Padrões, PEAR e Frameworks PHP

Professor: FLÁVIO GOMES DA SILVA LISBOA (FGSL)

AULA 2 – A MISSÃO!

Ou A Vingança do Mapeamento Objeto-Relacional

Plano de Aulas

Dia	Conteúdo
1	Motivação para o uso de frameworks. Instalação e uso do Eclipse com plugin PDT. Padrão de Projeto MVC. Apresentação do Zend Framework. Projeto Mínimo. Padrões de Projeto Singleton, Controller Front e Controller Page. Controle de Erros.
2	Componente de acesso ao banco. Mapeamento Objeto-Relacional. Abstração da camada do banco X Uso de funções específicas. Encapsulamento da sessão como objeto. Padrões de Projeto Factory, Gateway, Iterator e Active Record.
3	Implementação de código seguro com componentes do framework. Filtros e Validadores. Listas de Controle de Acesso. Autenticação. Segurança no acesso ao banco de dados.
4	Separação da aplicação em módulos. Uso de templates e subtemplates de página. Criação de formulários dinâmicos.
5	Encapsulamento de componentes de terceiros (PEAR, Smarty). Criação de novos componentes.

Mapeando para Bancos de Dados Relacionais

POR QUÊ?

De um lado:

Aplicações orientadas a objetos.

Componentes reutilizáveis.

De outro lado:

Bancos de dados relacionais.

Linguagem padronizada para comunicação com banco de dados (SQL).

Fatos

No desenvolvimento de sistemas, a Orientação a Objetos alcançou um grande nível de maturidade.

No gerenciamento de banco de dados, os bancos relacionais alcançaram um grande nível de maturidade.

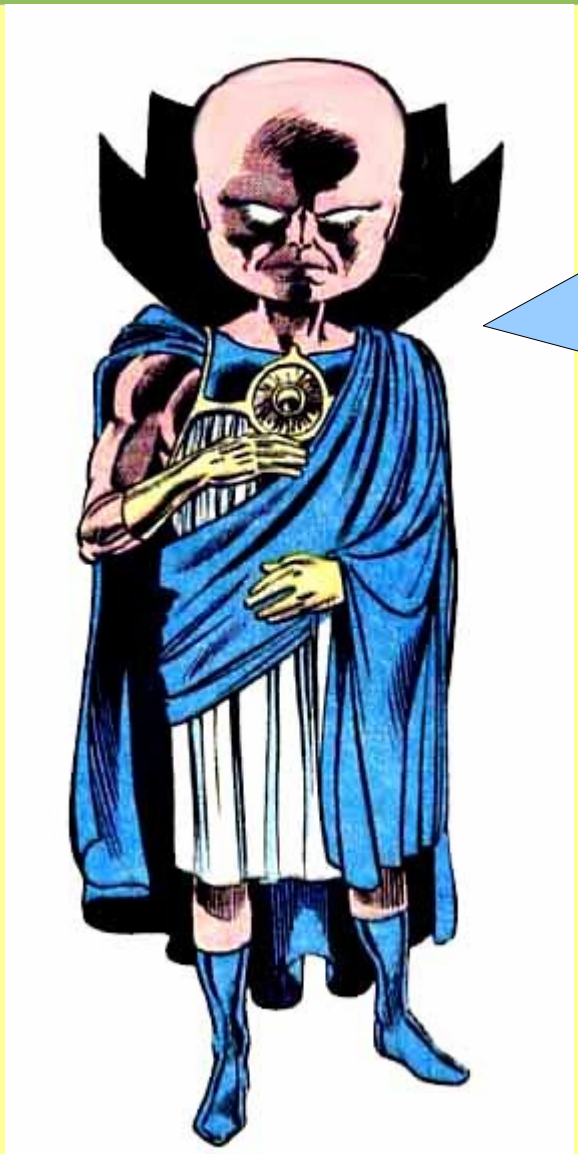
Os bancos relacionais são populares e têm uma grande base instalada. Bancos orientados a objeto ainda são promessa e ninguém vive de promessa.

O que fazer?

