Работа с базами данных

Кулаков Кирилл Александрович

LAMP

- LAMP = Linux, Apache, MySQL, PHP
  - концепция создания Web-приложений
  - свободно распространяемое ПО, альтернатива Microsoft
  - P=Perl или Python
- По аналогии BAMP, SAMP
  - B=BSD-системы
  - S=Solaris

Взаимодействие с БД

- API = Application Programming Interface
- Интерфейсы для MySQL
  - PHP API
  - Perl API
  - C++ API (MySQL++)
  - C API
    - базовый для других интерфейсов
    - библиотека mysqlclient
- Пример использования C API:
  - многие клиенты MySQL написаны на C

Взаимодействие с БД

- Утилита mysql = клиент MySQL

Шаг 1: Подключение к БД

mysql –p

Шаг 2: Выбор БД

mysql> use login;

Шаг 3: Работа с БД

1) вручную

2) из файла (напимер, users.mysql)

mysql> sourse users.mysql;

Пример файла users.mysql

```
SET NAMES utf8;
DROP TABLE IF EXISTS users;
CREATE TABLE users (
    id INT NOT NULL AUTO_INCREMENT,
    username VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    PRIMARY KEY (id) );
INSERT INTO users (username, password) VALUES
("ivanov", "qwerty");
```

PHP + mysql

- PHP API для MySQL
  - MySQL – набор функций работы с MySQL
    - не включает новые возможностей
  - mysqli – обновленный набор функций
    - для MySQL 4.1 и >
    - процедурный и ООП синтаксис

Алгоритм работы с БД

- Шаг 1: Подключение к БД

  $link=mysqli_connect('localhost','login','password','db');

- Шаг 2: Выбор БД

  mysqli_select_db($link, $db_name);

- Шаг 3: SQL-запросы

  $result = mysqli_query($link,"SELECT * FROM users");

- Шаг 4: Работа с данными запроса

  ...

- Шаг N: Закрыть соединение

  mysqli_close($link);

Совместимость версий

- SQLite 2.X:

```
$db = sqlite_open("example.db") or die("failed to open/create the database");
$result = sqlite_query($db, 'select bar from foo');
var_dump(sqlite_fetch_array($result));
```

- SQLite 3.X:

```
$db_handle  = new SQLite3($filename);
$db_handle->exec($query_string);
$result     = $db_handle->query($query_string);
$row        = $result->fetchArray();
```

Расширение Объекты данных PHP (PDO)

- Простой и согласованный интерфейс для доступа к базам данных в PHP
- Реализация на C
- Не является абстрактным слоем (в отличии от PEAR DB, ADOdb)
- Доступно в PHP 5.0, внедрено в PHP 5.1
- Достоинства:
  - унифицированный интерфейс для разных БД
  - поддержка изменений в API БД
  - скорость работы
  - во многих проектах не требуются особенности БД (простота)

Архитектура

- PDO состоит из двух компонент:
  - ядро (интерфейс взаимодействия)
  - драйверы (интерфейс к БД)
- ядро ставится всегда, драйвера — нет
- Поддерживаемые БД:
  - Microsoft SQL Server / Sybase
  - Firebird / Interbase
  - DB2 / INFORMIX (IBM)
  - MySQL
  - OCI (Oracle Call Interface)
  - ODBC
  - PostgreSQL
  - SQLite

Подключение

- Общий шаблон
  - drivername:<driver-specific-stuff>
- Примеры:
  - mysql:host=name;dbname=dbname
  - odbc:odbc_dsn
  - oci:dbname=dbname;charset=charset
  - sqlite:/path/to/db/file
  - sqlite::memory:

Пример: подключение к mysql

```
try {
    $dbh = new PDO("mysql:host=localhost;dbname=test",
    $username, $password);
    echo "Соединение установлено";
} catch (PDOException $e) {
    echo $e->geMessage();
}
```

Пример: подключение к sqlite

- version 3 = "sqlite", version 2 = "sqlite2"
- Файл:

```
try {
    $dbh = new PDO("sqlite:/path/to/db/file.sqlite");
} catch (PDOException $e) {
    echo $e->getMessage();
}
```

- Память:

```
try {
    $dbh = new PDO("sqlite::memory");
} catch (PDOException $e) {
    echo $e->getMessage();
}
```

Постоянное соединение

- Подключение — медленный процесс
  - повторное использование подключения между запросами

```
try {
    $dbh = new PDO($dsn, $user, $pass,
    array(PDO_ATTR_PERSISTENT => true));
} catch (PDOException $e) {
    echo $e->getMessage();
}
```

Закрытие соединения

```
try {
    $dbh = new PDO("mysql:host=localhost;dbname=test",
    $username, $password);
    echo "Соединение установлено";
    do_something($dbh);
    $dbh = null;
} catch (PDOException $e) {
    echo $e->geMessage();
}
```

