The PHP Company

# Zend\Db in ZF 2.0

As of Zend Framework Beta 3 (going on Beta 4)

# Who Am I?

- **Ralph Schindler (ralphschindler)**
  - Software Engineer on the Zend Framework team
    - At Zend for almost 4 years
    - Before that TippingPoint/3Com
  - Programming PHP for 13+ years
  - Live in New Orleans, LA.
    - Lived in Austin, Tx for 5 years

# ZF 2.0

- **Next generation of Zend Framework**
- **Embrace 5.3 (and 5.4 in some places)**
- **Embrace multiple programming paradigms**
  - AOP
  - Event driven programming
- **More SOLID**
  - http://en.wikipedia.org/wiki/SOLID_(object-oriented_design)
  - More interfaces, more possibility for extension
  - Practice dependency injection
- **More Agile and open!**
  - No more CLA
  - Code on github.com

# Zend\Db first pass...

- Converted to namespaces
- Converted to new Exception standards
- Then, left alone while other things were developed on
- Rewrite initial release in place of old Zend\Db during beta3
  - March 2012

# Zend\Db Requirements In Short

- A more clear and concise API (single responsibility)
- Open for extension (open/close principle)
- Practice dependency injection
- Favor composition over inheritance
- Do not solve Model Domain problems

# What does that mean in practice

- **No statics anywhere**
  - 1 Exception:
    - Zend\Db\TableGateway\StaticAdapterTableGateway
- **Lots of (simple) interfaces**
  - No setters
  - No dependencies
- **Zend\Db\Adapter can be treated like a collaborator (dependency) in other components**
  - Zend\Db\TableGateway
  - Zend\Db\Sql
  - Zend\Db\Metadata
  - … Zend\Db\ActiveRecord?

# What does that mean in practice, cont.

- **Very few Abstract classes ...**
  - Only in places where it is clear there is some shared implementation details
- **No final keyword on classes**
- **No privates inside classes**
- **All API are database centric, not model centric**
  - columns
  - rows
  - tables
  - schemas

# Walking through Zend\Db

- **Under the Zend\Db Namespace:**
  - Adapter
  - ResultSet
  - Sql
  - TableGateway
  - RowGateway
  - Metadata

# Zend\Db\Adapter\Adapter & Zend\Db\Adapter\ResultSet

# Zend\Db\Adapter

- Zend\Db\Adapter is a namespace
- Zend\Db\Adapter\Adapter is the “kernel” of Zend\Db
- Zend\Db\Adapter* are all the interfaces and implementation details of the Adapter

```php
namespace Zend\Db\Adapter;

use Zend\Db\ResultSet;

class Adapter
{
    const QUERY_MODE_EXECUTE = 'execute';
    const QUERY_MODE_PREPARE = 'prepare';
    const PREPARE_TYPE_POSITIONAL = 'positional';
    const PREPARE_TYPE_NAMED = 'named';

    /* beta4 */
    const HELPER_FORMAT_PARAMETER_NAME = 'formatParameterName';
    const HELPER_QUOTE_IDENTIFIER = 'quoteIdentifier';
    const HELPER_QUOTE_VALUE = 'quoteValue';
    const HELPER_IDENTIFIER_SEPARATOR = 'identifierSeparator';

    public function __construct($driver, Platform\PlatformInterface $platform = null, ResultSet\ResultSet $queryResultPrototype = null);
    public function getDriver(); // @return Driver\DriverInterface
    public function setQueryMode($queryMode);
    public function getQueryMode();
    public function getPlatform(); // @return Platform\PlatformInterface
    public function getDefaultSchema();
    public function query($sql, $parametersOrQueryMode = self::QUERY_MODE_PREPARE); // @return Driver\ResultInterface, ResultSet
    public function createStatement($initialSql = null, $initialParameters = null); // @return Driver\StatementInterface
    public function getHelpers(/* helperlist */); /* in beta4, @return functions and strings */
    public function __get($name);
}
```

https://github.com/zendframework/zf2/blob/master/library/Zend/Db/Adapter/Adapter.php

# Two Primary Parts of Zend\Db\Adapter

- Driver
- Platform

# Driver Responsibilities

- Connecting to proper PHP extension
- Reporting the capabilities of the extension
- Coordinating 3 primary areas of interaction from driver object:
  - Connections
  - Statements
  - Results

