Основы баз данных и использования программных модулей. Записная книжка.

Лабораторная работа № В-1.

ЦЕЛЬ РАБОТЫ

Получение навыков оптимизации программы *PHP* с помощью использования программных модулей. Знакомство с основами использования Базы данных в *PHP* для реализации различных задач.

Очень часто абсолютно одинаковый программный код используется несколько раз в разных местах программы. Для *PHP*, который используется в веб-приложениях, это актуально при использовании нескольких страниц сайта со схожими функциями. Если просто "копировать" кусок кода и вставлять его на прочие страницы, да и на ту же самую страницу, то, во-первых, объем кода быстро разрастается, что приводит к ухудшению его восприятию программистом и увеличению числа ошибок в нем. Во-вторых, исправление или изменение одного фрагмента кода приводит к необходимости искать все остальные его включения, что не всегда удается – после нескольких таких модификаций код, который должен быть одинаковым и работать по одному алгоритму, довольно сильно отличается от своего аналога в других файлах проекта.

Решением этой задачи может быть применение модулей: отдельных файлов с *PHP*-кодом программы. При необходимости они "встраиваются" в любое место основной *PHP*-программы или в другой модуль. Это позволяет выделить часто повторяющие фрагменты, например, пользовательские функции, записать их в отдельный файл и использовать на разных страницах одного сайта.

Более того, многие функции или фрагменты кода без изменения могут быть использованы в разных проектах: например, для написания административной части сайта или для типовой обработки информации. По такому принципу строятся многочисленные библиотеки функций, которые позволяют *PHP*-программистам сразу использовать довольно мощный функционал без отрыва от основного задания. Так широко известна библиотека "*pclzip.lib.php*", позволяющая использовать *ZIP*-архивацию для файлов. Тогда, если необходимо исправить ошибку или реализовать новый алгоритм для того или иного действия, правка кода производится только один раз, после чего при необходимости файл с модулем просто копируется в другие проекты.

Использование модулей позволяет не только сократить код за счет использования одного и того же программного модуля на разных страницах, но и упростить сам процесс разработки. Нет необходимости работать с огромным файлом, содержащим тысячи строчек кода: разные его фрагменты очень просто реализовывать последовательно, модуль за модулем. Более того, этот процесс можно поручить сразу нескольким программистам, которые будут выполнять этот процесс параллельно.

Базы данных – не менее важная тема при Веб-разработке. Это оптимальный способ организации длительного хранения данных на сервере. Причем, в отличие от выполняющихся на локальных компьютерах пользовательских программ, он зачастую и единственный. Действительно, на локальном компьютере совсем не обязательно использовать специальные базы данных, можно просто сохранить промежуточный или окончательный результат в файл. Для веб-сайтов это не так: дело в том, что одну и ту же страницу могут одновременно просматривать, а значит и выполнять *PHP*-программу, десятки или даже сотни посетителей. Поэтому при использовании файлов неизбежно возникнет ситуация, когда к одному и тому же файлу будет обращение от разных системных процессов, что приведет к конфликту.

Делать специальную систему идентификации посетителей, привязанную к его *ip*-адресу, и выделять для каждого из них свой файл можно, но очень сложно: адреса могут меняться. База данных же значительно упрощают работу, позволяя не только согласовывать доступ и изменения

данных разными посетителями сайта, но и предоставляя современные средства доступа, организации и манипуляции данными.

Одним из самых распространенных серверов баз данных в настоящий момент времени является *MySQL*. Скорее всего именно он будет установлен на хостинге, который Вы решите использовать. Для его настройки, создания баз и таблиц используется специальная панель администратора *phpMyAdmin*. Как и большинство современных БД *MySQL* относится к реляционным базам данных, а значит для его использования применяется *SQL* – язык структурированных запросов. С его помощью осуществляется быстрый поиск требуемых данных, их фильтрация, добавление, удаление или редактирование. Знание и понимание принципов организации реляционных баз данных и *SQL* – важная и необходимая часть навыков успешного *PHP*-программиста.

ПРОДОЛЖИТЕЛЬНОСТЬ

4 академических часа работы в аудитории, 4 академических часа – самостоятельно.

РЕЗУЛЬТАТ РАБОТЫ

Размещенные на Веб-сервере и доступные по протоколу *HTTP* документы:

- *index.php* – единственный загружаемый в браузер документ, осуществляющий всю работу сайта;
- *menu.php* – формирующий меню и регламентирующий его работу модуль;
- *viewer.php* – модуль для вывода содержимого базы данных в браузер;
- *add.php* – модуль для добавления новой записи в базу данных;
- *edit*.php – модуль для редактирования существующей записи базы данных;
- *delete.php* – модуль для удаления записи из базы данных.

ДОПОЛНИТЕЛЬНЫЕ ТРАБОВАНИЯ К РАБОТЕ

Сайт должен предоставлять удобный сервис для хранения, редактирования и доступа к данным о людях, т.е. фактически представлять из себя записную книжку с контактами. Каждый контакт характеризуется следующими атрибутами:

- фамилия;
- имя;
- отчество;
- пол;
- дата рождения;
- телефон;
- адрес;
- E-майл;
- комментарий.

Для работы сайта в качестве части URL используется только файл "*index.php*", все остальные файлы являются программными модулями и не доступны напрямую в браузере.

Модуль "*menu.php*" содержит функцию без параметров которая возвращает в виде строки *HTML*-код, содержащий основное меню сайта. Меню состоит из следующих пунктов:

- Просмотр;
- Добавление записи;
- Редактирование записи;
- Удаление записи.

Все пункты меню оформляются в виде созданных с помощью тега `<a>..</a>` кнопок синего цвета. При первой загрузке активным считается пункт меню "*Просмотр*", при переходе на другой пункт меню – тот по которому перешли. Активный пункт меню должен выделяться красным цветом.

В случае выбора пункта "*Просмотр*" ниже основных пунктов меню выводятся дополнительные пункты для определения вида сортировки (см. ниже). Визуально они должны быть меньше

основных, но оформлены том же стиле. При переходе по дополнительным пунктам выделение основного пункта меню не должно исчезать. У дополнительных пунктов меню также есть активный, выделенный красным цветом пункт меню (по умолчанию первый по порядку).

Вывод меню осуществляется в верхней части экрана с помощью подключения указанного модуля к файлу "*index.php*" и вызова сформированной функции. Вне зависимости от переданных на сайт *GET*- и *POST*-параметров один из пунктов меню всегда должен быть активным.

При первой загрузке сайта в браузере (либо при выборе пункта меню "*Просмотр*") выводится содержимое записной книжки в виде таблицы с не более чем 10 строками. Если записей в базе данных больше – под таблицей выводится пагинация (ссылки на другие страницы в виде их номеров; при наведении на ссылку курсора мыши она обводится в рамочку толщиной 2*px*), которая позволяет выводить другие фрагменты базы данных по 10 записей каждый. Для удобства использования должна быть предусмотрена возможность сортировки результата в порядке добавления записей в базу, а также по фамилии или по дате рождения (в порядке возрастания).

Для подготовки *HTML*-кода с таблицей и ссылками пагинации используется модуль "*viewer.php*", который должен содержать пользовательскую функцию с параметрами, определяющими тип сортировки и номер выводимой страницы пагинации. Функция вызывается из модуля "*index.php*" который и выводит в браузер возвращаемую ей строку с контентом.

При переходе по ссылке "*Добавление записи*" на месте таблицы выводится форма для добавления новой записи в базу данных. После ее заполнения и отправки страница перезагружается и выводится та же форма с надписью: "*Запись добавлена*" (зеленым цветом) или "*Ошибка: запись не добавлена*" (красным цветом) в зависимости от успешности выполнения операции. Форма задается в модуле "*add.php*" в статическом виде. *PHP*-код для добавления записи также располагается в этом модуле.

При переходе на пункт меню "*Редактирование записи*" загружается страница с формой, аналогичной форме для добавления записи. Перед формой выводятся ссылки с текстом, соответствующим именам и фамилиям из базы данных (список сортируется по фамилии, затем по имени). При переходе по ссылке страница перезагружается и в полях формы отображаются соответствующие выбранной записи значения. Текущая запись в списке выделяется цветом или рамкой; если запись не была выбрана пользователем – то текущей считается первая запись по порядку. Весь *HTML*- и *PHP*-код полностью находится в модуле "*edit.php*".

Доступная при переходе по ссылке "*Удалить запись*" страница содержит список ссылок, текст которых соответствует фамилии и инициалам из базы данных. При переходе по ссылке страница перезагружается, соответствующая запись удаляется из базы данных, выводится надпись: "*Запись с фамилией Иванов удалена*" (вместо "*Иванов*" указывается фамилия из удаляемой записи). Весь *HTML*- и *PHP*-код полностью находится в модуле "*delete.php*".

ПРИМЕНЕНИЕ ПОЛУЧАЕМЫХ В РАБОТЕ ЗНАНИЙ И НАВЫКОВ

РЕКОМЕНДАЦИИ К СТРУКТУРЕ ПРОГРАММЫ

Использование модульного подхода позволяет значительно упростить программирование, реализуя каждый из них по очереди. Всего по заданию лабораторной работы необходимо реализовать один главный файл и пять программных модулей. Начнем с модуля *menu.php*, реализующего основное меню сайта.

Листинг В-1. 1

```
<div id="menu">
<?php
	// если нет параметра меню – добавляем его
	if( !is_array($_GET['p']) ) $_GET['p']='view';

	echo '<a href="/?p=viewer"'; // первый пункт меню
```

```
        if( $_GET['p'] == 'viewer' ) // если он выбран
              echo ' class="selected"'; // выделяем его
        echo '>Просмотр</a>';
        echo '<a href="/?p=add"';   // второй пункт меню
        if( $_GET['p'] == 'add' ) echo ' class="selected"';
        echo '>Добавление записи</a>';

        if( $_GET['p'] == 'viewer' )     //если был выбран первый пункт меню
        {
            echo '<div id="submenu">';   // выводим подменю

              echo '<a href="/?p=viewer&sort= byid "';// первый пункт подменю
              if( !isset($_GET['sort']) || $_GET['sort'] == 'byid' )
                    echo ' class="selected"';
              echo '>По-умолчанию</a>';

              echo '<a href="/?p=viewer&sort=fam"';    // второй пункт подменю
              if( isset($_GET['sort']) && $_GET['sort'] == 'fam' )
              echo ' class="selected"';
              echo '>По фамилии</a>';

            echo '</div>';          // конец подменю
        }
?>
</div>
```

В модуле могут одновременно располагаться как статический *HTML*-код, так и выполняемый *PHP*-код. Все статические фрагменты модуля будут перенесены в результат без изменений: это дает возможность выбора и добавления в документ в виде его фрагментов полностью статических файлов. Фактически это позволяет при программировании не думать о том, как будет использоваться получаемый файл. В данном случае представим себе, что он просто будет напрямую открываться в браузере: тогда неизменную часть *HTML*-кода (тег `<div>`) логично сделать в статическом виде, а динамически формируемую часть – в виде *PHP*-программы.

