1. Модель
2. Классы Zend_Db_Table и Zend_Config
3. Запросы
4. Свойства и методы таблицы

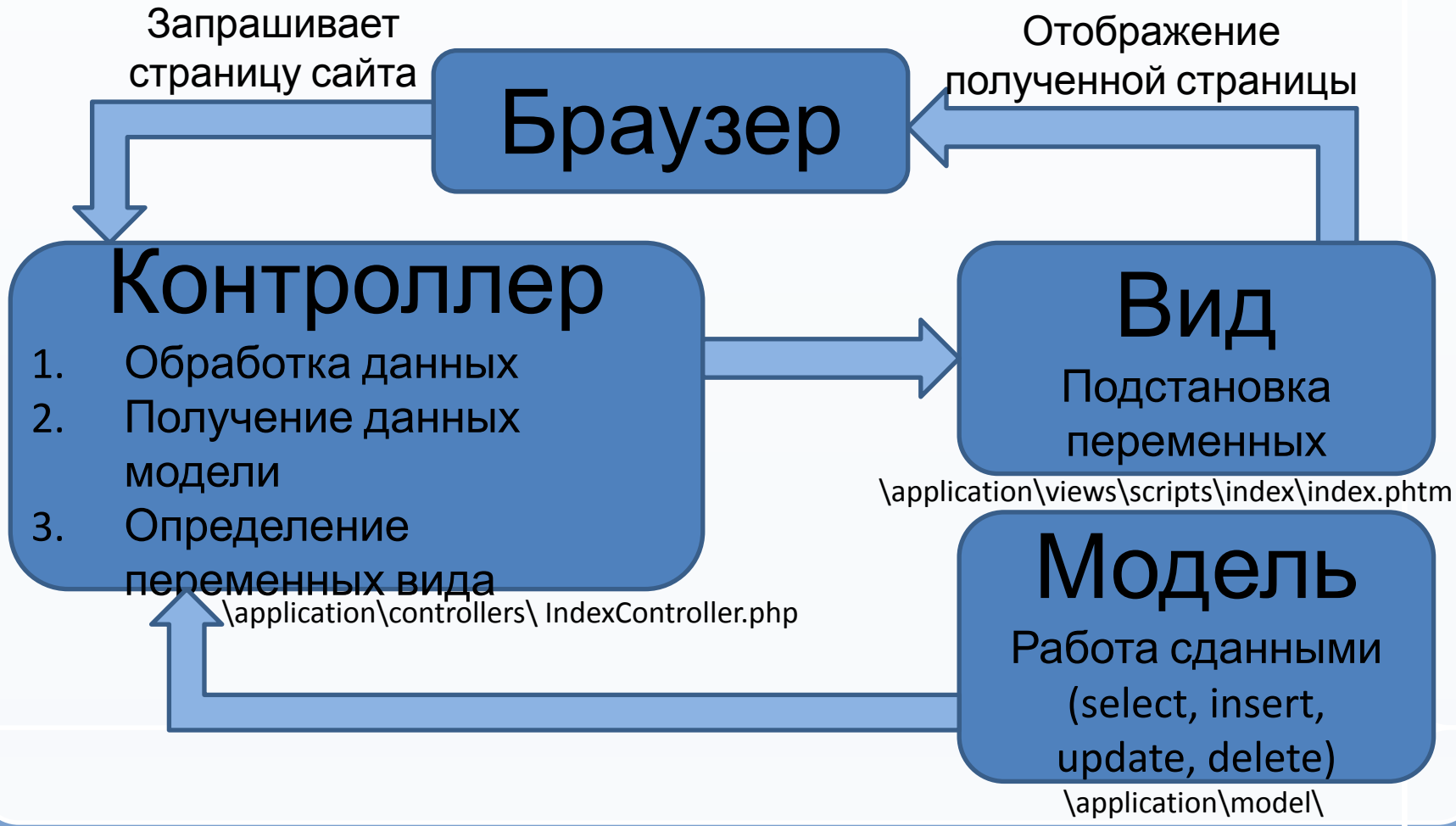
Работа с БД в Zend

Лекция №4

Модель

- Модель – основная логическая часть системы, которая относится к работе с данными.
- Модели широко применимы в управляющей логике приложений, имеющих в своей основе базы данных.

Взаимодействие Модель – Вид – Контроллер



Zend Framework Zend_Db_Table

- Модель — базовая часть приложения, которая реализует его основные функции, следовательно, модель выполняет работу с базой данных.