"Seria melhor acessar os dados usando mecanismos que se encaixem com a linguagem de desenvolvimento da aplicação" Fowler (2006)

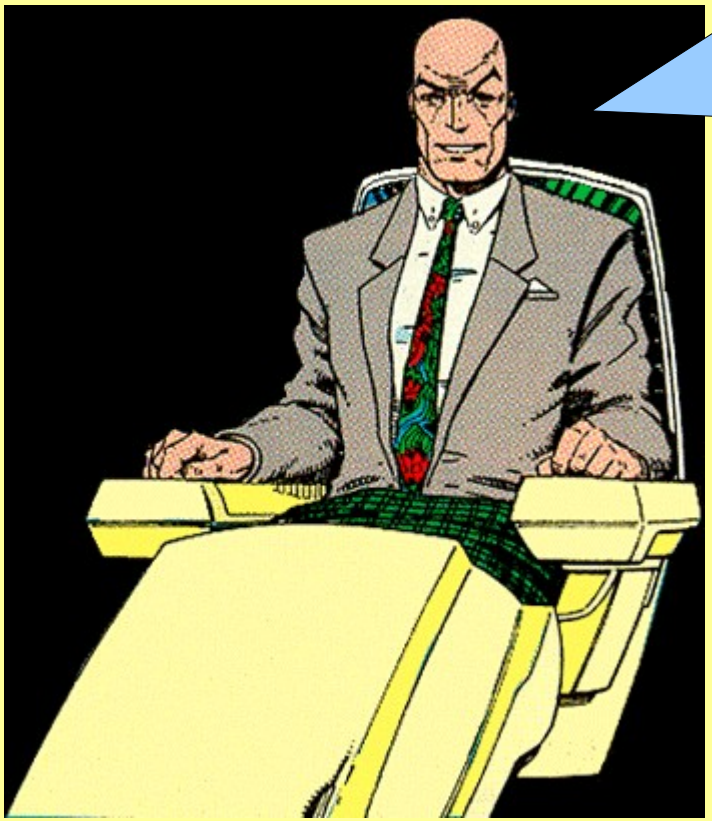
Por Tutatis! Como não pensei nisso antes?

O que fazer, Mestre Fowler?



"É sábio separar o acesso SQL da lógica de domínio e colocá-la em classes separadas."

E como fazer isso, Mestre?



Uma maneira é basear essas classes na estrutura da tabela do banco de dados de modo que você tenha uma classe por tabela.

PADRÃO DE PROJETO ***GATEWAY***

E como usarei o Gateway, Mestre?

Uma forma é ter uma instância da classe para cada linha retornada por uma consulta.

PADRÃO DE PROJETO
GATEWAY DE LINHAS DE DADOS (ROW DATA GATEWAY)

E qual a outra forma, Mestre?

A outra forma é ter uma instância da classe para cada tabela, ou melhor, conjunto de registros.

PADRÃO DE PROJETO
GATEWAY DE TABELAS DE DADOS (TABLE DATA GATEWAY)

O Gateway é o único caminho, Mestre?

Não, outra maneira é usar o padrão de projetos **ACTIVE RECORD.**

ESSE PADRÃO POR SUA VEZ USA O ***DOMAIN MODEL***, *EM SUA VERSÃO SIMPLES,*

E eu aprenderei a usar esses padrões?

Claro, assim que aprender o padrão para se conectar com o banco de dados!

Como espera mapear os dados se nem acesso a eles você tem?

Padrões, PEAR e Frameworks PHP

CHEGA DE PERGUNTAS, XARÁ! TÁ NA HORA DAS RESPOSTAS!

COMO ME CONECTO AO BANCO DE DADOS?

USANDO O PADRÃO DE PROJETOS **ADAPTER,** VULGARMENTE CONHECIDO COMO **WRAPPER.**

O OBJETIVO DESSE PADRÃO É CONVERTER A INTERFACE DE UMA CLASSE EM OUTRA, ESPERADA PELOS CLIENTES.

ADAPTER PERMITE QUE CLASSES COM INTERFACES INCOMPATÍVEIS TRABALHEM EM CONJUNTO – O QUE, DE OUTRA FORMA, SERIA IMPOSSÍVEL**.**

ISSO SE ENCAIXA PERFEITAMENTE NA CONEXÃO DE BANCOS, POIS CADA UM TEM UMA INTERFACE DIFERENTE.

