Взаимодействие с СУБД

Лекция 5

Интерфейсы доступа к базам данных

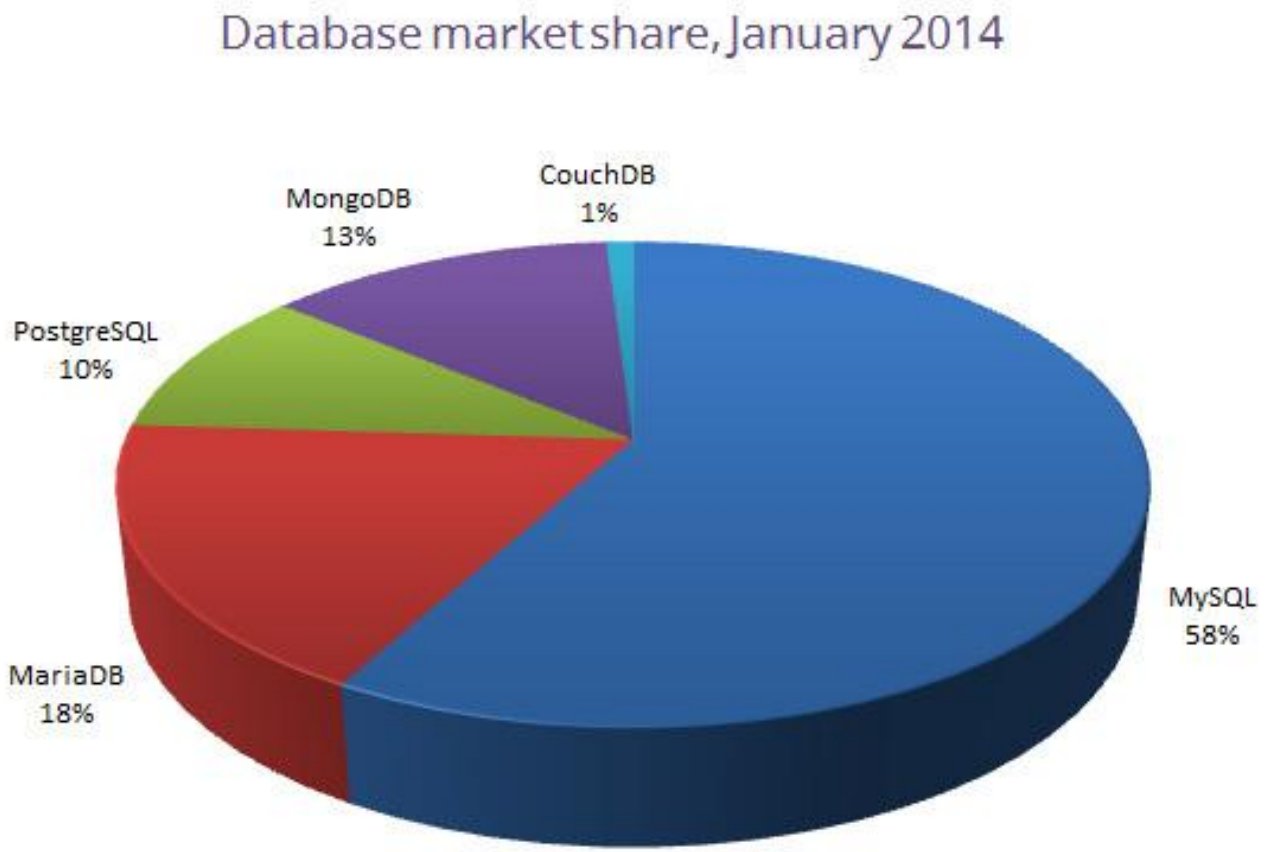
- **Программный интерфейс (API) доступа к базам данных** – промежуточное звено для связи СУБД и языка программирования.
- В языке PHP существуют как **универсальные**, так и **специализированные** интерфейсы доступа к БД.

