Exercise Solutions

Section 5 Lesson 1. Database in Magento

Exercise 1: List Root Categories by Store

Use your IDE to do this exercise.

Step 1. Get a list of stores using `Mage_Core_Model_Resource_Store_Collection`

- In a new controller action, instantiate the store collection object.

```
public function rootCategoryListAction()
{
      $stores = Mage::getResourceModel('core/store_collection');
      /* @var $stores Mage_Core_Model_Resource_Store_Collection */
}
```

Step 2. Get root category IDs using `Mage_Core_Model_Store::getRootCategoryId()`

- Using the the store collection's `walk()` method, call each store's `getRootCategoryId()` method and build an array of root category IDs.

```
public function rootCategoryListAction()
{
      $stores = Mage::getResourceModel('core/store_collection');
      /* @var $stores Mage_Core_Model_Resource_Store_Collection */

      $rootIds = $stores->walk('getRootCategoryId');
}
```

  The `walk()` method accepts a callback parameter and will iterate over each member model instance in the collection and call that method. The result of that callback is an array of numerically indexed results.

Step 3. Create a category collection and filter it by the root category IDs

a) Instantiate the category collection.

```
public function rootCategoryListAction()
{
      $stores = Mage::getResourceModel('core/store_collection');
      /* @var $stores Mage_Core_Model_Resource_Store_Collection */

      $rootIds = $stores->walk('getRootCategoryId');

      $cats = Mage::getResourceModel('catalog/category_collection');
      /* @var $stores Mage_Catalog_Model_Resource_Category_Collection */
}
```

b) Using the category collection's `addIdFilter()` method, apply the array of root category IDs from the store collection.

```
public function rootCategoryListAction()
{
      $stores = Mage::getResourceModel('core/store_collection');
      /* @var $stores Mage_Core_Model_Resource_Store_Collection */

      $rootIds = $stores->walk('getRootCategoryId');

      $cats = Mage::getResourceModel('catalog/category_collection');
      /* @var $stores Mage_Catalog_Model_Resource_Category_Collection */

      $cats->addIdFilter($rootIds);
}
```


Step 4. Add the category name attribute to the result

- Use the EAV collection method `addAttributeToSelect()` to find join the name attribute data on to the collection query.

```
public function rootCategoryListAction()
{
      $stores = Mage::getResourceModel('core/store_collection');
      /* @var $stores Mage_Core_Model_Resource_Store_Collection */

      $rootIds = $stores->walk('getRootCategoryId');

      $cats = Mage::getResourceModel('catalog/category_collection');
      /* @var $stores Mage_Catalog_Model_Resource_Category_Collection */

      $cats->addIdFilter($rootIds);
      $cats->addAttributeToSelect('name');
}
```

Step 5. Display stores with the associated root category names

- Having set up the category filter and added the desired attribute, iterate over the store collection and display the store name with its root category name.

```
public function rootCategoryListAction ()
{
      $stores = Mage::getResourceModel('core/store_collection');
      /* @var $stores Mage_Core_Model_Resource_Store_Collection */

      $rootIds = $stores->walk('getRootCategoryId');

      $cats = Mage::getResourceModel('catalog/category_collection');
      /* @var $stores Mage_Catalog_Model_Resource_Category_Collection */

      $cats->addIdFilter($rootIds);
      $cats->addAttributeToSelect('name');

      $this->getResponse()->setHeader('Content-Type', 'text/plain');

      foreach ($stores as $store){
            $this->getResponse()->appendBody($store->getName() . ": ");
            $category = $cats->getItemById($store->getRootCategoryId());
            $this->getResponse()->appendBody($category->getName() . "\n");
      }
}
```

Exercise 2: Create a Tree of Categories Using the Database Adapter

Use your IDE to output a list of children categories for each store, building on the previous exercise.