Componente da vez: Zend_Db_Adapter

Provê classes adaptadoras para as seguintes marcas de bancos de dados:

- Firebird
- IBM DB2
- Interbase
- MySQL
- Microsoft SQL Server
- Oracle
- PostgreSQL
- SQLite

Você pode usar o construtor!

```
require_once('Zend/Db/Adapter/Pdo/Mysql.php');
   $db = new Zend_Db_Adapter_Pdo_Mysql(array(
        'host'      => '127.0.0.1',
        'username'  => 'usuario',
        'password'  => '12345',
        'dbname'    => 'escola'
));
```

Você pode usar o padrão Factory!

```
require_once('Zend/Db.php');
/* Carrega automaticamente a classe
Zend_Db_Adapter_Pdo_Mysql
 *  e cria uma instância dela.
*/
$db = Zend_Db::factory('Pdo_Mysql', array(
    'host'     => '127.0.0.1',
    'username' => 'usuario',
    'password' => '12345',
    'dbname' => 'escola'
));
```

Você pode usar sua classe adaptadora!

```
require_once('Zend/Db.php');
/* Carrega automaticamente a classe
MyProject_Db_Adapter_Pdo_Mysql e cria uma
instância dela. */
$db = Zend_Db::factory('Pdo_Mysql', array(
    'host' => '127.0.0.1',
    'username' => 'usuario',
    'password' => '12345',
    'dbname' => 'escola',
'adapterNamespace' => 'MyProject_Db_Adapter'
));
```

Você pode usar Zend_Config com Zend_Db!

```
require_once('Zend/Db.php');
require_once('Zend/Config.php');
$config = new Zend_Config(
        array(
           'database' => array(
              'adapter'  => 'Mysqli',
              'params'   => array(
                 'dbname'   => 'escola',
                 'username' => 'usuario',
                 'password' => '12345',
                 'host'     => '127.0.0.1'
              )
           )
        )
);
$db = Zend_Db::factory($config->database);
```

Você pode usar Zend_Config com Zend_Db!

Ainda podemos carregar a configuração do banco de um arquivo externo, usando as classes filhas de Zend_Config:

Zend_Config_Ini e Zend_Config_Xml

Zend_Config_Ini, com sua simplicidade:

Arquivo **config.ini**

```
[database]
db.adapter = PDO_MYSQL
db.config.username = usuario
db.config.password = 12345
db.config.host = 127.0.0.1
db.config.dbname = escola
```

Zend_Config_Ini, com sua simplicidade:

```
require_once('Zend/Db.php');
require_once('Zend/Config/Ini.php');
$config = new
Zend_Config_Ini('config.ini','database');
$db = Zend_Db::factory(
      $config->db->adapter,
      $config->db->config->toArray()
      );
```

Zend_Config_Xml, com sua versatilidade:

Arquivo **config.xml**

```
<?xml version="1.0"?>
<configdata>
    <production>
        <webhost>www.escola.com</webhost>
            <db>
            <adapter>pdo_mysql</adapter>
            <config>
                <host>db.escola.com</host>
                <username>pro_user</username>
                <password>pro_secret</password>
                <dbname>escola</dbname>
            </config>
            </db>
    </production>
    <development extends="production">
         <db>
            <config>
                <host>127.0.0.1</host>
                <username>usuario</username>
                <password>12345</password>
            </config>
            </db>
    </development>
</configdata>
```

Zend_Config_Xml, com sua versatilidade:

```
require_once('Zend/Db.php');
require_once('Zend/Config/Xml.php');
$config = new
Zend_Config_Xml('config.xml','development');

$db = Zend_Db::factory(
      $config->db->adapter,
      $config->db->config->toArray()
      );
```

A CONEXÃO COM O BANCO É IMEDIATA?

Você quer saber se na hora em que criar a instância de Zend_Db será feita a conexão e um atributo protegido em algum lugar receberá um tipo resource?

A resposta é **NÃO.** A conexão é feita sob demanda, a partir da primeira consulta ao banco.