В самом ее начале осуществляется проверка наличия передаваемого при переходе по пунктам меню параметра "`p`". Если он отсутствует, т.е. если это первая загрузка страницы и необходимо по-умолчанию выделить пункт меню "*Просмотр*", мы принудительно добавляем в массив `$_GET` элемент с ключом "`p`", тем самым имитируя переход по пункту меню "*Просмотр*". Для повышения безопасности выполнения программы здесь также следует проверить допустимость значения параметра: если оно недопустимо (например, `p=dhgjfkehgb`) – то программу стоит принудительно остановить. Сделайте это самостоятельно.

Формирование пунктов меню стандартно и уже рассматривалась в предыдущих лабораторных работах. В листинге В-1. 1 приведены лишь два пункта, еще два (Редактирование и Удаление записи) необходимо сделать самостоятельно. Для определенности договоримся, что значение передаваемого параметра для них будет равно "`p=edit`" и "`p=delete`" соответственно.

Подменю реализуется аналогичным образом: если выбран пункт меню "*Просмотр*", т.е. значение элемента массива `$_GET['p']` равно "`view`", выполняется формирующий подменю *PHP*-код. Обратите внимание: для управления подменю используется другой параметр "`sort`", а параметр "`p`" передается в него без изменений. Кроме того, в данном примере существование параметра `sort` явно проверяется перед его сравнением с возможным значением: первый пункт меню становится активным, если соответствующего параметру элемента массива не существует или его значение равно `byid`. Остальные – если он существует и его значение равно передаваемому в них параметру. Доработайте приведенный пример в соответствии с заданием: добавьте отсутствующие пункты меню и подменю; создайте в *css*-файле необходимые стили.

Теперь, когда у нас есть модуль, полностью формирующий работоспособное меню, можно сформировать и код главного файла сайта (*index.php*). В нем будет осуществлен вызов описанного

и других, еще не сформированных модулей. Опустим всю статическую часть этого файла (реализуйте ее самостоятельно) и приведем в листинге только его PHP-код.

Листинг В-1. 2

```
require 'menu.php';  // главное меню
// модули с контентом страницы
if( $_GET['p'] == 'viewer' ) { include 'viewer.php'; } else
if( $_GET['p'] == 'add' ) { include 'add.php'; } else
if( $_GET['p'] == 'edit' ) { include 'edit.php'; } else
if( $_GET['p'] == 'delete' ) { include 'delete.php'; }
```

Модуль *menu.php* с главным меню подключается всегда, поэтому стоит использовать конструкцию `require`: при ее наличии в *PHP*-программу перед ее выполнением будет вставлен код из указанного в ней модуля. Далее, в зависимости от значения переданного параметра, подключается тот или иной модуль с контентом. Причем в этом случае используется конструкция `include`, которая добавляет код не перед выполнением программы, а во время выполнения. Таким образом, размер программы не увеличится за счет добавления в него кода сразу из всех теоретически используемых модулей (*viewer.php*, *add.php*, *edit.php*, *delete.php*) – будет выбран и добавлен код только из того модуля, который нам необходим (обратной стороной такой гибкости является уменьшение скорости работы).

Обратите внимание: нет необходимости проверять существование элемента массива `$_GET['p']` – это уже было сделано в модуле *menu.php*. Кроме того, можно несколько сократить код за счет того, что значение переменной и имя файла с модулем совпадает.

Листинг В-1. 3

```
require 'menu.php';  // главное меню
// модули с контентом страницы
if( file_exists($_GET['p'].'.php') ) { include $_GET['p'].'.php'; }
```

Действительно, имя подключаемого модуля совпадает со значением параметра "`p`" плюс соответствующее разрешение. Если бы использовалась конструкция `require` – динамически сформировать имя подключаемого файла было бы невозможно, т.к. оно должно быть известно до начала выполнения *PHP*-программы. Но т.к. `include` добавляет код во время выполнения, имя модуля может быть сформировано динамически. Теперь последовательно реализуем программы оставшихся трех модулей.

В модуле "*viewer.php*", согласно условиям лабораторной работы, должна размещаться пользовательская функция, которая по типу сортировки и диапазону выводимых записей формирует содержимое записной книжки. Фактически в этом примере демонстрируется возможность создания библиотеки функций.

Листинг В-1. 4

```
function getFriendsList($type, $page)
{
      // осуществляем подключение к базе данных
      $mysqli = mysqli_connect('localhost', 'user', 'password', 'friends');

       if( mysqli_connect_errno() ) // проверяем корректность подключения
             return 'Ошибка подключения к БД: '.mysqli_connect_error();

      // формируем и выполняем SQL-запрос для определения числа записей
      $sql_res=mysqli_query($mysqli, 'SELECT COUNT(*) FROM friends');

      // проверяем корректность выполнения запроса и определяем его результат
      if( mysqli_errno($mysqli) && $row=mysqli_fetch_rows($sql_res) )
       {
           if( !$TOTAL=$row[0] )      // если в таблице нет записей
                 return 'В таблице нет данных'; // возвращаем сообщение
```

```
            $PAGES = ceil($TOTAL/10); // вычисляем общее количество страниц

            if( $page>=$TOTAL )  // если указана страница больше максимальной
                     $page=$TOTAL-1;  // будем выводить последнюю страницу
            // формируем и выполняем SQL-запрос для выборки записей из БД
            $sql='SELECT * FROM friends LIMITS '.$page.', 10';
            $sql_res=mysqli_query($mysqli, $sql);

            $ret='<table>';       // строка с будущим контентом страницы
            while( $row=mysqli_fetch_assoc($sql_res) ) // пока есть записи
            {
                  // выводим каждую запись как строку таблицы
                  $ret.='<tr><td>'.$row['name'].'</td>
                                 <td>'.$row['mail'].'</td>
                                 <td>'.$row['telephone'].'</td></tr>';
               }
               $ret.='</table>';  // заканчиваем формирование таблицы с контентом

            if( $PAGES>1 )  // если страниц больше одной - добавляем пагинацию
            {
                  $ret.='<div id="pages">';       // блок пагинации
                  for($i=0; $i<$TOTAL; $i++)      // цикл для всех страниц пагинации
                        if( $i != $page )    // если не текущая страница
                              $ret.='<a href="?p=viewer&pg='.$i.'">'.($i+1).'</a>';
                        else                 // если текущая страница
                              $ret.='<span>'.($i+1).'</span>';
                  $ret.='</div>';
            }
            return $ret;          // возвращаем сформированный контент
      }
                                  // если запрос выполнен некорректно
      return 'Неизвестная ошибка';  // возвращаем сообщение
}
```

Входными параметрами функции служат две переменные: `$type` – для определения типа сортировки и `$page` – для указания страницы пагинации записи которой добавляются в контент (напомним, что одна страница согласно заданию, это 10 записей). В теле функции в первую очередь происходит подключение к серверу базы данных. В качестве адреса сервера используется "*localhost*" (локальный компьютер), в качестве имени пользователя и его пароля "*user*" и "*password*" соответственно (в зависимости от технического оснащения и настроек указанные реквизиты доступа могут меняться – уточняйте их перед выполнением работы). На сервере будет выбрана для работы база данных "*friends*", поэтому ее и содержащиеся в ней таблицы необходимо предварительно создать – сделайте это самостоятельно.

Если вы это не сделали или доступ к серверу невозможен – в следующих строках PHP-программы будет осуществлена проверка успешности подключения. Если произошла ошибка, то функция `mysqli_connect_errno()` вернет ее код (код "*0*" – успешное подключение). Поэтому, согласно правилам преобразования типов, любой отличный от нуля результат функции будет воспринят условным оператором `if` как `TRUE`, и, следовательно, он будет выполнен. В данном случае будет выведено сообщение об ошибке и его текст, возвращаемый функцией `mysqli_connect_error()`.

После успешного подключения можно непосредственно выполнять *SQL*-запросы, т.е. запрашивать необходимые для отображения данные. Это можно делать сразу, но в требованиях указана необходимость пагинации. Для ее формирования необходимо знать число записей на одной странице (в данном случае это известно: 10) и количество таких страниц. Очевидно, что количество страниц пагинации равно общему количеству выводимых записей деленное на размер одной страницы с округлением в большую сторону. Поэтому, необходимо как-то вычислить количество выводимых записей. Это можно сделать двумя путями: во-первых, выполнить *SQL*-запрос и посчитать количество записей в результирующей таблице. Во-вторых, выполнить *SQL*-запрос, который бы средствами *SQL*-сервера подсчитал количество записей.

Плюсом первого варианта является отсутствие необходимости выполнять какие-либо предварительные действия: выполняется сразу окончательный *SQL*-запрос, возвращающий

нужные нам результаты. Но, при большом объеме данных и результат будет очень большим (именно для уменьшения передаваемой и отображаемой информации и применяется пагинация) – даже для отображения небольшой страницы из 10 записей сервер будет формировать полный результат выборки, возможно содержащий тысячи записей. Такой минус на порядок перевешивает все плюсы, поэтому необходимо применять второй вариант: предварительный запрос с функцией `COUNT()`.

В следующем условном операторе проверяется корректность выполнения *SQL*-запроса с помощью функции `mysqli_errno($mysqli)` и одновременно формируется и проверяется наличии первой (и для данного запроса единственной) записи результирующей таблицы. В итоге оператор срабатывает при возможности корректной работы с таблицей `friends` (она существует и доступна пользователю), если оператор не выполняется – в конце функции возвращается сообщение об неизвестной ошибке базы данных.

