Файл header.inc

Файл `header.inc` импортируется в верхний раздел каждой страницы. Код использует линейные спецификации стиля для создания верхнего баннера. Файл содержит не слишком много кода, но его можно расширить, чтобы включить многие другие свойства, которые могут понадобиться в заголовке страниц.

```
header.inc

<span style="font-size:32pt; font-weight:bold">softWarehouse.com</span>
```

Файл menu.inc

Файл menu.inc импортируется в крайний левый раздел всех страниц, чтобы создать общедоступное меню поиска продуктов. Полный код этого файла, различные части которого были рассмотрены ранее, показан ниже.

menu.inc

```
<a href="home.php">Home</a>
<a href="shopcart.php">Shopping Cart</a>
<br>
<br>
<span class="head4">Software Categories:</span>
<table>
<?php

//Создание соединения с данными

  $conn = odbc_connect
  ('Driver={Microsoft Access Driver (*.mdb)};
DBQ=c:\inetpub\wwwroot\PHPTutorial\Ecommerce\databases\ecommerce.mdb','','');

//Формирование оператора SQL SELECT

  $sql = "SELECT DISTINCT ItemType FROM Products ORDER BY ItemType";

//Выполнение оператора SQL и создание множества записей

  $rs = odbc_exec($conn, $sql);

//Цикл по множеству записей и ввод необходимых записей

 while($row = odbc_fetch_array($rs))

        {
echo "<tr style=\"color:seagreen; line-height:8pt; font-size:9pt\"
                        onMouseOver=\"this.style.backgroundColor='lightgreen';
                        this.style.color='darkgreen';
this.style.cursor='hand'\"
                        onMouseOut=\"this.style.backgroundColor='white';
                        this.style.color='seagreen'\"


        onClick=\"location.href='search.php?Category=$row[ItemType]'\">


                        <td>$row[ItemType]</td>
                </tr>";
                }

//Закрытие соединения с БД

odbc_close($conn);

?>
</table>

<form action="search.php" method="get">
  <span class="head4">Search for:</span><br>
  <input type="text" name="Criterion" class="textbox" size="12"
    value="<?php echo $_GET[Criterion]?>">
  <input type="submit" class="buttonS" name="Submit" value="Go"
    onMouseOver="OverMouse(this)"; onMouseOut="OutMouse(this)">
</form>
```

Пример I.5.

Файл jscript.inc

Наконец, ниже воспроизводится файл jscript.inc.

```
jscript.inc
<script language="javascript">
function OverMouse(button)
{
  button.style.backgroundColor="lightgreen"
  button.style.color='darkgreen'
  button.style.cursor='hand'
}
function OutMouse(button)
{
  button.style.backgroundColor="seagreen"
  button.style.color='white'
}
</script>
```

Важно отметить, что файлы `INCLUDE` содержат только показанный здесь код. В них нет тегов `<HTML>` или `<BODY>` либо какого-либо другого кода, кроме того, который нужен для определенных целей файла. Эти файлы импортируются в полностью сформированную страницу HTML, и можно создать проблемы с выводом, если включить в эти файлы больше информации, чем требуется.

Имея это общее представление и функции, теперь станет значительно легче сконцентрироваться на кодировании основного контента каждой страницы сайта.

Поиск продуктов

Страница `search.php` вызывается для выполнения поиска в таблице `Products` с помощью одного из двух методов. *Поиск* категории продукта находит все названия программных продуктов в этой категории на основе соответствующего значения `ItemType` в их записях. *Поиск* по ключевым словам находит все названия программных продуктов, содержащие *ключевое слово* в одном из своих полей. В обоих случаях создается *список* из полей `ItemNumber`, `ItemTitle`, и `ItemPrice` соответствующих записей о продуктах. *Список* продуктов соединяется также со страницей `detail.php`, где представлена вся *информация* о продукте.

Получение строк запросов

Когда загружается страница search.php она получает строку запроса со страницы, на которой был сделан запрос поиска категории или поиска по критерию. Возможны два вида строк запроса:

```
?Category=CategoryName
?Criterion=KeywordValue
```

Одной из первых задач на этой странице поэтому является получение этого значения из массива строки запроса Request.QueryString, который создается, когда пара имя/значение попадает на сервер. Обычно в массиве доступно одно из двух значений. Однако в одном случае нет ни одного значения. Это происходит, когда форма поиска отправляется с пустым полем критерия поиска. В этом случае нет имени категории или значения ключевого слова на этой странице. Все эти ситуации учитываются в следующем сценарии, который появляется в верхней части страницы search.php.

```
<?php
$Category = $_GET[Category];
$Criterion = $_GET[Criterion];

if ($Category == "" && $Criterion == "")
{
 header("Location:home.php");
 }

?>
```

Значения строки запроса присваиваются переменным Category (Категория) и *Criterion* (Критерий). Отметим также, что если оба эти значения будут null, в соответствии с описанной выше ситуацией, то посетитель немедленно перенаправляется на страницу home.php. В этом случае нечего искать.

Структура страницы

Основное содержимое этой страницы кодируется внутри третьего раздела страницы (напомним, что первый раздел включает файл header.inc, а второй раздел включает файл menu.inc). Именно в этом разделе используются две процедуры поиска. Страница поэтому имеет следующую структуру:

```
<?php
$Category = $_GET[Category];
$Criterion = $_GET[Criterion];

if ($Category == "" && $Criterion == "")

{
 header("Location:home.php");

}

?>

<html>
<head>
        <title>Сайт eCommerce</title>
        <link href="stylesheetEC.css" rel="stylesheet">
        <<?php require("jscript.inc") ?>
</head>
<body>


<div style="position:absolute; top:0px; left:0px; width:780px;
  background-color:seagreen; color:white; padding:5px">

  <?php
  require("header.inc")
```

```
  ?>

</div>

<div style="position:absolute; top:75px; left:10px; width:175px">

 <?php
  require("menu.inc")
  ?>

</div>

<div style="position:absolute; top:75px; left:200px; width:550px">

 if ($Category != "") {
          ...code for category search

}

if ($Criterion != "") {
          ...code for keyword search
}
</div>
</body>
</html>
```
Пример I.6.

Если имеется значение `Category`, то это значение было передано на эту страницу и должно использоваться для поиска категории. Если, с другой стороны, на страницу было передано значение *`Criterion`*, то должен выполняться этот тип поиска.

Программирование поиска категории

Для продукта соответствующего значению `Category`, переданного на эту страницу, подходящие записи представляют в таблице. Из подходящих записей выводятся три поля: `ItemNumber`, `ItemTitle`, и `ItemPrice`. Код начинается с вывода заголовка таблицы и заголовков столбцов над значениями таблицы. На следующем листинге значение переменной `Category` выводится в заголовке, чтобы помочь идентифицировать результаты поиска.

```
if ($Category != "") {
          <span class="head3">Поиск</span>результатов для категории
          <span class="head3"><?php echo $Category ?></span>:
          <br>
          <br>
          <table border="0" cellpadding="3">
          <tr>
                    <th>Item Number</th>
                    <th>Item Title</th>
                    <th>Item Price</th>
          </tr>
```

Вслед за этим на странице выводятся результаты поиска. Каждая запись таблицы `Product`, которая имеет значение `ItemType`, совпадающее со значением переменной `Category`, выбирается для вывода полей `ItemNumber`, `ItemTitle`, и `ItemPrice`. Каждая запись продукта появляется в отдельной строке таблицы вывода. Реализующий это код показан ниже.

```
<?php
    $conn = odbc_connect
  ('Driver={Microsoft Access Driver (*.mdb)};
DBQ=c:\inetpub\wwwroot\PHPTutorial\Ecommerce\databases\ecommerce.mdb','','');
```

```
  //Формирование оператора SQL SELECT
$sql = "SELECT ItemNumber,ItemTitle,ItemPrice FROM Products WHERE ItemType =
'$Category' ORDER BY ItemNumber";

//Выполнение оператора SQL для создания множества записей
$rs = odbc_exec($conn, $sql);

//Цикл по множеству записей и вывод необходимых записей
while($row = odbc_fetch_array($rs))

        {
        $ItemNumber = $row[ItemNumber];
        $ItemTitle = $row[ItemTitle];
        $ItemPrice = number_format($row[ItemPrice],2);

echo " <tr style=\"color:seagreen; line-height:8pt\"

onMouseOver=\"this.style.backgroundColor='lightgreen';this.style.color='darkg
reen';
          this.style.cursor='hand'\"

onMouseOut=\"this.style.backgroundColor='white';this.style.color='seagreen'\"
        onClick=\"location.href='detail.php?ItemNumber=$row[ItemNumber]\" .
          \"&Category=$Category'\">
        <td>$ItemNumber</td>
        <td>$ItemTitle</td>
        <td align=\"right\">$ItemPrice</td>
      </tr>";

        }

        odbc_close($conn);

  ?>
          </table>
<?php

}

?>
```

Пример I.7.

Сценарий начинается с соединения с базой данных `eCommerce.mdb` и создания объекта множества записей с именем `RSCategory` для извлечения подходящих записей из таблицы `Products`. Выбираются записи, в которых значение `ItemType` совпадает со значением переменной `Category`, содержащей значение строки запроса:

```
SQL="SELECT ItemNumber,ItemTitle,ItemPrice FROM Products
WHERE ItemType= '$Category' ORDER BY ItemNumber"
```

Сценарий выполняет итерации на множестве выбранных записей, выводя каждую по очереди в строке таблицы. Значения множества записей присваиваются переменным, чтобы упростить с ними работу.

```
$ItemNumber = $row[ItemNumber]
$ItemTitle = $row[ItemTitle]
$ItemPrice = number_format($row[ItemPrice],2)
...
<td>echo $ItemNumber</td>
<td>echo $ItemTitle</td>
<td align="right">$echo $ItemPrice</td>
```

Отметим, что для переменной `$ItemPrice` перед присваиванием была использована функция PHP `number_format`, чтобы вывод был представлен в долларах и центах. Значение также выравнивается вправо в ячейке таблицы.

Соединение из строк таблицы

В этой таблице, как и в таблице ссылок на категории в файле `menu.inc`, вся строка таблицы является ссылкой на страницу `detail.php`, где приводится полное описание продукта. Эти ссылки и их визуальные эффекты создают, программируя каждый тег `<tr>` реагировать на события перемещения указателя мыши над строкой и на щелчок мыши.

```
<tr style="color:seagreen; line-height:8pt"

onMouseOver="this.style.backgroundColor='lightgreen';this.style.color='darkgr
een';
    this.style.cursor='hand'"
  onMouseOut="this.style.backgroundColor='white';this.style.color='seagreen'"
  onClick="location.href='detail.php?ItemNumber=echo
$ItemNumber&Category=echo $Category'"
  >
```

Здесь также линейная спецификация стиля задает для строки используемый по умолчанию цвет текста и толщину линий. Затем встроенные обработчики событий JavaScript динамически изменяют визуальные стили оформления строки на события перемещения курсора мыши. Кроме того, событие `onClick` создает запрос URL для страницы `detail.php`. Этот URL форматируется для передачи на страницу `detail.php` строки запроса, содержащей `ItemNumber` продукта этой строки, вместе с переменной `Category`, которая была передана на эту страницу из меню поиска. Использование этого последнего элемента строки запроса рассматривается при обсуждении страницы `detail.php`.

После вывода всех строк таблицы множество записей и соединение закрываются, и записывается закрывающий тег `</table>`. Все продукты запрошенной категории выводятся в таблице вместе со ссылками на страницу `detail.php`, где предоставляется полная информация о продукте.

Программирование поиска по ключевым словам

Поиск по ключевым словам действует почти таким же образом, как и поиск по категории. Основное различие состоит в способе извлечения записей о продуктах из таблицы `Products`. Первая часть сценария будет идентичной, за исключением вывода значения *`Criterion`*, переданного в строке запроса, а не значения `Category`.

```
<% If Criterion <> "" Then %>
  <span class="head3">Search</span>results for criterion
  <span class="head3">echo $Criterion</span>:
  <br>
  <br>
  <table border="0" cellpadding="3">
  <tr>
    <th>Item Number</th>
    <th>Item Title</th>
    <th>Item Price</th>
  </tr>
```

Следующий листинг кода показывает процедуру поиска всех продуктов, содержащих значение в одном из своих полей, которое совпадает со значением *`Criterion`*, переданным на страницу из меню поиска.

```
<?php

      $conn = odbc_connect
```

```
  ('Driver={Microsoft Access Driver (*.mdb)};
DBQ=c:\inetpub\wwwroot\PHPTutorial\Ecommerce\databases\ecommerce.mdb','','');

    $sql = "SELECT ItemNumber,ItemTitle,ItemPrice FROM Products WHERE";
    $sql = $sql . " ItemNumber LIKE '%" . $Criterion . "%'";
    $sql = $sql . " OR ItemType LIKE '%" . $Criterion . "%'";
    $sql = $sql . " OR ItemProducer LIKE '%" . $Criterion . "%'";
    $sql = $sql . " OR ItemTitle LIKE '%" . $Criterion . "%'";
   $sql = $sql . " OR ItemDescription LIKE '%" . $Criterion . "%'";
    $sql = $sql . " ORDER BY ItemNumber";

 //Выполняется оператор SQL и создается множество записей

  $rs = odbc_exec($conn, $sql);

//Цикл по множеству записей и вывод необходимых записей

 while($row = odbc_fetch_array($rs))

        {

        $ItemNumber = $row[ItemNumber];
        $ItemTitle = $row[ItemTitle];
        $ItemPrice = number_format($row[ItemPrice],2);


        echo "<tr style=\"color:seagreen; line-height:8pt\"

onMouseOver=\"this.style.backgroundColor='lightgreen';this.style.color='darkg
reen';
        this.style.cursor='hand'\"

onMouseOut=\"this.style.backgroundColor='white';this.style.color='seagreen'\"
      onClick=\"location.href='detail.php?ItemNumber=$ItemNumber\" .
        \"&Criterion=$Criterion'\">
      <td>$ItemNumber</td>
      <td>$ItemTitle</td>
      <td align=\"right\">$$ItemPrice</td>
    </tr>";

        }

        odbc_close($conn);

  ?>

  </table>

<?php

}

?>
```

Пример I.8.