Para forçar a conexão, use o método *getConnection()*

Obtendo registros na forma de matrizes com o método fetchAll()

```
$sql = 'SELECT * from alunos WHERE id > ?';
$registros = $db->fetchAll($sql,array(2));
```

Obtendo registros na forma de matrizes com o método fetchAll()

O resultado de *fetchAll()* por padrão é uma matriz de duas dimensões, onde a primeira equivale aos registros e a segunda aos campos. Na segunda, as chaves dos elementos são os nomes dos campos, se nada for dito em contrário.

Esse último detalhe pode ser configurado com o método *setFetchMode()*, usando as constantes Zend_Db::FETCH...

Obtendo somente a primeira coluna de uma consulta

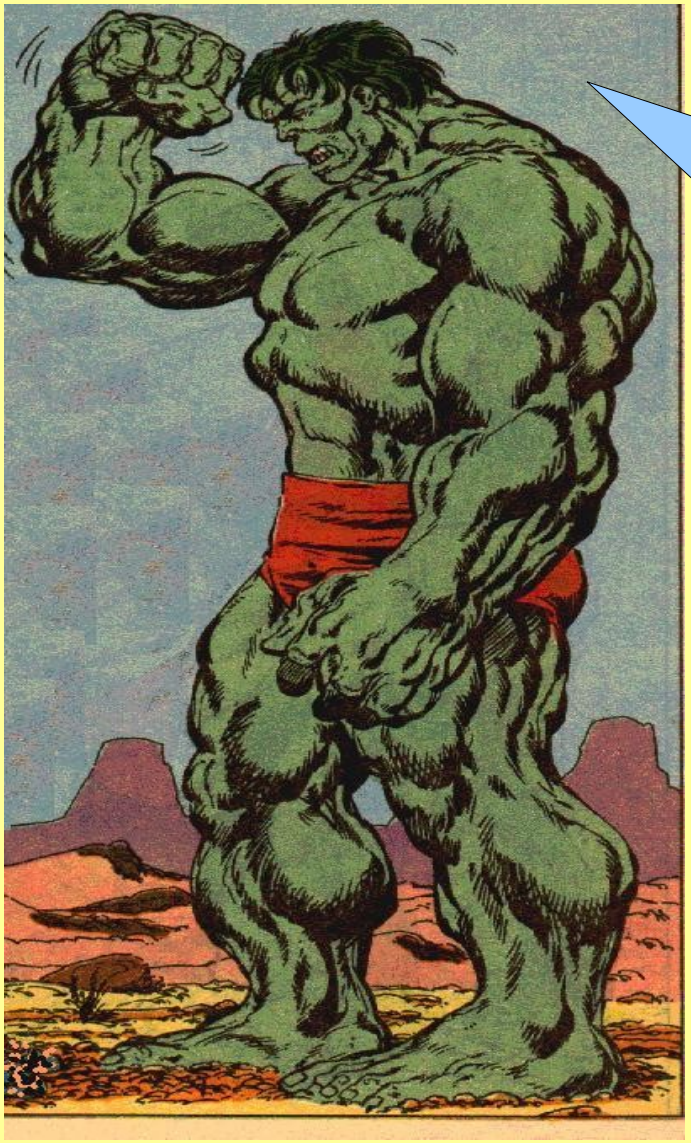
Se quiser somente a primeira coluna de uma consulta (o primeiro campo especificado na consulta, ou o primeiro definido na tabela), use o método *fetchCol()*. Ele retornará uma matriz uma dimensão.

Para cada elemento, a chave é o número ordinal do registro na consulta (não na tabela) e o valor é conteúdo do campo.

Obtendo somente a primeira linha de uma consulta

Se quiser somente a primeira linha de uma consulta (o primeiro registro retornado pela mesma), use o método *fetchRow()*. Ele retornará uma matriz com uma dimensão.

Para cada elemento, a chave é o nome (ou o ordinal) do campo e o valor é o conteúdo do mesmo.



CHEGA DE CONSULTAS! EU QUERO VER AS OUTRAS OPERAÇÕES!

Inclusão de registros

```
$data = array(
        "nome" => 'Peninha'
);
$db->insert('alunos',$data);
```

Pergunta:

E se ao invés de colocar um valor diretamente, eu queira atribuir ao campo o valor de retorno de uma função do banco?

SEM PROBLEMAS!