Класс Zend Framework Zend_Db_Table используется для

- поиска
- вставки
- обновления
- удаления записей
- и т.д.

Если в классе таблицы не были указаны первичные ключи, то эта таблица не может использоваться через Zend_Db_Table.

- Для использования Zend_Db_Table, понадобится сообщить ему, к какой базе данных, под каким именем пользователя и с каким паролем он будет обращаться. Принимая во внимание, что эту информацию предпочтительно не забивать в код, используем конфигурационный файл для ее хранения.

- Zend Framework предоставляет для этой цели класс Zend_Config, обеспечивающий гибкий объектно-ориентированный доступ к конфигурационным файлам в форматах INI и XML.

Пример Zend_Config

- Пример файла config.ini:

```
[general]
db.adapter = PDO_MYSQL
db.config.host = localhost
db.config.username = rob
db.config.password = 123456
db.config.dbname = zftest
```

- Загружать конфигурационный файл из файла начальной загрузки /index.php:

```
Zend_Loader::loadClass('Zend_Controller_Front');
Zend_Loader::loadClass('Zend_Config_Ini');
Zend_Loader::loadClass('Zend_Registry');
// load configuration
$config = new Zend_Config_Ini('./application/config.ini',
'general');
$registry = Zend_Registry::getInstance();
$registry->set('config', $config);
// setup controller
```

Использование Zend_Db_Table

- Для того, чтобы использовать класс Zend_Db_Table, нам понадобится передать в него параметра доступа к базе данных, которые были ранее загружены. Для этого необходимо создать объект Zend_Db и зарегистрировать его функцией Zend_Db_Table::setDefaultAdapter().

Пример (/index.php):

```
1.  Zend_Loader::loadClass('Zend_Controller_Front');
2.  Zend_Loader::loadClass('Zend_Config_Ini');
3.  Zend_Loader::loadClass('Zend_Registry');
4.  Zend_Loader::loadClass('Zend_Db');
5.  Zend_Loader::loadClass('Zend_Db_Table');
6.      // load configuration
7.      $config = new Zend_Config_Ini('./application/config.ini',
    'general'); $registry = Zend_Registry::getInstance();
8.      $registry->set('config', $config);
9.      // setup database
10.     $db = Zend_Db::factory($config->db->adapter,
    $config->db->config->toArray());
11.     Zend_Db_Table::setDefaultAdapter($db);
12.     // setup controller
```

Создание нового класса

- Zend_Db_Table — абстрактный класс, поэтому на его основе необходимо создать специализированный класс-наследник. Имя нового класса не имеет принципиального значения, но такие классы стоит называть аналогично соответствующим им таблицам БД (это повысит уровень самодокументируемости кода). Например, для таблицы album имя класса будет Album.
- Для того, чтобы передать в Zend_Db_Table имя таблицы, которой он будет управлять, необходимо присвоить это значение защищенному свойству класса $_name. Стоит отметить, что в классе Zend_Db_Table по-умолчанию всегда используется ключевое автоинкрементное поле таблицы с именем id. При необходимости значение имени этого поля так же можно менять.

Пример:

Класс Album будет храниться в директории моделей.

/application/models/Album.php:

```
<?php
class Album extends Zend_Db_Table {
protected $_name = 'album';} ?>
```

Создание таблицы

- Мы будем использовать базу данных MySQL, поэтому SQL запрос на создание таблицы будет выглядеть так:

```
1.  CREATE TABLE album (
2.  id int(11) NOT NULL auto_increment, artist varchar(10 0)
    NOT NULL, title varchar(100) NOT NULL, PRIMARY KEY (id)
3.  );
```

Добавление тестовой записи

- Добавим несколько тестовых записей в таблицу для проверки функциональности главной страницы, на которой они в последствии должны отображаться.

```
2.  INSERT INTO album (artist, title) VALUES
3.  ('James Morrison', 'Undiscovered'), ('Snow Patrol', 'Eyes Open');
```

Запросы

При помощи класса Zend_Db_Select

- Объектно-ориентированные методы для "покусочного" построения SQL-запросов;
- Заключение в кавычки идентификаторов и значений для снижения угрозы атак с использованием SQL-инъекций.