Как можно видеть, единственное реальное различие в двух сценариях состоит в том, что поиск по ключевым словам использует более сложную команду SQL `SELECT` для поиска подходящих продуктов:

```
$sql = "SELECT ItemNumber,ItemTitle,ItemPrice FROM Products WHERE";
$sql = $sql . " ItemNumber LIKE '%" . $Criterion . "%'";
$sql = $sql . " OR ItemType LIKE '%" . $Criterion . "%'";
$sql = $sql . " OR ItemProducer LIKE '%" . $Criterion . "%'";
$sql = $sql . " OR ItemTitle LIKE '%" . $Criterion . "%'";
```

```
$sql = $sql . " OR ItemDescription LIKE '%" . $Criterion . "%'";
$sql = $sql . " ORDER BY ItemNumber";
```

Так как получается очень длинный оператор `SELECT`, то он составлен из отдельных частей. Каждая последующая часть строки поиска присоединяется в конце предшествующей строки для создания всего оператора `SELECT` в переменной `$sql`.

Как и раньше, извлекаются только поля `ItemNumber`, `ItemTitle`, и `ItemPrice`. Сам поиск совпадения со значением *`Criterion`* происходит по пяти полям записей. Строка критерия может присутствовать в любом месте поля, так как для поиска используется оператор `LIKE`.

Поэтому если происходит совпадение с частью содержимого полей `ItemNumber`, `ItemType`, `ItemProducer`, `ItemTitle`, или `ItemDescription`, то эта запись выбирается.

После извлечения множества записей отдельные поля выводятся как строки таблицы, точно так же, как для поиска по категории. Из этих строк также сделаны ссылки, чтобы передать URL на страницу `detail.php`, добавляя строку запроса — она содержит `ItemNumber` вместе со значением *`Criterion`*, использование которого будет рассмотрено при обсуждении страницы `detail.php`.

Давайте теперь посмотрим на законченную страницу `search.php`.

`search.php`

```
<?php
$Category = $_GET[Category];
$Criterion = $_GET[Criterion];

if ($Category == "" && $Criterion == "")

{
 header("Location:home.php");

}

?>

<html>
<head>
  <title>Сайт eCommerce </title>
  <link href="stylesheetEC.css" rel="stylesheet">

  <?php
  require("jscript.inc");

  ?>

</head>
<body>


<div style="position:absolute; top:0px; left:0px; width:780px;
  background-color:seagreen; color:white; padding:5px">

  <?php
  require("header.inc")
  ?>

</div>
```

```
<div style="position:absolute; top:75px; left:10px; width:175px">

 <?php
  require("menu.inc")
  ?>

</div>

<div style="position:absolute; top:75px; left:200px; width:550px">

<?php If ($Category != "")
{
 ?>
  <span class="head3">Поиск</span>результатов для категории
  <span class="head3"><?php echo $Category ?></span>:
  <br>
  <br>
  <table border="0" cellpadding="3">
  <tr>
    <th>Item Number</th>
    <th>Item Title</th>
    <th>Item Price</th>
  </tr>

  <?php

    $conn = odbc_connect
  ('Driver={Microsoft Access Driver (*.mdb)};
DBQ=c:\inetpub\wwwroot\PHPTutorial\Ecommerce\databases\ecommerce.mdb','','');

  //Формирование оператора SQL SELECT

  $sql = "SELECT ItemNumber,ItemTitle,ItemPrice FROM Products WHERE ItemType
= '$Category' ORDER BY ItemNumber";

//Выполнение оператора SQL и создание множества записей

  $rs = odbc_exec($conn, $sql);

//Цикл по множеству записей и вывод необходимых записей

 while($row = odbc_fetch_array($rs))

        {
        $ItemNumber = $row[ItemNumber];
        $ItemTitle = $row[ItemTitle];
        $ItemPrice = number_format($row[ItemPrice],2);

        echo "<tr style=\"color:seagreen; line-height:8pt\"

onMouseOver=\"this.style.backgroundColor='lightgreen';this.style.color='darkg
reen';
         this.style.cursor='hand'\"

onMouseOut=\"this.style.backgroundColor='white';this.style.color='seagreen'\"
      onClick=\"location.href='detail.php?ItemNumber=$row[ItemNumber]\" .
        \"&Category=$Category'\"
      >
      <td>$ItemNumber</td>
      <td>$ItemTitle</td>
      <td align=\"right\">$$ItemPrice</td>
    </tr>";
```

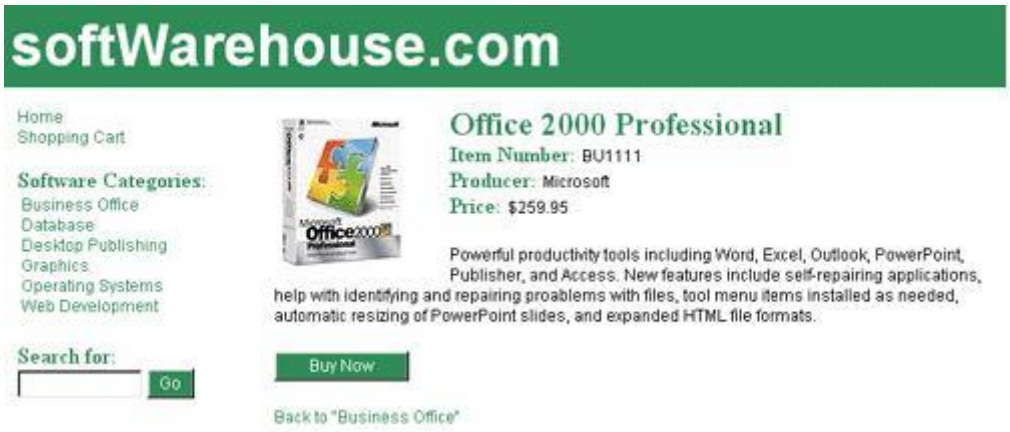
```
        }

        odbc_close($conn);

  ?>

  </table>

<?php

}

 if ($Criterion != "")

{
?>

  <span class="head3">Поиск</span>результатов для критерия
  <span class="head3"><?php echo $Criterion ?></span>:
  <br>
  <br>
  <table border="0" cellpadding="3">
  <tr>
    <th>Item Number</th>
    <th>Item Title</th>
    <th>Item Price</th>
  </tr>

  <?php

      $conn = odbc_connect
  ('Driver={Microsoft Access Driver (*.mdb)};
DBQ=c:\inetpub\wwwroot\PHPTutorial\Ecommerce\databases\ecommerce.mdb','','');

    $sql = "SELECT ItemNumber,ItemTitle,ItemPrice FROM Products WHERE";
    $sql = $sql . " ItemNumber LIKE '%" . $Criterion . "%'";
    $sql = $sql . " OR ItemType LIKE '%" . $Criterion . "%'";
    $sql = $sql . " OR ItemProducer LIKE '%" . $Criterion . "%'";
    $sql = $sql . " OR ItemTitle LIKE '%" . $Criterion . "%'";
   $sql = $sql . " OR ItemDescription LIKE '%" . $Criterion . "%'";
    $sql = $sql . " ORDER BY ItemNumber";

       //Выполнение оператора SQL и создание множества записей

  $rs = odbc_exec($conn, $sql);

//Цикл по множеству записей и вывод необходимых записей

 while($row = odbc_fetch_array($rs))

        {

        $ItemNumber = $row[ItemNumber];
        $ItemTitle = $row[ItemTitle];
        $ItemPrice = number_format($row[ItemPrice],2);

        echo "<tr style=\"color:seagreen; line-height:8pt\"

onMouseOver=\"this.style.backgroundColor='lightgreen';this.style.color='darkg
reen';
        this.style.cursor='hand'\"

onMouseOut=\"this.style.backgroundColor='white';this.style.color='seagreen'\"
```

```
      onClick=\"location.href='detail.php?ItemNumber=$ItemNumber\" .
        \"&Criterion=$Criterion'\">
      <td>$ItemNumber</td>
      <td>$ItemTitle</td>
      <td align=\"right\">$$ItemPrice</td>
    </tr>";

        }

        odbc_close($conn);

  ?>

  </table>

<?php

}

?>

</div>

</body>
</html>
```

Пример I.9.

Вывод описания продукта

Когда делается соединение со страницей `detail.php`, соответствующая *строка запроса* предоставляет странице `ItemNumber` продукта и *значение* `Category` или *`Criterion`* из предшествующего поиска. Страница `detail.php` применяет `ItemNumber` для извлечения всех связанных с ним полей из таблицы `Product` и выводит эту информацию на странице. *Значение* `Category` или *`Criterion`* используется для создания *обратной ссылки* на страницу поиска.



Программирование страницы достаточно просто. `ItemNumber` используется для извлечения соответствующей записи из таблицы `Product` и для доступа к изображению продукта в папке `Pictures`. Вся эта *информация* выводится затем на странице.

```
detail.php
<%
$ItemNumber = $_GET[ItemNumber]
$Category = $_GET[Category]
$Criterion = $_GET[Criterion]
%>
```

```
<html>
<head>
  <title>Сайт eCommerce </title>
  <link href="stylesheetEC.css" rel="stylesheet">
  <?php require("jscript.inc") ?>
</head>
<body>

<div style="position:absolute; top:0px; left:0px; width:780px;
    background-color:seagreen; color:white; padding:5px">
  <?php require("header.inc") ?>
</div>

<div style="position:absolute; top:75px; left:10px; width:175px">
  <?php require("menu.inc") ?>
</div>

<div style="position:absolute; top:75px; left:200px; width:550px">

<?php

 $conn = odbc_connect
  ('Driver={Microsoft Access Driver (*.mdb)};
DBQ=..\Ecommerce\databases\ecommerce.mdb','','');

  //Формирование оператора SQL SELECT

  $sql = "SELECT * FROM Products WHERE ItemNumber = '$ItemNumber'";

//Выполнение оператора SQL и создание множества записей

  $rs = odbc_exec($conn, $sql);

//Присваивание записей

$ItemType = odbc_result($rs,ItemType);
$ItemProducer= odbc_result($rs,ItemProducer);
$ItemTitle = odbc_result($rs,ItemTitle);
$ItemDescription = odbc_result($rs,ItemDescription);
$ItemPrice = number_format(odbc_result($rs,ItemPrice),2);

?>

<img src="Pictures/<?php echo $ItemNumber; ?>.jpg" align="left"
style="margin-right:30px">
<span class="head1"><?php echo $ItemTitle; ?></span><br/>
<span class="head4">Item Number: <?php echo $ItemNumber; ?></span><br/>
<span class="head4">Producer: <?php echo $ItemProducer; ?></span><br/>
<span class="head4">Price: $<?php echo $ItemPrice; ?></span>
<p><?php echo $ItemDescription; ?></p>

<form>
<input type="submit" class="buttonL" name="BuyButton" value="Buy Now"
         onMouseOver="OverMouse(this)"; onMouseOut="OutMouse(this)">
</form>

<a href="search.php?Category=<?php echo $Category; ?> &Criterion=<?php echo
$Criterion; ?>">
         Back to <?php echo $Category; echo $Criterion; ?>
</a>

</div>
```

```
</body>
</html>
```
Пример I.10.

Как мы уже делали на предыдущих страницах, прежде всего необходимо получить посланную на страницу строку запроса. В данном случае имеется три значения строки запроса, которые могут быть посланы. Будут получены *значение* `ItemNumber` вместе со значением `Category` или *`Criterion`*. Не имеет значения, что одно из двух последних значений будет отсутствовать в зависимости от типа выполненного на странице `search.php` поиска. *Присваивание* все равно можно выполнить, просто одно из значений будет `null`.

```
$ItemNumber = $_GET[ItemNumber]
$Category = $_GET[Category]
$Criterion = $_GET[Criterion]
```

В основном разделе страницы извлекается *информация* о продукте. *Поиск* записи о продукте основывается на значении `ItemNumber`, переданного со страницы поиска. Когда *запись* извлекается, ее значения присваиваются переменным для упрощения записи. Сохраняются все поля кроме `ItemQuantity`. Это *поле* используется в дальнейшем, а не для вывода на этой странице.

Изображение, связанное с продуктом, выводится с помощью тега `<IMG>`, используя `ItemNumber` для идентификации соответствующего изображения. Все изображения именуются в соответствии с регистрационными номерами соответствующих продуктов. Затем форматируется последовательность заголовков, чтобы представить данные о продукте, вместе с расширенным описанием продукта. Никакое специальное *форматирование* не применяется.

Затем появляется кнопка `"Buy Now"`. Эта кнопка используется посетителем для покупки продукта, добавляя позицию в корзину покупателя. В данное время мы проигнорируем эту кнопку и обратимся к ней и связанному с ней программированию в дальнейшем. Можно отметить, однако, что эта кнопка имеет такие же характеристики стиля оформления, как и кнопка `"Go"` в *меню* поиска. Ее обработчики событий `onMouseOver` и `onMouseOut` связываются с процедурами JavaScript, импортированными на страницу в файле `INCLUDE jscript.inc`.

Ссылка внизу страницы возвращает на предыдущую страницу `search.php`. Но не только это: *URL* возвращает назад строку запроса, что приводит к повторному выводу результатов предыдущего поиска. (Возврат *URL* без строки запроса не предоставит странице `search.php` *значение* `Category` или *`Criterion`*. Страница `search.php` тогда по умолчанию вернется на домашнюю страницу (`home`), и результаты предыдущего поиска будут потеряны.)

Строка запроса передает оба значения `Category` и *`Criterion`*, даже если только одно из них было послано на эту страницу. Это означает просто, что одно из значений строки запроса будет `null`. Такая ситуация не имеет значения для страницы `search.php`. Она заново выполняет *поиск* на основе возвращаемого значения.