Базы данных и PHP

- Поддерживаемые интерфейсы СУБД:

MySQL	MySQLi	SQLite
PostgreSQL	Oracle (OCI8)	Oracle
Microsoft SQL Server	Sybase	**ODBC**
mSQL	IBM DB2	Informix
Lotus Notes	DB++	DBM
dBase	DBX	FrontBase
Ingres II	SESAM	Firebird / InterBase
Paradox File Access	MaxDB	**PHP Data Objects (PDO)**

Доминирующая роль MySQL



Способы доступа к MySQL в PHP

Для доступа к СУБД MySQL из PHP существует три стандартных решения:

- Стандартная библиотека MySQL *(устарела)*
- Объектно-ориентированная библиотека MySQLi *(громоздка)*
- Универсальное расширение PHP Data Objects (PDO) *(в самый раз!)*.

Интерфейс PHP Data Objects

- **PHP Data Objects (PDO)** — расширение для PHP, предоставляющее разработчику универсальный интерфейс для доступа к множеству баз данных, в т.ч. PostgreSQL, MySQL, Oracle (OCI8), SQLite.
- Чтобы узнать, для каких СУБД установлены драйверы в PDO, необходимо выполнить:

```
print_r(PDO::getAvailableDrivers());
```

Имя источника данных (DSN)

- **Имя источника данных** *(Data Source Name, DSN)* – структура данных (строка), используемая для описания соединения к источнику данных. Используется во многих библиотеках, включая PDO и ODBC.
- DSN включает:
  - Имя источника данных
  - Месторасположение источника данных
  - Имя драйвера для доступа к источнику данных
  - Идентификатор пользователя для доступа к данным (если требуется)
  - Пароль для доступа к данным (если требуется)

Соединение с базой данных через PHP Data Objects

```php
$host    = "localhost"; // сервер баз данных
$db_name = "smiig";     // имя базы данных
$charset = "utf-8";     // кодировка базы данных
$user    = "pavel";     // имя пользователя
$pass    = “123456";    // пароль
$dsn = "mysql:host=$host;dbname=$db_name;charset=$charset";

// Дополнительные опции
$opt = array(
   PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
   PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC );

// указатель на соединение
$dbh = new PDO($dsn, $user, $pass, $opt);
```

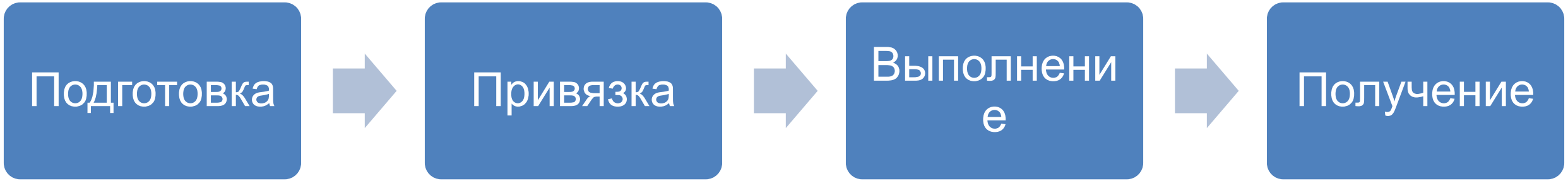
Соединение с базой данных через PHP Data Objects

```
try {
    # MS SQL Server и Sybase через PDO_DBLIB
    $dbh = new PDO("mssql:host=$host;dbname=$db_name", $user, $pass);
    $dbh = new PDO("sybase:host=$host;dbname=$db_name", $user, $pass);
    # MySQL через PDO_MYSQL
    $dbh = new PDO("mysql:host=$host;dbname=$db_name", $user, $pass);
    # SQLite
    $dbh = new PDO("sqlite:my/database/path/database.db");
}
catch(PDOException $e)
{
    echo $e->getMessage();
}
```

Обработка исключений

```
try
{
   $dbh = new PDO($dsn, $user, $password);
}
catch (PDOException $e)
{
   die('Подключение не удалось: ' . $e->getMessage());
}
```

Этапы выполнения запроса



- Каждый запрос должен быть выполнен в 3 (или 4 этапа)
  - prepare() – подготовка SQL-запроса
  - bindValue() / bindParam() – привязка данных к запросу (необязательно)
  - execute() – выполнение запроса (и привязка данных)
  - fetch() и аналоги – получение данных.

