Article Categories

Magento
Laravel
OroCRM
Modern PHP
SugarCRM
Drupal
WebOS (HP/Palm)
Python
AppleScript
Links (beta)

Quickies Sites

Magento Quickies
OroCRM Quickies
Wordpress Quickies

Quickies

Development oriented blogs with less detailed, but still vital, programming tips.

Magento Quickies
Wordpress Quickies
Oro Quickies

Alan Storm is a human being living in Portland, OR by way of Seattle, WA by way of Portland, OR by way of Rochester, NY. He likes making computers do things, and talks about that here.

He also likes to make things on the web. If you need something made on the web, drop him a line.

We're a little worried about his penchant for slipping into the third person narrative form.

Follow the Feed

Follow the Twitter

Magento ORM: Entity Attribute Value; Part 1

In our last article we told you there were two kinds of Models in Magento. Regular, or "simple" Models, and Entity Attribute Value (or EAV) Models. We also told you this was a bit of a fib. Here's where we come clean.

The following is part of a longer series about Magento aimed at developers familiar with PHP MVC development. While each article can be read stand alone, each article does build on concepts and code covered in previous articles. If you're confused, be sure to catch up on the older stuff first. Also, later versions of Magento have introduced changes that require you to take a different approach to creating new tables. Zachary Schuessler did a bit of detective work and figured out the new science. Highly recommended reading.

ALL Magento Models inherit from the `Mage_Core_Model_Abstract` / `Varien_Object` chain. What makes something either a simple Model or an EAV Model is its **Model Resource**. While all resources extend the base `Mage_Core_Model_Resource_Abstract` class, simple Models have a resource that inherits from `Mage_Core_Model_Mysql4_Abstract`, and EAV Models have a resource that inherits from `Mage_Eav_Model_Entity_Abstract`

If you think about it, this makes sense. As the end-programmer-user of the system you want a set of methods you can use to talk to and manipulate your Models. You don't care what the back-end storage looks like, you just want to get properties and invoke methods that trigger business rules.

What is EAV

Wikipedia defines EAV as

> Entity-Attribute-Value model (EAV), also known as object-attribute-value model and open schema is a data model that is used in circumstances where the number of attributes (properties, parameters) that can be used to describe a thing (an "entity" or "object") is potentially very vast, but the number that will actually apply to a given entity is relatively modest. In mathematics, this model is known as a sparse matrix.

Another metaphor that helps me wrap my head around it is "EAV is normalization applied to the database **table schema**". In a traditional database, tables have a fixed number of columns

```
+------------------+
| products         |
+------------------+
| product_id       |
| name             |
| price            |
| etc..            |
+------------------+
```

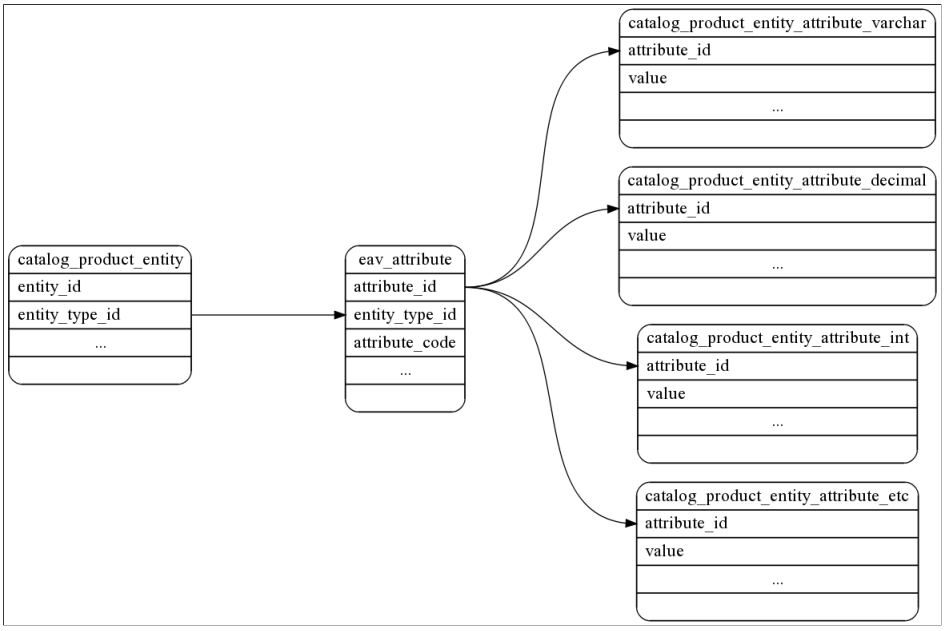
```
+------------+----------------+------------------+---------+
| product_id | name           | price            | etc...  |
+------------+----------------+------------------+---------+
| 1          | Widget A       | 11.34            | etc...  |
+------------+----------------+------------------+---------+
| 2          | Dongle B       | 6.34             | etc...  |
+------------+----------------+------------------+---------+
```

Every product has a name, every product has a price, etc.