Это отсутствующее *значение* `Category` или *`Criterion`* также не влияет на *вывод* внутри *обратной ссылки*.

```
Back to "<?php echo $Category; echo $Criterion; ?>"
```

Одно из этих двух значений будет `null`, поэтому будет выводиться имеющееся *значение*, а другое выводиться не будет.

Страница `detail.php` имеет другое важно назначение, кроме вывода информации о продукте. На этой странице посетитель покупает продукты, заполняя свою корзину покупок с помощью кнопки `"Buy Now"`. Мы вернемся к обсуждению сценария для этой задачи немного позже.

Отслеживание заказчиков

Прежде чем следовать дальше, необходимо создать способ идентификации посетителей сайта и отслеживания их, когда они перемещаются со страницы на страницу. Когда посетители что-то покупают, они

становятся заказчиками с покупками, и необходимо иметь возможность связать определенных заказчиков с определенными покупками. Мы рассмотрим два способа присваивания уникальных идентификаторов посетителям и два способа их отслеживания.

Генерация случайных чисел

Один метод создания уникального идентификатора состоит в генерации случайного числа. В PHP это делают с помощью следующего кода:

```
$RandomNumber = rand(int min, int max)
```

где `min` и `max` являются целыми значениями, которые определяют диапазон чисел, в котором должно находится случайно сгенерированное число. Результат присваивается переменной `RandomNumber`. Поэтому, если требуется получить случайное число между, скажем, 1111111111 и 9999999999, необходимо использовать оператор:

```
$OrderNo = rand(1111111111, 9999999999)
```

В этом случае мы присвоили случайное число переменной с именем `OrderNo`, то есть, если посетитель решит что-нибудь купить, мы будем использовать это число в качестве номера заказа. Всегда существует вероятность, какой бы ни была она маленькой, что сгенерированное число совпадает с уже существующим номером заказа. Поэтому для безопасности это число необходимо сравнить с использованными ранее номерами заказов, прежде чем присваивать его посетителю.

Использованные ранее номера заказов появятся в двух местах. Они будут храниться в таблице корзин покупателя посетителей, которые покупают в данный момент; и они будут храниться в таблице заказов, содержащей информацию о заказчиках предыдущих продаж. Эти таблицы пока не обсуждались, поэтому подождем, когда они будут рассмотрены, и затем вернемся к повторяющимся номерам заказов.

Итак, где же поместить процедуру генератор `OrderNo`? Возможно на начальной странице (`home.php`) сайта, так как мы хотим начинать отслеживать посетителей немедленно при их появлении. В таком случае можно ожидать, что можно будет отследить посетителей сайта, даже если они ничего не будут покупать. Мы здесь не будем это делать, но такая возможность существует.

Отслеживание заказчиков с помощью строки запроса

Однако возникает небольшая проблема. Всякий раз, когда пользователь возвращается на домашнюю страницу, ему будет присваиваться новый номер заказа. В этом случае необходимо проверять, не существует ли уже номер заказа для этого посетителя, и не создавать новый номер, если он уже существует.

Вспомните из предыдущего обсуждения, что для сохранения информации при переходе со страницы на страницу можно задействовать для передачи этой информации строки запроса. Мы ранее показывали, как использовать строку запроса для передачи информации поиска между страницами `search.php` и `detail.php`. То же самое можно делать со значением `OrderNo`. Можно передавать его как строку запроса со страницы на страницу, чтобы сохранить его и, для решения текущей проблемы, получить способ узнать, что номер заказа уже был создан.

Каждой странице сайта необходимо будет получать строку запроса, содержащую `OrderNo` вместе с любой другой информацией, которая может передаваться на эту страницу. Так как страница `home.php` также будет получателем строки запроса с `OrderNo` (когда посетитель возвращается на домашнюю страницу с других страниц сайта), то можно проверять присутствие этой строки запроса, чтобы убедиться, что номер заказа уже был присвоен, и пропускать процедуру генерации номера заказа.

Все эти объяснения сводятся к некоторому очень простому коду. Мы помещаем процедуру генерации случайного числа внутри условного оператора проверки значения строки запроса:

```
<?php
```

```
if ($_GET[OrderNo]) == "")
{
  $OrderNo = rand(1111111111,9999999999);
}
?>
```

Если домашняя страница получает строку запроса со значением `OrderNo`, равным `null`, то будет создан новый номер заказа. Это будет происходить при первом посещении домашней страницы, так как на сайт в начале не передается никакой строки запроса. Однако, если домашняя страница получает строку запроса с `OrderNo`, отличным от `null` (номер существует), то эта процедура пропускается. Это будет происходить при всех последующих возвращениях на домашнюю страницу с любой другой страницы сайта.

Конечно, если посетитель покидает сайт и возвращается, будет создаваться новый номер заказа, так как внешний сайт не передает строку запроса с `OrderNo`. Такая ситуация вызывает серьезные проблемы при отслеживании заказов покупателя. Если человек уже поместил товары в свою корзину покупателя под одним номером, а затем покинул сайт и вернулся, то нет удобного способа узнать, что это тот же самый человек, добавляющий новые товары в свою корзину покупателя. Фактически первый вариант выбора товаров будет потерян.

Можно обойти эту проблему, требуя от всех посетителей регистрации на сайте и присваивая им постоянный номер заказчика, который они используют во время всех посещений. Таким образом, номер заказчика будет всегда связан с одним и тем же человеком. Это неплохая идея, но требует более сложного программирования, чем то, что мы хотим здесь использовать. Для текущей задачи более удобным решение будет соединение номера заказа с объектом сеанса (`Session`).

Использование сеансов

В PHP посетители сайта Web создают `"сеансы"`. Сеанс создается автоматически, когда посетитель впервые появляется на любой странице сайта. Сеанс автоматически завершается, если посетитель не осуществляет обращение к странице в течение 20-минутного интервала времени. Поэтому сеанс начинается, когда посетитель прибывает, и заканчивается через 20 минут после того, как посетитель покидает сайт. Этот 20-минутный интервал времени позволяет также посетителю покидать сайт и возвращаться позже, не становясь новым посетителем. (Между прочим, 20-минутный интервал времени можно увеличить, если понадобится. Мы этого делать не будем.)

С сеансом посетителя связан ассоциативный массив `$_SESSION[]`. Можно считать `$_SESSION[]` глобальной областью хранения, где поддерживается относящаяся к посетителю информация. Все посетители сайта имеют свой собственный индивидуальный объект сеанса (`Session`) для отслеживания индивидуальной активности. Как можно было бы предположить, объект сеанса посетителя может быть хорошим механизмом для отслеживания заказов покупателя, когда человек перемещается со страницы на страницу, уходит на другой сайт и возвращается. Если такое значение, как `OrderNo`, поместить в объект сеанса посетителя, когда он впервые приходит на сайт, то это значение будет доступно на любой странице, нет необходимости тащить это значение со страницы на страницу с помощью строки запроса.

Идентификатор (ID) сеанса

PHP отслеживает посетителя с помощью уникального идентификатора, который присваивается ему при создании сеанса. Это уникальное значение хранится в свойстве сеанса `session_id()` объекта сеанса. К этому значению можно обратиться после запуска нового сеанса:

```
<?php
session_start();
$_SESSION[OrderNo] = session_id();
?>
```

Функция `session_id()` автоматически генерирует `id` сеанса. Если желательно создать свой собственный случайный `id`, можно использовать функцию `rand()` для генерации значения, а затем сохранить его в переменной сеанса:

```
<?php
session_start();
$_SESSION[OrderNo] = rand(1111111111,9999999999)
?>
```

Возвращаясь к вопросу присваивания уникальных значений `OrderNo` для новых посетителей сайта и вопросу отслеживания этого номера при перемещении со страницы на страницу, мы можем с помощью переменной сеанса легко реализовать эти вещи. Делается это так: когда посетитель впервые появляется на сайте, мы присваиваем `session_id()` (или случайным образом созданный `ID`) как `OrderNo`, и, чтобы сделать этот номер доступным всем страницам, мы сохраняем его в `$_SESSION[]`, где он будет оставаться еще 20 минут после ухода посетителя с сайта. Конечно, когда посетитель решает стать заказчиком и оформляет покупку, мы просто "разрушим" этот сеанс с помощью `session_destroy()`, чтобы создать новый сеанс с другим номером заказа.

Отслеживание заказчиков с помощью массива $_SESSION[]

Итак, давайте применим все это для задачи присваивания уникального `ID` (переменной `OrderNo`) посетителю и отслеживания этого номера при перемещении посетителя со страницы на страницу.

`page1.php`

```
<?php
  session_start();
  $_SESSION[OrderNo] = session_id();
  header("Location:page2.php");
?>
```

Значение `id` сеанса доступно теперь на последующих страницах для задач отслеживания и идентификации. К нему можно обратиться с помощью `$_SESSION[OrderNo]`. Следующий пример показывает, как это значение будет извлекаться и выводиться на второй странице:

`page2.php`

```
<?php
  session_start();
  echo "Привет! Ваш идентификационный номер " . $_SESSION[OrderNo];
?>
```

Может показаться, что такая техника будет единственно разумным способом поддержания состояния между различными страницами. Однако, существуют потенциальные проблемы. Например, если во время посещения заказчик будет неактивен в течение 20 минут, то PHP автоматически завершит сеанс, и не будет никакой возможности восстановить действия этого человека. Это может произойти, например, если заказчика приглашают к телефону, он отвлекается на другой сайт или просто уснул. Будет создан совершенно новый сеанс, когда он вернется, и мы вынуждены будем искать способ удалить все ожидающие объекты покупки, так как нет способа завершить предыдущие транзакции.

Поэтому необходимо будет сопоставить преимущества использования сеансов с неудобством наличия брошенных сеансов. Тем не менее, разумно предположить, что большинство посетителей не будут создавать перерывы активности, превышающие 20 минут, и что простота программирования с помощью объекта сеанса превышает другие соображения. Как отмечено выше, всегда можно изменить 20-минутный интервал, если он покажется слишком коротким.

ID посетителя и номер заказа

В предыдущем примере мы присваивали номера заказов посетителям как способ уникальной идентификации посетителей, и, в конечном счете, для уникальной идентификации их заказов. Мы отчасти предполагаем, что все посетители станут заказчиками. Это не является проблемой, если мы собираемся использовать случайные числа в обоих случаях. Однако компания может иметь более систематический способ присваивания номеров заказов. Номера заказов могут, например, выбираться из последовательности предварительно определенных номеров. В этом случае присвоенный заранее номер заказа для посетителя, который не стал покупателем, оставляет пробел в последовательности номеров заказов.

В такой ситуации необходимо присваивать сначала случайное число в качестве "номера посетителя", чтобы отслеживать перемещения этого человека по сайту, а затем присвоить окончательный "номер заказа", когда будет принято решение о покупке. Из того, что можно увидеть в Web, можно заключить, что номера заказов не имеют очевидной логики, и поэтому можно использовать один и тот же номер для отслеживания посетителей и для идентификации их заказов.

Подсчет посетителей

Пока мы обсуждаем тему объекта сеанса, давайте рассмотрим процедуру, которая активно использует его возможности, - подсчет посетителей. *Объект* сеанса в действительности предопределяет "посетителя", поскольку поддерживает *сеанс* для каждого человека в течение 20 минут с момента последнего обращения посетителя к странице. Поэтому простое существование сеанса является доказательством присутствия посетителя. Итак, чтобы подсчитать посетителей, необходимо только добавлять 1 к счетчику посетителей, когда запускается *сеанс*.

Создание счетчика посетителей

Прежде всего, необходимо создать счетчик посетителей и инициализировать его значение. Обычно начальное значение счетчика задается как 0, но если есть желание произвести впечатление на посетителей или инвесторов, то можно инициализировать его значением 1000000. Счетчик может быть простым текстовым файлом, содержащим значение счетчика. Так как мы работаем в среде базы данных, то можно также создать таблицу. В этой таблице будет только одна запись с одним полем, в котором накапливается счетчик посетителей.

Здесь мы создаем таблицу с именем `Counters` (так как необходимо подумать о некоторых дополнительных счетчиках, которые понадобятся позже). Таблица содержит одно поле с именем VisitorCounter, определенное как числовое поле. В таблице имеется только одна эта запись. Теперь все готово к подсчету посетителей.

Увеличение значения счетчика посетителей

Когда посетитель прибывает на сайт, необходимо увеличить счетчик посетителей на 1. Мы узнаем, что прибыл посетитель, когда PHP создает сеанс и создает объект сеанса для этого человека. Поэтому мы создаем глобальную переменную сеанса с именем `"Count"` (`$_SESSION[Count]`) и будем увеличивать ее значение на 1 всякий раз, когда запускается новый сеанс. Это значение будет использоваться для увеличения `VisitorCounter`.

Чтобы не считать дважды одного и того же пользователя во время следующего сеанса, можно поместить код обновления счетчика внутрь оператора `if`, который проверяет, что переменной `Count` уже было присвоено значение. Если `Count` не определено, то это указывает на новый сеанс или на нового пользователя и

удовлетворяется условие для обновления счетчика `Visitor`. Если `Count` имеет значение, то текущий сеанс пользователя уже был учтен и блок обновления игнорируется.

Так как `home.php` является "входом" на сайт, давайте вернемся к файлу `home.php` и добавим необходимый код для учета посетителей:

```
session_start();

   if (!$_SESSION[Count])
   {
   $_SESSION[Count] = 1;
   $conn = odbc_connect('Driver={Microsoft Access Driver

(*.mdb)};DBQ=c:\inetpub\wwwroot\PHPTutorial\Ecommerce\databases\ecommerce.mdb
','','');
   $sql = "UPDATE Counters SET VisitorCounter = VisitorCounter +
$_SESSION[Count]";
   $rs = odbc_exec($conn, $sql)
   odbc_close($conn);
   }
```

Мы соединяемся и открываем таблицу `Counters` и добавляем 1 в поле `VisitorCounter`. Это действие происходит всякий раз, когда кто-то новый приходит на сайт. ПРИМЕЧАНИЕ: не забудьте сделать вызов функции `session_start()` перед проверкой статуса переменной сеанса `Count`. Если сеанс не запущен, `$_SESSION[Count]` не будет опознаваться как переменная сеанса, и оператор if будет выполняться каждый раз при перезагрузке страницы. Другими словами счетчик будет увеличиваться всякий раз, когда обновляется страница, а не тогда, когда создается новый сеанс.