Для удобства работы и понимания текста программы результат из массива `$row[0]` переносится в переменную `$TOTAL` и, если она равна нулю, функция возвращает сообщение об отсутствии в таблице данных. Если же записи есть, то вычисляется общее количество страниц в пагинации и заносится в переменную `$PAGE`. Не лишним будет проверить корректность переданной в функцию в качестве параметра номера выводимой страницы пагинации, т.е. переменной `$page`. Если она больше максимально возможной – она будет уменьшена: теперь мы можем в дальнейшем безбоязненно использовать эту переменную без проверок.

После всех предварительных вычислений можно выполнять и окончательный вариант *SQL*-запроса. В нем используется конструкция `LIMIT`, указывающая диапазон передаваемых записей результирующей таблицы. При этом диапазон вычисляется так, чтобы он полностью совпал с диапазоном соответствующей страницы пагинации, т.е. все полученные в результирующей таблице записи можно смело отображать. Проверять корректность работы запроса в данном случае не обязательно и даже нежелательно, т.к. успешное выполнение предыдущего запроса гарантирует успешное выполнение и окончательного (он не использует никакие другие таблицы, условия, поля и т.д.).

Поэтому, введя переменную `$ret` (в ней будем накапливать результат работы функции, т.е. контент страницы), в цикле переберем все записи результирующей таблицы с помощью функции `mysqli_fetch_assoc()`. Она возвращает ассоциативный массив с ключами, соответствующими именам полей результирующей таблицы. Каждый последующий вызов функции меняет значения элементов возвращаемого ей массива на значения следующей записи: таким образом она позволяет в цикле с предусловием `while` осуществить последовательный перебор всех записей результирующей таблицы (функция вернет `FALSE` при достижении конца таблицы). На каждой итерации в переменную `$ret` добавляется строка *HTML*-таблицы из значений соответствующей записи. После цикла тег `<table>` закрывается.

На этом вывод непосредственно данных из БД закончен, но, если количество страниц больше одной – необходимо вывести ссылки на них. Это условие проверяется и, при его выполнении, в переменную `$ret` добавляются тег `<div>` для пагинации, содержащий блок с надписью: "*Страницы:*" и ссылки на них. Ссылки формируются и выводятся в цикле: обратите внимание что в тексте ссылки страницы пагинации нумеруются, начиная с единицы, а в адресе – начиная с нуля. Действительно, для корректности вычисления диапазона выводимых записей первая страница должна быть на самом деле нулевой, что не совсем удобно для восприятия человеком.

Ясно, что не стоит выводить на HTML-странице ссылку на саму себя (это бессмысленно), кроме того следует выделить текущую страницу пагинации в списке. Для этого в цикле осуществляется проверка: если выводится текущая страница из нее формируется не ссылка, а тег `<span>`, что позволит средствами *CSS* как-то выделить страницу.

Добавление блока пагинации заканчивает формирование контента, и функция возвращает содержимое переменной `$ret` как результат своей работы. Для выполнения условий лабораторной работы необходимо самостоятельно добавить в формируемую HTML-таблицу оставшиеся поля (*фамилия*, *пол*, *дата рождения*, *адрес*, *комментарий*); добавить подписи полей

вверху таблицы; доработать *CSS*-файлы так, чтобы таблица и блок пагинации был представлен в требуемом виде. Кроме того, необходимо добавить в функцию обработку параметра `$type`: как в *SQL*-запросах, так и для сохранения типа сортировки при переходе на другую страницу пагинации.

Листинг B-1. 5

```
require 'menu.php';  // главное меню
if( $_GET['p'] == 'viewer' ) // если выбран пункт меню "Просмотр"
{
      include 'viewer.php'; // подключаем модуль с библиотекой функций

      // если в параметрах не указана текущая страница - выводим самую первую
      if( !isset($_GET['pg']) || $_GET['pg']<0 ) $_GET['pg']=0;

      // если в параметрах не указан тип сортировки или он недопустим
      if(!isset($_GET['sort']) || ($_GET['sort']!='byid' && $_GET['sort']!='fam' &&
         $_GET['sort']!='birth'))
            $_GET['sort']='byid'; // устанавливаем сортировку по умолчанию

      // формируем контент страницы с помощью функции и выводим его
      echo getFriendsList($_GET['sort'], $_GET['pg']);
}
else // подключаем другие модули с контентом страницы
if( file_exists($_GET['p'].'.php') ) { include $_GET['p'].'.php'; }
```

Теперь, когда был реализован модуль "*viewer.php*", необходимо вновь вернуться к файлу "*index.php*": ведь доступность функции не означает ее выполнение. Поэтому немного модифицируем код *PHP*-программы главной страницы, отделив отображение содержимого базы данных от подключения всех остальных модулей.

Итак, если был выбран пункт меню "*Просмотр*" или загружена первая страница сайта, то подключается модуль "*viewer.php*" – в нем мы только что реализовали функцию `getFriendsList()`, поэтому она становится доступна и в основной *PHP*-программе (просто мысленно замените строку `include 'viewer.php';` на содержимое этого файла). В качестве параметров ей передается значения элементов массива `$_GET`, т.е. *GET*-параметры страницы. Т.к. эти параметры могут быть какими угодно (они формируются через *URL*, который можно легко изменить вручную), то предварительно следует проверить их наличие и корректность. Если номер страницы не существует или он меньше нуля – то принудительно этот параметр устанавливается равным нулю (номер страницы не может быть меньше минимального значение, т.е. нуля; превышение максимального значения проверяется внутри функции `getFriendsList()`).

Для типа сортировки проверяется его существование и равенство его значения одному из трех возможных вариантов. Если это условие не выполняется – тип сортировки устанавливается по порядку добавления записей в базу данных. После всех проверок функция вызывается и в *HTML*-код страницы добавляется результат ее работы.

Теперь перейдем к добавлению записей в таблицу, т.е. к модулю "*add.php*". В отличии от библиотеки функций, этот модуль самостоятельно формирует *HTML*-код, поэтому редактировать *PHP*-код "*index.php*" необходимости нет.

Листинг B-1. 6

```
<form name="form_add" method="post" action="/?p=add">
       <input type="text" name="name" id="name" placeholder="Имя">
       <textarea name="comment" placeholder="Краткий комментарий"></textarea>
       <input type="submit" name="button" value="Добавить запись">
</form><?php
      // если были переданы данные для добавления в БД
      if( isset($_POST['button']) && $_POST['button']== 'Добавить запись')
      {
            $mysqli = mysqli_connect('localhost', 'user', 'password', 'friends');

              if( mysqli_connect_errno() ) // проверяем корректность подключения
                    echo 'Ошибка подключения к БД: '.mysqli_connect_error();
```

```
            // формируем и выполняем SQL-запрос для добавления записи
            $sql_res=mysqli_query($mysqli, 'INSERT INTO friends VALUES ("'.
                                        htmlspecialchars($_POST['name']).'", "'.
                                        htmlspecialchars($_POST['comment']).'")');
            // если при выполнении запроса произошла ошибка - выводим сообщение
            if( mysqli_errno($mysqli) )
                 echo '<div class="error">Запись не добавлена</div>';
            else // если все прошло нормально - выводим сообщение
                 echo '<div class="ok">Запись добавлена</div>';
      }
?>
```

По условию лабораторной работы форма формируется в виде статического *HTML*-кода. При его подключении в главной программе статический код будет без изменений передан в *HTML*-код страницы так, как будто бы эта была строка, выводимая оператором `echo`.

PHP-код модуля в данном случае должен определить: есть ли необходимость добавления новой записи в таблицу. Признаком этого является переданные в программу *POST*-параметры из формы: если форма была заполнена и отправлена, то эти параметры есть, т.е. в массиве `$_POST` присутствуют элементы. Как уже говорилось в предыдущих лабораторных работах, для проверки необходимости обработки формы не следует использовать параметры, имя и значение которых может быть использовано в других формах. Действительно, если признаком добавления новой записи будет проверка наличия параметра `name`, то он может использоваться и на других формах (например, для редактирования существующей записи) – тогда не будет уверенности какое действие необходимо произвести программе. Конечно, в данном случае такая ситуация невозможна т.к. эта страница обрабатывает только одну форму, но в общем случае лучше проверять наличие уникальных параметров или их значения: например, имя кнопки или специально введенного скрытого поля.

Если признак добавления новой записи был подтвержден, т.е. в массиве `$_POST` присутствует элемент с ключом "`button`" и значением "*Добавить запись*", то осуществляется подключение к серверу баз данных, проверяется его корректность, формируется и выполняется соответствующий *SQL*-запрос (см. описание предыдущего модуля). Если запрос выполнен с ошибкой – выводится соответствующее сообщение, если без ошибок – также выводится сообщение.

Обратите внимание, что при формировании *SQL*-запроса используемые в нем параметры обрабатываются функцией `htmlspecialchars()`: она заменяет специальные символы на *HTML*-сущности. Например, символ двойной кавычки будет преобразован в "`&quot;`". Это делается для возможности использования в вводимых пользователем элементах формы кавычек, слешей и других символов, которые могут повлиять на корректность *SQL*-запроса. Действительно, если в поле комментариев пользователь ввел часть строки в кавычках (например, название чего-либо), то при формировании запроса эти кавычки будут интерпретированы сервером БД как окончание строки, и, следовательно, сам *SQL*-запрос станет некорректным: `INSERT INTO friends VALUES ("Алексей", "Участник конкурса "WorldSkills"")` – окончание строки (а именно, `WorldSkills""`) будет воспринято *SQL*-сервером как ошибочное добавление после закрытия кавычки. Поэтому преобразование таких символов – один из способов избежать такой ситуации (другой способ – использование псевдопеременных, см. ниже).

Самостоятельно доработайте *CSS*-файл таким образом, чтобы элементы формы были одинакового размера, располагались друг под другом, надписи имели требуемый в условиях лабораторной работе цвет. Добавьте на форму остальные элементы для формирования записи таблицы согласно заданию лабораторной работы. Добавьте в таблицу базы данных числовое поле id и сделайте его первичным ключом. Доработайте *SQL*-запрос так, чтобы он корректно добавлял такую запись в базу данных (например, используйте автоинкремент).