In an EAV Model, each "entity" (product) being modeled has a **different** set of attributes. EAV makes a lot of sense for a generic e-commerce solution. A store that sells laptops (which have a CPU speed, color, ram amount, etc) is going to have a different set of needs than a store that sells yarn (yarn has a color, but no CPU speed, etc.). Even within our hypothetical yarn store, some products will have length (balls of yarn), and others will have diameter (knitting needles).

There aren't many open source or commercial databases that use EAV by default. There are none that are available on a wide variety of web hosting platforms. Because of that, Varien engineers have built an EAV system out of PHP objects that use MySQL as a data-store. In other words, they've built an EAV database system **on top of** a traditional relational database.

In practice this means any Model that uses an EAV resource has its attributes spread out over a number of MySQL tables.



The above diagram is a rough layout of the database tables Magento consults when it looks up an EAV record for the `catalog_product` entity. Each individual product has a row in `catalog_product_entity`. All the available attributes in the **entire** system (not just for products) are stored in `eav_attribute`, and the actual attribute values are stored in tables with names like `catalog_product_entity_attribute_varchar`, `catalog_product_entity_attribute_decimal`, `catalog_product_entity_attribute_etc`.

Beyond the mental flexibility an EAV system gives you, there's also the practical benefit of avoiding ALTER TABLE statements. When you add a new attribute for your products, a new row is inserted into `eav_attribute`. In a traditional relational database/single-table system, you'd need to ALTER the actual database structure, which can be a time consuming/risky proposition for tables with large data-sets.

The downside is there's no one single simple SQL query you can use to get at all your product data. Several single SQL queries or one large join need to be made.

Advanced EAV

That's EAV in a nutshell. The rest of this articles is a run-through of what's needed to create a new EAV Model in Magento. It's the hairiest thing you'll read about Magento and it's something that 95% of people working with the system will never need to do. However, understanding what it takes to build an EAV Model Resource will help you undersatnd what's going on with the EAV Resources that Magento uses.

Because the EAV information is so dense, I'm going to assume you've studied up and are already very familiar with Magento's MVC and grouped class name features. We'll help you along the way, but training wheels are off.

Weblog, EAV Style

We're going to create another Model for a weblog post, but this time using an EAV Resource. To start with, setup and create a new module which responds at the the following URL

```
http://example.com/complexworld
```

If you're unsure how to do this, be sure you've mastered the concepts in the previous tutorials.

Next, we'll create a new Model named Weblogeav. Remember, it's the **Resource** that's considered EAV. We design and configure our Model the exact same way, so let's configure a Model similar to one we created last time.

```
<global>
    <!-- ... -->
    <models>
        <!-- ... -->
        <complexworld>
            <class>Alanstormdotcom_Complexworld_Model</class>
            <resourceModel>complexworld_resource_eav_mysql4</resourceModel>
        </complexworld>
        <!-- ... -->
    </models>
    <!-- ... -->
</global>
```

You'll notice the one different to setting up a regular Model is the `<resourceModel/>` name looks a bit more complex (`weblog_resource_eav_mysql4`).

We'll still need to let Magento know about this resource. Similar to basic Models, EAV Resources are configured in the same `<model/>` node with everything else.

```
<global>
    <!-- ... -->
    <models>
        <!-- ... -->
        <complexworld_resource_eav_mysql4>
            <class>Alanstormdotcom_Complexworld_Model_Resource_Eav_Mysql4</class>
```

```
            <entities>
                <eavblogpost>
                    <table>eavblog_posts</table>
                </eavblogpost>
            </entities>
        </complexworld_resource_eav_mysql4>
        <!-- ... -->
    </models>
    <!-- ... -->
</global>
```

Again, so far this is setup similarly to our regular Model Resource. We provide a `<class/>` that configures a PHP class, as well as an `<entities/>` section that will let Magento know the base table for an individual Model we want to create. The `<eavblogpost/>` tag is the name of the specific Model we want to create, and its inner `<table/>` tag specifies the base table this Model will use (more on this later).

We're also going to need `<resources/>`. Again, this is identical to the setup of a regular Model. The resources are the classes that Magento will use to interact with the database back-end.

```
<global>
    <!-- ... -->
    <resources>
        <complexworld_write>
            <connection>
                <use>core_write</use>
            </connection>
        </complexworld_write>
        <complexworld_read>
            <connection>
                <use>core_read</use>
            </connection>
        </complexworld_read>
    </resources>
    <!-- ... -->
</global>
```

Where Does That File Go?

Until wide adoption of PHP 5.3 and namespaces, one of the trickier (and tedious) parts of Magento will remain remembering how `<classname/>`s relate to file paths, and then ensuring you create the correctly named directory structure and class files. After configuring any `<classname/>`s or URIs, I find it useful to attempt to instantiate an instance of the class it in a controller **without** fist creating the class files. This way PHP will throw an exception telling me it can't find a file, along with the file location. Give the following a try in your Index Controller.

```
public function indexAction() {
    $weblog2 = Mage::getModel('complexworld/eavblogpost');
    $weblog2->load(1);
    var_dump($weblog2);
}
```

As predicted, a warning should be thrown

```
Warning: include(Alanstormdotcom/Complexworld/Model/Eavblogpost.php) [function.include]: fai
```

In addition to telling us the path where we'll need to define the new resource class this also serves as a configuration check. If we'd been warned with the following

```
Warning: include(Mage/Complexworld/Model/Eavblogpost.php) [function.include]: failed to oper
```

we'd know our Model was misconfigured, as Magento was looking for the Model in `code/core/Mage` instead of `code/local/Alanstormdotcom`.