Использование оператора SQL UPDATE

Использованный выше оператор SQL `UPDATE` имеют следующую общую форму:

```
UPDATE TableName SET FieldName1 = expression1 [,FieldName2 = expression2] ...
```

где `TableName` является именем таблицы, а *`FieldName`* указывает на поле таблицы, в которой происходит обновление. Выражения (`expression1,...`) могут быть одиночными значениями или арифметическими выражениями, создающими значение. Предложение `WHERE` можно добавлять к этому оператору для ограничения обновлений только определенными записями, соответствующими заданному критерию.

Вывод счетчика посетителей

Счетчик посетителей выводится на странице `home.php`. Для считывания и вывода значения счетчика используется следующий сценарий:

```
<?php
$conn = odbc_connect
  ('Driver={Microsoft Access Driver
(*.mdb)};DBQ=c:\inetpub\wwwroot\PHPTutorial\
     Ecommerce\databases\ecommerce.mdb','','');
$sql = "SELECT VisitorCounter FROM Counters";
$rs = odbc_exec($conn, $sql);
$Counts = odbc_result($rs,VisitorCounter);
echo "Вы являетесь $Counts посетителем с 12/2006";
?>
```

При тестировании этой процедуры счетчика на странице помните об одном важном факте. Счетчик не будет увеличиваться просто при перезагрузке страницы. Счетчик увеличивается только при появлении нового посетителя и создании нового сеанса. Поэтому, чтобы увеличить счетчик, понадобится закрыть браузер (завершить текущий сеанс) и снова его открыть (запуская новый сеанс).

Создание корзины покупателя

Нам потребуется *корзина* покупателя, с помощью которой заказчики могут выбирать продукты для покупки. Это делается с помощью кнопки `"Buy Now"` (Купить), появляющейся на странице `detail.php`. Когда происходит щелчок на кнопке, в корзину добавляется *запись*, *идентифицирующая* выбранный продукт. Выбранные продукты накапливаются, пока *покупатель* не оформит покупку, в этом случае объекты из корзины покупателя используются для подготовки заказа на покупку.

Создание таблицы корзины покупателя

Удобным способом создания корзины покупателя является использование таблицы базы данных. Записи добавляются в таблицу, когда заказчик выбирает объекты для покупки; записи удаляются из таблицы, когда заказчик решает не покупать выбранный ранее объект; и запись изменяется, когда заказчик выбирает другое количество товара для покупки. Таблица является динамической, ее записи добавляются, изменяются и удаляются во время процесса покупки.

Запись корзины покупателя должна содержать только минимальное количество информации. Она должна отслеживать заказчика, продукт и выбранное количество. Можно добавить дополнительные поля для поддержки других действий. Например, можно включить поле даты для отметки времени покупки (хорошая идея для очистки таблицы при отказе от покупки) и поле цены, если имеется риск, что цена объекта может измениться до окончательного оформления. Для текущих задач мы создаем следующую структуру таблицы `ShopCart`, которая добавляется в базу данных `eCommerce.mdb`.

ShopCart (таблица)			
поле Название	**поле Тип**	**поле Размер**	**поле Количество**
OrderNo	Text	10	Номер заказа
OrderItem	Text	6	Код продукта
OrderDate	date/Time		Дата заказа продукта
OrderQuantity	Number	*Long Integer*	Выбранное количество продукта

Добавление объектов в корзину покупателя

Выбор продукта и добавление его в корзину покупателя происходят на странице `detail.php`. Когда покупатель щелкает на кнопке `"Buy Now"`, информация о выбранном продукте записывается как новая запись в таблицу `ShopCart`. Объекты, накопленные в корзине покупателя, записываются как дополнительные покупки. После окончательного оформления и записи окончательного заказа корзину покупателя можно очистить от этих объектов.

Прежде всего, мы должны рассмотреть кнопку `"Buy Now"` на странице `detail.php` и запрограммировать ее для запуска процесса покупки.

```
<form action="detail.php?OrderItem=<?php echo $ItemNumber ?>" method="post">
  <input type="submit" class="buttonL" name="BuyButton" value="Buy Now"
    onMouseOver="OverMouse(this)"; onMouseOut="OutMouse(this)">
```

```
</form>
```

Атрибут ACTION, конечно, указывает на текущую страницу detail.php, кроме того, нам нужно передать на эту страницу, когда она перезагружается, информацию об ItemNumber продукта, добавляемого в корзину покупателя. Эта информация передается через строку запроса ?OrderItem=<?php echo $ItemNumber ?> (вспомните, что переменная ItemNumber доступна на этой текущей странице). Для передачи используется метод POST.

Здесь есть над чем подумать. Обычно теги <FORM> окружают поля данных, которые передаются, когда делается щелчок на кнопке отправки. Однако в данном случае полей формы нет - только одна кнопка. Поэтому, когда форма посылается, передаются только имя и значение кнопки. Однако нам надо передать значение ItemNumber продукта, выбранного на этой странице, чтобы его можно было добавить в корзину покупателя. Поэтому мы просто добавляем его как строку запроса в URL из ACTION. Теперь обе переменные, POST[BuyButton] (содержащая BuyButton) и $_GET[OrderItem] (содержащая OrderItem), становятся доступными на получающей странице. Отметим, что мы использовали имя строки запроса OrderItem для покупаемого продукта, чтобы избежать путаницы с ItemNumber рассматриваемого продукта.

Когда форма посылается, сценарий в начале страницы detail.php записывает объект в корзину покупателя.

```
<?php
session_start();
$ItemNumber = $_GET[ItemNumber];
$Category = $_GET[Category];
$Criterion =$_GET[Criterion];
$OrderNo = $_SESSION[OrderNo];
$OrderDate = date('n/d/y');

 if ($_POST[BuyButton] == "Buy Now")
  {

$OrderItem = $_GET[OrderItem];
$conn2 = odbc_connect('Driver={Microsoft Access Driver
(*.mdb)};DBQ=c:\inetpub\wwwroot\PHPTutorial\Ecommerce\databases\ecommerce.mdb
','','');
$sqlCart = "SELECT OrderNo,OrderItem FROM ShopCart WHERE OrderNo
='$OrderNo'";
$rsCart = odbc_exec($conn2,$sqlCart);

        while ($row = odbc_fetch_array($rsCart))
                {
                        if ($row[OrderNo] == $OrderNo && $row[OrderItem] ==
$OrderItem)
                {
                                $update = true;

                }

                }

        if (!$update)
        {
        $sqlInsert = "INSERT INTO ShopCart
(OrderNo,OrderItem,OrderDate,OrderQuantity) Values ('$OrderNo',
'$OrderItem','$OrderDate,1)";
        $rsInsert = odbc_exec($conn2,$sqlInsert);
            }
          else
          {
```

```
          $sqlUpdate = "Update ShopCart SET OrderQuantity = OrderQuantity + 1
WHERE OrderNo = '$OrderNo' AND
          OrderItem = '$OrderItem'";
          $rsUpdate = odbc_exec($conn2,$sqlUpdate);
          }
   header("Location:shopcart.php");
  }
?>
```

Пример I.11.

Новая запись `ShopCart` заполняется четырьмя значениями. `OrderNo` получают из объекта сеанса — `$_SESSION[OrderNo]`. `OrderItem` передается на эту страницу через строку запроса. `OrderDate` определяется текущей датой системы, полученной с помощью функции `date()`. `OrderQuantity` задается как 1. Когда объект выбирается в начале для покупки, мы автоматически задаем заказанное количество как 1. Покупатель может изменить это количество на странице корзины покупателя.

Мы имеем здесь также дело с ситуацией, в которой заказчик может попытаться купить один и тот же продукт несколько раз, намеренно или по неосторожности. Наше "бизнес-правило" говорит, что единовременно может быть заказан только один объект (хотя покупатели будут иметь возможность изменить заказанное количество на странице корзины покупателя). Мы не хотим, чтобы один и тот же объект появлялся несколько раз в корзине покупателя.

Такая ситуация легко решается начальной проверкой того, что запись для этого продукта уже существует в таблице `ShopCart`. Оператор SQL `SELECT` и цикл по возвращаемому множеству записей выполняют нужную проверку:

```
$sqlCart = "SELECT OrderNo,OrderItem FROM ShopCart WHERE OrderNo
='$OrderNo'";
$rsCart = odbc_exec($conn2,$sqlCart);

        while ($row = odbc_fetch_array($rsCart))
                {
                        if ($row[OrderNo] == $OrderNo && $row[OrderItem] ==
$OrderItem)
                {
                                $update = true;

                }

                }
```

Если найдены подходящие `OrderNo` и `OrderItem`, то потребуется оператор SQL `UPDATE` для обновления существующей записи. Если совпадений не найдено, требуется оператор SQL `INSERT` для создания новой записи. Для этого создается переменная (`$update`), которой присваивается значение `true`, если будут найдены подходящие `OrderNo` и `OrderItem`. Следующий блок кода содержит операторы `INSERT` и `UPDATE`. Значение переменной `$update` зависит от используемого оператора SQL:

```
if (!$update)
        {
        $sqlInsert = "INSERT INTO ShopCart
(OrderNo,OrderItem,OrderDate,OrderQuantity)
        Values ('$OrderNo', '$OrderItem',$OrderDate,1)";
        $rsInsert = odbc_exec($conn2,$sqlInsert);
            }
          else
          {
          $sqlUpdate = "Update ShopCart SET OrderQuantity = OrderQuantity + 1
WHERE OrderNo =
          = '$OrderNo' AND
          OrderItem = '$OrderItem'";
```

```
        $rsUpdate = odbc_exec($conn2,$sqlUpdate);
        }
   header("Location:shopcart.php");
```

После добавления новой записи множество записей и соединение закрываются, и покупатель посылается прямо на страницу корзины покупателя с помощью метода `header("Location:url")`. Предполагается, что покупатель захочет после выбора просмотреть объекты корзины покупателя. Дальше мы перейдем к программированию страницы `shopcart.php`.

Обслуживание корзины покупателя

Страница `shopcart.php` предоставляет заказчику *доступ* к объектам корзины покупателя. Она суммирует его покупки, позволяет изменить заказанное количество и является переходом к оформлению заказа. При программировании этой страницы существуют некоторые сложности, поэтому будьте внимательны.

Вывод корзины покупателя

Следующий код создает вывод корзины покупателя. Этот код появляется в основном разделе страницы, чтобы создать приведенный выше вывод. Отметим, что весь раздел включает форму, которая пересылается снова на эту же страницу, чтобы можно было изменить заказанное количество единиц товара.

```
<div style="position:absolute; top:75px; left:200px; width:550px">

<form name="ShopCart" action="shopcart.php" method="post">

<span class="head1">Shopping Cart</span><br>
<br>
<span class="head4">Date: </span><?php echo date('n/d/y')?><br>
<span class="head4">Order No: </span><?php echo $_SESSION[OrderNo] ?><br>

<table border="0">
<tr>
  <th>Item Number</th>
  <th>Title</th>
  <th>Quantity</th>
  <th>Price</th>
  <th>Amount</th>
</tr>

<tr>
    <td colspan="4" style="text-align:right">Shipping </td>
    <td style="text-align:right">$<?php echo number_format($OrderShipping,2)
?></td>
```

```
</tr>
<tr>
    <th colspan="4" style="text-align:right">Order Total </th>
    <td style="border-style:solid"><b>$<?php echo
number_format($OrderTotal,2) ?></b></td>
</tr>
</table>

<?php if($OrderTotal != 0) { ?>
  <div style="width:375px; line-height:8pt">
  <input type="submit" name="UpdateButton" class="buttonM"
    style="float:left; margin-right:5px" value="Update"
    onMouseOver="OverMouse(this)"; onMouseOut="OutMouse(this)">
  <span class="small">Click to update changed quantities. Enter new quantity
  or enter 0 to cancel purchase of item. (Щелкните, чтобы обновить
измененные количества. Введите новое количество или введите 0, чтобы отменить
покупку товара.)</span>
  </div>
<?php } ?>

</form>

<?php if ($OrderTotal != 0) { ?>
  <div style="width:375px; line-height:8pt">
  <form action="http://.../creditcheck.php" method="post">
    <input type="hidden" name="ReturnURL"
value="http://.../ordercapture.php">
    <input type="hidden" name="CompanyID" value="softWarehouse.com">
    <input type="hidden" name="CustomerID" value="<?php echo
$_SESSION[OrderNo]?>">
    <input type="hidden" name="Amount" value="<?php echo $OrderTotal ?>">
    <input type="submit" name="CheckoutButton" class="buttonM"
      style="float:left;margin-right:5px" value="Checkout"
      onMouseOver="OverMouse(this)"; onMouseOut="OutMouse(this)">
    <span class="small">Click to finalize on-line purchase through secure
connection
    to Credit Payment Systems. Щелкните здесь, чтобы оформить онлайн покупку
через безопасное соединение с системой оплаты по кредитным картам.</span>
  </form>
  </div>
<?php } ?>

</div>
```

Пример I.12.

Первая часть кода создает заголовки страницы и заголовки таблицы для списка корзины покупателя. Вместе с номером заказа покупателя, который содержится в переменной `$_SESSION[OrderNo]`, выводится текущая системная дата

Извлечение объектов корзины покупателя

Объекты в корзине покупателя заказчика соответствуют номеру заказа и извлекаются с помощью следующей команды SQL:

```
$sql="SELECT OrderItem, OrderQuantity FROM ShopCart WHERE OrderNo =
'$_SESSION[OrderNo]'";
```

Получаемое множество записей `$rsCart` содержит для каждой записи два выбранных поля `OrderItem` и `OrderQuantity`. Эти поля присваиваются переменным. Два других поля в записях (`OrderNo` и `OrderDate`) не требуются для нашей цели. Теперь необходимо выполнить итерации по записям и создать таблицу строк с выбранными объектами.