Очень часто в информацию из базы данных необходимо вносить изменения. Кончено, это можно сделать, удаляя старую запись и добавляя новую, уже исправленную. Но такой подход в корне неверен: любой сбой, любая ошибка оператора может привести к потере данных. Поэтому на сайте должны быть функции изменения содержимого таблиц: обычно они совмещаются на одной

HTML-странице с функциями добавлением записей, но для обучения и упрощения программы в условиях лабораторной работы функционал по манипуляции с данными разнесен на разные страницы сайта.

Листинг В-1. 7

```
$mysqli = mysqli_connect('localhost', 'user', 'password', 'friends');

if( mysqli_connect_errno() ) // если при подключении к серверу произошла ошибка
{
       // выводим сообщение и принудительно останавливаем PHP-программу
       echo 'Ошибка подключения к БД: '.mysqli_connect_error();
       exit();
}
// если были переданы данные для изменения записи в таблице
if( isset($_POST['button']) && $_POST['button']== 'Изменить запись')
{
       // формируем и выполняем SQL-запрос на изменение записи с указанным id
       $sql_res=mysqli_query($mysqli, 'UPDATE friends SET name="'.
                                       htmlspecialchars($_POST['name']).'"
                                       WHERE id='.$_POST['id']);
      echo 'Данные изменены';    // и выводим сообщение об изменении данных
      $_GET['id']=$_POST['id']; // эмулируем переход по ссылке на изменяемую запись
}

// формируем и выполняем запрос для получения требуемых полей всех записей таблицы
$sql_res=mysqli_query($mysqli, 'SELECT * FROM friends');

if( !mysqli_errno($mysqli) )    // если запрос успешно выполнен
{
      $currentROW=array(); // создаем массив для хранения текущей записи

      echo '<div id="edit_links">';
      while( $row=mysqli_fetch_assoc($sql_res) ) // перебираем все записи выборки
      {
            // если текущая запись пока не найдена и ее id не передан
               // или передан и совпадает с проверяемой записью
            if( !$currentROW &&
                  (!isset($_GET['id']) || $_GET['id']==$row['id']) )
               {
                     // значит в цикле сейчас текущая запись
                     $currentROW=$row;// сохраняем информацию о ней в массиве
                     echo '<div>'.$row['name'].'</div>';    // и выводим ее в списке
               }
               else    // если проверяемая в цикле запись не текущая
                  // формируем ссылку на нее
                  echo '<a href="?p=edit&id='.$row['id'].'">'.$row['name'].'</a>';
      }
      echo '</div>';

      // формируем HTML-код формы
       echo '<form name="form_edit" method="post" action="/?p=edit">
             <input type="text" name="name" id="name"';
       // если текущая запись определена - устанавливаем значение полей формы
       if( $currentROW ) echo ' value="'.$currentROW['name'].'"';
       echo '><input type="submit" name="button" value="Изменить запись">
             <input type="hidden" name="id" value="';
       if( $currentROW )             // если текущая запись определена
             echo $currentROW['id'];      // передаем ее id как POST параметр формы
       echo '"></form>';
}
else                              // если запрос не может быть выполнен
      echo 'Ошибка базы данных';      // выводим сообщение об ошибке
```

Как и в других модулях в начале модуля "*edit.php*" производится подключение к серверу баз данных. Однако, в отличии от предыдущих рассмотренных вариантов проверки корректности подключения, здесь в случае неудачи предусмотрено принудительное завершение программы с

помощью функции `exit()`. Это упрощает программу, уменьшает количество фигурных скобок, позволяет реализовывать ошибку *404*.

Далее следует проверка необходимости изменить запись таблицы: обратите внимание, все манипуляции с данными необходимо делать до их запроса из базы данных для вывода на экран. Если сделать по-другому, то при изменении, например, фамилии в списке ссылок будет выводится ее старый вариант, что существенно запутает пользователя сайта. Итак, если запись необходимо изменить, признаком чего является наличие переданного методом *POST*-параметра "`button`" и его равенство надписи на кнопке отправки формы, формируется *SQL*-запрос с оператором `UPDATE`, который и изменяет необходимую запись. При этом `id` изменяемой записи также должно передаваться через *POST*-параметр.

После вывода сообщения об успешном изменении записи происходит имитация перехода по ссылке из списка на изменяемую запись. Учитывая, что признаком перехода по ссылке является передача в *PHP*-программу *GET*-параметра `id`, в котором хранится *id* выбранной записи, то для этого достаточно присвоить элементу массива `$_GET['id']` *id* измененной записи, т.е. `$_POST['id']`. Таким образом, дальнейшее формирование *HTML*-кода будет таким же, как если бы мы перешли по ссылке на эту запись, т.е. она будет выбрана, а элементы формы будут заполнены значениями соответствующих полей. Поэтому в дальнейшем нет необходимости проверять: была ли изменена запись и ее необходимо выделить, также, как если бы она была выбрана в списке соответствующих ссылок.

Далее необходимо вывести все хранящиеся в базе данных записи в виде списка ссылок, при переходе по каждой из которых соответствующая ей запись становится текущей. Т.е. необходимый элемент списка ссылок выделяется, элементы формы заполняются значениями полей этой записи, при изменении значений элементов форм и ее отправки изменяются и соответствующие значения полей записи в базе данных.

Для этого уже известными методами формируем *SQL*-запрос, и если он успешно выполнен – выводим список ссылок, если нет – выводим сообщение об ошибке. Перед выводом списка создается пустой массив, в котором будет хранится информация о текущей записи. Т.к. при выводе все равно перебираются в цикле все записи из базы данных, то нет необходимости делать отдельный запрос для определения текущей: если *id* выводимой записи совпадает с *id* текущей, значит выводимая на данной итерации цикла запись и есть текущая и информацию о ней можно сохранить для дальнейшего использования в массиве `$currentROW`.

Именно это и происходит в цикле: если текущая запись еще не найдена и информация о ней не передана в программу (первая загрузка страницы, переход по ссылкам еще не осуществлялся), то первая же выводимая запись станет текущей, ее поля будут сохранены в массиве, а сама она будет выведена не в виде ссылки, а в виде блока `<div>`, т.е. будет выделена в списке. Это же будет сделано если на данной итерации цикла получена запись с переданным в качестве текущего *id*.

Если же текущая запись определена, и она не совпадает с рассматриваемой на данной итерации цикла записью, то будет выведена ссылка с *GET*-параметром `id` равным ее *id*. Как видно из предыдущего описания, при переходе по этой ссылке именно эта запись станет текущей, что и требовалось сделать.

Последнее задания в данном модуле – вывод формы. Она практически полностью совпадает с формой из модуля "*add.php*", но формирует значения полей в соответствии с текущей записью. Т.е. если текущая запись определена, то в *HTML*-код элементов форм добавляется свойства `value` в котором выводится соответствующее элементу значение поля записи (для некоторых элементов формы используется не свойство value, а другие способы определения его значения). Кроме того, в программу должно быть передано в качестве *POST*-параметра значение текущей записи, иначе не будет понятно какую именно запись необходимо изменить. Для этого используется скрытое текстовое поле со свойство `value` равным *id* текущей записи. В принципе, возможна передача этого параметра через свойство `action` формы путем добавления параметра *id* в *URL* страницы-обработчика, но метод со скрытым полем более корректный и безопасный.

Имеет смысл еще раз остановиться на получении информации о текущей записи из базы данных. В приведенном листинге В-1. 7 она копируется в переборе всех записей при их выводе из базы данных. Но, если необходимо выводить не все записи, а только часть из них (например, при реализации постраничной пагинации) то есть вероятность что текущая запись не попадет в перебираемые – в этом случае информация о ней не будет получена.

Кроме того, если комментарий достаточно объемен или просто в результирующей таблице много полей – т.е. если размер в памяти для каждой записи таблицы достаточно большой (а это особенно актуально если в базе данных хранятся, например, фотографии или другие цифровые данные), то такой подход будет очень затратным для ресурсов сервера. Поэтому, в таких случаях необходимо выполнять два разных *SQL*-запроса: для определения полной информации о текущей записи и для получения краткой информации о всех выводимых данных. Скорректируем программу в соответствии с приведенными аргументами, кроме того для примера будем передавать id параметр редактируемой записи через *GET*-параметр.

Листинг В-1. 8

```
$mysqli = mysqli_connect('localhost', 'user', 'password', 'friends');

if( mysqli_connect_errno() ) // если при подключении к серверу произошла ошибка
{
      // выводим сообщение и принудительно останавливаем PHP-программу
      echo 'Ошибка подключения к БД: '.mysqli_connect_error();
      exit();
}

// если были переданы данные для изменения записи в таблице
if( isset($_POST['button']) && $_POST['button']== 'Изменить запись')
{
      // формируем и выполняем SQL-запрос на изменение записи с указанным id
      $sql_res=mysqli_query($mysqli, 'UPDATE friends SET name="'.
                                      htmlspecialchars($_POST['name']).'"
                                      WHERE id='.$_GET['id']);
     echo 'Данные изменены';    // и выводим сообщение об изменении данных
}

$currentROW=array();            // информации о текущей записи пока нет
                                 // если id текущей записи передано -
if( isset($_GET['id']) )        // (переход по ссылке или отправка формы)
{
      // выполняем поиск записи по ее id
      $sql_res=mysqli_query($mysqli,
                  'SELECT * FROM friends WHERE id='.$_GET['id'].' LIMIT 0, 1');
     $currentROW=mysqli_fetch_assoc($sql_res); // информация сохраняется
}
if( !$currentROW )   // если информации о текущей записи нет или она некорректна
{
      // берем первую запись из таблицы и делаем ее текущей
      $sql_res=mysqli_query($mysqli, 'SELECT * FROM friends LIMIT 0, 1);
      $currentROW=mysqli_fetch_assoc($sql_res);
}

// формируем и выполняем запрос для получения требуемых полей всех записей таблицы
$sql_res=mysqli_query($mysqli, 'SELECT id, name FROM friends');

if( !mysqli_errno($mysqli) )    // если запрос успешно выполнен
{
     echo '<div id="edit_links">';
     while( $row=mysqli_fetch_assoc($sql_res) ) // перебираем все записи выборки
     {
          // если текущая запись пока не найдена и ее id не передан
             // или передан и совпадает с проверяемой записью
          if( $currentROW['id']==$row['id']) )
                  // значит в цикле сейчас текущая запись
                  echo '<div>'.$row['name'].'</div>';    // и выводим ее в списке
            else    // если проверяемая в цикле запись не текущая
               // формируем ссылку на нее
```

```
                echo '<a href="?p=edit&id='.$row['id'].'">'.$row['name'].'</a>';
        }
        echo '</div>';

        if( $currentROW ) // если есть текущая запись, т.е. если в таблице есть записи
        {
              // формируем HTML-код формы
                 echo '<form name="form_edit" method="post"
                       action="/?p=edit&id='.$currentROW['id'].'">
                       <input type="text" name="name" id="name" value="'.
                       $currentROW['name'].'"><input type="submit" name="button"
                       value="Изменить запись"></form>';
        }
        else echo 'Записей пока нет';
}
else                                  // если запрос не может быть выполнен
        echo 'Ошибка базы данных';      // выводим сообщение об ошибке
```

При выполнении оператора `UPDATE` поиск изменяемой записи теперь осуществляется по полученному *GET*-параметру `id`. По этой причине нет необходимости имитировал переход по ссылке: этот параметр уже будет в массиве `$_GET`. Далее, как и в прошлом варианте, создаем пустой массив, в который попытаемся записать информацию о текущей записи.