Use uma instância de Zend_Db_Expr e passe a função como parâmetro para o construtor.

```
<?php
$data = array(
'created_on' => new Zend_Db_Expr('CURDATE()'),
'bug_description' => 'Something wrong',
'bug_status' => 'NEW'
);
$db->insert('bugs', $data);
```

Pergunta:

E se eu quiser saber qual o id do registro incluído?

Use a função *getLastInsertId()*

Atualização de registros:

```
$data = array(
        "nome" => 'Gansolino'
);
$where = $db->quoteInto('nome
= ?','Peninha');
$db->update('alunos',$data,$where);
```

Remoção de registros:

```
$where = $db->quoteInto('nome
= ?','Gansolino');
$db->delete('alunos',$where);
```

O problema dos campos de texto

O conteúdo de campos do tipo texto em SQL é cerceado por apóstrofos. Como isso pode gerar uma confusão com os apóstrofos, a classe Zend_Db fornece o método *quote()*, que envolve o texto com os mesmos.

Controle de Transações

```
<?php
$db->beginTransaction();
try {
  $db->query(...);
  $db->commit();
} catch (Exception $e) {
  $db->rollBack();
  echo $e->getMessage();
}
```

Informações sobre as Tabelas

```
$lista = $db->listTables();

foreach($lista as $table)
{
   echo "<h1>Tabela $table</h1>";
   $campos = $db->describeTable($table);
   foreach ($campos as $nome => $dados)
   {
      echo "<p><b>$nome</b><br>";
      foreach ($dados as $chave => $valor)
      {
         echo "$chave = $valor<br>";
      }
      echo '</p>';
   }
}
```

Encerrando a Conexão

Normalmente não é necessário fechar uma conexão de banco de dados. PHP automaticamente elimina todos os recursos e finaliza uma requisição. Extensões de bancos de dados são desenhadas para fechar a conexão assim que a referência para o objeto de recurso seja eliminada.

Encerrando a Conexão

```
$db->closeConnection();
```

Declarações: para quê servem?

RECUPERAR TUDO? NÃO, UM REGISTRO DE CADA VEZ!

Preparando Declarações SQL com o construtor de Zend_Db_Statement

```
require_once('Zend/Db/Statement.php');
require_once('Zend/Config/Xml.php');
$config = new Zend_Config_Xml('config.xml','development');

$db = Zend_Db::factory($config->db->adapter,$config->db->config-
>toArray());

$sql = 'SELECT * from alunos WHERE id > ?';

$stmt = new Zend_Db_Statement_Pdo($db,$sql);

$stmt->execute(array(2));

while ($registro = $stmt->fetch())
{
    foreach ($registro as $campo => $conteudo)
    {
        echo "$campo = $conteudo<br>";
    }
}
```

Preparando Declarações SQL com o próprio objeto Zend_Db

```
require_once('Zend/Db/Statement.php');
require_once('Zend/Config/Xml.php');
$config = new Zend_Config_Xml('config.xml','development');

$db = Zend_Db::factory($config->db->adapter,$config->db->config-
>toArray());

$sql = 'SELECT * from alunos WHERE id > ?';

$stmt = $db->query($sql,array(2));

$stmt->execute();

while ($registro = $stmt->fetch())
{
    foreach ($registro as $campo => $conteudo)
    {
        echo "$campo = $conteudo<br>";
    }
}
```

Recupere uma única coluna

```
require_once('Zend/Db/Statement.php');
require_once('Zend/Config/Xml.php');
$config = new Zend_Config_Xml('config.xml','development');

$db = Zend_Db::factory($config->db->adapter,$config->db->config-
>toArray());

$sql = 'SELECT * from alunos WHERE id > ?';

$stmt = $db->query($sql,array(2));

$stmt->execute();

while ($coluna = $stmt->fetchColumn(1))
{
   echo "$coluna<br>";
}
```

Consultas Orientadas a Objetos: Zend_Db_Select

```
$db = Zend_Db::factory($config->db->adapter,$config->db->config->toArray());

$select = $db->select();

$select->from('alunos');
$select->where('id > ?',2);
$select->order('nome');

$stmt = $select->query();

$stmt->execute();

while ($registro = $stmt->fetch())
{
    echo '<p>';
    foreach ($registro as $campo => $conteudo)
    {
        echo "$campo = $conteudo<br>";
    }
    echo '</p>';
}
```