Однако мы еще не получили всю информацию, которая выводится в строке. Кроме `OrderItem` и `OrderQuantity`, содержащихся в корзине покупателя, таблица выводит название товара и его цену. Эти значения находятся в таблице `Products`. Поэтому эти поля `ItemTitle` и `ItemPrice` извлекают для соответствующего продукта, создавая множество записей `$rsProd` и выполняя оператор SQL.

```
$sqlProd = "SELECT ItemTitle, ItemPrice FROM Products WHERE ItemNumber =
'$OrderItem'";
```

Эти значения также присваиваются переменным.

Для каждой выводимой строки объекта выводится также объем покупки. Переменная $OrderAmount вычисляется умножением заказанного количества (`$OrderQuantity`) на цену продукта (`$OrderPrice`). Также, когда сценарий выполняет итерации по записям корзины покупателя, значение `$OrderTotal` накапливается путем сложения по очереди всех `$ItemAmount`. Отметим, что переменная `OrderTotal` инициализируется как 0 перед началом *цикла вывода*.

Теперь собрана вся информация, необходимая для вывода строки таблицы:

```
echo "<tr>
    <td>$OrderItem </td>
    <td>$OrderTitle</td>
    <td><input type=\"text\" name=\"Q$OrderItem\" size=\"2\" class=\"qtybox\"
         value=\"$OrderQuantity\"></td>
    <td style=\"text-align:right\">$$OrderPrice</td>
    <td style=\"text-align:right\">$$OrderAmount</td>
  </tr>";
```

Мы выводим значения переменных, содержащих извлеченные из таблиц `ShopCart` и `Products` данные и вычисленный объем. В случае объектов `$OrderPrice` и `$OrderAmount` используется функция PHP `number_format()`, чтобы вывести значения в формате с двумя десятичными позициями после запятой. Теперь надо объяснить, что происходит с полем `OrderQuantity`.

Именование полей количества

Раздел страницы заключен в теги `<FORM>`, так как заказчику дается возможность изменить заказанное количество товара и заново послать форму, чтобы обновить это количество. Для этого необходимо поместить `$OrderQuantity` для каждого покупаемого объекта в поле формы, как это делается с помощью кода:

```
<input type="text" name="Q<?php echo $OrderItem ?>" size="2" class="qtybox"
    value="<?php echo $OrderQuantity ?>">
```

Ключом к тому, чтобы эта форма правильно работала, является соглашение об именах, используемых для этого поля. Обратим внимание, что этому текстовому полю присваивается имя `"Q<?php echo $OrderItem ?>"`, то есть буква `"Q"` (которая выбрана просто произвольно), — за ней следует код (`$OrderItem`) продукта, значение `$OrderQuantity` которого выводится в этом поле. Если взять в качестве примера корзину покупателя, показанную выше, то именем поля количества первого товара будет `"QBU1111"`, а именем поля для второго товара будет `"QOS1111"`.

Для использования такого способа именования имеются две причины. Прежде всего, мы не знаем, сколько различных полей количества появится в форме. В ней будет столько полей, сколько будет купленных продуктов. Поэтому в отличие от предыдущего опыта с формами, мы не можем заранее определить и назвать эти поля. Во-вторых, мы не можем использовать одно имя для всех полей. Если, например, мы бы назвали поле просто `"Quantity"`, то было бы столько полей `Quantity`, сколько было бы купленных продуктов. Это могло бы создать реальные, непреодолимые проблемы при попытке определить, какое поле `quantity` связано с каким продуктом. Поэтому требуется соглашение об именах, которое, первое, позволяет определить по мере необходимости неопределенное число полей количества, и, второе,

уникальным образом определяет продукт, связанный с количеством, представленным в этом поле. Текущее соглашение об именах решает обе проблемы достаточно хорошо. Символ `"Q"` используется для идентификации поля количества (важный идентификатор, который будет использоваться ниже), а добавленное `OrderItem` предоставляет уникальное имя, которое идентифицирует также продукт, связанный с указанным в поле количеством. Мы обсудим ниже, как сценарий сортирует эту информацию для обновления полей количества измененными значениями.

Суммарные строки заказа и кнопки

После того как сценарий закончил итерации по объектам корзины покупателя и перечислил все покупки, вычисляется стоимость доставки (`$OrderShipping`) как 2% от `OrderTotal` и выводится пара суммарных строк.

```
$OrderShipping = number_format($OrderTotal * .02,2);
$OrderTotal = number_format($OrderTotal + $OrderShipping,2);
<tr>
    <td colspan="4" style="text-align:right">Shipping </td>
    <td style="text-align:right"><?php echo $OrderShipping ?></td>
</tr>
<tr>
    <th colspan="4" style="text-align:right">Order Total </th>
    <td style="border-style:solid"><b><?php echo $OrderTotal ?></b></td>
</tr>
```

Наконец, выводятся две кнопки: кнопка `"Update"` для отправки всех сделанных изменений количества продуктов, и кнопка `"Checkout"` для оформления заказа. Последняя кнопка обсуждается дальше вместе с оформлением покупки с помощью кредитной карты.

Заказчики могут посещать страницу `shopcart.php` в любое время, даже до того, как будут сделаны какие-либо покупки. Однако, когда это происходит, вывод кнопок или доступ к ним для посетителя не обязателен. Они нужны только тем заказчикам, которые выбрали продукты для покупки. Поэтому их вывод происходит только для тех заказчиков, которые имеют объекты в своей корзине покупателя, т.е. когда значение `OrderTotal` не равно $0.00. Это условие используется для определения, будут или нет выводиться кнопки.

```
<?php if($OrderTotal != 0) {?>
  <div style="width:375px; line-height:8pt">
  <input type="submit" name="UpdateButton" class="buttonM"
    style="float:left; margin-right:5px" value="Update"
    onMouseOver="OverMouse(this)"; onMouseOut="OutMouse(this)">
  <span class="small">Щелкните, чтобы обновить измененные значения
количества. Введите новое значение количества или введите 0, чтобы отменить
покупку товара.</span>
  </div>
<?php } ?>

</form>

<?php if ($OrderTotal != 0) {?>
  <div style="width:375px; line-height:8pt">
  <form action="http://.../creditcheck.php" method="post">
    <input type="hidden" name="ReturnURL"
value="http://.../ordercapture.php">
    <input type="hidden" name="CompanyID" value="WebWarehouse.com">
    <input type="hidden" name="CustomerID" value="<?php echo
$_SESSION[OrderNo]?>">
    <input type="hidden" name="Amount" value="<?php echo $OrderTotal ?>">
    <input type="submit" name="CheckoutButton" class="buttonM"
      style="float:left;margin-right:5px" value="Checkout"
      onMouseOver="OverMouse(this)"; onMouseOut="OutMouse(this)">
```

```
    <span class="small">Щелкните, чтобы оформить онлайн покупку с помощью
защищенного соединения с кредитной платежной системой.</span>
  </form>
  </div>
<?php } ?>
```

Пример I.13.

Итерации по отправленной форме

Когда пользователь изменяет значение количества одного или нескольких продуктов и щелкает на кнопке `"Update"` для отправки формы, то информация снова передается на страницу `shopcart.php`. Страница `shopcart.php` содержит сценарий для обновления этих изменений в таблице `ShopCart`. Этот сценарий находится в начале страницы для перехвата изменений, когда посылается форма.

Назначение этого сценария состоит в обновлении таблицы `ShopCart` значениями количества продуктов, переданных из формы. Мы знаем, что имеется неопределенное число полей количества, имя каждого из которых начинается с буквы `"Q"`, и мы знаем, что значения формы, приходящие на страницу, могут быть в любом порядке.

До сих пор мы всегда знали имена и номера полей на формах. Это связано с тем, что мы проектировали формы, именовали поля и определяли, сколько будет полей. Но здесь это не так. Мы не знаем, сколько имеется полей (это зависит от того, сколько объектов имеется в корзине покупателя), и мы не знаем их имен (за исключением того, что они начинаются с буквы `"Q"`). Поэтому мы не можем выполнить обработку формы, как это делалось ранее.

Однако с помощью PHP можно решить эту проблему. Можно выполнить итерации по всем элементам $*_REQUEST*, чтобы определить имена отправленных полей и соответствующие значения. Это реализуется с помощью специального оператора цикла For Each...Next, общая структура которого применительно к формам имеет следующий вид:

```
foreach($_REQUEST as $key => $value)
```

где `$_REQUEST` является ассоциативным массивом, содержащим все пары имя/значение для элементов управления формы на странице. Этот цикл выполняется для каждой пары имя/значение, присланной из формы. На каждой итерации переменная `$key` содержит имя из пары имя/значение, а `$value` указывает на значение, связанное с этим именем. Массив имеет следующий формат `$_REQUEST[$key] = $value` для каждой пары имя/значение, связанной с формой.

Затем можно использовать эту итерационную структуру для просмотра присланных позиций для обновления корзины покупателя и определения имен полей и значений. Когда встречается поле с именем, начинающимся с буквы `"Q"`, то мы знаем, что это поле количества, содержащее значение для обновления таблицы `ShopCart`. Мы можем также определить из названия продукт, для которого задано значение количества.

Обновление корзины покупателя

Ниже показан весь сценарий обновления корзины покупателя, когда заказчик щелкает на кнопке формы `"Update"`.

```
if ($_POST[UpdateButton]  == "Update")

 {

$conn2 = odbc_connect('Driver={Microsoft Access Driver
(*.mdb)};DBQ=c:\inetpub\wwwroot\PHPTutorial\Ecommerce\databases\ecommerce.mdb
','','');

foreach($_REQUEST as $key => $value)

        {
```

```
        if (strpos($key,"Q") === 0 )

        {

                $OrderItem = substr($key,1);
                $OrderQuantity = $value;

                if (is_numeric($OrderQuantity))

                {

                        if ($OrderQuantity == 0)

                        {

                        $sqlCartUpdate = "DELETE FROM ShopCart WHERE
OrderNo='$_SESSION[OrderNo]' AND OrderItem='$OrderItem'";

                        }

        else

                        {

          $sqlCartUpdate = "UPDATE ShopCart SET
OrderQuantity='$OrderQuantity' WHERE OrderNo='$_SESSION[OrderNo]'
          AND OrderItem='$OrderItem'";

                        }
                $rsCartUpdate = odbc_exec($conn2,$sqlCartUpdate);

                }
        }

        }
                odbc_close($conn2);
        }
```
Пример I.14.

Прежде всего, создается соединение с базой данных `eCommerce.mdb`. Объект множества записей (`Recordset`) здесь не требуется, так как обновления делаются непосредственно командами SQL, выполняемыми с помощью объекта соединения (`Connection`).

Затем применяется цикл `foreach` для итераций по именам и значениям, помещенным в массив `$_REQUEST[]` после отправки формы на сервер. Переменная с именем `$key` используется для ссылки на имена полей, находящиеся в массиве.

При просмотре элементов массива выполняется проверка того, что имя поля начинается с буквы `"Q"`:

```
if (strpos($key,"Q") === 0 ) {
}
```

Функция `PHPstrpos()` применяется для проверки, что первый символ (или символ в позиции 0) имени элемента управления формы совпадает с `"Q"`. Если первый символ `"Q"`, то это будет поле количества, содержащее значение для обновления корзины покупателя. Здесь важно отметить, что используется оператор сравнения PHP "===" ("идентично"), а не оператор сравнения "==" ("равно"). Это необходимо, потому что функция *`strpos()`* требует для проверки своих возвращаемых значений "===".

Затем сценарий определяет, какую надо обновить запись о продукте и каким является фактическое значение количества:

```
$OrderItem = substr($key,1);
$OrderQuantity = $value;
```

Мы знаем, что код продукта, связанный с полем количества, содержится в переменной `$key`. Фактически это будут шесть правых символов имени поля. Поэтому эти шесть символов извлекаются из переменной с помощью функции PHP `substr()` и помещаются в переменную `$OrderItem`. Затем значение переменной `$value`, связанное с этим полем, присваивается переменной `$OrderQuantity`. Теперь у нас есть два фрагмента информации, необходимых для обновления поля `$ItemQuantity` этого продукта в таблице `ShopCart`.

Прежде чем обновлять количество продукта, необходимо проверить, что заказчик ввел число. В поле можно случайно ввести недействительный символ. Поэтому для `$OrderQuantity` выполняется числовая проверка.

```
if (is_numeric($OrderQuantity))
{
}
```

Если в поле находится что-то отличное от числа, то обновление количества для этого продукта пропускается и происходит обращение к следующему полю формы.

Количества, присланные с помощью формы, могут представлять дополнительные купленные количества, или они могут иметь значение 0, указывающее, что товар удален из корзины покупателя. Это значение проверяется, и выполняется один из двух операторов SQL.

```
if ($OrderQuantity == 0)
{
$sqlCartUpdate = "DELETE FROM ShopCart WHERE OrderNo='
$_SESSION[OrderNo]' AND OrderItem='$OrderItem'";
}

else
{
$sqlCartUpdate = "UPDATE ShopCart SET OrderQuantity='$OrderQuantity'
WHERE OrderNo='$_SESSION[OrderNo]' AND OrderItem='$OrderItem'";
}

$rsCartUpdate = odbc_exec($sqlCartUpdate,$conn2);
```

В случае удаления или обновления, действие применяется к записи `ShopCart` с полем `OrderNo` соответствующим текущему `$_SESSION[OrderNo]`, и с полем `$OrderItem` соответствующим `$OrderItem` из отправленной формы. Оператор `DELETE` удаляет из таблицы всю запись; оператор `UPDATE` изменяет поле `$OrderQuantity` в таблице на значение `$OrderQuantity` из отправленной формы.