Для этого проверим наличие в массиве элемента `$_GET['id']`, т.е. был ли передан в программу *id* текущей записи. Если он был передан, то попытаемся найти эту запись и получить информацию о ней. Для этого формируем *SQL*-запрос с условием выборки равенства поля *id* переданному параметру. Причем, т.к. нам точно известно, что такая запись всего одна, можно оптимизировать работу сервера явно указав ему это с помощью конструкции "`LIMIT 0, 1`": в этом случае в результирующей таблице будет только одна запись из всей выборки (после ее получения *SQL*-сервер не будет искать другие, а сразу вернет результат – это экономит ресурсы). Полученная с помощью запроса запись и является текущей (сохраняем ее в массив `$currentROW`).

Интересно, что если параметр не был передан или же в нем был передан неправильный *id*, то массив `$currentROW` останется пустым. Поэтому если это произошло, то необходимо повторно сформировать и выполнить *SQL*-запрос для получения первой записи из таблицы. Ее и будем считать текущей согласно заданию лабораторной работы. Если же и после этого массив пуст – это значит, что в таблице нет записей.

Далее происходит выполнение *SQL*-запроса для вывода списка записей: причем в отличии от предыдущего примера отбираются не все поля, а только необходимые для формирования ссылок. В цикле также убраны все операторы для сохранения информации о текущей записи: она уже получена. Достаточно только сравнения *id* выводимой записи и текущей: если они совпали, то вывод осуществляется не в виде ссылки, а в виде блока `<div>`. Обратите внимание: в условии нет необходимости проверять существование элемента массива `$currentROW['id']`. Если этот цикл выполняется, значит в таблице есть записи; если в таблице есть записи – значит текущая запись была гарантирована выбрана.

Для завершения коррекции кода остается только поменять вывод формы. Имеет смысл перед этим проводить проверку наличия текущей записи: если текущей записи нет, то изменять нам нечего и форму выводить не надо – вместо нее выводится предупреждение "*Записей пока нет*". Тогда код формы можно сформировать одним оператором `echo`. Из него было убрано скрытое поле для передачи *id* текущей записи в виде *POST*-параметра, но добавлен в *URL* обработчика формы *GET*-параметр *id* с этим же значением: т.е. передаваемые параметры приведены в соответствие с *PHP*-кодом обработки формы.

Реализуйте оба предложенные варианта модуля, убедитесь в идентичности их работы для пользователя. Кроме того, для приведения функционала модуля "*edit.php*" в соответствие с требованиями лабораторной работы самостоятельно доработайте *CSS*-файл, добавьте в приведенный код вывод текста ссылок в виде фамилии и имени (при необходимости модифицируйте *SQL*-запросы), добавьте сортировку в *SQL*-запросы.

Для удаления записи на основе разобранного материала самостоятельно разработайте и реализуйте модуль "*delete.php*", функционирующий в соответствии с заданием лабораторной работы. Обратите внимание – в списке записей необходимо выводить фамилию и инициалы, а не ФИО целиком.

СПРАВОЧНАЯ ИНФОРМАЦИЯ

Исключительные ситуации

Как правило в объектно-ориентированном подходе к программированию, который будет рассмотрен в последующих заданиях, и довольно часто в процедурном программировании вместо многочисленных проверок данных на корректность и хитроумных сообщений об ошибках, которые передаются из функции в функцию, удобно использовать так называемые исключительные ситуации (исключения).

Исключение – это отправляемое интерпретатором *PHP* сообщение о возникновении нештатной или исключительной ситуации во время выполнения кода программы, которая, как правило, является сигналом о наличии ошибки. К сожалению, не все ошибки в *PHP* приводят к исключительным ситуациям: деление на ноль, ошибки при использовании стандартных функций и т.д. необходимо обрабатывать традиционными способами, интерпретируя коды ошибок и проверяя валидность входных данных.

Но объектно-ориентированные расширения *PHP*, (например, *PDO*), использующие объекты, поддерживают исключения. Для их обработки используются три блока: `try`, `catch` и `finally` – последний необязателен. Интерпретируются они очень просто: если в любой строке блока `try` произошла исключительная ситуация, то дальнейший код блока не выполняется, а начинает выполняться код блока `catch` (исключительная ситуация поймана). Если присутствует блок `finally` – его код выполнится после завершения выполнения кода `try` или `catch`, было исключение поймано или нет.

Листинг В-1. 9

```
function someDBoperation()
{
       try                          // начало блока
       {
/* 01 */   $DBH= new PDO('mysql:host=localhost;dbname='.$dbname, $user, $pass);
/* 02 */   $DBH->exec('DELETE FROM table');      // что-то делаем с БД
       }
       catch(Exception $e)            // начало обработчика исключений
       {
           // вывод сообщения об исключительной ситуации
/* 03 */   echo 'Ошибка БД: '.$e->getMessage();
/* 04 */   return;    // возврат из функции
       }
       finally
       {
           // код, который выполнится даже если случилось исключение
/* 05 */   $DBH = null;
       }
}
```

В приведенном выше примере в блоке `try` осуществляется подключение к базе данных и выполнение *SQL*-запроса к ней. Если все происходит штатно – выполняется блок `finally` в котором освобождается память. Т.е. в этом случае будут выполнены операторы в строках 01, 02 и 05.

Если же в блоке `try` произошло исключение, например при подключении к БД в строке 01, – выполняется блок `catch` в котором выводится сообщение и функция `someDBoperation()` прекращает свою работу: программа выполнит операторы в строках 01, 03, 04 и 05. При этом код блока `finally` все равно гарантировано сработает и память будет освобождена: если код блока

try начал выполняться, то вне зависимости от наличия исключительных ситуаций и любых инструкций (кроме принудительной остановки программы), блок finally будет гарантировано выполнен. Если же очистку памяти вынести за пределы блока, то в случае исключения этот код не выполнится.

Исключительные ситуации – мощный инструмент организации программной архитектуры. Их использование упрощает код, делает его более наглядным и облегчает его понимание другими разработчиками. Для этого в *PHP* реализована возможность самостоятельной генерации исключительных ситуаций и их обработки.

Листинг В-1. 10

```
function div($a, $b)       // пользовательская функция
{
     if( !$b )             // если делитель равен нулю
          throw new Exception('Деление на ноль! '); // выбрасываем исключение
     return $a/$b;              // возвращаем результат деления
}

try
{
     $r = div($a, $b);               // вызываем функцию с попыткой деления на ноль
     echo 'Результат: '.$r.'. ';          // выводим результат
     if( $r > 255 )             // если результат больше 255
          throw new Exception('Выход за диапазон. ');     // выбрасываем исключение
     echo 'Calculation ok! ';        // сообщение об успехе
}
catch(Exception $e)
{
     echo $e->getMessage();     // сообщение об ошибке
}
echo 'Done!';
```

Программа начинает свое выполнение с захода в блок try и вызова из него пользовательской функции div(). Если второй аргумент $b не равен нулю, то она вернет результат деления $a на $b и выведет его в *html*-код. Если результат меньше или равен 255 – то следующим действием будет вывод слов Calculation ok! и Done! Если же он выйдет за пределы 255 – то в блоке try с помощью команды throw выбрасывается исключение, которое отлавливается в блоке catch, где произойдет вывод сообщения об ошибке. Даже если генерация исключительной ситуации происходит в функции, вызываемой в блоке try – она все равно будет обработана и отловлена в блоке catch: в данном примере таким образом проверяется деление на ноль. Таким образом, в зависимости от значений переменных $a и $b результат работы функции можно записать в следующей таблице.

Значения аргументов	$a=10; $b=10;	$a=10; $b=0;	$a=1000; $b=1;
Исключительные ситуации	Нет, нормальное выполнение.	Исключение в функции div()	Исключение непосредственно в блоке try
Выводимый результат	Результат: 1. Calculation ok! Done!	Деление на ноль! Done!	Результат: 1000. Выход за диапазон. Done!

Помимо текстового сообщения о произошедшей исключительной ситуации, *PHP* предоставляет возможность и более детального анализа исключения. Для этого у события Exception есть соответствующие методы, представленные в следующей таблице.

$e->getMessage();	Строка с сообщеним о исключительной ситуации.
$e->getCode();	Строка с кодом исключительной ситуации.
$e->getFile();	Строка с именем *PHP*-файла в котором произошло исключение.
$e->getLine();	Номер строки в PHP-коде в которой произошло исключение.
$e->getTrace();	Массив с содержанием стека вызванных функций.

`$e->getTraceAsString();`	Стек вызванных функций в виде строки.

Внутри блока `try` или `catch` могут размещаться вложенные обработчики исключений. В этом случае исключительная ситуация будет обработана ближайшем по вложенности блоком `catch`.