Вставка, обновление и удаление записей

```
$query_insert = $dbh->prepare("INSERT INTO folks ( first_name ) values ( 'Пушкин' )");
$query_update = $dbh->prepare("UPDATE folks SET name = 'Пушкин' ");
$query_delete = $dbh->prepare("DELETE FROM folks WHERE name = 'Пушкин' ");
$query_insert->execute();
$query_update->execute();
$query_delete->execute();
```

Подготовленные выражения

- **Подготовленные выражения (Prepared statements)** – заранее скомпилированное выражение, которое может быть многократно выполнено путем отправки серверу лишь различных наборов данных.
- Каждое скалярное значение, подставляемое в запрос динамически, должно быть представлено именованным (:name) или обычным placeholder’ом.

Примеры подготовленных выражений

```
# без placeholders - дверь SQL-инъекциям открыта!
$query = $dbh->prepare("INSERT INTO folks (name, addr, city) values ($name, $addr, $city)");
# безымянные placeholders
$query = $dbh->prepare("INSERT INTO folks (name, addr, city) values (?, ?, ?)");
# именные placeholders
$query = $dbh->prepare("INSERT INTO folks (name, addr, city) values (:name, :addr, :city)");
```

Безымянные placeholder’ы

```
# безымянные placeholders
$query = $dbh->prepare("INSERT INTO folks (name, addr, city) values (?, ?, ?)");

# назначаем переменные каждому placeholder, с индексами от 1 до 3
$query->bindParam(1, $name);
$query->bindParam(2, $addr);
$query->bindParam(3, $city);

# вставляем одну строку
$name = "Александр Пушкин";
$addr = "набережная реки Мойки, д. 12";
$city = "Санкт-Петербург";
$query->execute();

# вставляем еще одну строку, уже с другими данными
$name = "Михаил Лермонтов";
$addr = "набережная реки Фонтанки, д. 14";
$city = "Санкт-Петербург";
$query->execute();
```

Безымянные placeholder’ы

- Если в запросе слишком много параметров, можно задать их в виде одного массива вместо нескольких переменных.

```
# набор данных, которые мы будем вставлять
$data = array('Александр Пушкин',
   'набережная реки Мойки, д. 12',
   'Санкт-Петербург');
$query = $dbh->prepare("INSERT INTO folks (name, addr,
city) values (?, ?, ?)");
$query->execute($data);
```

Именные placeholder’ы

```
# первым аргументом является имя placeholder’а
# его принято начинать с двоеточия
# хотя работает и без них
$query->bindParam(':name', $name);
$query->bindParam(':addr', $addr);
$query->bindParam(':city', $city);
```

Именные placeholder’ы

```
# здесь тоже можно передавать массив, но он должен
# быть ассоциативным в роли ключей выступают
# placeholder’ы
# данные, которые мы вставляем
$data = array( 'name' => ‘Александр Пушкин',
          'addr' => 'набережная реки Мойки, д.12',
          'city' => ‘Санкт-Петербург' );

$query = $dbh->prepare("INSERT INTO folks (name,
addr, city) VALUES (:name, :addr, :city)");

$query->execute($data);
```

Именные placeholder’ы

```
# класс для простенького объекта
class person {
    public $name;
    public $addr;
    public $city;
    function __construct($n,$a,$c)
    {
        $this->name = $n;
        $this->addr = $a;
        $this->city = $c;
    } # так далее...
}
$pushkin = new person('Александр Пушкин', 'набережная реки Мойки, д.12', 'Санкт-Петербург');
# а тут самое интересное
$query = $dbh->prepare("INSERT INTO folks (name, addr, city) values (:name, :addr, :city)");
$query->execute((array)$pushkin);
```

Выборка данных

Данные могут быть получены методом `->fetch`.