Сценарий выполняет итерации по всем позициям, отправленным через форму, проверяя имя поля на наличие символа `"Q"`. Если символ найден, то из имени извлекается код продукта и присваивается вместе со значением количества переменным, которые используются при обновлении таблицы `ShopCart`. После завершения обработки массива `$_REQUEST[]` соединение с базой данных закрывается и сценарий заканчивается.

Итерации по массиву $_REQUEST[]

Может быть полезно использовать пример с отправленными значениями для визуализации массива `Request.Form`. Предположим, что корзина покупателя содержит объекты, показанные на иллюстрации выше, тогда при нажатии кнопки `"Update"` будет создан следующий массив `Request.Form`.

Массив $_*REQUEST*[]	
Имя	**Значение**
QOS1111	2
UpdateButton	Update
QBU1111	1

Здесь необходимо отметить один принципиальный момент, а именно, что элементы массива не обязательно находятся в том же порядке, в котором они появляются в форме; кроме того, нажатая кнопка также является частью массива. Рассмотрим теперь итерации по элементам с помощью цикла `foreach`, в котором переменная `$key` указывает имя, а `$value` указывает значение:

```
foreach($_REQUEST as $key => $value) {

  $key = "QOS1111"
  $value = "2"
  $key = "UpdateButton"
  $value = "Update"
  $key = "QBU1111"
  $value = "1"
}
```

Здесь можно видеть элементы данных, с которыми должен иметь дело сценарий, и можно лучше понять действие сценария по извлечению кода продуктов и обработке количества продуктов.

Покупка онлайн

Когда заказчик завершил выбор программного обеспечения для покупки и готов оплатить продукты, он нажимает кнопку `"Checkout"` ("Оплатить"), чтобы завершить свою покупку.

Обычно заказчики соединяются с онлайновой процессинговой компанией кредитных карт, которая получает информацию о кредитной карте, санкционирует покупку и соединяется с национальной банковской системой, чтобы дебетовать и кредитовать соответствующие банковские счета. Когда эти транзакции завершаются, заказчик возвращается на *сайт* магазина, где получает подтверждение своего заказа.

В то же самое время компания по обслуживанию кредитных карт возвращает на *сайт* полученную от заказчика несекретную информацию, имеющую *отношение* к выставлению счета. Эта *информация* затем может использоваться для создания постоянной записи заказа для этого клиента для будущих ссылок и для воссоздания исходного заказа, если потребуется. Возвращаемая *информация* может, на самом деле, применяться для соединения с системами бухгалтерского учета и склада компании. Хотя мы не можем соединиться с федеральной банковской системой для этого модельного приложения, мы можем достаточно точно промоделировать обработку в компании по обслуживанию кредитных карт и можем архивировать информацию о заказе, даже хотя и не будем соединяться непосредственно с какими-либо другими системами.

Отправка информации о заказе

Страница `shopcart.php` содержит кнопку `"Checkout"` в отдельной форме, отличной от той, которая используется для обновления покупаемого количества товара. Это связано с тем, что форма оформления передается компании по обслуживанию кредитных карт, в то время как форма обновления снова соединяется со страницей `shopcart.php`.

```
<?php if ($OrderTotal != 0) {?>

  <div style="width:375px; line-height:8pt">
  <form action="https://.../creditcheck.php" method="post">
    <input type="hidden" name="ReturnURL"
value="https://..../ordercapture.php">
    <input type="hidden" name="CompanyID" value="Webwarehouse.com">
    <input type="hidden" name="CustomerID" value="<?php echo
$_SESSION[OrderNo]?>">
    <input type="hidden" name="Amount" value="<?php echo $OrderTotal ?>">
    <input type="submit" name="CheckoutButton" class="buttonM"
      style="float:left;margin-right:5px" value="Checkout"
      onMouseOver="OverMouse(this)"; onMouseOut="OutMouse(this)">
    <span class="small">Щелкните, чтобы оплатить онлайн покупку через
защищенное соединение с Системами оплаты с помощью кредитных карт.
   </span>

  </form>
  </div>

<?php }
```

Пример I.15.

Для целей этого примера `ACTION URL` формы будет

```
../php/ecommerce/creditcheck.php
```

Этот адрес соединяется с модельной компанией обслуживания кредитных карт, которая является просто страницей в каталоге сайта. Но можно точно так же просто соединиться с реальной компанией.

Обычный метод взаимодействия с компанией по обслуживанию кредитных карт состоит в передаче информации, связанной с текущей покупкой с помощью множества скрытых полей формы. Как минимум компании необходимо знать: 1) общую стоимость покупки, 2) идентификатор сайта, присылающего информацию (`ID` счета в компании по обслуживанию кредитных карт), и 3) URL страницы, на которую будет возвращено подтверждение о выполнении транзакции. Сайту продаж необходимо также отправить 4) идентификатор заказчика, который может возвращаться вместе с подтверждением. Эти четыре скрытых поля содержатся в форме с кнопкой `"Checkout"`.

Поле `ReturnURL` используется для предоставления адреса, в который возвращается информация компании кредитных карт. В данном примере информация возвращается на страницу `ordercapture.php` по адресу

`../php/ecommerce/ordercapture.php`

Заказчик возвращается из компании кредитных карт на эту промежуточную страницу, прежде чем попасть на страницу `salesorder.php`, на которой представляется окончательный заказ на покупку и подтверждающее сообщение. На этой странице `ordercapture.php` мы получаем возвращаемую из компании кредитных карт информацию для создания записи о заказе на продажу и очистки корзины покупателя. Заказчик не видит эту страницу. Она содержит только сценарий PHP для обработки возвращаемой информации и затем автоматически переадресуется на страницу `salesorder.php` для вывода заказчику. Позже мы рассмотрим это подробнее.

Поле `CompanyID` содержит идентификатор счета сайта, отправившего заказ. В качестве такого идентификатора мы применяем `"Webwarehouse.com"`. При использовании этого URL для проверки кредитных карт можно ввести в это поле любую текстовую строку.

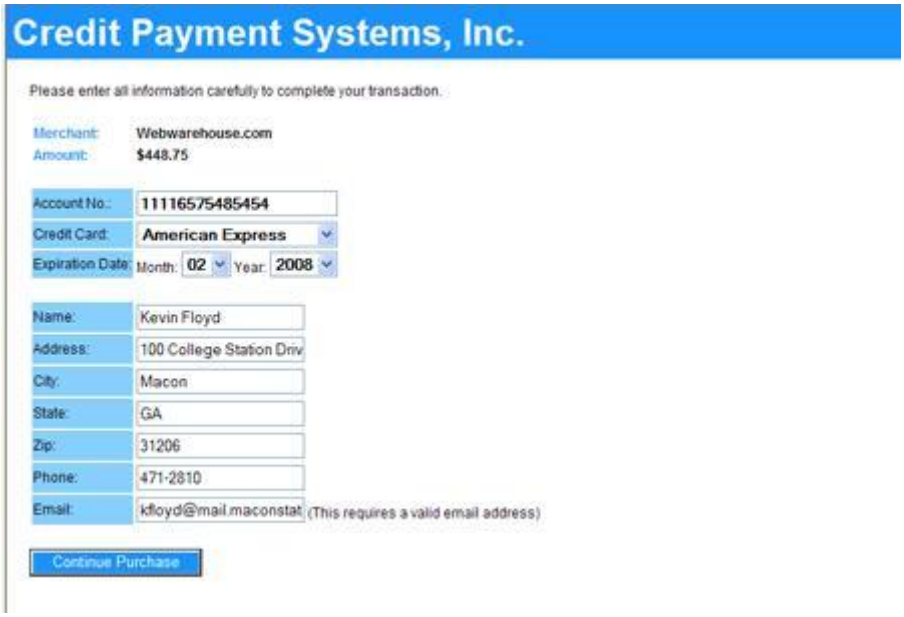
Поле `CustomerID` содержит идентификатор заказчика. Это значение возвращается компанией кредитных карт, чтобы мы знали, какому заказчику или какому заказу соответствует возвращаемая информация. Здесь для идентификации заказчика используется значение `$_SESSION[OrderNo]`.

Поле `Amount` содержит общую стоимость заказа. Значение этого поля задается с помощью переменной `OrderTotal`, которая доступна на этой странице.

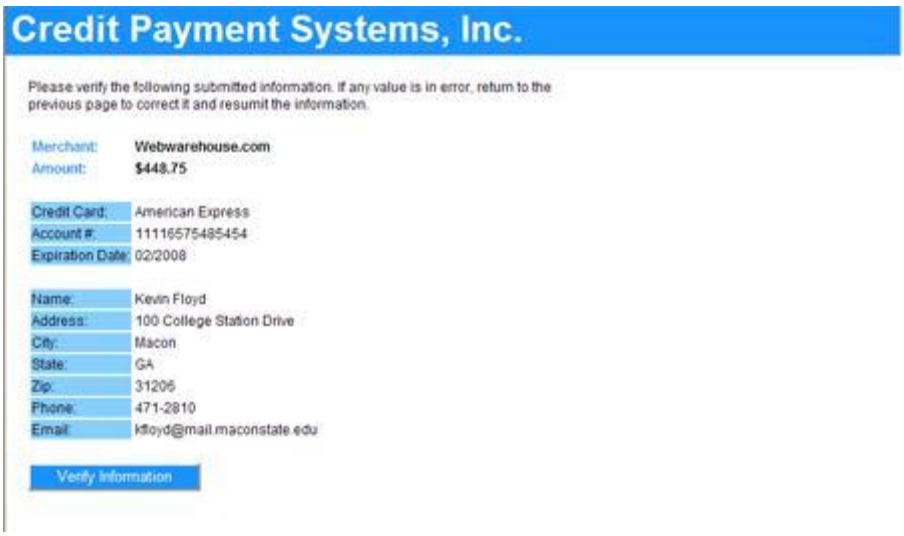
После отправки формы остается только подождать. Компания кредитных карт выполняет в это время обработку. Когда компания завершит обработку, она автоматически создает адрес `ReturnURL` для возврата на страницу `ordercapture.php`, где продолжается обработка.

Обработка кредитных карт

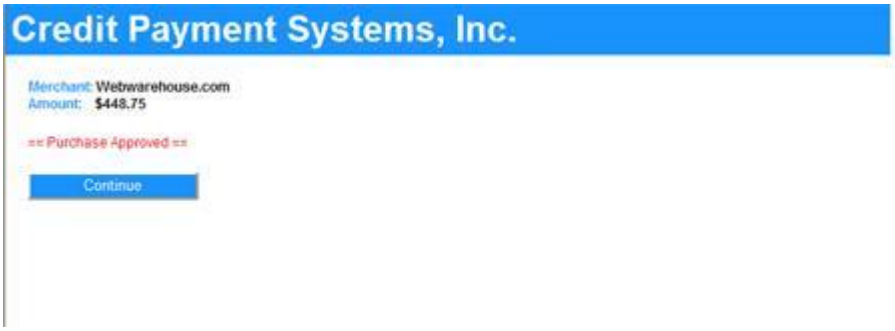
Когда заказчик направляется в компанию кредитных карт, первая представленная форма появляется в следующем виде:



Заказчик заполняет информацию о кредитной карте и выставленном счете и щелкает на кнопке `"Continue Purchase"`. Форма проверяется, чтобы убедиться, что имеется вся информация, и затем выводится страница подтверждения:



Когда заказчик проверяет информацию, щелкая на кнопке `"Verify Information"`, проверяется информация о кредитной карте. Используется следующая процедура: если первые четыре цифры номера счета (`Account`) будут "0000", то заказ отвергается, если первые четыре цифры будут любыми другими цифрами, то заказ принимается. После обработки заказчик видит экран с подтверждением или отказом:



Когда заказчик щелкает на кнопке `"Continue"`, делается *обратная ссылка* на страницу, определенную в отправляемой форме как `ReturnURL`. Эта страница, в примере `ordercapture.php`, получает информацию о заказе и выставленном счете, которая была получена от заказчика. Ее получение и обработка обсуждаются далее.

Запись информации о продажах

Подтверждение покупки и информация о выставленном заказчику счете возвращается из компании кредитных карт на страницу, указанную в `ReturnURL`, представленной формы. Это будет в нашем случае страница `ordercapture.php`. Информация посылается как скрытые поля и доступна на этой странице в массиве `$_POST[]` с помощью следующих имен:

```
$_POST[Approval]        true (заказ принят) или false (заказ отвергнут)
$_POST[Amount]          Общая сумма заказа в долларах
$_POST[CustomerID]      Номер заказа заказчика
$_POST[Name]            Имя в счете к оплате заказчика
$_POST[Address]         Адрес в счете к оплате заказчика
$_POST[City]            Город в счете к оплате заказчика
$_POST[State]           Штат в счете к оплате заказчика
$_POST[Zip]                     Zip-код в счете к оплате заказчика
$_POST[Phone]           Номер телефона заказчика
$_POST[Email]           Адрес e-mail заказчика
```

Страница `ordercapture.php` является страницей на PHP без какого-либо кода XHTML. Эта страница не выводится заказчику, но используется для выполнения фоновой обработки возвращаемой информации, прежде чем направить заказчика на страницу `salesorder.php`.

Существенная часть работы, которая должна быть выполнена на этой странице, состоит в фиксации информации о заказа на продажу, чтобы о нем существовала постоянная запись. Эту запись можно использовать для восстановления заказа, если понадобится, и для предоставления информации для взаимодействия с системами бухгалтерии и склада (если они существуют, так как в данном примере они отсутствуют).

Записи заголовка заказа и деталей заказа

Традиционный способ отслеживания заказов на продажу состоит в создании двух типов записей о продажах. Запись заголовка заказа содержит идентификационную информацию заказа, такую как номер заказа, дату заказа, и информацию о счете к оплате заказчика. Для каждого заказа существует одна запись заголовка заказа. Запись деталей заказа содержит детали о купленных товарах - код товара, количество, и цену. Существует одна запись деталей заказа для каждого купленного товара. Такая информация поддерживается в этих файлах, чтобы можно было восстановить исходный заказ.