Листинг В-1. 11

```
try                                   // 01
{
      echo 1;                         // 02
      try                             // 03
      {
            echo 2;                   // 04
            throw new Exception('A');       // 05
            echo 3;                   // 06
      }
      catch(Exception $e)                   // 07
      {
            echo $e->getMessage();          // 08
            echo 4;                   // 09
            throw new Exception('B');       // 10
            echo 5;                   // 11
      }
}
catch (Exception $e)                  // 12
{
      echo $e->getMessage();                // 13
      echo 6;                         // 14
}
```

Данный код начинается с `try` блока в котором в строке 02 выводится 1. Далее идет вложенный `try` блок, в котором после вывода символа «`2`» выбрасывается исключение «`A`». исключение обрабатывается в блоке `catch`, в котором выполняются строки 08, 09 и 10 – в последней выбрасывается исключение «`B`», которое отлавливается в блоке `catch`, выполняющего строки 13 и 14. Таким образом, приведенный пример сформирует *html*-код, содержащий строку «`12A4B6`».

Использование программных модулей

Добавление кода из другого файла во время выполнения программы	`include 'filename';` `// всегда`
	`include_once 'filename';` `// только один раз`
	При добавлении в программу конструкции `include` *PHP* приостанавливает выполнение основного файла и начинает выполнение программы из файла, указанного в конструкции. После завершения программы из модуля продолжается выполнение основного файла. Подключаемые модули также могут подключать модули, являясь для них основным файлом. Для предотвращения ситуаций подключения одного и того же модуля несколько раз используется конструкция `include_once` – в этом случае *PHP* сам будет проверять использовался ли модуль ранее. И если он уже был подключен – то его код выполняться не будет.

Добавление кода из другого файла перед выполнением программы	`require 'filename';` `// всегда`
	`require_once 'filename';` `// только один раз`
	Данная конструкция, в отличии от `include`, не приостанавливает выполнение программы, а заменяет себя на содержимое указанного в ней файла непосредственно перед началом ее выполнения. Это ускоряет процесс подключения, но также и увеличивает размер программы. Конструкция require_once предварительно проверяет был ли модуль подключен ранее. Если был – конструкция игнорируется.

Язык структурированных запросов SQL

Добавление записей в таблицу	`INSERT INTO table VALUES (val_1, … )`
	`INSERT INTO table (col_name_1, …) VALUES (val_1, … )`
	`INSERT INTO table SELECT val_1, … FROM another_table`
	Для добавления записи в таблицу `table` используется оператор `INSERT INTO`. В

	качестве добавляемых данных в скобках перечисляются все значения столбцов по порядку (если имена столбцов не указаны явно) или в том порядке, который указан перед словом `VALUES`. Во втором случае возможен пропуск каких-либо столбцов таблицы, в качестве их значений будут подставлены значения "по умолчанию". Для добавления в таблицу сразу нескольких записей, которые являются результатом выполнения *SQL* запроса, существует оператор `INSERT … SELECT`. Результатом работы оператора `SELECT` всегда является таблица с записями, которые и добавляются в таблицу `table`. Важно чтобы типы и количество столбцов возвращаемой `SELECT` таблицы совпадали с формируемой таблицей.

Примеры использования оператора INSERT	`INSERT INTO users VALUES (1, "Иванов", "qwerty")`
	Добавляем в таблицу `users` одну запись. Порядок следования значений полей строго соответствует порядку следования полей в таблице.
	`INSERT INTO users (id, name, password) VALUES (1, "Иванов", "qwerty")`
	Добавляем в таблицу `users` одну запись. Явно указываем имена полей таблицы `user`. Порядок следования значений соответствует порядку, указанному в операторе.
	`INSERT INTO users VALUES (1, "Иванов", "qwerty"), (2, "Петров", "1234"), (1, "Сидоров", "")`
	Добавляем в таблицу `users` три записи. Порядок следования значений полей строго соответствует порядку следования полей в таблице. Таким способом возможно добавление любого количества записей в таблицу.
	`INSERT INTO users (id, name, password) VALUES (1, "Иванов", "qwerty"), (2, "Петров", "1234"), (1, "Сидоров", "")`
	Добавляем в таблицу `users` три записи. Явно указываем имена полей таблицы `user`. Порядок следования значений соответствует порядку, указанному в операторе. Таким способом возможно добавление любого количества записей в таблицу.
	`INSERT INTO users SELECT id, fio, code FROM admins WHERE age>21`
	Получаем из таблицы admins все записи для которых столбец age имеет значение более `21`. Из этих записей отбираем столбцы `id`, `fio` и `code` значения которых вставляем в таблицу `users`. Имена столбцов в операторе `SELECT` могут не совпадать с необходимыми – главное соблюсти соответствие их типов. Таким образом будет добавлено столько строк, сколько удовлетворяющих условию записей нашлось в таблице `admins`. Оператор `SELECT` может использоваться в любой своей форме (см. ниже).
	`INSERT INTO users (name, password) VALUES ("Иванов", "qwerty")`
	В таблицу users добавляется одна запись. Если поле `id` было объявлено как ключевое с автоинкрементом, оно будет автоматически увеличено на `1` от прошлого значения. Если в дальнейшем необходимо знать вычисленное значение поля, например для использования в другой таблице, можно воспользоваться *SQL*-функцией `LAST_INSERT_ID()`.
	`INSERT INTO news (id, user_id, article) VALUES (1, LAST_INSERT_ID(), "Текст новости …")`
	Добавляет в таблицу `news` одну запись со значением поля `user_id` равному последнему значению автоинкремента. Если до этого выполнялся предыдущий пример оператора, то этим значением будет `id` добавленной записи таблицы `users`.

Удаление записей из таблицы	`DELETE FROM table_name`
	Удаляет все записи из таблицы `table_name`.
	`DELETE FROM table_name WHERE …`
	Удаляет из таблицы `table_name` только те записи, которые удовлетворяют указанному после `WHERE` условию. Например, `DELETE FROM users WHERE id=2` удалит из таблицы `users` записи у которых поле `id` равно `2`. Если `id` – ключевое поле, это означает удаление данной конкретной записи.
	`TRUNCATE TABLE table_name`
	Очищает таблицу удаляя все ее записи.

Изменение записей в	`UPDATE table_name SET field_name_1 = value_1, … WHERE …` Изменяет значение полей таблицы `table_name`. В операторе явно указывается каким

таблице	полям какое значение присваивается. Если поля не указаны, то они не будут изменены. Условие отбора записей в которых будут изменяться поля указывается после `WHERE`. Рассмотрим несколько примеров использования оператора.
	`UPDATE users SET name="Петров", age=19 WHERE id=1`
	В таблице `users` у записи с `id` равным `1` устанавливается значение поля `name` равным строке `"Петров"`, значение поля `age` равным `19`.
	`UPDATE users SET age=22 WHERE id>5`
	В таблице `users` для записей с `id` большим `5` устанавливается значение поля `age` равным `22`. Другие поля не изменяются.
	`UPDATE users SET age=age+1`
	В таблице `users` для всех записей значение поля `age` увеличивается на `1`.

Выборка записей в таблицах	`SELECT field_names FROM table_names WHERE conditions`
	`GROUP BY group_fields ORDER BY order_fields LIMIT rows, offset`
	Очень важно понимать что результатом выполнения оператора `SELECT` всегда является таблица. Наиболее часто встречающаяся форма оператора представлена выше. После слова `SELECT` указываются поля, которые должны входить в результирующую таблицу. После слова `FROM` указываются участвующие в формировании результата таблицы (одна или несколько). Условие отбора записей из исходных таблиц указывается после слова `WHERE`.
	Поля, по которым будет сортироваться результат и порядок сортировки (`ASC` – по возрастанию, `DESC` – по убыванию) указывается после слова `ORDER BY`. Если порядок сортировки не указан – производится сортировка по возрастанию значения. Поля для группировки записей указываются после `GROUP BY` (см. ниже разделе "Функции *SQL*") – не путайте его с полем для сортировки записей! После слова `LIMITS` указывается максимальное количество записей в результирующей таблице и отступ первой из них от начала результирующей таблицы (см. примеры). Все части оператора кроме перечня полей результирующей таблицы и исходных таблиц являются необязательными. Подробнее разбор оператора `SELECT` производится ниже с помощью примеров.

Примеры использования оператора SELECT	`SELECT * FROM users`
	Возвращает все поля и все записи таблицы `users`.
	`SELECT id, name FROM users`
	Возвращает таблицу из всех записей таблицы `users`, но только с полями `id` и `name`.
	`SELECT id, name AS fio FROM users`
	Отбирает в таблице `users` значение полей `id` и name для всех ее записей. При этом в результирующей таблице поле `name` будет переименовано в `fio`.
	`SELECT name FROM users`
	Возвращает таблицу с единственным полем `name`. Количество записей соответствует количеству записей в таблице `users`. Если в ней есть записи с одинаковым полем `name` – в результате будут несколько повторяющихся записей.
	`SELECT DISTINCT name FROM users`
	В отличие от предыдущего варианта, все повторения в результирующей таблице будут убраны.
	`SELECT id FROM users WHERE age>21 AND name<>"Иванов"`
	Формирует таблицу из одного столбца `id` со значениями, взятыми из таблицы `users` при условии, что у записи значение поля `age` больше `21` и значение поля `name` не равна строке `"Иванов"`.
	`SELECT * FROM users WHERE age<22 OR age>25 ORDER BY age`
	В таблице `users` отбираются записи у которых значение поля `age` меньше `22` или больше `25` и из них формируется результирующая таблица. Причем результат будет упорядочен по значениям поля `age` по возрастанию.
	`SELECT * FROM users WHERE id<>2 ORDER BY age, name DESC, id ASC`
	Из таблицы `users` отбираются все записи, у которых поле `id` не равно `2`. В результирующей таблице записи сначала сортируются по полю `age` по возрастанию значений, если значения поля `age` нескольких записей совпадают, то они будут отсортированы по полю `name` по убыванию, если и тут есть совпадение – по

	возрастанию `id`. Ключевое слово `ASC` может не использоваться, т.к. тип сортировки по возрастанию используется по умолчанию.
	
	`SELECT id FROM users ORDER BY age, name DESC`
	Результат – таблица с одним столбцом `id` со значениями из записей таблицы `users` отсортированных сначала по возрастанию поля `age`, а затем по убыванию значения поля `name`.
	
	`SELECT name AS fio, age+1 AS age_1 FROM users WHERE age/2<20`
	`ORDER BY age_1 DESC`
	В результирующей таблице два поля: `fio` и `age_1`. В первое попадают значения поля `name` из таблицы `users`, во второе – значения `age` увеличенные на `1`. Для заполнения полей отбираются только те записи, у которых половина значения поля `age` меньше `20`.
	