So, lets create our Model class

```
File: app/code/local/Alanstormdotcom/Complexworld/Model/Eavblogpost.php

class Alanstormdotcom_Complexworld_Model_Eavblogpost extends Mage_Core_Model_Abstract {
    protected function _construct()
    {
        $this->_init('complexworld/eavblogpost');
    }
}
```

Remember, the Model itself is resource independent. A regular Model and an EAV Model both extend from the same class. It's the resource that makes them different.

Clear your Magento cache, reload your page, and you should see a **new** warning.

```
Warning: include(Alanstormdotcom/Complexworld/Model/Resource/Eav/Mysql4/Eavblogpost.php)
```

As expected, we need to create a class for our Model's resource. Let's do it!

```
File: app/code/local/Alanstormdotcom/Complexworld/Model/Resource/Eav/Mysql4/Eavblogpost.php
```

```
class Alanstormdotcom_Complexworld_Model_Resource_Eav_Mysql4_Eavblogpost extends Mage_Eav_Mc
{
    public function _construct()
    {
        $resource = Mage::getSingleton('core/resource');
        $this->setType('complexworld_eavblogpost');
          $this->setConnection(
                $resource->getConnection('complexworld_read'),
                $resource->getConnection('complexworld_write')
            );
    }
}
```

So, already we're seeing a few differences between a simple Model Resource and an EAV Model Resource. First off, we're extending the `Mage_Eav_Model_Entity_Abstract` class. While `Mage_Eav_Model_Entity_Abstract` uses the same `_construct` concept as a regular Model Resource, there's no `_init` method. Instead, we need to handle the init ourselves. This means telling the resource what connection-resources it should use, and passing a unique identifier into the `setType` method of our object.

Another difference in `Mage_Eav_Model_Entity_Abstract` is `_construct` is **not** an abstract method. It's not clear if this is an oversight on the part of the Magento team, something to do with backward compatibility, or if there's something deeper in the system preventing it. Regardless, it's a useful fact to know when you go spelunking in the depths of the Magento system code.

So, with that, let's clear the Magento cache and reload the page. You should see a new exception which reads

```
Invalid entity_type specified: complexworld_eavblogpost
```

Magento is complaining that it can't find a entity_type named `complexworld_eavblogpost`. This is the value you set above

```
$this->setType('complexworld_eavblogpost');
```

Every entity has a type. Types will, among other things, let the EAV system know which attributes a Model uses, and allow the system to link to tables that store the values for attributes. We'll need to let Magento know that we're adding a new entity type. Take a look in the MySQL table named `eav_entity_type`.

```
mysql> select * from eav_entity_type\G
*************************** 1. row ***************************
          entity_type_id: 1
        entity_type_code: customer
            entity_model: customer/customer
         attribute_model:
            entity_table: customer/entity
      value_table_prefix:
         entity_id_field:
         is_data_sharing: 1
        data_sharing_key: default
default_attribute_set_id: 1
          increment_model: eav/entity_increment_numeric
     increment_per_store: 0
    increment_pad_length: 8
      increment_pad_char: 0
*************************** 2. row ***************************
          entity_type_id: 2
        entity_type_code: customer_address
            entity_model: customer/customer_address
         attribute_model:
            entity_table: customer/address_entity
      value_table_prefix:
         entity_id_field:
         is_data_sharing: 1
        data_sharing_key: default
default_attribute_set_id: 2
          increment_model:
     increment_per_store: 0
    increment_pad_length: 8
      increment_pad_char: 0
```

This table contains a list of all the entity_types in the system. The unique identifier `complexworld_eavblogpost` corresponds to the `entity_type_code` column.

Systems and Applications

This illustrates the single most important Magento concept, one that many people struggle to learn.

Consider the computer in front of you. The OS (Mac OS X, Windows, Linux, etc.) is the software system. Your web browser (Firefox, Safari, IE, Opera) is the application. Magento **is a system** first, and an application second. You build e-commerce applications using the Magento system. What gets confusing is, there's a lot of places in Magento where the system code is exposed in a really raw form to the application code. The EAV system configuration living in the same database as your store's data is an example of this.

If you're going to get deep into Magento, you need to treat it like it's an old Type 650 machine. That is to say, it's the kind of thing you can't effectively program applications in unless unless you have a deep understanding

of the system itself.

Creating a Setup Resource

So, it's theoretically possible to manually insert the rows you'll need into the Magento database to get your Model working, but it's not recommended. Fortunately, Magento provides a specialized Setup Resource that provides a number of helper method that will automatically create the needed records to get the system up and running.