Файлы заголовка заказа и деталей заказа являются таблицами базы данных, содержащимися в базе данных `eCommerce.mdb`. Их формат показан ниже.

OrderHeader (таблица)		
Имя поля	**Тип поля**	**Размер поля**
OrderNo	Text	10
OrderDate	Date/Time	
CustomerName	Text	50
CustomerAddress	Text	50
CustomerCity	Text	50
CustomerState	Text	2
CustomerZip	Text	10
CustomerPhone	Text	15

CustomerEmail	Text	50
OrderDetail (таблица)		
Имя поля	**Тип поля**	**Размер поля**
OrderNo	Text	10
ItemNumber	Text	6
ItemTitle	Text	50
ItemPrice	Валюта	с 2 знаками после запятой
ItemQuantity	Числовое	*long integer*

Получение информации заказа

Первая часть сценария на странице `ordercapture.php` собирает информацию, посланную из компании кредитных карт через массив `$_POST[]` и присваивает ее переменным для упрощения записи кода.

```
<?php
'—Сбор информации, возвращаемой компанией кредитных карт
$Approval = $_POST[Approval];
$Amount = $_POST[Amount];
$OrderNo = $_POST[CustomerID];
$Name = $_POST[Name];
$Address = $_POST[Address];
$City = $_POST[City];
$State = $_POST[State];
$Zip = $_POST[Zip];
$Phone = $_POST[Phone];
$Email = $_POST[Email]; ...
```

Компания кредитных карт посылает флаг `Approval`, задаваемый как `"True"` или `"False"` в зависимости от того, был подтвержден заказ или нет. Если заказ подтвержден, необходимо сохранить информацию о заказе в таблицах `OrderHeader` и `OrderDetail` ; если заказ не был подтвержден, то информация не сохраняется, так как продажа не была сделана. В любом случае, однако, необходимо очистить корзину покупателя этого заказа. Не существует пока никаких ожидающих заказов.

Общая логика сценария для обработки этих задач обработки показана ниже.

```
if ($Approval) {

  '—Создать запись OrderHeader

  '—Создать запись OrderDetail
```

```
}

'—Создать таблицу ShopCart
```

Создание записи OrderHeader

Сценарий начинается с открытия соединения с базой данных `eCommerce`. Это соединение необходимо для очистки корзины покупателя, независимо от того, будут или нет созданы записи заказа. Затем, если было получено подтверждение от компании кредитных карт, сценарий создает запись `OrderHeader`.

```
$conn = odbc_connect('Driver={Microsoft Access Driver
(*.mdb)};DBQ=c:\inetpub\wwwroot\PHPTutorial\Ecommerce\
  databases\ecommerce.mdb','','');
if ($Approval)
{
$sqlInsert = "INSERT INTO OrderHeader (OrderNo,OrderDate,CustomerName,
  CustomerAddress,CustomerCity,CustomerState,
    CustomerZip,CustomerPhone,CustomerEmail)
Values('$OrderNo','$Date','$Name',
  '$Address','$City','$State','
$Zip','$Phone','$Email')"; $rsInsert = odbc_exec($conn,$sqlInsert);
}
```

Этот процесс состоит просто в создании новой записи в таблице `OrderHeader` и копировании полей, посланных из компании кредитных карт в новую запись. Поле `OrderDate` заполняется текущей системной датой. После того как запись заполнена и обновлена, множество записей закрывается.

Создание записей OrderDetail

Затем сценарий создает одну или несколько записей в таблице `OrderDetail`. Информация для этой таблицы, однако, приходит не из компании кредитных карт. Часть ее находится в таблице `ShopCart`, а часть — в таблице `Products`. Мы, фактически, переносим детали текущего заказа в подобное множество записей `OrderDetail`. Мы начинаем сценарий с создания трех множеств записей, необходимых для работы: множество записей `$RSDetail` для доступа к таблице `OrderDetail`, множество записей $RSShopCart для доступа к таблице `ShopCart` и множество записей `$RSProducts` для доступа к таблице `Products`.

```
'—Создание записи OrderDetail
$sqlDetail ="SELECT * FROM OrderDetail WHERE NULL";
$rsDetail = odbc_exec($conn,$sqlDetail);

$sqlShopCart ="SELECT * FROM ShopCart WHERE OrderNo ='$OrderNo'";
$rsShopCart = odbc_exec($conn,$sqlShopCart);

  while ($row = odbc_fetch_array($rsShopCart))

  {

  $sqlProd ="SELECT ItemTitle,ItemPrice FROM Products WHERE ItemNumber
='$row[OrderItem]'";
  $rsProd = odbc_exec($conn,$sqlProd);

  $ProdTitle = odbc_result($rsProd,ItemTitle);
  $ProdPrice = odbc_result($rsProd,ItemPrice);

        $sqlInsertDetail = "INSERT INTO OrderDetail
(OrderNo,ItemNumber,ItemQuantity,ItemTitle,ItemPrice) Values
('$row[OrderNo]','$row[OrderItem]','$row[OrderQuantity]','$ProdTitle','$ProdP
rice')";

        $rsInsertDetail = odbc_exec($conn,$sqlInsertDetail);

  }
```

Пример I.16.

Сценарий выполняет итерации по записям корзины покупок в поиске соответствующих OrderNo, копирует поля OrderNo, OrderItem и OrderQuantity в соответствующие поля записи OrderDetail. Затем поле OrderItem используется для извлечения ItemTitle и ItemPrice для этого продукта из таблицы Products и копирования их в поля the ItemTitle и ItemPrice записи OrderDetail. Подготовленная запись обновляется в таблице OrderDetail, и начинается обработка следующей записи корзины покупок. В конце мы будет иметь запись в таблице OrderDetail для каждой записи корзины покупателя.

Очистка корзины покупателя

Конечная часть сценария очищает корзину покупок от купленных товаров. Эти записи идентифицируются наличием OrderNo, соответствующим OrderNo подтвержденного или отвергнутого заказа.

```
'—Очистка ShopCart
$sqlDelete = "DELETE FROM ShopCart WHERE OrderNo='$OrderNo'";
$rsDelete = odbc_exec($conn,$sqlDelete);

header("Location:salesorder.php?Approval=$Approval")
```

Когда эта "фоновая" обработка завершается (вспомните, что на этой странице нет кода XHTML), происходит немедленная переадресация на страницу salesorder.php. Подтверждающее уведомление также передается на эту страницу для управления обработкой, которая там происходит.

Создание заказа на продажу

Мы приходим теперь на первую страницу, которую видит заказчик, когда покидает компанию кредитных карт. Эта страница salesorder.php представляет сводку подтвержденного заказа (или сообщение "простите" при отвергнутом заказе), которую заказчик может распечатать для справки. Также, поскольку заказ был завершен, выводится новый номер заказа, на тот случай, если заказчик захочет продолжить покупки.

Основной раздел этой страницы начинается с заголовка страницы и получения строки запроса Approval, переданной со страницы ordercapture.php. Если подтверждение заказа будет "true", то начинается процесс вывода информации о заказе.

```
<div style="position:absolute; top:75px; left:200px; width:550px">

<span class="head1">Sales Order</span><br>
<span class="head4">Date: </span><?php echo date('n/d/y') ?><br>
<span class="head4">Order No: </span><?php echo $_SESSION[OrderNo] ?><br>
<br>
```

```
<?php
$Approval = $_GET[Approval]

if ($Approval)
{

   $conn = odbc_connect('Driver={Microsoft Access Driver
(*.mdb)};DBQ=c:\inetpub\wwwroot\PHPTutorial\Ecommerce\databases\ecommerce.mdb
','','');

  $sql = "SELECT * FROM OrderHeader WHERE OrderNo='$_SESSION[OrderNo]'";
  $rsHeader = odbc_exec($conn,$sql);

    if (odbc_result($rsHeader,OrderNo) == "")
      {
      odbc_close($conn);
      ob_flush();
      header("Location:home.php")
    }

  $slqOrders = "SELECT * FROM OrderDetail WHERE
OrderNo='$_SESSION[OrderNo]'";
  $rsOrder = odbc_exec($conn,$sqlOrders);

  ?>
```

Пример I.17.

Создается *объект* соединения для связи с базой данных `eCommerce.mdb` и используется оператор *SQL* для извлечения информации из таблиц `OrderHeader` и `OrderDetail`. Эти *множества* записей имеют общий номер заказа, который находится в глобальной переменной `$_SESSION[OrderNo]`. Тогда будет доступна вся *информация* для восстановления заказа, начиная с информации о выставленном счете к оплате, извлеченной из таблицы `OrderHeader`.

Стоит обратить внимание на небольшой раздел кода, следующий за открытием таблицы `OrderHeader`.

```
if (odbc_result($rsHeader,OrderNo) == "")
      {
      odbc_close($conn);
      ob_flush();
      header("Location:home.php")
    }
```

Этот *сценарий* проверяет условие `EOF` на множестве записей, которое означает, что для `OrderNo` не было найдено подходящего значения. Как это может случиться, и зачем проверять? Это может случиться, если заказчик использует кнопку браузера `"Back"` ("назад") для возврата на предыдущую страницу (страница подтверждения компании кредитных карт), а затем снова нажимает кнопку `"вперед"` (`"forward"`).

Когда данная страница загружается впервые для представления заказа на продажу, конечная часть сценария создает для заказчика новый номер заказа (см. ниже). Поэтому, если заказчик возвращается на страницу назад, а затем снова перемещается вперед, номер заказа, вновь посланный компанией кредитных карт, не будет совпадать с номером текущего заказа, и будет порождаться ошибка сценария. Данный раздел кода позволяет избежать этой проблемы, перенаправляя заказчика на страницу `home.php`, если отсутствует совпадение. Невозможно запретить использовать кнопку браузера для возврата на предыдущую страницу, но можно перехватывать все проблемы, которые возникают при перемещении вперед.

Сценарий затем продолжает создание заказа на продажу, выводя информацию о счете платежа и деталях покупки.

```
<?php echo odbc_result($rsHeader,CustomerName) ?><br>
  <?php echo odbc_result($rsHeader,CustomerAddress) ?><br>
  <?php echo odbc_result($rsHeader,CustomerCity) ?>,
```

```
  <?php echo odbc_result($rsHeader,CustomerState) ?>
  <?php echo odbc_result($rsHeader,CustomerZip) ?><br>
  <br>
  <table border="0" cellpadding="3">
  <tr>
    <th>Item Number</th>
    <th>Title</th>
    <th>Quantity</th>
    <th>Price</th>
    <th>Amount</th>
  </tr>
  <?php
  $OrderTotal = 0
  while ($row = odbc_fetch_array($rsDetail) {
    $ItemNumber = $row[ItemNumber]
    $ItemTitle = $row[ItemTitle]
    $ItemPrice = $row[ItemPrice]
    $ItemQuantity = $row[ItemQuantity]
    $ItemAmount = $ItemPrice * $ItemQuantity
    $OrderTotal = $OrderTotal + $ItemAmount

echo "<tr>
      <td>$ItemNumber</td>
      <td><$ItemTitle</td>
      <td style=\"text-align:right\">$ItemQuantity</td>
      <td style=\"text-align:right\">$number_format($ItemPrice,2)</td>
      <td style=\"text-align:right\">$number_format($ItemAmount,2)</td>
    </tr>";

  }
  odbc_close($conn);
  $OrderShipping = number_format($OrderTotal * .02,2);
  $OrderTotal = number_format($OrderTotal + $OrderShipping,2);

echo "<tr>
    <td colspan=\"4\" style=\"text-align:right\">Shipping </td>
    <td style=\"text-align:right\">$OrderShipping</td>
  </tr>
  <tr>
    <th colspan=\"4\" style=\"text-align:right\">Order Total </th>
    <td style=\"border-style:solid\"><b>$OrderTotal</b></td>
  </tr>";
?>
  </table>
  <br>
  Спасибо за ваш заказ. Пожалуйста, распечатайте эту страницу и указывайте
номер заказа при любых запросах. Ваш заказ будет доставлен немедленно.

<?php }
else
{

echo "Простите, но вы не можете завершить свой заказ. Когда вы решите
проблему,
 возвращайтесь, пожалуйста, на <span class=\"head4\">WebWarehouse.com</span>
чтобы купить необходимое программное обеспечение."

}
?>
```

Пример I.17.

Форматирование этого заказа на продажу аналогично тому, которое использовалось при выводе корзины покупателя. *Сценарий* выполняет цикл по записям, извлеченным из таблицы `OrderDetail`, создавая строку

для каждой записи. Как и раньше, вычисляется каждое *значение* `$ItemAmount`, накапливается общая сумма `OrderTotal` и вычисляется `$OrderShipping` на `$OrderTotal`.

(Здесь снова возникает тот случай, когда можно подумать о том, чтобы хранить плату за доставку в записи `OrderHeader`, а не вычислять ее во время создания заказа. Позже, когда этот заказ будет восстановлен, скорее всего, будет изменение в проценте оплаты и общая *стоимость* заказа будет отличаться от текущей. Здесь эта проблема рассматриваться не будет.)

Сброс номера заказа

Теперь важная часть этой обработки. Заказ клиента завершен. Поэтому необходимо присвоить новое значение `OrderNo` на тот случай, если заказчик захочет продолжить покупку. Если он сделает это без нового номера, то в результате получится дублирование номера заказа и возникнет путаница при попытке разделить заказы клиента. Конечный шаг обработки на этой странице состоит тогда в присваивании нового номера с помощью той же процедуры, которая использовалась, когда клиент впервые прибыл на сайт.