	`SELECT name FROM users LIMIT 10, 20`
	Результирующая таблица формируется следующим образом: из таблицы `users` отбираются все значения поля `name`. При этом первые `20` записей пропускаются, а следующие за ними `10` (если они есть) формируют результирующую таблицу.
	
	`SELECT * FROM users WHERE age<23 ORDER BY age LIMIT 0, 15`
	Из таблицы `users` отбираются записи у которых значение поля `age` меньше `23`. Записи сортируются по возрастанию поля `age`. Первые `15` записей из получившегося списка попадают в результирующую таблицу.
	
	`SELECT users.name AS fio, news.article AS text FROM users, news`
	`WHERE user_id=users.id AND age<20 ORDER BY users.id, news.id`
	В результирующей таблице два поля: `fio` и `text`. Первое поле заполняется значениями из поля `name` таблицы `users`, второе – значениями из поля `article` таблицы `news`. Причем из таблицы `users` отбираются только те записи, у которых значение поля `age` меньше `20`. Значения из двух полей разных таблиц комбинируются следующим образом: берется запись таблицы `users`, если в таблице `news` есть записи у которых значение поля `user_id` равно значению `id` этой записи – все эти комбинации добавляются в результат.

Условия отбора в SQL	С помощью конструкций `WHERE` возможен отбор записей из таблицы с помощью указанного за ней условия. В условии могут фигурировать имена полей, вместо которых при проверке будут подставляться значения записей; скобки; математические операторы "`+`", "`-`", "`/`" и "`*`"; логические операторы "`=`", "`AND`", "`OR`" и "`NOT`"; функции. Приведем пример. `… WHERE age>22 AND (name="Иван" OR name="Петр")` Условие соответствует записям, у которых значение поля `age` больше `22`. Кроме того, значение поля `name` этой записи должно быть равно строке `"Иван"` или `"Петр"`. Другими словами, такое условие позволит отобрать из таблицы все людей по имени Иван или Петр которые старше `22` лет.

Функции в SQL	Для обработки значений в *SQL* существуют специальные арифметические, строковые и другие функции. Например, функция `CONCAT()` применяется для соединения нескольких строк.
	
	`SELECT CONCAT(name, "ович") FROM users`
	Возвращает таблицу со значениями, полученными как конкатенация значений поля `name` таблицы `users` и строки "`ович`".
	
	`SELECT * FROM users WHERE CONCAT(name, "ович")=secname`
	Возвращаются все записи таблицы, в которых значение поля `name` в соединении со строкой "`ович`" совпадает со значением поля `secname`.
	Функции `ROUND(X)`, `FLOOR(X)` и `CEILING(X)` – используются для округления аргумента. Причем первая использует правила математики, вторая округляет в меньшую сторону, вторая – в большую.
	Существует масса других функций, подробное описание которых будет размещено в блоке "Базы данных".

Группировка в операторе SELECT	При использовании слова `GROUP BY` в операторе `SELECT` результирующие записи группируются по указанным после слова полям. Это означает что если в таблице есть несколько записей с совпадающими значениями такого поля, то в результат будет

	добавлена только одна запись (первая). Этот метод используется вместе со специальными функциями, которые позволяют обработать значения разных записей для одного поля. Наиболее часто используются следующие функции.
	• `MAX()` – максимальное значение поля;
	• `MIN()` – минимальное значение поля;
	• `COUNT()` – общее количество различных значений в поле;
	• `SUM()` – сумма всех значений поля таблицы;
	• `AVG()` – среднее значение поля таблицы.
	`SELECT MIN(age), MAX(age), AVG(age) FROM users`
	Запрос возвращает таблицу с одной записью, включающей столбцы с минимальным, максимальным и средним возрастом людей информация о которых хранится в таблице.
	`SELECT COUNT(*) FROM users WHERE age>20`
	Возвращает таблицу с одной записью и одним полем в котором указывается число записей таблицы `users` у которых значение поля `age` больше `20`.
	`SELECT name, MAX(age), MIN(age), AVG(age) FROM users GROUP BY name`
	Для каждого имени определяется максимальный, минимальный и средний возраст для всех людей информация о которых хранится в таблице `users`. Запрос возвращает таблицу со столькими записями, сколько уникальных имен содержится в поле `name`.
	Если в операторе `SELECT` не указаны столбцы, по которым осуществляется группировка – то функции действуют на все значения поля.

Язык *SQL* также используется и для других манипуляций с базой данных: создание и удаление таблиц, создание ключей, самих баз данных, полная очистка таблиц и т.д. Выше приведена лишь небольшая часть основных сведений и вариантов использования языка *SQL* предназначенная для получения общего представления о его возможностях. Полностью язык *SQL*, включая подзапросы, будет рассматриваться в блоке "Базы данных". В качестве финального примера, для усвоения понимания, возьмем таблицы *book* и *box* со следующим содержимым.

book

id	Author	Name	Year	b_id
1	Пушкин	Сборник	2016	1
2	Толстой	Петр I	2005	1
3	Пушкин	Метель	1952	2

box

id	Name	Placement	Size
1	Шкаф №1	Комната №1	40
2	Полка №1	Аудитория №2	12
3	Шкаф №2	Комната №1	36

Рассмотрим некоторые SQL-запросы и результат их выполнения.

`SELECT * FROM box`
Запрос всех записей таблицы box.

1	Шкаф №1	Комната №1	40
2	Полка №1	Аудитория №2	12
3	Шкаф №2	Комната №1	36

`SELECT Name, Placement FROM box`
Запрос полей `Name` и `Placement` у всех записей таблицы `box`.

1	Шкаф №1	Комната №1	40
2	Полка №1	Аудитория №2	12
3	Шкаф №2	Комната №1	36

`SELECT id, Name FROM book WHERE Year>1980`
Запрос всех значений полей `id` и `name` у записей таблицы `book`, у которых значение поля `Year` больше `1980`.

1	Пушкин
2	Толстой

`SELECT * FROM book WHERE Author="Пушкин"`
Запрос всех записей таблицы `book` у которых значение поля `Author` равно строке `"Пушкин"`.

1	Пушкин	Сборник	2016	1
2	Толстой	Петр I	2005	1
3	Пушкин	Метель	1952	2

```
SELECT * FROM box LIMIT 1, 2
```
Запрос двух записей из таблицы `box` с отступом в одну запись от первой.

2	Полка №1	Аудитория №2	12
3	Шкаф №2	Комната №1	36

```
SELECT * FROM book WHERE box=1 LIMIT 1, 10
```
Запрос записей таблицы `book` у которых значение поля `box` равно `1`. В результат отбираются десять записей с отступом в одну запись от первой.

2	Толстой	Петр I	2005	1

```
SELECT book.id, book.Name, box.Name,
Placement FROM book, box WHERE b_id=box.id
```
Запрос записей из двух таблиц. В результирующей таблице поля `id` и `Name` из таблицы `book`, а также поля `Name` и `Placement` из таблицы `box`. При этом к записи таблицы `book` присоединяются соответствующие поля таблицы `box` при условии, что значение поля `b_id` равно `id` таблицы `box`.

1	Сборник	Шкаф №1	Комната №1
2	Петр I	Шкаф №1	Комната №1
3	Метель	Полка №1	Аудитория №2

```
SELECT MIN(Year) FROM book
```
Запрос минимального значения поля `Year`.

1952

```
SELECT MAX(Size) FROM box WHERE
Placement="Комната №1"
```
Запрос максимального значения поля `Size` среди всех записей таблицы `box`, у которых поле `Placement` равно строке `="Комната №1"`.

40

```
SELECT Author, MIN(Year) FROM book GOUP BY
Author
```
Запрос минимального значения поля `Year` для каждого автора отдельно.

Толстой	2005
Пушкин	1952

```
SELECT COUNT(*) FROM box WHERE Size>20
```
Запрос количества записей в таблице `box`, у которых значение поля `Size` больше `20`.

2

Доступ к базам данных MySQL с помощью расширения MySQLi

MySQLi – это расширение языка *PHP* для доступа к серверу баз данных *MySQL*. Поддерживает как процедурный стиль программирования, так и объектно-ориентированный. Может не работать на старых версиях *PHP* – в этом случае следует использовать функции предыдущего расширения *PHP* для *MySql*.

Процедурный стиль	
Подключение к серверу баз данных	`$mysqli = mysqli_connect($host, $user, $password, $database);` Функция осуществляет подключение пользователя `$user` с паролем `$password` к базе данных `$database` на сервере `$host`. Возвращает идентификатор подключения.
Получение кода ошибки подключения к серверу	`$errno = mysqli_connect_errno($mysqli);` Возвращает код ошибки при подключении к северу; `NULL`, если соединение успешно установлено.
Получение текста ошибки подключения к серверу	`$error = mysqli_connect_error();` Возвращает текстовое описание ошибки при подключении к северу.
Выполнение SQL-запроса	`$res = mysqli_query($mysqli, $query, $resultmode);` Для успешного подключения с идентификатором `$mysqli` выполняется *SQL*-запрос `$query`. Функция возвращает `FALSE` во всех случаях ошибочного выполнения. Для предполагающих возврат данных запросов будет возвращен идентификатор (объект) с данными, в остальных – `TRUE`. Параметр `$resultmode` определяет тип результата запроса: буферизованный (`MYSQLI_STORE_RESULT`, значение по умолчанию) или не буферизованный (`MYSQLI_USE_RESULT`). Первый тип занимает место в ОП и не выдает данные, пока запрос не будет выполнен до конца.