При помощи прямых SQL -запросов

- Для очень простых запросов SELECT обычно проще указать SQL-запрос целиком в виде строки и выполнить его, используя такие методы адаптера, как query() или fetchAll()

- При построении запроса вы можете добавлять по одному его предложения. Предложение - это часть SQL-оператора, не представляющая собой законченный оператор; например, предложение WHERE. Для каждого предложения есть свой метод Zend_Db_Select.

```
1.   // Создание объекта Zend_Db_Select
2.  $select = $db->select();
3.   // Добавление предложения FROM
4.  $select->from( ...определение таблицы и столбцов... )
5.   // Добавление предложения WHERE
6.  $select->where( ...определение критериев поиска... )
7.   // Добавление предложения ORDER BY
8.  $select->order( ...определение критериев сортировки... );
```

Свойства таблицы

Указывайте таблицу, для которой определен этот класс, используя защищенную переменную $_name. Переменная должна содержать **имя таблицы** в том виде, в котором она представлена в БД.

```
1.  class bugs extends Zend_Db_Table_Abstract
2.  {
3.   // имя таблицы соответствует имени класса
4.  }
```

Вы можете также объявить **схему таблицы** в защищенной переменной $_schema или через добавленное перед именем таблицы имя схемы в свойстве $_name. Схема, указанная с помощью свойства $_name, имеет более высокий приоритет, чем схема, объявленная с помощью свойства $_schema.

```
1.  class Bugs extends Zend_Db_Table_Abstract
2.  {
3.  protected $_schema = 'bug_db';
4.  protected $_name = 'bugs';
5.   }
```

Каждая таблица должна иметь **первичный ключ**. Вы можете объявить столбец для первичного ключа, используя защищенную переменную $_primary. Это может быть строка с именем одного столбца или массив имен столбцов, если первичный ключ является составным.

```
1. class Bugs extends Zend_Db_Table_Abstract
2. {
3. protected $_name = 'bugs';
4. protected $_primary = 'bug_id';
5. }
```

Если вы не задали первичный ключ, то Zend_Db_Table_Abstract пытается определить его, основываясь на данных, полученных через метод describeTable().

Методы

Вставка строк

- Для того, чтобы вставить новую строку в свою таблицу, просто вызывайте insert() с ассоциативным массивом из пар "имя столбца": "значение". В данные будут автоматически добавлены кавычки; метод возвращает последний добавленный ID. (Обратите внимание на то, что этим он отличается от Zend_Db_Adapter::insert(), который возвращает количество затронутых строк.)

```
1.  <?php
2.  // INSERT INTO round_table
3.  //     (noble_title, first_name, favorite_color)
4.  //     VALUES ("King", "Arthur", "blue")
5.  class RoundTable extends Zend_Db_Table {}
6.  $table = new RoundTable();
7.  $data = array(
8.  'noble_title' => 'King',
9.  'first_name'  => 'Arthur',
10. 'favorite_color' => 'blue',
11. )
12. $id = $table->insert($data);
13. ?>
```

Обновление строк

- Для того, чтобы обновить любое количество строк в своей таблице, вызывайте update() с ассоциативным массивом из пар "имя столбца": "значение для установки", наряду с этим передается условие WHERE для определения, какие строки дожны быть обновлены. Метод обновит таблицу и вернет количество затронутых строк.

- В данные для установки будут будут автоматически добавлены кавычки, но это не относится к условию WHERE, поэтому вам нужно самим добавить кавычки с помощью принадлежащего таблице объекта Zend_Db_Adapter.

```
1.  <?php
2.  //
3.  // UPDATE round_table
4.  //    SET favorite_color = "yellow"
5.  //    WHERE first_name = "Robin"
6.  //
7.  class RoundTable extends Zend_Db_Table {}
8.  $table = new RoundTable();
9.  $db = $table->getAdapter();
10. $set = array(
11.     'favorite_color' => 'yellow',
12. )
13. $where = $db->quoteInto('first_name = ?',
    'Robin');
14. $rows_affected = $table->update($set,
    $where);
15. ?>
```

Удаление строк

- Для того, чтобы удалить любое количество строк в своей таблице, вызывайте delete() с условием WHERE для определения, какие строки должны быть удалены. Метод будет возвращать количество удаленных строк.
- В условие WHERE не добавляются кавычки за вас, поэтому вам нужно самим добавить кавычки с помощью объекта Zend_Db_Adapter таблицы.

```
1.  <?php
2.  //
3.  // DELETE FROM round_table
4.  //    WHERE first_name = "Patsy"
5.  //
6.  class RoundTable extends
    Zend_Db_Table {}

7.  $table = new RoundTable();
8.  $db = $table->getAdapter();

9.  $where =
    $db->quoteInto('first_name = ?',
    'Patsy');

10. $rows_affected =
    $table->delete($where);
11. ?>
```

Извлечение одной строки

- Несмотря на то, что вы можете использовать метод find() для поиска строк по их первичным ключам, часто вам нужно добавлять различные условия, когда извлекате строки. Zend_Db_Table предоставляет fetchRow() только для этой цели. Вызывайте fetchRow() с условием WHERE (и необязательно условием ORDER), и Zend_Db_Table вернет Zend_Db_Table_Row, который будет с первой строкой, удовлетворяющей вашим условиям.