So, for starters, configure the Setup Resource like you would any other.

```
<resources>
  <!-- ... -->
    <complexworld_setup>
        <setup>
            <module>Alanstormdotcom_Complexworld</module>
            <class>Alanstormdotcom_Complexworld_Entity_Setup</class>
        </setup>
        <connection>
            <use>core_setup</use>
        </connection>
    </complexworld_setup>
  <!-- ... -->
</resources>
```

Next, create its class file.

```
File: app/code/local/Alanstormdotcom/Complexworld/Entity/Setup.php
class Alanstormdotcom_Complexworld_Entity_Setup extends Mage_Eav_Model_Entity_Setup {
}
```

Take note that we're extending from `Mage_Eav_Model_Entity_Setup` rather than `Mage_Core_Model_Resource_Setup`.

Finally, we'll setup our installer script. If you're not familiar with the naming conventions here, you'll want to review the previous tutorial on Setup Resources.

```
File: app/code/local/Alanstormdotcom/Complexworld/sql/complexworld_setup/mysql4-install-0.1.

<?php   $installer = $this;
throw new Exception("This is an exception to stop the installer from completing");
```

Clear your Magento Cache, reload you page, and the above exception should be thrown, meaning you've correctly configured your Setup Resource.

NOTE: We'll be building up our install script piece by piece. If you've read the previous tutorial, you'll know you need to remove the setup's row from the `core_resource` table and clear your cache to make an installer script re-run. For the remainder of this tutorial, please remember that anytime we add or remove an item from our installer and re-run it, you'll need to remove this row from the database and clear your Magento cache. Normally you would create this file and run it once, a tutorial is something of an edge case.

Adding the Entity Type

To begin, add the following to your Setup Resource installer script, and then run the script by loading any page (after removing the above exception)

```
$installer = $this;
$installer->addEntityType('complexworld_eavblogpost',Array(
//entity_mode is the URL you'd pass into a Mage::getModel() call
'entity_model'          =>'complexworld/eavblogpost',
//blank for now
'attribute_model'       =>'',
//table refers to the resource URI complexworld/eavblogpost
//<complexworld_resource_eav_mysql4>...<eavblogpost><table>eavblog_posts</table>
'table'         =>'complexworld/eavblogpost',
//blank for now, but can also be eav/entity_increment_numeric
'increment_model'       =>'',
//appears that this needs to be/can be above "1" if we're using eav/entity_increment_numeric
'increment_per_store'   =>'0'
));
```

We're calling the `addEntityType` method on our installer object. This method allows us to pass in the entity type (`complexworld_eavblogpost`) along with a list of parameters to set its default values. If you've run this script, you'll notice new rows in the `eav_attribute_group`, `eav_attribute_set`, and `eav_entity_type` tables.

So, with that in place, if we reload our complexworld page, we'll get a new error.

```
SQLSTATE[42S02]: Base table or view not found: 1146 Table 'magento.eavblog_posts' doesn't ex
```

Creating the Data Tables

So, we've told Magento about our new entity type. Next, we need to add the mysql tables that will be used to store all the entity values, as well as configure the system so it knows about these tables.

If you've spent any amount of time looking at the stock Magento installer files, you've seen a lot of manual SQL being used to create tables for EAV Models. Fortunately for us, this is no longer necessary. Our Setup Resource

has a method named `createEntityTables` which will automatically setup the tables we need, as well as add some configuration rows to the system. Let's add the following line to our setuup resource.

```
$installer->createEntityTables(
$this->getTable('complexworld/eavblogpost')
);
```

The `createEntityTables` method accepts two parameters. The first is the base table name, the second is a list of options. We're using the Setup Resource's `getTable` method to pull the table name from our config. If you've been following along, you know this should resolve to the string `eavblog_posts`. We've omitted the second parameter which is an array of options you'll only need to used it for advanced situations that are beyond the scope of this tutorial.

After running the above script, you should have the following new tables in your database

```
eavblog_posts
eavblog_posts_datetime
eavblog_posts_decimal
eavblog_posts_int
eavblog_posts_text
eavblog_posts_varchar
```

You'll also have an additional row in the `eav_attribute_set` table

```
mysql> select * from eav_attribute_set order by attribute_set_id DESC LIMIT 1 \G
*************************** 1. row ***************************
  attribute_set_id: 65
    entity_type_id: 37
attribute_set_name: Default
        sort_order: 6
```

So, let's go back to our page and reload.

```
http://example.com/complexworld
```

Success! You should see no errors or warnings, and and a dumped `Alanstormdotcom_Complexworld_Model_Eavblogpost` —- with no data.

Adding Attributes

The last step we need to take in our Setup Resource is telling Magento what attributes we want our Model to have. This would be equivalent to adding new columns in a single database table setup. Again, the Setup Resource will help us. The two methods we're interested in are `installEntities` and `getDefaultEntites`.

The naming here can be a little confusing. After all, we just ran some code that told Magento about our entities, but now we need to do it again?

The code from the previous section was simply telling Magento about a **type** of entity that we **may** add to the system. These next bits of code are what will actually add a single entity of your type to the system. If you wanted to, I'm pretty sure you could create multiple entities of the same type. You can also get tattoos on the inside of your eyelids. If you're not sure if either is for you, they're not.