# Driver continued

- **Connection interface**
  - wrapper for connection resource and/or functions

```php
namespace Zend\Db\Adapter\Driver;

interface ConnectionInterface
{
    public function getDefaultCatalog();
    public function getDefaultSchema();
    public function getResource();
    public function connect();
    public function isConnected();
    public function disconnect();
    public function beginTransaction();
    public function commit();
    public function rollback();
    public function execute($sql); // return result set
    public function getLastGeneratedId();
}
```

# Driver continued

- **Statement interface**
  - wrapper for statement resource and/or functions

```php
namespace Zend\Db\Adapter\Driver;

use Zend\Db\Adapter\ParameterContainerInterface;

interface StatementInterface
{
    public function getResource();
    public function setSql($sql);
    public function getSql();
    public function setParameterContainer(ParameterContainerInterface $parameterContainer);
    public function getParameterContainer();
    public function prepare($sql = null);
    public function isPrepared();
    public function execute($parameters = null);
}
```

# Driver continued

- **Result interface**
  - wrapper for result resource and/or function

```php
namespace Zend\Db\Adapter\Driver;

interface ResultInterface extends \Countable, \Iterator
{
    public function isQueryResult();
    public function getAffectedRows();
    public function getResource();
}
```

# Platform

- **Responsible for:**
  - Knowing how to quote in a vendor RDBMS specific way
  - Knowing the "name" of the database in a code-neutral way

```php
namespace Zend\Db\Adapter\Platform;

interface PlatformInterface
{
    public function getName();
    public function getQuoteIdentifierSymbol();
    public function quoteIdentifier($identifier);
    public function getQuoteValueSymbol();
    public function quoteValue($value);
    public function getIdentifierSeparator();
    public function quoteIdentifierInFragment($identifier, array $additionalSafeWords = array());
}
```

# Creating an Adapter

```php
use Zend\Db\Adapter\Adapter as DbAdapter,
    Zend\Db\Adapter\Driver,
    Zend\Db\Adapter\Platform;

// explicit, injecting all dependencies
$dbAdapter = new DbAdapter(
    new Driver\Pdo\Pdo(
        new Driver\Pdo\Connection(array('pdoDriver' => 'sqlite', 'database' => 'path/to/db')),
        new Driver\Pdo\Statement,
        new Driver\Pdo\Result
    ),
    new Platform\Sqlite
);

// OR, better:
$dbAdapter = new DbAdapter(array(
    'driver' => 'Pdo-Sqlite',
    'database' => __DIR__ . '/tmp/sqlite.db',
));
```

# Connection abstraction

- **first parameter of Adapter takes an array**
- **Keys:**
  - driver, database, hostname, port, username, password, (pdodriver)
  - any key you might find in php.net documentation

```php
use Zend\Db\Adapter\Adapter as DbAdapter

// as specificed in the SQL documentation
$connectParams = array(
    'driver' => 'Sqlsrv',
    'hostname' => 'RALPH-PC\SQLEXPRESS',
    'UID' => 'developer',
    'PWD' => 'developer',
    'Database' => 'zend_db_example'
);

// same as:
$connectParams = array(
    'driver' => 'Sqlsrv',
    'hostname' => 'RALPH-PC\SQLEXPRESS',
    'username' => 'developer',
    'password' => 'developer',
    'database' => 'zend_db_example'
);

// OR, better:
$dbAdapter = new DbAdapter($connectParams);
```

# Adapter Convenience

- Convenience API

```php
use Zend\Db\Adapter\Adapter as DbAdapter;

list($qi, $qv, $is, $fp) = $dbAdapter->getHelpers(
    DbAdapter::HELPER_QUOTE_IDENTIFIER,
    DbAdapter::HELPER_QUOTE_VALUE,
    DbAdapter::HELPER_IDENTIFIER_SEPARATOR,
    DbAdapter::HELPER_FORMAT_PARAMETER,
);


$sql = 'INSERT INTO '
    . $qi('artist')
    . ' (' . $qi('name') . ', ' . $qi('history') . ') VALUES ('
    . $fp('name') . ', ' . $fp('history') . ')';

// mysqli: INSERT INTO `artist` (`name`, `history`) VALUE (:name, :history)

// OR

$sql = 'SELECT ' . $qi('mytable') . $is . '* FROM '
    . $qi('mytable') . ' WHERE ' . $qi('id') . ' = ' . $qv($value);

// pdo sqlite: SELECT "mytable".* FROM "mytable" WHERE "id" = 'myvalue'
```