PDO: insert

```
$dbh = new PDO ("mysql:host=localhost;dbname=test", $login, $password);

$count = $dbh->exec("INSERT INTO users(name, password) values ('user2', 'password')");

echo $count;

$dbh = null;
```

- число добавленных записей или false
  - неправильно if (!$count)
  - правильно if ($count !== false)

PDO: update

```
$dbh = new PDO ("mysql:host=localhost;dbname=test", $login, $password);

$count = $dbh->exec("UPDATE users set password='gjgjgljbej' where name='user2'");

echo $count;

$dbh = null;
```

- число измененных записей или false
  - неправильно if (!$count)
  - правильно if ($count !== false)

PDO: select

```
$dbh = new PDO ("mysql:host=localhost;dbname=test", $login, $password);

$sql = "SELECT * from users";

foreach ($dbh->query($sql) as $row) {

    print $row['name'].':'.$row['password'];

}

$dbh = null;
```

- возвращает объект PDOStatement или FALSE, если запрос выполнить не удалось.

Обработка ошибок

- 2 метода для получения информации об ошибке (результат = false)
  - errorCode() – код ошибки SQLSTATE последней операции с БД (PDO::) или выборки (PDOStatement::)

    Пример: 42000 == Syntax Error
  - errorInfo() – полная информация об ошибке последней операции с БД (PDO::) или выборки (PDOStatement::)

    Пример: array(

    [0] => 42000, # код ошибки SQLSTATE

    [1] => 1064, # код ошибки драйвера

    [2] => You have an error in your SQL syntax; # текст ошибки от драйвера

)
- Атрибут PDO::ERRMODE_EXCEPTION – Передача ошибок через исключения

Получение данных

- Различные способы получения данных
  - Array (индексный или ассоциативный)
  - Strings (для результата из одной колонки)
  - Objects (поля результата = свойства объекта)
  - Callback function
  - Lazy fetching
  - Iterators
  - и т.д.

Пример: получение массивом

```
$res = $db->query("SELECT * FROM foo");
while ($row = $res->fetch(PDO::FETCH_NUM)){
    // $row == array with numeric keys
}
$res = $db->query("SELECT * FROM foo");
while ($row = $res->fetch(PDO::FETCH_ASSOC)){
    // $row == array with associated (string) keys
}
$res = $db->query("SELECT * FROM foo");
while ($row = $res->fetch(PDO::FETCH_BOTH)){
    // $row == array with associated & numeric keys
}
```

Пример: получение строкой

- Многие запросы возвращают один столбец

```
$u = $db->query("SELECT users WHERE login='login' AND
password='password'");


// fetch(PDO::FETCH_COLUMN)
if ($u->fetchColumn($columnIndex)) {
    // login OK
} else { /* authentication failure */ }
```

- проблема с булевыми значениями у fetchColumn

Пример: получение анонимным объектом

- Получение записи в виде объекта (колонка = свойство)

```
$res = $db->query("SELECT * FROM users");
while ($obj = $res->fetch(PDO::FETCH_OBJ)) {
    // $obj == instance of stdClass
    echo $obj->name;
}
```

Пример получение определенным объектом

```
$res = $db->query("SELECT * FROM foo");
$res->setFetchMode(
    PDO::FETCH_CLASS, # режим выборки
    "className",  # имя класса
    array('optional'='Constructor Params')); # массив параметров конструктора
while ($obj = $res->fetch()) {
    // $obj == instance of className
}
```

- Аналог:

  DOStatement::fetchObject ([string $class_name = "stdClass" [, array $ctor_args]])

Пример получение определенным объектом

- Возможность извлечения имени класса из результата выборки

```
$res = $db->query("SELECT * FROM foo");
$res->setFetchMode(PDO::FETCH_CLASS |
    PDO::FETCH_CLASSTYPE);
while ($obj = $res->fetch()) {
    // $obj == объект класса,
    // указанного в первой колонке
}
```

Пример: получение в существующий объект

```
$u = new userObject; # создание объекта

$res = $db->query("SELECT * FROM users");
$res->setFetchMode(PDO::FETCH_INTO, $u);

while ($res->fetch()) {
    // заполнение $u значениями из строки
    // (имя столбца = имя свойства)
}
```

Итеративный результат

- Реализация интерфейса Iterator

```
$res = $db->query("SELECT * FROM users", PDO::FETCH_ASSOC);
foreach ($res as $row) {
    // $row == ассоциативный массив содержащий
    // значения строки
}
```

Пример: ленивое получение

- Результат: объект PDORow (содержит sql запрос в свойстве queryString)
- Переменная не содержит сами запрошенные данные, а отдаёт их по запросу (экономия памяти)
- Чтение данных с текущей позиции курсора

```
$res = $db->query(“SELECT * FROM users”, PDO::FETCH_LAZY);

foreach ($res as $row) {

    echo $row[‘name’]; // получаем только одну колонку

}
```

