Q&A for Zend Framework Database Access

Questions about Zend_Db component

Q: Where can I find the slides to review the whole presentation after we end here?

A: The recording of this webinar, and also the slides and Q&A document will be posted at http://www.zend.com/webinar.

Database connections and adapters

Q: Adapters: why using PDO MSSQL instead of PDO ODBC, which I thought was the preferred PDO adapter for MS SQL server?

A: When the Zend Framework project began, it was assumed that PDO MSSQL was the best way to access Microsoft SQL Server. Recently, it has become clear that PDO MSSQL is not being maintained, and it would be better to use a Zend_Db adapter class for ODBC.

There is a prototype for an ODBC adapter under development currently (check out ZF from subversion and look in the incubator). Hopefully this will be ready for ZF 1.1.

Q: Exactly how reliable is the LIMIT function when used with RDBMS's that don't support it directly?

A: MySQL, PostgreSQL, and SQLite support the LIMIT clause, although officially this syntax is not part of the ANSI/ISO SQL specification.

Other RDBMS brands do not support LIMIT, so in the case of Oracle and IBM DB2, Zend_Db does some query rewriting, to emulate LIMIT using some proprietary features of those respective databases. This works fine.

In the case of Microsoft SQL Server, LIMIT cannot be emulated with their “TOP N” syntax, and there is no solution to make it work in the general case.

Queries

Q: What's the syntax for passing multiple parameters to $db->prepare() ?

A: When you prepare a SQL query, the query string may contain one or more parameter placeholders. You leave the values unspecified at prepare time. Then when you’re ready to execute the query, pass parameter values in the optional array argument of the execute() method. You must supply the same number of values in this array as the number of parameter placeholders in the SQL statement you prepared. Example:

```
$stmt = $db->prepare(‘SELECT * FROM mytable WHERE a = ? and b = ?’);

$stmt->execute( array(10, ‘string’) );
```

Q: If there are errors upon execution, is there enough info known as to why it couldn't be deleted? As in foreign key violations, etc.?

A: If your SQL statement violates database constraints, this is reported in an exception. Currently you need to examine the text of the exception message to know the exact cause of the error. The exception message is produced by the underlying database, and there is not much consistency between database brands. There has been some talk of a future enhancement to Zend_Db exceptions to include the standard SQL error code, but at this time it is not provided.

Q: Not seeing it on the documentation... is there a way to do outer joins?

A: Certainly. If you use the query methods of the Adapter class, you can form any SQL:

```
$stmt = $db->query('SELECT * FROM table1 LEFT OUTER JOIN table2 ON …');
```

If you use the Zend_Db_Select query builder, you can use the joinLeft() or joinRight methods:

```
$select = $db->select()->from('table1')->joinLeft('table2', '…condition…');
```

The Table Data Gateway does not support join queries. It is an object-oriented interface to a single table at a time.

Q: Using query builder am I sure that query will be accepted? That is, SQL syntax is valid for MySQL, Oracle , DB2 ... ?

A: The query builder (Zend_Db_Select) produces standard SQL where possible. The cases where this fails to work for a given RDBMS brand are few.

For example, SQLite does not support "RIGHT OUTER JOIN", DB2 does not support "CROSS JOIN", PostgreSQL does not support expressions in "GROUP BY" clause, etc.

LIMIT is not standard SQL syntax, but it's supported for each respective RDBMS brand if possible (though it is not possible in Microsoft SQL Server).

Table Data Gateway

Q: Can you define columns in the table as non returnable by fetchAll(), ie. I don't want to return passwords from the user table?

A: No, currently the Table Data Gateway functions always fetch all columns.

There is a feature currently under development to make these queries much more flexible, by allowing you to pass a Zend_Db_Select object to fetchAll(), so you can specify optional SQL clauses. This is in the Zend Framework incubator at this time, and should be ready for ZF 1.1.

By the way, I would offer caution about storing passwords in plaintext. It's better for the sake of security to store the password as a digest, for example by using MD5.

Q: Using the Table Data Gateway, how are complex queries done? (JOINS, subqueries, etc)

A: Table Data Gateway is necessarily an object-oriented interface to a single table at a time. It does not support JOIN queries. If you need to run queries with complex conditions, you can add a custom method to a given concrete Table class in your project.

The work that is currently in development to support a Zend_Db_Select object for Table queries will enable new possibilities for making complex queries, but as far as I know it should still be true that querying a given Table class is limited to one database table – JOINs are not supported.

Q: How to use Table Data Gateway to “UPDATE bugs SET status=CLOSED WHERE owner='Bob'”?

A: This is hard to do with the Row class, since the Row’s save() method applies only to that instance of a Row. But the Table class has an update() method. It’s similar to the update() method of the Adapter object, but of course you don’t need to name the table. Example:

```
$table = new TableClass();

$table->update( array(‘status’ => ‘CLOSED’), “owner = ‘Bob’” );
```

Q: Can I customize Zend_Db_Table_Abstract to do a custom operation on every table?