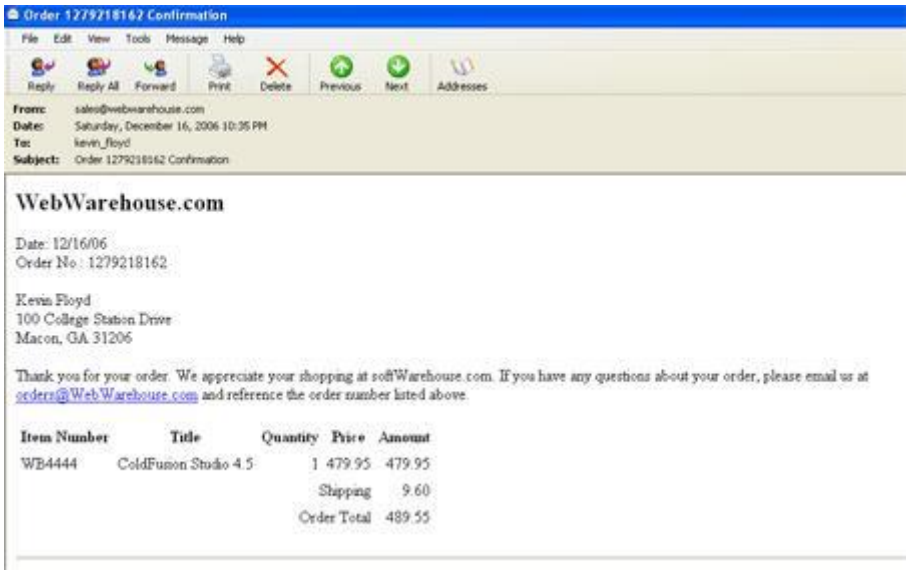
```
<?php
'-- Assign new OrderNo
$OrderNo = rand(1111111111,9999999999)
$_SESSION[OrderNo] =$OrderNo
?>

</div>
```

Все прошло полный круг. В этой точке бывший заказчик становится новым заказчиком и может начинать покупку с самого начала.

На этом завершается обсуждение базовых механизмов работы онлайнового сайта `eCommerce`. Можно, конечно, добавить еще множество свойств и функций. Мы рассмотрим еще одно - автоматическое создание подтверждения заказа по e-mail.

Отправка подтверждения по E-mail

Обычно компании посылают подтверждение по *e-mail* заказов, когда они приняты к выполнению. С помощью сценария множно создавать автоматические сообщения *e-mail*, и такая возможность будет добавлена в *приложение* `eCommerce`. Следующая иллюстрация показывает формат подтверждающего сообщения, которое получит заказчик.



Функция mail()

В PHP имеется функция mail(), предназначенная для создания сообщений email под управлением сценария.

mail() содержит параметры для создания и отправки email, по большей части также, как это происходит в обычной почтовой программе. Среди свойств и параметров нам понадобятся следующие:

Свойство	Описание
to	Строка, представляющая адрес e-mail получателя сообщения
subject	Строка, представляющая строку тему сообщения
message	Строка, представляющая основное тело сообщения
headers	Строка, представляющая дополнительную информацию, которая включается в начале сообщения e-mail

В простейшем виде сценарий e-mail может выглядеть следующим образом:

```
<?php
$to      = 'youraddress@domain.com';
$subject = 'the subject';
$message = 'hello';
$headers = 'From: webmaster@example.com' . "\r\n" .
    'Reply-To: webmaster@example.com' . "\r\n" .
    'X-Mailer: PHP/' . phpversion();

mail($to, $subject, $message, $headers);

?>
```

Конечно необходимо еще с ним поработать, чтобы составить сообщение о подтверждении заказа, особенно если оно форматируется для представления в виде формы заказа.

Размещение сценария

Лучшим местом для сценария подтверждения заказа по email является страница `ordercapture.php`. Эта страница получает информацию от компании кредитных карт и создает для заказчика записи `OrderHeader` и `OrderDetail`, а также содержит всю информацию, необходимую для составления сообщения e-mail. Размещение сценария показано на следующем эскизе страницы:

```
<div style="position:absolute; top:75px; left:200px; width:550px">
  ...

<?php
$Approval = $_POST[Approval];

if ($Approval)

{
```

```
  $sqlInsert = "INSERT INTO OrderHeader
(OrderNo,OrderDate,CustomerName,CustomerAddress,
  CustomerCity,CustomerState,CustomerZip,CustomerPhone,CustomerEmail) Values
('$OrderNo','$Date','$Name','$Address','$City','$State','$Zip','$Phone','$Ema
il')";

 $rsInsert = odbc_exec($conn,$sqlInsert);

//Создание записи OrderDetail

$sqlDetail ="SELECT * FROM OrderDetail WHERE NULL";
$rsDetail = odbc_exec($conn,$sqlDetail);

$sqlShopCart ="SELECT * FROM ShopCart WHERE OrderNo ='$OrderNo'";
$rsShopCart = odbc_exec($conn,$sqlShopCart);

  while ($row = odbc_fetch_array($rsShopCart))

  {

  $sqlProd ="SELECT ItemTitle,ItemPrice FROM Products WHERE ItemNumber
='$row[OrderItem]'";

  $rsProd = odbc_exec($conn,$sqlProd);
  $ProdTitle = odbc_result($rsProd,ItemTitle);
  $ProdPrice = odbc_result($rsProd,ItemPrice);

 $sqlInsertDetail = "INSERT INTO OrderDetail
(OrderNo,ItemNumber,ItemQuantity,
 ItemTitle,ItemPrice) Values ('$row[OrderNo]','$row[OrderItem]',
 '$row[OrderQuantity]','$ProdTitle','$ProdPrice')";

        $rsInsertDetail = odbc_exec($conn,$sqlInsertDetail);

  }

//ВСТАВЬТЕ СЮДА СЦЕНАРИЙ E-MAIL
-------------------------------------------------------------

$to      = 'youraddress@domain.com';
$subject = 'the subject';
$message = 'hello';
$headers = 'From: webmaster@example.com' . "\r\n" .
    'Reply-To: webmaster@example.com' . "\r\n" .
    'X-Mailer: PHP/' . phpversion();

mail($to, $subject, $message, $headers);
-------------------------------------------------------------

 //Удаление корзины покупателя

$sqlDelete = "DELETE FROM ShopCart WHERE OrderNo='$OrderNo'";
$rsDelete = odbc_exec($conn,$sqlDelete);

odbc_close($conn);

//echo $sqlInsertDetail;
header("Location:salesorder.php?Approval=$Approval");
}
?>
```

Пример I.18.

Программирование сообщения E-mail

Начнем теперь программирование сценария, который вставляется на страницу. Необходимо сделать так, чтобы поле e-mail было не обязательным на форме проверки кредитной карты. Некоторые заказчики могут не предоставлять адрес e-mail, и мы не сможем послать им подтверждающее сообщение. Поэтому весь сценарий находится внутри условного оператора.

```
if ($Email) {Then

    '—Послать подтверждение заказа по e-mail
    $to = $Email
    $subject = "Order $OrderNo Confirmation"
```

Прежде всего, необходимо инициализировать параметры mail() — `$to` и `$subject`.
Параметру `$to` присваивается один из элементов информации, посланной из компании кредитных карт и полученной в начале этой страницы. Переменная содержит адрес e-mail заказчика. Это должен быть реальный, действительный адрес e-mail.

Присваивание `$subject` использует текстовую строку, которая включает для идентификации значение переменной `$OrderNo`.

Теперь необходимо составить тело сообщения, форматируя его с помощью HTML для создания общей компоновки и внешнего вида. Вспомните из сказанного ранее, что тело должно быть одной строкой текста. Поэтому нельзя написать последовательность отдельных строк HTML и присвоить их параметру $message. Необходимо представить их в виде одной строки. Мы делаем это, задавая переменную $message, а затем помещаем в нее одну длинную строку, собирая ее из отдельных подстрок. Такой подход можно видеть в следующем фрагменте кода.

```
$message=""
    $message=$message & "<html>"
    $message=$message & "<$message>"
    $message=$message & "<h2>WebWarehouse.com</h2>"
    $message=$message & "Date: date('n/d/y') "<br>"
    $message=$message & "Order No.: $OrderNo "<br>"
    $message=$message & "<br>"
    $message=$message & Name & "<br>"
    $message=$message & Address & "<br>"
    $message=$message & City & ", " & State & " " & Zip & "<br>"
    $message=$message & "<p>"
    $message=$message & "Thank you for your order. We appreciate your
shopping at "
    $message=$message & "WebWarehouse.com. If you have any questions about
your order, "
    $message=$message & "please email us at "
    $message=$message & "<a
href=mailto:orders@WebWarehouse.com>orders@softWarehouse.com</a> "
    $message=$message & "and reference the order number listed above."
    $message=$message & "</p>"
    $message=$message & "<table border=0 cellpadding=3>"
    $message=$message & "<tr>"
    $message=$message & " <th>Item Number</th>"
    $message=$message & " <th>Title</th>"
    $message=$message & " <th>Quantity</th>"
    $message=$message & " <th>Price</th>"
    $message=$message & " <th>Amount</th>"
    $message=$message & "</tr>"
```

Пример I.19.

Каждая подстрока текста соединяется с переменной `$message`, создавая постепенно в переменной одну текстовую строку. Эта строка содержит также теги XHTML для форматирования вложенного текста. Отметим, что переменные `$Date`, `$OrderNo`, `$Name`, `$Address`, `$City`, `$State`, и `$Zip`, полученные из информации кредитной компанией, встраиваются в увеличивающуюся строку. Конечная часть кода создает заголовки таблицы для информации заказа на продажу, которая форматируется далее.

Затем необходимо собрать информацию заказа на продажу из таблицы `OrderDetail` и сформатировать ее как строки таблицы.

```
$sqlMail = "SELECT * From OrderDetail WHERE OrderNo = '$OrderNo'";
 $rsMail = odbc_exec($conn,$sqlMail);

 while ($row = odbc_fetch_array($rsMail))

 {

      $ItemNumber =$row[ItemNumber];
        $ItemTitle =$row[ItemTitle];
        $ItemQuantity = $row[ItemQuantity];
        $ItemPrice = $row[ItemPrice];
       $ItemAmount = $ItemQuantity * $ItemPrice;
        $OrderTotal = $OrderTotal + $ItemAmount;
        $message = $message . "<tr>";
       $message = $message ." <td>" . $ItemNumber . "</td>";
       $message = $message . " <td>" . $ItemTitle . "</td>";
       $message = $message . " <td align=right>". $ItemQuantity . "</td>";
       $message = $message . " <td align=right>" .
number_format($ItemPrice,2) . "</td>";
       $message = $message . " <td align=right>" .
number_format($ItemAmount,2) . "</td>";
       $message = $message . "</td>"

 }
```

Это делается здесь в цикле `while`: значения полей помещаются в переменные, которые форматируются в ячейки таблицы и соединяются со строкой `$message`. По мере выполнения цикла накапливается значение переменной `$OrderTotal`.

```
$ShippingCharge = $OrderTotal * .02;
    $OrderTotal = $OrderTotal + $ShippingCharge;
     $message = $message . "<tr>";
     $message = $message . "<td colspan=4 align=right>Shipping</td>";
    $message = $message . " <td align=right>" .
number_format($ShippingCharge,2) . "</td>";
    $message = $message . "</tr>";
     $message = $message . "<tr>";
     $message = $message . "<td colspan=4 align=right>Order Total</td>";
    $message = $message . "<td align=right>" . number_format($OrderTotal,2) .
"</td>";
     $message = $message . "</tr>";
     $message = $message . "</table>";
    $message = $message . "</body>";
    $message = $message . "</html>";

$headers = 'From: sales@webwarehouse.com' . "\r\n" .
    'Reply-To: sales@webwarehouse.com' . "\r\n" .
    'Content-Type:text/html;charset=us-ascii' . "\r\n" .
    'X-Mailer: PHP/' . phpversion();

mail($to, $subject, $message, $headers);
```

После того как сценарий выполнил итерации на множестве записей, форматируя отдельные строки таблицы для купленных продуктов, вычисляется и форматируется `$ShippingCharge`, вместе с `$OrderTotal`, как две последние строки таблицы. Закрывающие теги HTML присоединяются в конце строки `$message`, и сообщение e-mail завершается.

После завершения формирования сообщения e-mail мы создаем переменную `$headers` для хранения заголовочных данных, которые нужны для описания сообщения. В данном случае мы добавляем заголовочные сообщения `From` (указывает адрес e-mail, откуда было прислано сообщение), `Reply-To` (адрес e-mail, который появится в поле `TO` получателя, если он будет отвечать на автоматическое сообщение), `Content-Type` (определяет формат сообщения) и `X-Mailer` (определяет версию PHP, используемую для генерации сообщения). Хотя заголовочные сообщения являются необязательными, рекомендуется всегда их включать.

Если сообщение e-mail содержит HTML, который должен интерпретироваться соответствующим образом, то необходимо использовать следующее значение заголовка.

```
'Content-Type:text/html;charset=us-ascii'
```

Последний шаг состоит в вызове встроенной функции mail().

Необходимо упомянуть попутно, что прежде чем можно будет послать сообщение e-mail через сценарий PHP, сервер Web должен иметь выполняющуюся службу `SMTP` (Простой протокол пересылки почты), и может понадобиться внести изменения в файл `PHP.ini`. Особенности настроек различных серверов можно найти на сайте PHP.

Контрольные вопросы

1. Введение в PHP. История языка PHP. Возможности PHP (краткий перечень платформ, протоколов,
2. баз данных, приложений электронной коммерции и функций, которые поддерживаются PHP).
3. Области применения PHP (как серверное приложение, в командной строке, создание GUI приложений); Способы использования.
4. Установка и настройка программного обеспечения, необходимого для работы с PHP. Основы синтаксиса. Основной синтаксис PHP.
5. Способы разделения инструкций, создания комментариев. Переменные, константы и типы данных, операторы. Управляющие конструкции. Условный оператор (if, switch). Циклы (while, for, fo-reach).
6. Операторы включения (include, require).
7. Обработка запросов с помощью PHP. Способы отправки данных на сервер и их обработке с помощью PHP.
8. Основы клиент-серверных технологий.
9. HTML-формы и отправка данных с ее помощью.
10. Краткая характеристика методов Post и Get. Механизм получения данных из HTML-форм, и их обработка с помощью PHP.
11. Работа с массивами данных. Массивы. Сортировка массивов.
12. Применение функции ко всем элементам массива.