# Zend\Db\ResultSet

- A collection of Rows
- Model iteration in an Adapter/Driver neutral way
- Present Rows as objects or arrays
  - support positional and name based indexes
- Should use PHP's Iterator & Countable interface

```php
namespace Zend\Db\ResultSet;

use ArrayIterator,
    ArrayObject,
    Countable,
    Iterator,
    IteratorAggregate;

class ResultSet implements Countable, Iterator
{
    const TYPE_OBJECT = 'object';
    const TYPE_ARRAY  = 'array';

    /**
```

# Zend\Db\ResultSet\RowObjectInterface

- **When using objects:**
  - This interface promotes new object creation though the prototype pattern
  - Only requirements are:
    - Countable, ArrayAccess & populate() method

# Zend\Db\Sql

# Zend\Db\Sql

- **Abstraction layer for creating DML (Data Manipulation Language in SQL)**
  - Select, Insert, Update, Delete
- **2 Modes**
  - Statement preparation
  - SQL string generation

# Zend\Db\Sql cont.

- Serializable, clone-able objects
- Preform Adapter specific parameterization
- Abstraction for DDL (Data Definition Language in SQL)
  - (not completed yet, planned for beta5)

# Zend\Db\Sql\Select

- Object for building queries
- columns, joins, where, offset, limit
  - Offset and limit support in beta4
- Ability to track identifiers and values in vendor and driver neutral way
- Fluent API (LINQ style)

```php
use Zend\Db\Sql;

$select2 = new Select;
$select2->from('foo')->columns(array('bar', 'baz'));
$sql2 = 'SELECT "bar", "baz" FROM "foo"';
```

# Zend\Db\Sql\Expression

- **Query/Statement agnostic value and identifier placeholders**
  - Uses “?” as common placeholder
- **Object to use for “pass through” of SQL fragments**
  - Much like Zend_Db_Expr in ZF1

```php
// take from unit tests:

// columns where value is Expression
$select6 = new Select;
$select6->from('foo')->columns(array(new Expression('COUNT(some_column) AS bar')));
$sql6 = 'SELECT COUNT(some_column) AS bar FROM "foo"';

// using replacements
$select7 = new Select;
$select7->from('foo')->columns(
    array(
        new Expression(
            '(COUNT(?) + ?) AS ?',
            array('some_column', 5, 'bar'),
            array(Expression::TYPE_IDENTIFIER, Expression::TYPE_VALUE, Expression::TYPE_IDENTIFIER)
        )
    )
);
$sql7 = 'SELECT (COUNT("some_column") + ?) AS "bar" FROM "foo"';
```

# Zend\Db\Sql\Where & Predicates

- **Zend\Db\Sql\Where models SQL WHERE statements**
  - Is the first node of a tree
- **Zend\Db\Sql\Predicate are SQL predicates**
  - http://en.wikipedia.org/wiki/Where_(SQL)
  - Truth values
  - Combined by parenthesizes, AND and OR
  - Predicate sets can be nested:
    - by API: $predicateSet->andPredicate($predicate);
    - or via Fluent:
      - $where->NEST->like('name', 'Ralph%')->UNNEST;
    - example to follow in later slides

# Supported Predicates

- Between
- Expression
  - SQL Fragment / Specialized functions, etc
- In
- IsNull
- Like
- Operator (>, <, =, <>, >=, <=)

  - https://github.com/zendframework/zf2/tree/master/library/Zend/Db/Sql/Predicate

# Zend\Db\TableGateway
# Zend\Db\RowGateway

# Zend\Db\TableGateway

- Table Data Gateway Pattern
- http://martinfowler.com/eaaCatalog/tableDataGateway.html
- “An object that acts as a Gateway to a database table. One instance handles all the rows in the table.”

# TableGateway Interface

```php
namespace Zend\Db\TableGateway;

interface TableGatewayInterface
{
    public function getTable();
    public function select($where = null);
    public function insert($set);
    public function update($set, $where = null);
    public function delete($where);
}
```

# Zend\Db\RowGateway

- Row Data Gateway
- http://martinfowler.com/eaaCatalog/rowDataGateway.html
- “An object that acts as a Gateway to a single record in a data source. There is one instance per row.”

