PHP. Работа с базами данных.

- Доступ к базам данных
  - Пример сценария
  - Установка соединения и выбор базы
  - Проверка и фильтрация данных
  - Выполнение запроса
  - Получение результатов запроса
  - Внесение новой информации
  - Освобождение ресурсов

Пример сценария

- Пример сценариев, реализующих интерфейс к базам данных MySQL, представлен здесь: http://217.71.139.74/~gun/samples/lab3
- В любом сценарии, который обеспечивает доступ к базе данных из Web, имеется несколько базовых шагов:
  1. Установка соединения с требуемой базой данных.
  2. Проверка и фильтрация данных пользователя.
  3. Передача запроса в базу данных.
  4. Получение результатов.
  5. Представление результатов пользователю.

Установка соединения

- Для подключения к серверу MySQL в сценарии есть такая строка:
- `@$db = mysqli_connect("localhost", "bookorama", "bookorama");`
- Требуется указать имя узла (**host**), на котором размещен сервер MySQL, имя пользователя (**user**), чтобы войти в него, и пароль (**password**). Если не указать все вышеперечисленное, функция воспользуется значениями по умолчанию — локальная машина вместо узла, имя пользователя, под которым запущен PHP, и пустой пароль.

Установка соединения

- При успехе функция вернет идентификатор связи с базой данных (который следует сохранить для дальнейшего использования), а при неудаче — значение **false**. Результат не стоит игнорировать, поскольку без соединения с базой данных работа невозможна. Это делает следующий код:
- `if (!$db) {`
- `echo "Error: Could not connect to database. Please try again later."; exit; }`

Установка соединения

- Можно использовать функцию **`mysqli_connect()`**. Отличие состоит в том, что **`mysql_pconnect()`** устанавливает постоянное соединение с базой данных.
- Соединение с базой данных закрывается, когда сценарий завершается или когда обращается к функции **`mysql_close()`**. Постоянное соединение остается открытым и после того, как сценарий выполнен, а функцией **`mysql_close()`** его закрыть нельзя.
- Когда вызывается **`mysql_pconnect()`**, она проверит, нет ли уже открытого постоянного соединения. Если есть, она не станет открывать новое. Это и время экономит, и предотвращает перегрузку сервера.

Установка соединения

- Если PHP выполняется как CGI, то постоянное соединение окажется не таким уж и постоянным. (Каждый вызов сценария PHP запускает новую копию механизма PHP и закрывает ее, когда сценарий завершает свою работу. Это, в свою очередь, также закрывает любое постоянное соединение.)
- Количество соединений в MySQL, которые существуют одновременно, ограничено параметром max_connections. Его задача — заставить сервер отвергать новые запросы на соединение, когда ресурсы узла заняты или когда программное обеспечение не функционирует.

Выбор базы данных

- Работая с MySQL, необходимо указывать, какая база данных нужна. Это может сделать PHP-функиия **`mysql_select_db()`**:
- **`mysqli_select_db($cs, "books");`**
- Прототип этой функции выглядит так:
- **`int mysqli_select_db (int database_conn,string database);`**
- Можно использовать соединение с базой данных, для которого требуется выполнить эту операцию, однако, если его не указать, будет использоваться последнее открытое соединение. Если открытое соединение не существует, оно открывается по умолчанию, как если бы вызывалась **`mysql_connect()`**.

Проверка и фильтрация данных

- Перед выполнением запроса к базе данных, возможно, потребуются его параметры, принятые от пользователя через поля форм. Эти параметры подклеиваются к запросу операцией конкатенации.
- Однако сначала необходимо убрать все лишние пробелы по краям слова, которые мог случайно набрать пользователь. Справиться с этим поможет функция **`trim()`**, применяемая к содержимому поля формы **$search** (критерий поиска).
- **`trim($search);`**

Проверка и фильтрация данных

- Еще необходимо фильтровать вводимые данные от управляющих символов. Если записывать данные, введенные пользователем, в базу данных типа MySQL, следует вызывать **`addslashes()`**, а при возврате пользователю выходных данных — **`stripslashes()`**.
- **`$search=addslashes ($search);`**
- К данным, исходящим из базы, применяется **`stripslashes()`**. Сохраняя их в базе данных, мы обращаемся к **`addslashes()`**, а это означает, что при извлечении данных необходимо будет вызывать **`stripslashes()`**.