So, let's add the following method to our Setup Resource.

```
class Alanstormdotcom_Complexworld_Entity_Setup extends Mage_Eav_Model_Entity_Setup {
    public function getDefaultEntities()
    {
        die("Calling ".__METHOD__);
    }
}
```

And then add the following call to the end of our setup script.

```
$installer->installEntities();
```

Reload your page and you should see the die/exit message specified above.

```
Calling Alanstormdotcom_Complexworld_Entity_Setup::getDefaultEntities
```

You **may** see an exception something like the following

```
[message:protected] => SQLSTATE[23000]: Integrity constraint violation: 1217 Cannot delete c
```

If that's the case, it's because you're re-running your setup script and calling the `createEntityTables` method again. Magento generates SQL statements for the `DROP`ing and `CREATE`ing your tables. Unfortunately, it doesn't take into account the `FORIGN KEY` relationships, and tries to `DROP/CREATE` the primary entity table first.

For the purposes of this tutorial, just comment out the line(s) calling `createEntityTables`. Again, this installer would normally run once and `createEntityTables` would only be called once.

Configuring our New Entity

When you call `installEntites`, Magento will need to do several things to install your entities. Before it can do this, it needs to know what your entities are. The contract here is for you to have `getDefaultEntities` return the entities. (It's also possible to pass installEntities a list of entities to install, but I'm trying to stick to the Magento core conventions).

Side Node: Strangely, Magento entities are configured using a simple nested `array` structure. It seems an odd choice in a system that's been described by some as OO'd to death.

To start with, we'll add our Eavblogpost entity and give it a single attribute named title.

```
class Alanstormdotcom_Complexworld_Entity_Setup extends Mage_Eav_Model_Entity_Setup {
    public function getDefaultEntities()
    {
        return array (
            'complexworld_eavblogpost' => array(
                'entity_model'      => 'complexworld/eavblogpost',
                'attribute_model'   => '',
                'table'             => 'complexworld/eavblogpost',
                'attributes'        => array(
                    'title' => array(
                        //the EAV attribute type, NOT a mysql varchar
                        'type'              => 'varchar',
                        'backend'           => '',
                        'frontend'          => '',
                        'label'             => 'Title',
                        'input'             => 'text',
                        'class'             => '',
                        'source'            => '',
                        // store scope == 0
                        // global scope == 1
                        // website scope == 2
                        'global'            => 0,
                        'visible'           => true,
                        'required'          => true,
                        'user_defined'      => true,
                        'default'           => '',
                        'searchable'        => false,
                        'filterable'        => false,
                        'comparable'        => false,
                        'visible_on_front'  => false,
                        'unique'            => false,
                    ),
                ),
            )
        );
    }
}
```

Alight, that's a pile of code. Let's break it apart.

Expected Return Value

The `getDefaultEntities` method should return a php array of key/value pairs. Each key should be the name of the entity type (setup with `$installer->addEntityType('complexworld_eavblogpost',...`, each value should be an array that describes the entity.

Array that Describes the Entity

The array that describes the entity is also a list of key/value pairs. Some of these should look familiar

```
'entity_model'      => 'complexworld/eavblogpost',
'attribute_model'   => '',
'table'             => 'complexworld/eavblogpost',
'attributes'        => array(
```

These should match the values you used in the call to `$installer->addEntityType(...`. The final key, `attributes`, should contain yet another array that describes the attributes themselves.

Yet Another Array that Describes the Attributes Themselves

This next array is, yet again, an array of key value pairs. This time the key is the attribute name (`title`) and the values are a final array of key value pairs that define the attribute. For the sake of simplicity we've chose to define a single attribute, but you could go on to define as many as you'd like.

Final Array of Key Value Pairs that Define the Attribute

Finally, we have a long list of attribute properties.

```
//the EAV attribute type, NOT a mysql varchar
'type'              => 'varchar',
'backend'           => '',
'frontend'          => '',
'label'             => 'Title',
'input'             => 'text',
'class'             => '',
'source'            => '',
// store scope == 0
// global scope == 1
// website scope == 2
'global'            => 0,
'visible'           => true,
'required'          => true,
'user_defined'      => true,
'default'           => '',
'searchable'        => false,
'filterable'        => false,
```

```
'comparable'        => false,
'visible_on_front'  => false,
'unique'            => false,
```

Unfortunately, this is where your author has to 'fess up and tell you he's unsure what most of these do. Many involve driving features of the Magento back-end UI, such as `label` and `input`. Varian engineers have chosen to tightly bind their UI implementation with their back-end Model structure. This allows them certain advantages, but it means there are large parts of the system that remain opaque to outsiders, particularly web developers who've been chanting to mantra of back-end/front-end separation for near on a decade.

That said, the one important property you'll want to make note of is

```
'type' => 'varchar'
```

This defines the type of the value that the attribute will contain. You'll recall that we added table for each attribute type

```
eavblog_posts_datetime
eavblog_posts_decimal
eavblog_posts_int
eavblog_posts_text
eavblog_posts_varchar
```

While these do not refer to the MySQL column types, (but instead the EAV attribute types), their names (varchar, datetime, etc.) are indicative of the values they'll hold.