# RowGateway Interface

```php
namespace Zend\Db\RowGateway;

interface RowGatewayInterface
{
    public function save();
    public function delete();
}
```

# TableRowGateway

- A component that will couple Table and Row Gateway implementations
- For beta4

# Zend\Db\Metadata

# Zend\Db\Metadata

- Component capable of interrogating a database for schema information
- Describe schema to consumers in a vendor neutral way
- Cacheable and serializable

# Metadata Interface

```php
namespace Zend\Db\Metadata;

interface MetadataInterface
{
    public function getSchemas();

    public function getTableNames($schema = null, $database = null);
    public function getTables($schema = null, $database = null);
    public function getTable($tableName, $schema = null, $database = null);

    public function getViewNames($schema = null, $database = null);
    public function getViews($schema = null, $database = null);
    public function getView($viewName, $schema = null, $database = null);

    public function getColumnNames($table, $schema = null, $database = null);
    public function getColumns($table, $schema = null, $database = null);
    public function getColumn($columnName, $table, $schema = null, $database = null);

    public function getConstraints($table, $schema = null, $database = null);
    public function getConstraint($constraintName, $table, $schema = null, $database = null);
    public function getConstraintKeys($constraint, $table, $schema = null, $database = null);

    public function getTriggerNames($schema = null, $database = null);
    public function getTriggers($schema = null, $database = null);
    public function getTrigger($triggerName, $schema = null, $database = null);

}
```

# Select, TableGateway, RowGateway Examples

# Examples

- **Working example repository:**
  - https://github.com/ralphschindler/Zend_Db-Examples/
  - Follow instructions there to setup working PDO sqlite file
    - essentially just: *php setup/up.php*

# Example of building a raw statement, using driver & platform to create a portable query...

```php
/* @var $adapter Zend\Db\Adapter */
$adapter = include ((file_exists('bootstrap.php')) ? 'bootstrap.php' : 'bootstrap.dist.php');
refresh_data($adapter);

// create completely portable SQL by hand
$sql = 'SELECT * FROM '
    . $adapter->platform->quoteIdentifier('artist')
    . ' WHERE id = ' . $adapter->driver->formatParameterName('id');

/* @var $statement Zend\Db\Adapter\DriverStatementInterface */
$statement = $adapter->query($sql);
$parameters = array('id' => 2);

/* @var $results Zend\Db\ResultSet\ResultSet */
$results = $statement->execute($parameters);

$row = $results->current();
$name = $row['name'];
assert_example_works($name == 'Bar Artist');
```

https://github.com/ralphschindler/Zend_Db-Examples/blob/master/example-01.php

```php
$adapter = include ((file_exists('bootstrap.php')) ? 'bootstrap.php' : 'bootstrap.dist.php');
refresh_data($adapter);

$artistTable = new Zend\Db\TableGateway\TableGateway('artist', $adapter);
$rowset = $artistTable->select(array('id' => 2));
$row = $rowset->current();

$name = $row['name'];
$name2 = $row->name;
assert_example_works($name == 'Bar Artist' && $name2 == 'Bar Artist');
```

https://github.com/ralphschindler/Zend_Db-Examples/blob/master/example-06.php

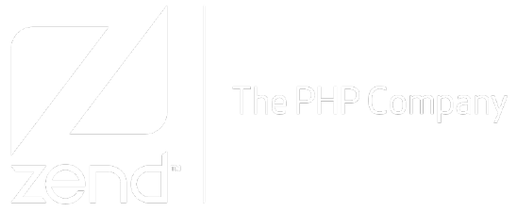
```php
$adapter = include ((file_exists('bootstrap.php')) ? 'bootstrap.php' : 'bootstrap.dist.php');
refresh_data($adapter);

$metadata = new Zend\Db\Metadata\Metadata($adapter);
// get the table names
$tableNames = $metadata->getTableNames();


foreach ($tableNames as $tableName) {
    echo 'In Table ' . $tableName . PHP_EOL;

    $table = $metadata->getTable($tableName);


    echo '    With columns: ' . PHP_EOL;
    foreach ($table->getColumns() as $column) {
        echo '        ' . $column->getName()
            . ' -> ' . $column->getDataType()
            . PHP_EOL;
    }

    echo PHP_EOL;
    echo '    With constraints: ' . PHP_EOL;
    foreach ($metadata->getConstraints($tableName) as $constraint) {
        echo '        ' . $constraint->getName()
            . ' -> ' . $constraint->getType()
            . PHP_EOL;
        foreach ($constraint->getKeys() as $key) {
            echo '            column: ' . $constraint->getTableName() . '.'
                . $key->getColumnName();
            if ($constraint->isForeignKey()) {
                echo ' => ' . $key->getReferencedTableName()
                    . '.' . $key->getReferencedColumnName();
            }
            echo PHP_EOL;
        }

    }

    echo '----' . PHP_EOL;
}
```