Проверка и фильтрация данных

- Следующий этап — убедиться, что пользователь указал критерий поиска:
- `if (!$search) {`
- `echo "You have not entered search details. Please go back and try again."; exit; }`
- В этом случае выдается сообщение о том, что критерий поиска не введен.
- Другой вариант – предусмотреть формирование запроса к базе данных без параметров.

Проверка и фильтрация данных

- Функцию **`htmlspecialchars()`** применяют для кодировки символов, которые в HTML имеют особое значение. В наших тестовых данных нет амперсандов (**&**), знаков "меньше" (<), "больше" (>), двойных кавычек ("), однако в названиях многих книг может повстречаться амперсанд. Использование этой функции страхует от грядущих ошибок.
- Проверки критериев поиска на соответствие форматам обычно реализуют через регулярные выражения.

Выполнение запроса

- Чтобы осуществить запрос, можно воспользоваться функцией **`mysql_query()`**. Однако прежде запрос необходимо настроить:
- **`if($search) {`**
- **`$SQL.=" and client.lastname like %".$search. "%";}`**
- В этом случае будет отыскиваться значение, близкое к введенному пользователем (**$search**). Обратите внимание на то, что мы употребили like, отдав ему предпочтение перед equal.
- Теперь можно выполнить запрос:
- **`$rs = mysql_query ($SQL);`**

Выполнение запроса

- Прототип функции **`mysqli_query()`** таков:
- **`int mysqli_query(int database_connection, string query);`**
- В функцию передается запрос, который должен быть выполнен; можно также передать еще и соединение с базой данных. Если его не указать, будет использоваться последнее открытое соединение. Если такового нет, функция откроет соединение точно так же, как при выполнении **`mysqli_connect()`**.
- Вместо этого можно воспользоваться функцией **`mysqli_db_query()`**. Ее прототип:

Выполнение запроса

- `int mysqli_db_query(int db_connection, string database, string query);`
- Здесь можно указать базу данных, в которой требуется производить поиск. В каком-то смысле это комбинация функций **`mysqli_select_db()`** и **`mysqli_query()`**. Обе функции возвращают идентификатор результата (что позволяет получить результаты поиска) в случае успеха и значение **false** в случае неудачи. Идентификатор результата (в нашем случае **`$rs`**) следует сохранить, чтобы извлечь из него некоторую пользу.

Получение результатов запроса

- Разнообразие функций дает возможность получить результат различными способами.
- В нашем примере использовались функции **`mysqli_num_rows($cs, $rs)`**, **`mysqli_num_fields($cs, $rs)`**, **`mysqli_fetch_array($cs, $rs)`**.
- **`mysqli_num_rows()`** сообщает количество строк, которые возвращает запрос В нее следует передать идентификатор результата:
- **`$nr = mysqli_num_rows ($rs);`**

Получение результатов запроса

- Зная количество столбцов
- `$nf=mysqli_num_fields($cs, $rs);`
- можно организовать цикл:
- `for ($i=0; $i <nr;  $i++)`
- `{ //обработка  результатов }`
- Знание количества столбцов позволит вывести шапку таблицы, подписав ее названиями полей базы данных:
- `for ($i=0; $i<$nf; $i++) {`
- `echo"<th>".mysql_field_name($rs,$i );`

Получение результатов запроса

- Последняя функция в интерфейсе mysqli отсутствует, вместо нее можно применить такую конструкцию:
- `$fields=mysqli_fetch_fields(@ss);`
- `Foreach ($fields as $value)`
- `$field[]=$value->name;`