A: Yes, you can create your own abstract Table class, extending Zend_Db_Table_Abstract. Then your concrete Table classes can extend your customized abstract class. Example:

```
class My_Table_Abstract extends Zend_Db_Table_Abstract
{
    // … customizations …
}

class SomeTable extends My_Table_Abstract
{
    protected $_name = ‘dbtable’;
}
```

Q: Zend_Db_Table_Abstract::setDefaultAdapter() stores a default adapter, is there a way to store multiple adapters to this class. If I made a My_Db_Table_Abstract it might have different adapters. Or is it just preferred to use the Zend_Registry?

A: The setDefaultAdapter() method stores only a single Adapter object. This is convenient if your application needs only one database connection, and this is probably the more common type of application.

If your application needs multiple database connections, I would recommend storing each of them in the Zend_Registry and referencing them by their registry key. Strictly speaking, you could store one connection with setDefaultAdapter(), and then store any secondary connections in Zend_Registry, but I think it would make your code more clear if you used the same type of solution for all connections.

Q: Is it preferred to use the “Path_To_Table_Account” class naming format when creating table classes?

A: Yes, Zend Framework encourages the PEAR naming convention, in which each class belongs in its own file, and the class name maps to a directory hierarchy. Some methods of Zend_Db rely on this, for instance when fetching parent or dependent rowsets through the table-relationships methods, the related Table classes are loaded on demand, according to the mapping from class name to pathname.

Q: When is it preferred to use a Row Data Gateway relationship rather than a query with a join?

A: If you know you want to fetch the whole result set of a join, you should use a join. The Row Data Gateway relationship methods are good if you already have the Rowset from one table, and you may want related data from another table, on a row-by-row basis.

Q: Are there any speedy ways to turn the returned Rowset/Row objects into XML?

A: Both Zend_Db_Table_Rowset and Zend_Db_Table_Row have methods called toArray(), which turns them into plain PHP arrays. From this, you can iterate over them and generate build an XML tree using the DOM or SimpleXML APIs in PHP.

Q: Are there any special considerations when using the Table Data Gateway against database views rather than against tables directly?

A: No issues are known -- but I admit that the Zend_Db_Table unit tests do not currently assure compatibility with database views. There may be issues supporting views for some brands of database.

You can certainly query views using Zend_Db_Select or the Adapter's query() method.

Q: How well does using the Table Data Gateway scale?

A: The Table Data Gateway runs ordinary SQL queries as you request. It just provides you an object-oriented interface for invoking these queries. So it should scale about as well as the equivalent SQL queries scale. I haven't heard any complaints.

The only extra thing Table Data Gateway does is to query a table's metadata from the system catalog during construction of a Table object. If this is too costly, the Table class supports caching the metadata using the Zend_Cache component of Zend Framework.

Questions about Db from Zend Framework Overview webinar

Database brands supported

Q: Why is there a limitation of only using PDO? Is there a work around?

A: Zend_Db already provides Adapters for several non-PDO database extensions. Currently you can use MySQL, Oracle, and IBM DB2 with no requirement to have PDO.

See a list of supported adapters here:
http://framework.zend.com/manual/en/zend.db.html#zend.db.adapter

Q: Does the Zend_Db Adapter allow you to access databases without having the database DB extension loaded?

A: No, Zend_Db_Adapter merely provides a consistent class interface so that you can use the PHP drivers for different databases without being required to change your application code. You do need to have the PHP extension for the respective database present and enabled.

Q: Is the DB2 adapter based on ODBC driver or is it independent?

A: The Zend_Db Adapter for IBM DB2 is based on the ibm_db2 extension in PHP. In addition, there is a new Adapter under development, which uses the pdo_ibm extension.

We do intend to develop a Zend_Db Adapter for the ODBC extension.

Q: Is there a preferred adapter to use for MySQL databases? How mature are Mysqli and Pdo_Mysql adapters?

A: Zend_Db offers two Adapters for accessing MySQL. One utilizes the pdo_mysql extension, and the other utilizes the mysqli extension. They are both functioning fine and we test them both with an identical suite of tests. Both adapters are passing with flying colors. You can run the test suites yourself – they're part of the ZF download.

ActiveRecord

Q: I would also be interested in knowing more about plans for a higher-level DB abstraction, such as ActiveRecord.

A: ZF has classes Zend_Db_Table and Zend_Db_Table_Row, implementing the design patterns known as "Table Data Gateway" and "Row Data Gateway". These patterns are very similar to ActiveRecord.

Q: What is the difference between Table Gateway and Zend_Db_Table?

A: Table Data Gateway is an abstracted design pattern described in Martin Fowler's book "Patterns of Enterprise Architecture." Zend_Db_Table is an implementation of that design pattern in PHP 5.

Q: Can you specify model associations in the models and do the model find/fetch function automatically retrieve associated model data by doing SQL joins?

A: Yes, you can define table relationships with a declarative syntax, which in general matches foreign key/primary key relationships in the database metadata. There are convenient methods for fetching parent and dependent rows based on these relationships. For more details, see: http://framework.zend.com/manual/en/zend.db.table.relationships.html