Padrões, PEAR e Frameworks PHP

Zend_Db_Select: Interface Fluente

```
require_once('Zend/Db.php');
require_once('Zend/Config/Xml.php');
$config = new Zend_Config_Xml('config.xml','development');

$db = Zend_Db::factory($config->db->adapter,$config->db->config->toArray()
          );

$select = $db->select();

$select->from('alunos')->where('id > ?',2)->order('nome');

$stmt = $select->query();

$stmt->execute();

while ($registro = $stmt->fetch())
{
    echo '<p>';
    foreach ($registro as $campo => $conteudo)
    {
        echo "$campo = $conteudo<br>";
    }
    echo '</p>';
}
```

Zend_Db_Select: Obtendo a expressão SQL

```
require_once('Zend/Db.php');
require_once('Zend/Config/Xml.php');
$config = new
Zend_Config_Xml('config.xml','development');

$db = Zend_Db::factory(
        $config->db->adapter,
        $config->db->config->toArray()
        );

$select = $db->select();

$select->from('alunos')->where('id > ?',2)-
>order('nome');

echo $select->__toString();
```

Tabelas como Objetos: Zend_Db_Table

É A IMPLEMENTAÇÃO DO GATEWAY DE TABELA DE DADOS!

Zend_Db_Table : Criando o Modelo

```
<?php
require_once('Zend/Db/Table.php');
class Alunos extends Zend_Db_Table
{
  protected $_name = 'alunos';
}
?>
```

Zend_Db_Table : Usando o Modelo para Recuperar Registros

```
require_once('Zend/Db.php');
require_once('Zend/Config/Xml.php');
require_once('Alunos.php');

$config = new Zend_Config_Xml('config.xml','development');

$db = Zend_Db::factory($config->db->adapter,$config->db-config-
>toArray());

Zend_Db_Table_Abstract::setDefaultAdapter($db);

$alunos = new Alunos();

$where = $alunos->getDefaultAdapter()->quoteInto('id > ?',2);
$registros = $alunos->fetchAll($where,'nome');
```

Zend_Db_Table : Usando o Modelo para Recuperar Registros

```
foreach($registros as $registro)
{
   echo '<p>';
   foreach ($registro->toArray() as $campo => $conteudo)
   {
      echo "$campo = $conteudo<br>";
   }
   echo '</p>';
}
```

Zend_Db_Table : Incluindo registros

```
require_once('Zend/Db.php');
require_once('Zend/Config/Xml.php');
require_once('Alunos.php');

$config = new Zend_Config_Xml('config.xml','development');

$db = Zend_Db::factory(
          $config->db->adapter,
          $config->db->config->toArray()
          );

Zend_Db_Table_Abstract::setDefaultAdapter($db);

$alunos = new Alunos();

$data = array(
             "nome" => 'Urtigão'
);

$alunos->insert($data);
```

Zend_Db_Table : Atualizando registros

```
require_once('Zend/Db.php');
require_once('Zend/Config/Xml.php');
require_once('Alunos.php');

$config = new Zend_Config_Xml('config.xml','development');

$db = Zend_Db::factory(
          $config->db->adapter,
          $config->db->config->toArray()
          );

Zend_Db_Table_Abstract::setDefaultAdapter($db);

$alunos = new Alunos();

$data = array(
              "nome" => 'Vovó Donalda'
);

$where = $alunos->getAdapter()->quoteInto('id = ?',14);
$alunos->update($data,$where);
```

Zend_Db_Table : Apagando registros

```
require_once('Zend/Db.php');
require_once('Zend/Config/Xml.php');
require_once('Alunos.php');

$config = new Zend_Config_Xml('config.xml','development');

$db = Zend_Db::factory(
          $config->db->adapter,
          $config->db->config->toArray()
          );

Zend_Db_Table_Abstract::setDefaultAdapter($db);

$alunos = new Alunos();

$data = array(
              "nome" => 'Vovó Donalda'
);

$where = $alunos->getAdapter()->quoteInto('id = ?',15);
$alunos->delete($where);
```

Conjuntos de Linhas como Objetos: Zend_Db_Rowset

É A IMPLEMENTAÇÃO DO GATEWAY DE LINHAS DE DADOS!