Получение результатов запроса

- Однако выяснять число строк в результате запроса не обязательно. Функции извлечения данных из результатов запроса возвращают нулевой указатель, если данных в результате запроса больше нет. Таким образом, можно организовать цикл while:
- `while`
- `($row= mysqli_fetch_array($rs)) {`
- `for ($j=0; $j<$nf; $j++) {`
- `echo "<td>". stripslashes ($row[$j]);}`
- `}`

Получение результатов запроса

- На каждой итерации цикла происходит вызов **`mysqli_fetch_array()`**. Эта функция берет каждую строку из списка результата и возвращает ее в виде ассоциативного массива, с ключом как именем атрибута и значением как соответствующим значением массива.
- **`$row = mysqli_fetch_array($rs);`**
- **`stripslashes()`** вызывают для того, чтобы "подчистить" от слэшей извлекаемое значение, прежде чем отображать его пользователю.

Получение результатов запроса

- Существуют несколько вариантов получения результата из идентификатора результата. Вместо ассоциативного массива можно воспользоваться нумерованным массивом, применив **`mysqli_fetch_row()`**:
- **`$row = mysqli_fetch_row($rs);`**
- Значения атрибутов будут храниться в каждом порядковом значении **`$row[0]`**, **`$row[1]`**…
- С помощью функции **`mysqli_fetch_object()`** можно выбрать строку внутрь объекта:
- **`$row = mysqli_fetch_object ($rs);`**

Получение результатов запроса

- Во многих ситуациях требуется узнать количество записей, участвующих в запросе SQL с командами INSERT, UPDATE, REPLACE или DELETE. Задача решается функцией **`mysqli_affected_rows()`**. Синтаксис функции:
- **`int mysqli_affected_rows([int идентификатор_соединения])`**
- Параметр идентификатор_соединения не является обязательным. Если он не указывается, **`mysqli_affected_rows()`** пытается использовать последнее открытое соединение.

Отсоединение от базы данных

- Для закрытия непостоянного соединения применяется функция:
- `mysql_close(database_connect±on);`
- Однако в этом нет особой необходимости, поскольку с завершением выполнения сценария соединение закроется автоматически.
- **Внесение новой информации в базу данных.** Внесение новой информации очень похоже на получение существующей. Нужно пройти те же шаги — установить соединение, отправить запрос и проверить результаты. Только в данном случае вместо SELECT будет использоваться INSERT.

Освобождение ресурсов

- Если во время выполнения сценария возникают проблемы, связанные с нехваткой памяти, пригодиться функция **`mysql_free_result()`**. Она вызывается с идентификатором результата:
- **`int mysql_free_result(int rs);`**
- **В** итоге освобождается память, занимаемая результатом. Очевидно, что до обработки результата эта функция вызываться не должна.

Создание и удаление баз данных

- Для создания новой базы данных MySQL из PHP-сценария применяется функция **`mysql_create_db()`**, а для удаления базы данных — **`mysql_drop_db:`**
- **`int mysql_create_db (string database, [int database_connection]);`**
- **`int mysql_drop_db(string database, [int database_connection]);`**
- Обе функции используют имя базы данных и соединение. Если соединения нет, будет использоваться последнее открытое. Функции создают либо удаляют указанную базу данных.

Создание и удаление баз данных

- В случае успеха функции возвращают значение **true**, а в случае неудачи — **false**.
- Аналогично производятся действия с таблицами базы данных.
- Рекомендуется подобные действия не программировать вручную, поскольку это чревато нарушением безопасности доступа к базе данных, а использовать готовое решение, например, пакет phpMyAdmin, см. http://www.phpmyadmin.net

Другие интерфейсы

- PHP поддерживает различные библиотеки, что дает возможность подключаться к огромному количеству баз данных, включая Oracle, Microsoft SQL Server, mSQL и PostgreSQL. В целом принципы подключения и подачи запросов любой из этих баз данных одни и те же. Названия функций могут быть разными, разные базы данных могут иметь разную функциональность, но если вы можете подключиться к MySQL, то другие базы вряд ли поставят вас в безвыходное положение.
- Если необходимо использовать базу данных, которая не имеет специфической библиотеки, доступной в PHP, можно прибегнуть к обобщенным функциям ODBC.