So, now that we have everything in place, lets refresh things one last time to run our installer script. After calling `installEntities`, we should have

1. A new row in `eav_attribute` for the title attribute
2. A new row in `eav_entity_attribute`

Tying it all Together

This is clearly the lamest.blogmodel.ever, but lets try adding some rows and iterating through a collection and get the heck out of here before our heads explode. Add the following two actions to your Index Controller.

```
public function populateEntriesAction() {
    for($i=0;$i<10;$i++) {
        $weblog2 = Mage::getModel('complexworld/eavblogpost');
        $weblog2->setTitle('This is a test '.$i);
        $weblog2->save();
    }

    echo 'Done';
}

public function showcollectionAction() {
    $weblog2 = Mage::getModel('complexworld/eavblogpost');
    $entries = $weblog2->getCollection()->addAttributeToSelect('title');
    $entries->load();
    foreach($entries as $entry)
    {
        // var_dump($entry->getData());
        echo '<h3>'.$entry->getTitle().'</h3>';
    }
    echo '<br>Done<br>';
}
```

Let's populate some entries! Load up the following URL

```
http://magento.dev/index.php/complexworld/index/populateEntries
```

If you take a look at your database, you should see 10 new rows in the `eavblog_posts` table.

```
mysql> select * from eavblog_posts order by entity_id DESC;
+-----------+----------------+------------------+--------------+-----------+----------+-----
| entity_id | entity_type_id | attribute_set_id | increment_id | parent_id | store_id | crea
+-----------+----------------+------------------+--------------+-----------+----------+-----
|        10 |             31 |                0 |              |         0 |        0 | 2009
|         9 |             31 |                0 |              |         0 |        0 | 2009
|         8 |             31 |                0 |              |         0 |        0 | 2009
|         7 |             31 |                0 |              |         0 |        0 | 2009
|         6 |             31 |                0 |              |         0 |        0 | 2009
|         5 |             31 |                0 |              |         0 |        0 | 2009
|         4 |             31 |                0 |              |         0 |        0 | 2009
|         3 |             31 |                0 |              |         0 |        0 | 2009
|         2 |             31 |                0 |              |         0 |        0 | 2009
|         1 |             31 |                0 |              |         0 |        0 | 2009
+-----------+----------------+------------------+--------------+-----------+----------+-----
```

as well as 10 new rows in the `eavblog_posts_varchar` table.

```
mysql> select * from eavblog_posts_varchar order by value_id DESC;
+----------+----------------+-------------+----------+-----------+------------------+
```

```
| value_id | entity_type_id | attribute_id | store_id | entity_id | value           |
+----------+----------------+--------------+----------+-----------+-----------------+
|       10 |             31 |          933 |        0 |        10 | This is a test 9 |
|        9 |             31 |          933 |        0 |         9 | This is a test 8 |
|        8 |             31 |          933 |        0 |         8 | This is a test 7 |
|        7 |             31 |          933 |        0 |         7 | This is a test 6 |
|        6 |             31 |          933 |        0 |         6 | This is a test 5 |
|        5 |             31 |          933 |        0 |         5 | This is a test 4 |
|        4 |             31 |          933 |        0 |         4 | This is a test 3 |
|        3 |             31 |          933 |        0 |         3 | This is a test 2 |
|        2 |             31 |          933 |        0 |         2 | This is a test 1 |
|        1 |             31 |          933 |        0 |         1 | This is a test 0 |
+----------+----------------+--------------+----------+-----------+-----------------+
```

Notice that `eavblog_posts_varchar` is indexed to `eavblog_posts` by `entity_id`.

Finally, let's pull our Models back out. Load the following URL in your browser

```
http://magento.dev/index.php/complexworld/index/showCollection
```

This should give us a

```
Warning: include(Alanstormdotcom/Complexworld/Model/Resource/Eav/Mysql4/Eavblogpost/Collecti
```

So Close! We didn't make a class for our collection object! Fortunately, doing so is just as easy as with a regular Model Resource. Add the following file with the following contents

```
File: Alanstormdotcom/Complexworld/Model/Resource/Eav/Mysql4/Eavblogpost/Collection.php

class Alanstormdotcom_Complexworld_Model_Resource_Eav_Mysql4_Eavblogpost_Collection extends
{

    protected function _construct()
    {
        $this->_init('complexworld/eavblogpost');
    }
}
```

This is just a standard Magento `_construct` method to initialize the Model. With this in place, reload the page, and we'll see all the titles outputted.

Which Attributes?

Those of you with sharp eyes may have noticed something slightly different about the collection loading.

```
$entries = $weblog2->getCollection()->addAttributeToSelect('title');
```

Because querying for EAV data can be SQL intensive, you'll need to specify which attributes it is you want your Models to fetch for you. This way the system can make only the queries it needs. If you're willing to suffer the performance consequences, you can use a wild card to grab **all** the attributes

```
$entries = $weblog2->getCollection()->addAttributeToSelect('*');
```

Jumping Off

So, that should give you enough information to be dangerous, or at least enough information so you're not drowning the next time you're trying to figure out why the yellow shirts aren't showing up in your store. There's still plenty to learn about EAV; here's a few topics I would have liked to cover in greater detail, and may talk about in future articles

1. EAV Attributes: Attributes aren't limited to datetime, decimal, int, text and varchar. You can create your own class files to model different attributes. This is what the blank `attribute_model` is for.
2. Collection Filtering: Filtering on EAV collections can get tricky, especially when you're dealing with the above mentioned non-simple attributes. You need to use the `addAttributeToFilter` method on your collection before loading.
3. The Magento EAV Hierarchy: Magento has taken their basic EAV Model and built up a hierarchy that's very tie to store functionality, as well as including strategies to reduce the number of queries an EAV Model generates (the concept of a **flat** Model, for example)

EAV Models are, without a doubt, the most complicated part of the Magento system that an ecommerce web developer will need to deal with. Remember to take deep breaths and that, at the end of the day, its just programming. Everything happens for a concrete reason, you just need to figure out why.