https://github.com/ralphschindler/Zend_Db-Examples/blob/master/example-10.php

```
(master) ~/Projects/Zend_Db-Examples$ php example-10.php
In Table album
    With columns:
        id -> INTEGER
        artist_id -> int(11)
        title -> varchar(255)
        release_date -> date

    With constraints:
        PRIMARY -> PRIMARY KEY
            column: album.id
        fk_album_1 -> FOREIGN KEY
            column: album.artist_id => artist.id
----
In Table artist
    With columns:
        id -> INTEGER
        name -> varchar(255)
        history -> text

    With constraints:
        PRIMARY -> PRIMARY KEY
            column: artist.id
----
In Table genre
    With columns:
        id -> INTEGER
        parent_id -> int(11)
        name -> varchar(255)

    With constraints:
        PRIMARY -> PRIMARY KEY
            column: genre.id
        sqlite_autoindex_genre_1 -> UNIQUE KEY
            column: genre.name
        fk_genre_1 -> FOREIGN KEY
            column: genre.parent_id => genre.id
----
In Table artist_genre
    With columns:
        artist_id -> int(11)
        genre_id -> int(11)
        added_on -> date
```

```php
$adapter = include ((file_exists('bootstrap.php')) ? 'bootstrap.php' : 'bootstrap.dist.php');
refresh_data($adapter);

$artistTable = new Zend\Db\TableGateway\TableGateway('artist', $adapter);

// All select()
//
// $artistTable->setSelectResultPrototype(
//     new Zend\Db\ResultSet\ResultSet(new Zend\Db\RowGateway\RowGateway($artistTable, 'id'))
// );


// find and update
$rowset = $artistTable->select(array('id' => 2));

// make sure all rows come back and RowGateway
$rowset->setRowObjectPrototype(new Zend\Db\RowGateway\RowGateway($artistTable, 'id'));

$row = $rowset->current();

$row['name'] = 'New Artist'; // array notation
$affected = $row->save();

// check
$rowset = $artistTable->select(array('id' => 2));
$row = $rowset->current();

assert_example_works($row->name == 'New Artist'); // object notation
```

https://github.com/ralphschindler/Zend_Db-Examples/blob/master/example-12.php




```php
/** @var $adapter Zend\Db\Adapter\Adapter */
$adapter = include ((file_exists('bootstrap.php')) ? 'bootstrap.php' : 'bootstrap.dist.php');
refresh_data($adapter);

$where = new Zend\Db\Sql\Where();
$where->equalTo('id', 1)->OR->equalTo('id', 2);
$where->OR
    ->NEST->like('name', 'Ralph%')->OR->greaterThanOrEqualTo('age', 30)->AND->lessThanOrEqualTo('age', 50)->UNNEST
    ->literal('foo = ?', 'bar');

$target =<<<EOS
SELECT "foo".* FROM "foo" WHERE "id" = '1' OR "id" = '2' OR ("name" LIKE 'Ralph%' OR "age" >= '30' AND "age" <= '50') AND foo = 'bar'
EOS;

$select = new Zend\Db\Sql\Select('foo');
$select->where($where);

assert_example_works($target == $select->getSqlString());
```

https://github.com/ralphschindler/Zend_Db-Examples/blob/master/example-13.php

```php
$adapter = include ((file_exists('bootstrap.php')) ? 'bootstrap.php' : 'bootstrap.dist.php');
refresh_data($adapter);

use Zend\Db\TableGateway\TableGateway,
    Zend\Db\Sql\Select;

$artistTable = new TableGateway('artist', $adapter);
$rowset = $artistTable->select(function (Select $select) {
    $select->where->like('name', 'Bar%');
});
$row = $rowset->current();

$name = $row['name'];
$name2 = $row->name;
assert_example_works($name == 'Bar Artist' && $name2 == 'Bar Artist');
```

https://github.com/ralphschindler/Zend_Db-Examples/blob/master/example-14.php

# Future

- **beta4**
  - Table & Row Gateway base implementation
    - Table interface where select() returns Row Gateway objects
  - Zend\Db\Sql enhancements, better organization, Expression object
  - Limit and Offset (Fetch) support for various vendor implementations
  - Fix Zend\Db consuming components
- **beta5**
  - More Driver support
    - Db2, Oracle, Postgres
  - Metadata integration
  - ActiveRecord?

# Thanks!

http://twitter.com/ralphschindler

http://framework.zend.com/zf2

http://github.com/zendframework/

http://github.com/ralphschindler