Drupal Active Records

Written by Henrique Recidive
October 24, 2007

This report accompanies the paper “A Data API for Drupal 7: Key steps to enabling transactional web service support” prepared by Nedjo Rogers and Henrique Recidive and aims to provide a concrete draft implementation of Active Records for Drupal as a basis for discussing and planning renewed core Drupal data APIs.

Motivation

- Provide consistency to Drupal internal APIs;
- Schema aware database operations;
- Schema level validation and filtering;
- All operations are 'observable';
- Do complex database operations while keeping the code clean and readable;
- Pave the way for simple, no thought, web services;

Background

There are several implementations of ORM (Object Relational Mappings) in different languages, examples of this are Ruby on Rails' Active Records and Hibernate. The first implements the Active Record Pattern while the other is a implementation of the Data Mapper Pattern. Each OMR implementation has its strengths and flaws. One thing I've observed is that as much as you try to abstract the whole SQL creation stuff, the most your code gets complex and inflexible. An example of this is Hibernate which implements among others an object query language (HQL). On the other hand, Ruby On Rails' Active records makes use of SQL fragments.

Implementation:

We've drafted a module called *record* for Drupal 6. We hope to implement this on Drupal 7 core. This module implements the Active Record Pattern for Drupal.

The module is at:

http://cvs.drupal.org/viewvc.py/drupal/contributions/modules/record/

Extend schema information using hook_schema_alter()

As the current Drupal 6 schema lacks some needed features, we make use of the _schema_alter() hook to add relationship information as well validation and filters to the Drupal schema. The new schema attributes are:

Reference types:

- *one*: describe relationships where the table contains one foreign record. E.g. a node has one current revision.
- *many*: describe relationships where the table contains one or more foreign records. E.g. a

user has many roles.

Reference properties:

- *required*: whether or not a foreign record is required. When building JOIN clauses, if required is set to TRUE then INNER JOIN is used.
- *join*: set the referenced table(s), mapping fields relationships.
- *validate*: it uses the the new filters available in PHP5. You can use this to do basic validation on the schema level. E.g. is it a valid email?
- save filter: allow processing of a field through this filter before saving the record. E.g. md5 encode passwords, and serialize array values.
- load filter: allow processing of a field when fetching a record from database. E.g. unserialize arrays.

The new hook_record()

The hook_record() is used to intercept operations done in the record. Record objects are passed by reference along with their types (table names) so modules that implement this hook can change the record object on every operation as well as perform additional procedures such as creating a record on another Drupal table. Several operations are available:

- *before save*: The record is about to be saved. Allow modules to perform operations before the record is saved to the database;
- *save:* The record was just saved. Allow modules to perform operations after the record is saved to the database;
- *before delete:* The record is about to be deleted. Allow modules to perform operations before the record is deleted from the database;
- *delete:* The record was just deleted. Allow modules to perform operations after the record is deleted from the database;
- *validate:* Allow modules to perform additional validations on a record;
- *load:* The record was just loaded. Allow modules to perform operations when the record is being loaded;

Examples

Loading records:

Shorthand

```
<?php

$node = record_load('node', $nid);

?>
```

Field and value pairs

```
<?php

$args = array('nid' => $nid);

$node = record_load('node', $args);

?>
```

Using a record object

```
<?php

$node = record_new('node');
$node->nid = $nid

node_load($node);

?>
```

Using where fragment

```
<?php

$args = array(
  '#where' => 'node.nid = %d',
  '#where values' => array($nid),
);

$node = record_load('node', $args);

?>
```

Finding records:

All records

```
<?php

$nodes = record_find('node');
```

```
?>
```

Using where fragment

```
<?php

$args = array(
  '#where' => "node.title LIKE '%%%s%%'",
  '#where values' => array($title),
);

$nodes = record_find('node', $args);

?>
```

Field and value pairs

```
<?php

$args = array('uid' => $uid);

$nodes = record_find('node', $args);

?>
```

Using foreign reference

```
<?php

$args = array(
  '#where' =>   user_users.name = '%s'  ,
  '#where values' => array($username),

  // Join user table using schema refereces.
  '#include' => array('user'),
);

$nodes = record_find('node', $args);

?>
```

Dealing with collections:

```
<?php

$args => array(
  'uid' => $uid,
  // Join roles table using schema references.
  '#include' => array('roles'),
);

$user = record_load('user', $args);

foreach ($user->roles as $role) {
  // Do something with user roles.
}

?>
```

Creating records (save):

```
<?php

$user = record_new('user');
$user->name = $username;

record_save($user);

$uid = $user->uid;

?>
```

Updating records (save):

```
<?php

$user = record_load('user', $uid);
$user->name = $username;

record_save($user);

?>
```

Validation:

```
<?php

$user = record_new('user');
$user->name = $username;

if (record_validate($user)) {
```

```
  record_save($user);
}
else {
  $errors = record_get_errors($user);

  // Do something with errors.
}

?>
```