Read more about Magento

Originally published December 6, 2009

Recommend 8 Share Sort by Oldest

Garry • 4 years ago

Maybe relatively complicated but this section here was one of the easiest to follow and get working with a minimum of fuss. It helped that I'm an old sql jokey - so the concepts were new but familiar and fascinating. I particularly liked the description of EAV. The idea of adding a row instead of a column for new properties is both flexible and handy!

• Share ›

AntonioAlonso • 4 years ago

Hi

I completted the section ,went to the whole process and I'm getting a memory error when trying to get the collection.

I'm running magento locally with XAMP and php.ini has assigned 128M. I'm using an empty fresh install of magento 1.5.1.0.

Thanks

• Share ›

alanstorm Mod ➔ AntonioAlonso • 4 years ago

Memory errors usually have something to do with the specifics of your system. Also "completing the section and going to the whole process" isn't descriptive enough to help. If you can describe your problem better, trying asking a question on Stack Overflow. Someone (possibly me) might be able to help.

• Share ›

Srdjan • 4 years ago

Thanks for this advanced tutorial. In most cases, one would have to pay for this.

• Share ›

alanstorm Mod ➔ Srdjan • 4 years ago

If you're looking for an advanced layout tutorial to pay for, look no further than

http://store.pulsestorm.net/pr...

1 • Share ›

Savita_phadte • 4 years ago

Hi
I need some help in the grid functionality.
I have created an attribute for product entity "buyer_id" . and this "buyer_id" will be the user_id from admin_user table.
In the grid i want to show all the products details with the buyer's name
The results are displayed but how to populate the dropdown with buyers and sort accordinly.

Thanks

• Share ›

alanstorm Mod ➔ Savita_phadte • 4 years ago

Sounds like a detailed request. Try asking on Stack Overflow instead.

• Share ›

Pete Mackie • 4 years ago

This 'Complexworld' tutorial code does not run with Magento 1.6.0.0. I tried it on two separate Mac OS X platforms. It used to run with Magento 1.5.1.0.

When this code:

```
$installer->createEntityTables(
$this->getTable('complexworld/eavblogpost')
);
```

is run in the file: app/code/local/Alanstormdotcom/Complexworld/sql/complexworld_setup/mysql4-install-0.1.0.php, in generates the following tables:

```
eavblog_posts
eavblog_posts_datetime
eavblog_posts_decimal
eavblog_posts_int
```

and then generates an exception--internal within Magento-when attempting to generate the table: eavblog_posts_text

I've spent way too much time on this tutorial exercise attempting to get it to work. I'm moving on without the ability to generate EAV database tables, at least in my Mac OS X dev environment.

• Share ›

alanstorm Mod ↗ Pete Mackie • 4 years ago

Looks like a Magento bug in 1.6. I ran through the setup and saw the same exception, and took a quick look at my SQL logs, and saw the following

[BLOG/TEXT column 'value' used in key specification with a key length]

Newer versions (1.6?) of Magento attempt to create a number of indexes to all the EAV tables as part of the create table syntax

INDEX `IDX_EAVBLOG_POSTS_TEXT_ATTRIBUTE_ID_VALUE` (`attribute_id`, `value`),
Indexing a text field with specifying a length isn't possible.http://www.google.com/search?hl=en&am... seems like a bug, or maybe the core team has dropped support for text tables in new EAV models for performance reasons, or there could be a new method for setting up these tables. Worth reporting to the core team if you can find the time.

▲ | ▼ • Share ›

Pete Mackie • 4 years ago

re: "Looks like a Magento bug in 1.6."

I did file a bug report, as suggested by Alan, at Magento Issue Tracking: http://www.magentocommerce.com...

For those who are interested in following the bug report (Issue #26535) comments and hopefully an issue resolution, go to:
http://www.magentocommerce.com...

▲ | ▼ • Share ›

Matis • 4 years ago

EAV Attributes: Attributes aren't limited to datetime, decimal, int, text and varchar. You can create your own class files to model different attributes. This is what the blank attribute_model is for.

Do you have any examples or links for this? What if I needed to create custom product attribute with custom form renderer? How could I do that?

▲ | ▼ • Share ›

alanstorm Mod ↗ Matis • 4 years ago

I don't do that often enough to have a quick example of it.

▲ | ▼ • Share ›

Gagan • 3 years ago

I am just wondering what would I do if your blog would not be dere ! Amazing explanation .

▲ | ▼ • Share ›

Francois99 • 3 years ago

When you add this (see tutorial) to mysql4-install-0.1.0.php (for second run) :

$installer->createEntityTables($this->getTable('complexworld/eavblogpost'));

remove first the record 'complexworld_setup' from table 'core_resource'

▲ | ▼ • Share ›

Adam ↗ Francois99 • 3 years ago

Where do you put this createEntityTables? before or after $installer->addEntityType?