Существует несколько констант для установки режимов получения данных:

- **PDO::FETCH_ASSOC:** возвращает массив с названиями столбцов в виде ключей
- **PDO::FETCH_NUM:** возвращает массив с ключами в виде порядковых номеров столбцов
- **PDO::FETCH_BOTH (по умолчанию):** возвращает массив с индексами как в виде названий столбцов, так и их порядковых номеров.
- **PDO::FETCH_OBJ:** возвращает анонимный объект со свойствами, соответствующими именам столбцов
- **PDO::FETCH_CLASS:** присваивает значения столбцов соответствующим свойствам указанного класса. Если для какого-то столбца свойства нет, оно будет создано
- И некоторые другие

Для установки режима получения данных существует следующий синтаксис:

```
$query->setFetchMode(PDO::FETCH_ASSOC);
```

Режим FETCH_ASSOC

```
# поскольку это обычный запрос без placeholder'ов,
# можно сразу использовать метод query()
$query = $dbh->query('SELECT name, addr, city FROM
folks');
# устанавливаем режим выборки
$query->setFetchMode(PDO::FETCH_ASSOC);
while($row = $query->fetch())
{
   echo $row['name'] . "\n";
   echo $row['addr'] . "\n";
   echo $row['city'] . "\n";
}
```

fetchAll()

Возвращает массив, содержащий все строки результирующего набора

```
$query = $dbh->query('SELECT name, addr, city FROM folks');
$query->setFetchMode(PDO::FETCH_ASSOC);
$row = $query->fetchAll())

   echo $row[0]['name'] . "\n";
   echo $row[1]['name'] . "\n";
```

fetchAll()

Использование этого метода для извлечения строк больших результирующих наборов может пагубно сказаться на производительности системы и сетевых ресурсов.

Вместо извлечения всех данных и их обработки в PHP рекомендуется использовать встроенные средства СУБД. Например, использование выражений WHERE и ORDER BY языка SQL может уменьшить размеры результирующего набора.

PDO и оператор LIMIT

- В режиме эмуляции данные, которые были переданы напрямую в execute(), форматируются как строки. То есть, экранируются кавычками. Поэтому `LIMIT ?,?` превращается в `LIMIT '10', '10'` и очевидным образом вызывает ошибку синтаксиса.

```php
$query = $dbh->prepare('SELECT * FROM table LIMIT ?, ?');
$query->bindParam(1, $limit_from, PDO::PARAM_INT);
$query->bindParam(2, $per_page, PDO::PARAM_INT);
$query->execute();
$data = $query->fetchAll();
```

Метод lastInsertID()

- Метод ->lastInsertId() возвращает id последней вставленной записи. Стоит заметить, что он всегда вызывается у объекта базы данных (у нас это $dbh), а не объекта с выражением ($query).

```
$query = $dbh->prepare("INSERT INTO folks ( first_name )
values ( 'Пушкин' )");
$query->execute();
# Выведет последний ID, вставленный в БД
echo $dbh->lastInsertId();
```

Метод exec()

- Запускает SQL запрос на выполнение и возвращает количество строк, задействованных в ходе его выполнения

```
/* Удалить все строки, где присутствует слово "набережная" в адресе */
$count = $dbh->exec("DELETE FROM folks WHERE addr LIKE '%набережная%'");
/* Возвращает количество удалённых строк */
print("Удалено $count строк.\n");
```

Метод rowCount()

- Возвращает количество строк, модифицированных последним SQL запросом.

Закрытие соединения

```
# закрывает подключение
$dbh = null;
```

Администрирование баз данных MySQL

- phpmyadmin (http://mati.su/db/phpmyadmin/)
- Команда mysql из командной строки операционной системы

Ссылки по теме

- http://phpfaq.ru/pdo
- http://habrahabr.ru/post/137664/
- http://www.php.net/manual/en/pdo.constants.php
- http://blog.jelastic.com/2014/02/06/software-stacks-market-share-january-2014/
- https://dev.mysql.com