Step 1. Create a recursive function `_displayChildCategories()` that takes a parent category

a) Create this method in the action controller with which we've been working.

```
protected function _displayChildCategories($category)
{
	$resource = $category->getResource();

	//Different tablename getter for flat vs EAV
	if (Mage::helper('catalog/category_flat')->isEnabled()) {
		$table = $resource->getMainTable();
	} else {
		$table = $resource->getEntityTable();
	}
}
```

Magento's Flat Catalog implementation uses parallel resource model classes for category and product entities. A curious by-product of this implementation can be seen above. When retrieved through its data model, the resource model object type will be different based on the entity's Flat Catalog status, and each resource model has a distinct method for retrieving the entity table name.

b) Next, get the read adapter so that we can work directly with the `select` object.

```
protected function _displayChildCategories($category)
{
	$resource = $category->getResource();

	//$table name logic (snip...)

	//Retrieve the read adapter to work with the select object
	$select = $resource->getReadConnection()->select()
		->from($table, '*')
		->where('parent_id=?', $category->getId());
}
```

Here we are setting a filter (`WHERE` clause) on the `parent_id` column. When a flat category is enabled, this query will contain the category name attribute value, but when EAV storage is being used, the attribute value table needs to be joined.

c) Test the category storage mode using the flat category helper's `isEnabled()` method and retrieve an instance of the "name" EAV attribute model.

```
protected function _displayChildCategories($category)
{
      $resource = $category->getResource();

      //$table name logic (snip...)

      //$select filter method (snip...)

      if (! Mage::helper('catalog/category_flat')->isEnabled()) {
        $attribute = $resource->getAttribute('name');
      }
}
```

d) Using the schema information from the attribute model, add a join clause to the select object.

```
protected function _displayChildCategories($category)
{
      $resource = $category->getResource();

      //$table name logic (snip...)

      //$select filter method (snip...)

      if (! Mage::helper('catalog/category_flat')->isEnabled()) {

            $attribute = $resource->getAttribute('name');
            $attributeTable = $attribute->getBackend()->getTable();

            $select->joinInner(
                  array($attributeTable),
                  "{$table}.entity_id={$attributeTable}.entity_id " .
                        "AND attribute_id={$attribute->getId()}",
                  array('name' => 'value')
            );
      }
}
```


e) Now that the "name" attribute value is certainly available, retrieve the result set from the database.

```
protected function _displayChildCategories($category)
{
	$resource = $category->getResource();

	//$table name logic (snip...)

	//$select filter method (snip...)

	//conditional "name" logic (snip...)

	$rows = $resource->getReadConnection()->fetchAll($select);
}
```

f) Next, iterate over the resulting rows, instantiate a fresh category data model instance, and apply the row results to it.

```
protected function _displayChildCategories($category)
{
	$resource = $category->getResource();

	//$table name logic (snip...)

	//$select filter method (snip...)

	//conditional "name" logic (snip...)

	$rows = $category->getResource()
		->getReadConnection()->fetchAll($select);

	foreach ($rows as $row) {
		$childCat = Mage::getModel('catalog/category')->setData($row);

		$output = str_pad("", $childCat->getLevel()*2, " ") .
			"{$childCat->getId()} {$childCat->getName()}\n";

		$this->getResponse()->appendBody($output);
		$this->_displayChildCategories($childCat);
	}
}
```