▲ | ▼ • Share ›

Francois99 ↗ Adam • 3 years ago

After

▲ | ▼ • Share ›

Anonymous • 3 years ago

1." eavblog_posts_varchar is indexed to eavblog_posts by entity_id. " - I understand that are you refering to the FK.

▲ | ▼ • Share ›

abagray • 3 years ago

While running this tutotrial I am having problem removing the complexworld_setup from the core_resource table. I am running magento 1.6.2. Once when changed the version number from 0.1.0 to 0.2.0, I was able to remove but not anymore. I have already disabled my Cache in Admin. Has anyone solution?

▲ | ▼ • Share ›

giannis • 3 years ago

Propably the best article on the web for magento, targeting developers...

About the adminhthml part and of course the edit/new/save controller, shall we try to mimic the ones of an already existing entity?

Why noone has tried to create a generic cck construction module for magento? starting from here and using what's being used in catalog module? maybe because intergrating magento with another platform is more easy and straigtforward?

• Share ›

allen • 3 years ago

HI,
I ain't good at english.
what's mean in (You can also get tattoos on the inside of your eyelids. If you're not sure if either is for you, they're not.)

2 • Share ›

alanstorm Mod → allen • 3 years ago

It's sarcasm. Getting tattoos on the inside of your eyelids would be a painful thing, and also a culturally silly thing to do in America. So while you "could create multiple entities of the same type", unless you really know you want to, you don't need to.

2 • Share ›

allen • 3 years ago

the part of Adding Attributes, I can't Calling Alanstormdotcom_Complexworld_Entity_Setup::getDefaultEntities!

• Share ›

alanstorm Mod → allen • 3 years ago

I realize there's a language barrier, but that sentence doesn't make sense.

• Share ›

allen • 3 years ago

HI, I found a strange problem!
I has a new row in eav_attribute table.
but I can't find a new row in eav_entity_attribute.

• Share ›

Wouter • 3 years ago

i keep getting this error:
"Error in file: "C:\xampp\htdocs\magento\app\code\core\Mage\Eav\sql\eav_setup\mysql4-install-0.7.0.php" - SQLSTATE[42S01]: Base table or view already exists: 1050 Table 'eav_attribute' already exists
"
i don't have any clue what i am doing wrong
please help!

• Share ›

alanstorm Mod → Wouter • 3 years ago

It looks like Magento is trying to reinstall the Mage_Eav module's setup script. My guess is you accidentally deleted THE ENTIRE resource table instead of just the row for your module.

• Share ›

Ayuhamaro • 3 years ago

"Creating the Data Tables" paragraphs, mysql4-install-0.1.0.php added (and removed from the "core_resource a" table the row the "complexworld_setup")

$ installer-> createEntityTables (
$ this-> getTable ('complexworld / eavblogpost')
);

The following error, can not be established the eavblog_posts table

Mage_Eav_Exception Object
(
[_messages:protected] => Array
(
)

[message:protected] => Can't create table: eavblog_posts
[string:Exception:private] =>
[code:protected] => 0
[file:protected] => /var/www/html/magento/app/Mage.php

see more

• Share ›

alanstorm Mod → Ayuhamaro • 3 years ago

You may be running into this bug

http://stackoverflow.com/quest...

1 • Share ›

Ayuhamaro alanstorm • 3 years ago
Modified does not fix

I am using the Magento 1.7.0.0

The modified code with 1.6.0.0, the code is as follows

$eavTable
-> addForeignKey ($ this-> getFkName ($ eavTableName, 'entity_id', $ baseTableName, 'entity_id'),
'entity_id', $ this-> getTable ($ baseTableName), 'entity_id',
Varien_Db_Ddl_Table :: ACTION_CASCADE, Varien_Db_Ddl_Table :: ACTION_CASCADE).....
(Other)Other problems?
Thank you

• Share ›

alanstorm Mod Ayuhamaro • 3 years ago
Magento's trying to run a create table command and can't. Determine why it can't, and you'll solve your problem.

• Share ›

Ayuhamaro alanstorm • 3 years ago
Okay, thank you

• Share ›

Matheus Gomes Ayuhamaro • 3 years ago
I had the same issue with 1.7.0.0.
I found that the problem was in the transaction (line 1359), but I don't know why... So I just copied the whole createEntityTables() method to my install script (I don't wanna mess with the core!), moved the foreach loop (line 1361) out of the transaction and everything worked fine!

Someone have any idea why the beginTransaction() method could be throwing an exception?

• Share ›

Kultibulb • 3 years ago
Thank you for this great explanation.
Everything works fine except that no row for the title attribute is added to the eav_entity_attribute table.

• Share ›

alanstorm Mod Kultibulb • 3 years ago
I imagine that's settable somehow. Check the setup resource models and setup script that ship with the core modules.

• Share ›

Kultibulb alanstorm • 3 years ago
it seems that it's because no 'group' value was specified in the attribute array of parameters in getDefaultEntities() function.

• Share ›

Subscribe Add Disqus to your site Privacy