Получение всего результата

- array PDOStatement::fetchAll ([int $fetch_style [, mixed $fetch_argument [, array $ctor_args = array()]]])
- Возвращает массив записей

```
$qry = “SELECT * FROM users”;
$res = $db->query($qry)->fetchAll(PDO::FETCH_ASSOC);
// $res == индексный массив строк, каждый элемент
// - ассоциативный массив полей
```

Пример: callback функции

```
// callback функция, параметры — имена столбцов
function draw_message($subject,$email) { … }

$res = $db->query("SELECT * FROM msg");
$res->fetchAll(
    PDO::FETCH_FUNC,
    "draw_message"
);
```

Подготовленные запросы

- Проблемы прямых запросов
  - невозможность повторного использования
  - проблемы безопасности (немаскированные данные, sql инъекции, …)
- функция quote() для маскировки специальных символов

```
$qry = “SELECT * FROM users WHERE login=“.
$db->quote($_POST[‘login’])..” AND passwd=“.
$db->quote($_POST[‘pass’]);
```

Подготовленные запросы

- Подготовленные запросы:
  - компилируются 1 раз, выполняются несколько раз
  - отделение структуры запроса от данных
  - быстрее чем прямые запросы

```
$stmt = $db->prepare("SELECT * FROM users WHERE id=?");
$stmt->execute(array($_GET['id']));
$stmt->fetch(PDO::FETCH_ASSOC);
```

Связывание параметров

- Привязка параметров к переменным
- Задание имен параметров

```
$stmt = $db->prepare("INSERT INTO users VALUES(:name,:pass,:mail)");
foreach (array('name','pass','mail') as $v)
    $stmt->bindParam(':'.$v,$$v);
$fp = fopen("./users", "r");
while (list($name,$pass,$mail) = fgetcsv($fp,4096)) {
    $stmt->execute();
}
```

Связывание результата

```
$qry = “SELECT :type, :data FROM images LIMIT 1”;
$stmt = $db->prepare($qry);

# привязка колонки type к переменной $type
$stmt->bindColumn(‘:type’,$type);
# привязка колонки data к выходному потоку
$stmt->bindColumn(‘:data’, STDOUT, PDO::PARAM_LOB);
# получение данных
$stmt->execute(PDO::FETCH_BOUND);

header(“Content-Type: “.$type);
```

Курсоры

- С результирующим набором связан курсор — указатель на текущую запись
- В методе fetch() указание направления движения:
  - PDO::FETCH_ORI_ABS — переход на указанную строку
  - PDO::FETCH_ORI_REL — смещение на указанное количество строк
  - PDO::FETCH_ORI_NEXT — следующая запись
  - PDO::FETCH_ORI_PRIOR — предыдущая запись
  - PDO::FETCH_ORI_FIRST — первая запись
  - PDO::FETCH_ORI_LAST — последняя запись
- closeCursor() - закрытие курсора и освобождение соединения с сервером

Транзакции

```
$db->beginTransaction();
if ($db->inTransaction() && $db->exec($qry1) === FALSE) {
    $db->rollback();
}
if ($db->exec($qry2) === FALSE) {
    $db->rollback();
}
$db->commit();
```

Получение ID добавленной записи

- При добавлении записи обычно ей присваивается уникальный ID
- Можно получить с помощью lastInsertId()

```
if ($db->exec("INSERT INTO …") !== false) {
    $id = $db->lastInsertId();
}
```

Информация

- Версия серверной БД

  $db->getAttribute(PDO::ATTR_SERVER_VERSION);
- версия клиентской библиотеки

  $db->getAttribute(PDO::ATTR_CLIENT_VERSION);
- Информация о сервере

  $db->getAttribute(PDO::ATTR_SERVER_INFO);
- Статус соединения

  $db->getAttribute(PDO::ATTR_CONNECTION_STATUS);
- Список доступных драйверов

  PDO::getAvailableDrivers();

Хранение логина/пароля к БД

- Проблема: PHP может выдать листинг скриптов
- Задача: скрыть данные для подключения к БД
- Решение 1:
  - файл с учетными данными за пределами дерева сайта

    include_once('../db-site.php');
- Решение 2:
  - определение глобальных переменных вне скрипта использование конфигурационных файлов

    $pass = getenv('sql_user_pass')

Хранение данных в Apache

- Конфигурация Apache httpd.conf

  SetEnv sql_user ivanov

  SetEnv sql_user_pass querty

- Файл .htaccess

  SetEnv sql_user ivanov

  SetEnv sql_user_pass querty

- Доступ:

  $_ENV['sql_user_pass']

  getenv('sql_user_pass')