- Заметьте, что в условие WHERE не добавляются кавычки за вас, поэтому вам нужно самим добавить кавычки с помощью объекта Zend_Db_Adapter таблицы.

```
1.  <?php
2.  //
3.  // SELECT * FROM round_table
4.  //     WHERE noble_title = "Sir"
5.  //     AND first_name = "Robin"
6.  //     ORDER BY favorite_color
7.  //
8.  class RoundTable extends
    Zend_Db_Table {}
9.  $table = new RoundTable();
10. $db = $table->getAdapter();
11. $where = $db->quoteInto('noble_title =
    ?', 'Sir')
12.       . $db->quoteInto('AND first_name =
    ?', 'Robin');
13. $order = 'favorite_color';
14. $row = $table->fetchRow($where,
    $order);
15. ?>
```

Извлечение множества строк

- Если вам нужно получить больше одной строки за раз, используйте метод fetchAll(). Как и fetchRow(), он принимает условия WHERE и ORDER, но, кроме этого, принимает еще количество строк и смещение для ограничения количества возвращаемых строк. Метод будет возвращать объект Zend_Db_Table_Rowset с выбранными строками.

```
1.  <?php
2.  class RoundTable extends Zend_Db_Table {}
3.  $table = new RoundTable();
4.  $db = $table->getAdapter();
5.  // SELECT * FROM round_table
6.  //     WHERE noble_title = "Sir"
7.  //     ORDER BY first_name
8.  //     LIMIT 10 OFFSET 20
9.  $where = $db->quoteInto('noble_title = ?', 'Sir');
10. $order = 'first_name';
11. $count = 10;
12. $offset = 20;
13. $rowset = $table->fetchAll($where, $order,
    $count, $offset);
14. ?>
```

Добавление кавычек против SQL-инъекций

- Вы должны всегда окружать кавычками значения, которые подставляются в оператор SQL - это поможет предотвратить атаки посредством SQL-инъекций. Zend_Db_Adapter предоставляет два метода (посредством включенного объекта PDO) для того, чтобы помочь вручную добавить кавычки.

- Первый из них - метод quote(). Он должным образом добавит кавычки в скалярное значение в соответствии с вашим адаптерои БД. Если вы пытаетесь добавить кавычки в массив, то метод вернет строку из значений массива, разделенных запятыми и с добавленными кавычками (это полезно для функций, которые принимают список параметров).

```
1.  <?php
2.  // создается объект $db, предполагается, что адаптером
    является Mysql
3.  // добавление кавычек в строку
4.  $value = $db->quote('St John"s Wort');
5.  // значением $value сейчас является строка '"St John\"s Wort"'
6.  // (обратите внимание на окружающие кавычки)
7.  // добавление кавычек в массив
8.  $value = $db->quote(array('a', 'b', 'c');
9.  // значением $value сейчас является '"a", "b", "c"'
10. // (разделенная запятыми строка)
11. ?>
```

- Вторым является метод quoteInto(). Вы передаете в него строку-основу с указанием меток заполнения в виде вопросительных знаков и после одно скалярное значение, либо массив для подстановки с добавлением кавычек в строку-основу. Это полезно для построения запросов и условий в порядке следования. Скалярные значения и массивы обрабатываются точно так же, как в методе quote().

```
1.  <?php
2.  // создается объект $db, предполагается, что адаптером является
    Mysql
3.  // подстановка скалярного значения в условие WHERE
4.  $where = $db->quoteInto('id = ?', 1);
5.  // значением $where сейчас является 'id = "1"'
6.  // (обратите внимание на окружающие кавычки)
7.  // подстановка массива в условие WHERE
8.  $where = $db->quoteInto('id IN(?)', array(1, 2, 3));
9.  // значением $where сейчас является 'id IN("1", "2", "3")'
10. // (разделенная запятыми строка)
11. ?>
```