Zend_Db_Rowset a partir de Zend_Db_Table

As consultas feitas com os métodos *fetchAll()* e *find()* retornam objetos Zend_Db_Rowset.

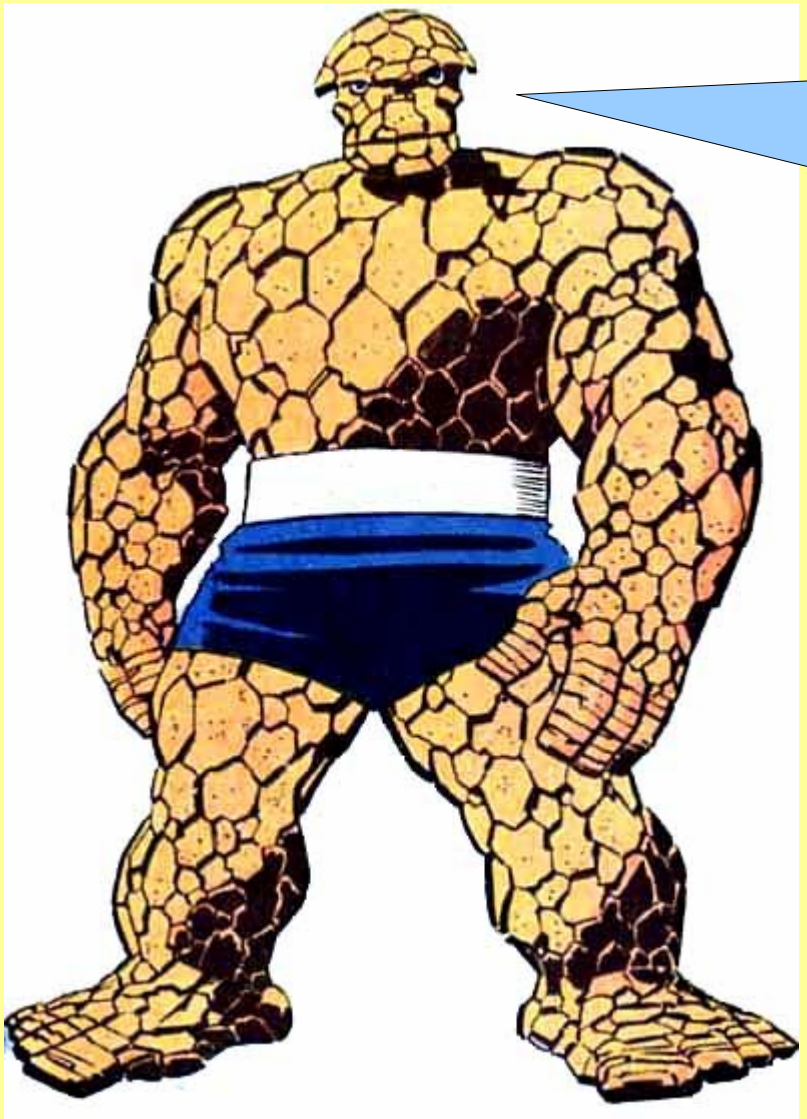
O método *find()* faz buscas diretamente pela chave primária.
Se um valor simples for passado como parâmetro, ele retorna um objeto Zend_Db_Rowset com somente um objeto Zend_Db_Row.
Se for passado uma matriz de valores, serão retornados tantos objetos Zend_Db_Row quantos os cujas chaves primárias combinarem com as da matriz.

Linhas como Objetos: Zend_Db_Row

```
$alunos = new Alunos();

$where = $alunos->getDefaultAdapter()->quoteInto('id > ?',2);

$registros = $alunos->fetchAll($where,'nome');

while ($registro = $registros->current())
{
    echo '<p>';
    foreach ($registro->toArray() as $campo => $conteudo)
    {
        echo "$campo = $conteudo<br>";
    }
    echo '</p>';
    $registros->next();
}
```

OPA, TEM UM PADRÃO **ITERATOR** AQUI!

Linhas como Objetos: Zend_Db_Row



TÁ, MAS PRA QUE SERVE AFINAL ESSE ZEND_DB_ROW?