This method of iterating over the result set and applying it to data model instances is essentially how collections build their items (except that they do not query the database inside a loop). The entire method follows.


```
protected function _displayChildCategories($category)
{
	$resource = $category->getResource();

	//Different tablename getter for flat vs EAV
	if (Mage::helper('catalog/category_flat')->isEnabled()) {
		$table = $category->getResource()->getMainTable();
	} else {
		$table = $resource->getEntityTable();
	}

	//Retrieve the read adapter to work with the select object
	$select = $resource->getReadConnection()
		->select()
		->from($table, '*')
		->where('parent_id=?', $category->getId());

	//Join "name" attribute table if necessary
	if (! Mage::helper('catalog/category_flat')->isEnabled()) {
		$attribute = $category->getResource()
			->getAttribute('name');
		$select->joinInner(
			array($attribute->getBackendTable()),
"{$table}.entity_id={$attribute->getBackendTable()}.entity_id " .
	"AND attribute_id={$attribute->getId()}",
			array('name' => 'value')
		);
	}

	//Retrieve results from database
	$rows = $category->getResource()
		->getReadConnection()->fetchAll($select);

	//Iterate, apply data, and call this method recursively
	foreach ($rows as $row) {
		$childCat = Mage::getModel('catalog/category')
			->setData($row);
		$output = str_pad("", $childCat->getLevel()*2, " ") .
			"{$childCat->getId()} {$childCat->getName()}\n";

		$this->getResponse()->appendBody($output);
		$this->_displayChildCategories ($childCat);
	}
}
```

g) Now call the `_displayChildCategories` method in the action that was created in the previous example.

```
public function rootCategoryListAction()
{
      $stores = Mage::getResourceModel('core/store_collection');
      /* @var $stores Mage_Core_Model_Resource_Store_Collection */

      $rootIds = $stores->walk('getRootCategoryId');

      $cats = Mage::getResourceModel('catalog/category_collection');
      /* @var $stores Mage_Catalog_Model_Resource_Category_Collection */

      $cats->addIdFilter($rootIds);
      $cats->addAttributeToSelect('name');
      $this->getResponse()->setHeader('Content-Type', 'text/plain');

      foreach ($stores as $store){
            $this->getResponse()->appendBody($store->getName() . ": ");
            $category = $cats->getItemById($store->getRootCategoryId());
            $this->getResponse()->appendBody($category->getName() . "\n");
            //Output category tree
            $this->_displayChildCategories($category);
      }
}
```

Note: This is not production ready code. The purpose of the code is to demonstrate how to execute queries to the database directly, and how to fetch the required table names. Querying the database in a loop or recursive method is not recommended for performance reasons.

h) The resulting out put should look something like the following:

```
English: Root Catalog
     10 Furniture
       22 Living Room
       23 Bedroom
     13 Electronics
       8 Cell Phones
       12 Cameras
         25 Accessories
         26 Digital Cameras
       15 Computers
         27 Build Your Own
         28 Laptops
         29 Hard Drives
         30 Monitors
         31 RAM / Memory
         32 Cases
         33 Processors
         34 Peripherals
     18 Apparel
       4 Teens
       5 Shoes
         16 Mens
         17 Womens
       36 Women
       37 Men
     41 Gift Cards
     42 Specials
     43 Members Specials
French: Root Catalog
     10 Furniture
       22 Living Room
```

Exercise 3: Create a New Module: `Training_Distributor`

Use your IDE to create a full module configuration and model classes.

Step 1. Create a module registration file

- Create an XML named *Training_Distributor.xml* in *app/etc/modules*.

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <modules>
        <Training_Distributor>
            <active>true</active>
            <codePool>local</codePool>
        </Training_Distributor>
    </modules>
</config>
```

Note: Of course, any file name in this directory will be loaded into the configuration DOM as long as it ends in ".xml".

Step 2. Create necessary folders

- Create *app/code/local/Training/Distributor/etc.*

  Based on the code pool and module configuration node from Step 1, the application will attempt to load a file named *config.xml* from this directory.

Step 3. Create the module configuration

- Create a *config.xml* file in the directory from Step 2.