	Не буферизованный тип не занимает место в ОП, выдает данные сразу после начала работы, но количество возвращаемых им записей нельзя определить функциями *MySQLi*. Также нельзя выполнять другой запрос, пока текущий не закрыт.
Код ошибки выполнения SQL-запроса	`mysqli_errno($mysqli);` Возвращает код ошибки при последнем выполнении *SQL*-запроса для данного соединения.
Текст ошибки выполнения SQL-запроса	`mysqli_error($mysqli);` Возвращает описание ошибки при последнем выполнении SQL-запроса для данного соединения.
Определение количества записей в результате SQL-запроса	`mysqli_num_rows($res);` Работает только для буферизированных запросов. В случае не буферизированного запроса будет возвращать некорректное значение.
Получение текущей записи результата SQL-запроса в виде списка	`$row=mysqli_fetch_row($res);` Возвращает запись из результата выполнения запроса `$res`. Каждый следующий вызов функции возвращает следующую запись результата, если следующей записи нет – будет возвращен `NULL`.
Получение текущей записи результата SQL-запроса в виде ассоциативного массива	`$row=mysqli_fetch_assoc($res);` Работает аналогично предыдущей функции, но в качестве ключей массива используется имена полей возвращаемой таблицы.
Очистка результата запроса	`mysqli_free_result($res);` Для не буферизированных запросов невозможно выполнение следующего запроса без вызова этой функции. Для буферизированного запроса очищается оперативная память с результатами его выполнения.
Разрыв подключения к серверу баз данных	`mysqli_close($link);` Соединение с базой данных принудительно разрывается. Возвращает `TRUE` в случае успеха, `FASLE` – в случае ошибки.
Объектно-ориентированный стиль	
Подключение к серверу базы данных	`$mysqli = new mysqli($host, $user, $password, $database);` Создается объект для подключения к базе данных `$database` на сервере `$host` для пользователя `$user` с паролем `$password`.
Получение кода ошибки подключения к серверу	`$mysqli->connect_errno;` Возвращает код ошибки при подключении к серверу. При отсутствии ошибки – `0`.
Получение текста ошибки подключения к серверу	`$mysqli->connect_error;` Возвращает текст с описание ошибки при подключении к серверу. Объект `$mysqli` обязательно должен быть предварительно создан.
Выполнение SQL-запроса	`$res = $mysqli->query($mysqli, $query, $resultmode);` Параметры и возвращаемый результат абсолютно аналогичны соответствующей функции процедурного стиля.
Код ошибки выполнения SQL-запроса	`$mysqli->errno;` Аналогично процедурному стилю функции.
Текст ошибки выполнения SQL-запроса	`$mysqli->error;` Аналогично процедурному стилю функции.
Определение количества записей в результате SQL-запроса	`$res->num_rows;` Аналогично процедурному стилю функции.
Получение текущей записи результата SQL-запроса в виде списка	`$row=$res->fetch_row();` Аналогично процедурному стилю функции.

Получение текущей записи результата SQL-запроса в виде ассоциативного массива	`$row=$res->fetch_assoc();` Аналогично процедурному стилю функции.
Очистка результата запроса	`$res->close();` Аналогично процедурному стилю функции `mysqli_free_result()`.
Разрыв подключения к серверу баз данных	`$mysqli->close();` Аналогично процедурному стилю функции `mysqli_close()`.

Смешивание процедурного и объектно-ориентированного стилей в рамках одного проекта затрудняет работу с программой и не рекомендуется. Выше рассмотрены лишь основные функции расширения, более полный их список приведен в блоке "Базы данных".

Доступ к базам данных MySQL с помощью расширения PDO

PDO – универсальное расширение PHP для работы с любым сервером баз данных.

Подключение к серверу базы данных MySQL	`$DBH= new PDO('mysql:host=server;dbname=name', $user, $pass);` Осуществляется подключение к серверу *MySQL* по адресу `server`. Используется база данных `name`, пользователь `user` с паролем `pass`. При завершении работы скрипта подключение будет автоматически разорвано.

Обработка ошибок при использовании PDO	`try { … }`
	`catch(PDOException $e) { echo $e->getMessage(); }`
	Для обработки ошибок используются генерируемые *PDO* исключительные ситуации, которые в свою очередь обрабатываются блоком `try/catch` *PHP*. Т.е. если в коде блока `try{}` произошла ошибка (исключительная ситуация) – выполнение программы не будет остановлено, а будет передано в блок `catch{}`. Если ошибок не было – код блока `catch{}` не будет выполнен. Информация об исключительной ситуации передается в объект `$e`. Вывод описания ошибки возможно методом `getMessage()`.

Выполнение простых SQL-запросов	`$STH = $DBH->prepare( $query ); // выполнение запроса`
	`$STH->execute(); // без псевдопеременных`
	Метод `prepare()` подготавливает запрос `$query` к выполнению, если это невозможно – возвращает `FALSE`. Метод `execute()` выполняет запрос.
	`$STH = $DBH->query( $query ); // выполнение простого запроса`
	Сразу выполняет запрос `$query` и возвращает результат.

Выполнение SQL-запросов с безымянными псевдопеременными	`// выполнение запроса с безымянными псевдопеременными`
	`$STH = $DBH->prepare("INSERT INTO person values (?, ?)");`
	`$STH->bindParam(1, $name);`
	`$STH->bindParam(2, $id);`
	`$name = "Николай Петров";`
	`$id = "8У-15А-247";`
	`// INSERT INTO person values ("Николай Петров", "8У-15А-247")`
	`$STH->execute();`
	`$name = "Ксения Иванова";`
	`$id = "9У-17А-132";`
	`// INSERT INTO person values ("Ксения Иванова", "9У-17А-132")`
	`$STH->execute();`
	В подготавливаемом запросе вместо данных ставятся символы "`?`", которые затем с помощью метода `bindParam()` привязываются к *PHP*-переменным. Тогда при выполнении запроса методом `execute()` вместо псевдопеременных в нем будут подставлены текущие значения привязанных переменных. При изменении значений переменных можно всего лишь повторно выполнить метод `execute()` – в подготовленный шаблон будут подставлены новые значения. Такой подход имеет смысл при

	многократном выполнении однотипных запросов: уменьшает нагрузку на сервер БД за счет использования им кэширования результатов; защищает от *SQL*-инъекций; позволяет автоматически экранировать символы.
	`// псевдопеременные размещаются в массиве`
	`$pholders=array("Николай Петров", "8У-15А-247");`
	`$ST->execute($pholders);`
	Для удобства псевдопеременные можно не привязывать методом `bindParam()`, а сразу передать их значения для выполнения запроса с помощью массива. Порядок элементов массива соответствует порядку следования псевдопеременных в шаблоне запроса.

Выполнение SQL-запросов с именованными псевдопеременными	`// с именными псевдопеременными`
	`$STH=$DBH->prepare("INSERT INTO person values (:name, :id)");`
	`$STH->bindParam(':id', $id);`
	`$STH->bindParam(':name', $name);`
	`$STH->execute();`
	Принцип работы абсолютно аналогичен предыдущему случаю, но в шаблоне запроса псевдопеременные указываются не символами "?", а с помощью их имен.
	`// псевдопеременные размещаются в массиве`
	`$pholders = array('name'=>'Катя', 'id'=>'98У-15А-247');`
	`$STH->execute($pholders);`
	Также как и для безымянных псевдопеременных, для именованных возможна передача их значений в запрос с помощью массива, но в этом случае не списка, а ассоциативного массива.

Определение количества затронутых в результате выполнения SQL-запроса строк	`$STH->rowCount();` Для операторов `INSRT`, `UPDATE` и `DELETE` возвращает количество затронутых при их выполнении строк. Для оператора `SELECT` возвращение точного количество строк в результирующей таблице не гарантируется.

Получение результата SQL-запроса (перебор записей)	`while( $row = $stmt->fetch() )`
	`{ … }`
	Для возвращающего результат запроса определяет текущую запись из результирующей таблицы. При первом вызове метода будет возвращена первая запись, при каждом последующем вызове – следующая запись таблицы. Если следующей записи нет – метод вернет `FALSE`.

Очистка результата запроса	`$stmt->closeCursor();` Рекомендуется вызывать метод перед выполнением нового *SQL*-запроса.
ID последней вставленной записи	`lastInsertId();` Возвращает значение индекса последней вставленной записи. Удобно использовать для индексов с автоматическим инкрементом.
Разрыв подключения к серверу баз данных	`$DBH = null;` Принудительно разрывает соединения с базой данных.

Расширение MySQLi также предоставляет функции и методы для предварительной подготовки запросов с использованием псевдопеременных, но их рассмотрение необходимо провести самостоятельно.

КОНТРОЛЬНЫЕ ВОПРОСЫ

Для успешной защиты работы помимо соответствующего требованиям результата необходимо уверенно отвечать на нижеперечисленные и другие вопросы.

1. Что такое внешние модули? Зачем они нужны? Как они подключаются?
2. Что такое библиотека функций? Зачем они нужны? Как они подключаются?

3. В чем отличие конструкций include и require?
4. В чем отличие конструкций include и include_once?
5. В чем отличие конструкций require и require_once?
6. Что такое SQL?
7. Основные операторы языка SQL?
8. Оператор INSERT. Для чего он предназначен, какие формы может иметь?
9. Источники данных для оператора INSETRT: как добавить записи в таблицу из другой таблицы?
10. Оператор UPDATE: назначение, синтаксис, форма записи?
11. Формы оператора SELECT?
12. Условие отбора в операторе SELECT?
13. Функции в операторе SELECT?
14. Сортировка записей и ее типы?
15. Группировка записей: назначение и отличие от сортировки?
16. РаботаSQL- функций с группировкой и без группировки?
17. Ограничение результата SQL-запроса: способы и параметры?
18. Псевдонимы полей в операторе SELECT?
19. Расширение MySQLi – назначение и способ применения?
20. Отличие процедурного и объектно-ориентированного стиля в MySQLi?
21. Буферизированные и не буферизированные запросы в MySQLi?
22. Функции и методы расширения MySQLi?
23. Что такое PDO?
24. Чем отличаются расширения PDO и MySQLi?
25. В чем отличие именованных и неименованных псевдопеременных В PDO?
26. Способы передачи значений в псевдопеременные запроса?
27. Какова цель и преимущества использования псевдопеременных в SQL?
28. Как изменится работа программы, если в листинге В-1. 1 в ссылках подменю не будет передаваться параметр "p"?
29. Почему в листинге В-1. 4 при выводе пагинации в тексте ссылок страницы нумеруются, начиная с единицы, а в адресе ссылок начиная с нуля?
30. В каких случаях имеет смысл делать отдельный SQL-запрос для определения текущей записи из списка выводимых записей таблицы базы данных?
31. Как изменится работа программы в листинге В-1.11, если внутренний блок try-catch вынести в отдельную функцию?
32. Как будет выглядеть сформированная программой на листинге В-1.11 строка, если во вложенный try-catch блок добавить finally, выводящий символ «Z»?