Zend_Db_Row: Objetos Persistentes

```
require_once('Zend/Db.php');
require_once('Zend/Config/Xml.php');
require_once('Alunos.php');

$config = new Zend_Config_Xml('config.xml','development');

$db = Zend_Db::factory($config->db->adapter,$config->db->config-
>toArray());

Zend_Db_Table_Abstract::setDefaultAdapter($db);

$alunos = new Alunos();

$where = $alunos->getDefaultAdapter()->quoteInto('nome = ?','zico');
$registro = $alunos->fetchRow($where);

$registro->nome = 'zeca';
$registro->save();
```

OPA, MAS ESSE É O PADRÃO **ACTIVE RECORD**!

Zend_Db_Row:
Criar Objetos = Incluir Registros

```
require_once('Zend/Db.php');
require_once('Zend/Config/Xml.php');
require_once('Alunos.php');

$config = new Zend_Config_Xml('config.xml','development');

$db = Zend_Db::factory(
          $config->db->adapter,
          $config->db->config->toArray()
          );

Zend_Db_Table_Abstract::setDefaultAdapter($db);

$alunos = new Alunos();

$registro = $alunos->createRow();
$registro->nome = 'Chiquinho';
$registro->save();
```

Zend_Db_Row:
O Objeto Apaga o Registro (Ele Mesmo!)

```
require_once('Zend/Db.php');
require_once('Zend/Config/Xml.php');
require_once('Alunos.php');

$config = new Zend_Config_Xml('config.xml','development');

$db = Zend_Db::factory(
          $config->db->adapter,
          $config->db->config->toArray()
          );

Zend_Db_Table_Abstract::setDefaultAdapter($db);

$alunos = new Alunos();

$registro = $alunos->fetchRow("nome = 'Chiquinho'");
$registro->delete();
```

Zend_Db_Row: E se os dados mudarem?

Quando o objeto Zend_Db_Row é recuperado através de uma consulta, ele retorna com os dados do banco naquele momento. Atualizações no banco não são propagadas automaticamente para o objeto.

Para carregar o objeto com os dados atualizados, use o método *refresh()*.

Monitore as Operações do Banco

```
$profiler = $db->getProfiler();
```

Isso retorna uma instância de objeto Zend_Db_Profiler. Com essa instância, o desenvolvedor pode examinar suas consultas usando uma variedade de métodos:

- getTotalNumQueries() retorna o número total de consultas que foram executadas.
- getTotalElapsedSecs() retorna o número total de segundos transcorridos para todas as consultas executadas.
- getQueryProfiles() retorna um vetor de todos os perfis de consulta.
- getLastQueryProfile() retorna o último (mais recente) perfil de consulta, não obstante a consulta tenha ou não terminado. (se não tiver, a hora de término será nula)

Monitore as Operações do Banco

 clear() limpa quaisquer perfis de consulta passados da pilha.

O valor de retorno de getLastQueryProfile() e os elementos individuais de getQueryProfiles() são objetos Zend_Db_Profiler_Query, que provêem a habilidade de inspecionar as consultas individuais por elas mesmas:

 getQuery() retorna o texto SQL da consulta. O texto SQL de uma declaração preparada com parâmetros é o texto no momento em que a consulta for preparada, assim ela contém espaços reservados para valores de parâmetro, não os valores usados quando a declaração é executada.

 getQueryParams() retorna um vetor de valores de parâmetro usados quando executar uma consulta preparada. Isso inclui tanto parâmetros de combinação quanto argumentos para o método execute() de declaração. As chaves do vetor são
índices de parâmetro posicionais (baseados em 1) ou nomeados (string).

 getElapsedSecs() retorna o número de segundo que a consulta levou para rodar.

Referências Bibliográficas

Fowler, M. *Padrões de Projeto de Aplicações Corporativas*. Porto Alegre. Bookman, 2006.

Freeman, E. e Freeman, E. *Use a Cabeça! Padrões de Projetos*. Rio de Janeiro. AltaBooks, 2005.

Gamma, E. et alli. *Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos*. Porto Alegre. Bookman, 2000.

Martin, J. e Odell, J. J. *Análise e Projeto Orientados a Objeto*. São Paulo. Makron Books, 1995.

http://framework.zend.com