```
<?xml version="1.0" encoding="UTF-8"?>
<config>
      <global>

            <blocks>
                  <training_distributor>
                        <class>Training_Distributor_Block</class>
                  </training_distributor>
            </blocks>

            <helpers>
                  <training_distributor>
                        <class>Training_Distributor_Helper</class>
                  </training_distributor>
            </helpers>

            <models>
                  <training_distributor>
            <class>Training_Distributor_Model</class>
            <resourceModel>training_distributor_resource</resourceModel>
                  </training_distributor>

                  <training_distributor_resource>
                        <class>Training_Distributor_Model_Resource</class>
                        <entities>
                              <distributor>
                              <table>training_distributor_entity</table>
                              </distributor>
                        </entities>
                  </training_distributor_resource>
            </models>

      </global>
</config>
```

The above configuration creates the classgroup `training_distributor` for the factory methods as well as a handle for the `traning_distributor_entity` table.

Step 4. Create a Model directory

- Based on the model-related class prefixes above (`Training_Distributor_Model` and `Training_Distributor_Model_Resource`), create the following directories:
  - *app/code/local/Training/Distributor/Model/*
  - *app/code/local/Training/Distributor/Model/Resource/*

Step 5. Create a model, a resource model, and collection classes

a) Create the file *app/code/local/Training/Distributor/Model/Distributor.php* and define the data model class.

```
<?php

class Training_Distributor_Model_Distributor
      extends Mage_Core_Model_Abstract
{}
```

b) Next, define the Magento `_construct()` method and set the resource model and collection model by calling the data model `_init()` method.

```
<?php

class Training_Distributor_Model_Distributor
      extends Mage_Core_Model_Abstract
{
      protected function _construct()
      {
            //Set the resource model and set the collection
            //class as resource + '_collection'
            $this->_init('training_distributor/distributor');
      }
}
```

The string passed in the `_init()` call is used as the resource model argument, is passed to `Mage::getResourceSingleton()`, and is used to set the resource collection class argument by having it appended with "`_collection`" and passed to `Mage::getResourceCollection()`.

c) Based on the initialization above and the resource model class prefix, create a file at *app/code/local/Training/Distributor/Model/Resource/Distributor.php*, calling its `_init()` method in the protected `_construct()` method.

```
<?php

class Training_Distributor_Model_Resource_Distributor
      extends Mage_Core_Model_Resource_Db_Abstract
{
      protected function _construct()
      {
            //Pass the tablename (handle) and primary key
            $this->_init('training_distributor/distributor','entity_id');
      }
}
```


The first argument is mapped to the distributors table value under the entities node in the configuration.

d) Having created the data model and resource model class definitions, the final class definition for this entity is the collection class. It is initialized in much the same way. Create the class definition at *app/code/local/Training/Distributor/Model/Resource/Distributor/Collection.php.*

```
<?php

class Training_Distributor_Model_Resource_Distributor_Collection
      extends Mage_Core_Model_Resource_Db_Collection_Abstract
{
      protected function _construct()
      {
            /* Typically only the first arguments is given and used for
                  both the resource model and data model arguments. */
            $this->_init('training_distributor/distributor');
      }
}
```


Step 6. Test the model, resource model, and collection classes

- Create a test file in the Magento root folder, include the bootstrap *Mage.php* file, set the developer mode to true, and initialize the application.

  Then initialize the distributor model, resource model and collection classes, output the results, and test the cross-references between the classes and the table.

```
<?php

include 'app/Mage.php';
Mage::setIsDeveloperMode(true);
Mage::app();

$model = Mage::getModel('training_distributor/distributor');
$resource = Mage::getResourceModel('training_distributor/distributor');
$collection =
      Mage::getResourceModel('training_distributor/distributor_collection'
);

$data = array(
    'model' => $model,
    'resource' => $resource,
    'collection' => $collection,

    'model-to-resource' => $model->getResource(),
    'model-to-collection' => $model->getCollection(),

    'collection-to-model' => $collection->getNewEmptyItem(),
    'collection-to-resource' => $collection->getResource(),

    'resource-table' => $resource->getMainTable(),
    'resource-id-field' => $resource->getIdFieldName()
);

foreach ($data as $key => $value)
{
    echo "$key: \n   ";
    echo (is_object($value) ? get_class($value) : $value);
    echo "\n\n";
}
```

