**Introduction**

In general, it’s not good to duplicate data across multiple tables in a database. Any time two or more columns across tables refer to the same real-life business information, it’s easier to introduce bugs that allow the data to become inconsistent. However, there are often problems that require building models, which refer to data in other models stored in other tables. The simplest solution for this is to include a foreign key in the model, which can be used in the application logic to load an instance of the other model and access the data. But, if only some of the data from the other model is needed, the additional database query can be avoided using a different approach. An alternative is to create a model that loads data with a single query that includes JOIN statements to pull in data from additional tables. Here’s a quick tip on how to implement such a model in Magento 2.

*NOTE: This article assumes basic knowledge of creating a new module and new entity in Magento 2.*

**Step 1: Model and Database schema**

For this example, suppose we want to allow customers to sign up to a mailing list on checkout, and we want to track data about the sales that correspond with customers signing up. We want a model that looks like this:

**MailingSignUp**

* Customer ID
* Customer Email
* Sale ID
* Sale Amount
* Allow email for promotions

We can use the following schema for our new entity:

**mailing\_sign\_up**

* entity\_id: int (primary key)
* sales\_order\_id: int (foreign key)
* allow\_promotions: boolean

We include the foreign key to the sales order, and we can use joins to get the rest of the data.

**Step 2: The ResourceModels**

When we create a new entity to use in our Magento 2 application, we must define a ResourceModel class. In our example, the path to that class would look something like this:

app/code/MyVendor/MyModule/Model/ResourceModel/MailingSignUp.php

In most cases, this file would simply contain a \_construct function that defines the main table to be used by the model, like so:

<?php

namespace MyVendor\MyModule\Model\ResourceModel;

use Magento\Framework\Model\ResourceModel\Db\AbstractDb;

class MailingSignUp extends AbstractDb

{

protected function \_construct()

{

$this->\_init('mailing\_sign\_up', 'entity\_id');

}

}

However, we need to also tell Magento to customize the query it uses to fetch the data. This is where the \_getLoadSelect function comes in. This is a protected function in the AbstractDb class, which means we can override it in our child class.

<?php

namespace MyVendor\MyModule\Model\ResourceModel;

use Magento\Framework\Model\ResourceModel\Db\AbstractDb;

class MailingSignUp extends AbstractDb

{

protected function \_construct()

{

$this->\_init('mailing\_sign\_up', 'entity\_id');

}

/\*\*

\* Retrieve select object for load object data

\*

\* @param string $field

\* @param mixed $value

\* @param \Magento\Framework\Model\AbstractModel $object

\* @return \Magento\Framework\DB\Select

\*/

protected function \_getLoadSelect($field, $value, $object)

{

$field = $this->getConnection()->quoteIdentifier(sprintf('%s.%s', $this->getMainTable(), $field));

$select = $this->getConnection()

->select()

->from($this->getMainTable())

->where($field . '=?', $value)

->join('sales\_order',

'mailing\_sign\_up.sales\_order\_id = sales\_order.entity\_id',

[

'customer\_id',

'grand\_total'

])->join('customer\_entity',

'sales\_order.customer\_id = customer\_entity.entity\_id',

[

'email'

]);

return $select;

}

}

This override of the \_getLoadSelect will add the necessary joins to the query Magento uses to fetch the data when you load an instance of the model. Similarly, we must also override the \_initSelect function in the ResourceModel Collection class. This class would be located in:

app/code/MyVendor/MyModule/Model/ResourceModel/MailingSignUp/Collection.php

<?php

namespace MyVendor\MyModule\Model\ResourceModel\MailingSignUp;

use MyVendor\MyModule\Model\MailingSignUp;

use MyVendor\MyModule\Model\ResourceModel\MailingSignUp as MailingSignUpResource;

use Magento\Framework\Model\ResourceModel\Db\Collection\AbstractCollection;

class Collection extends AbstractCollection

{

/\*\*

\* @var string $\_idFieldName

\*/

protected $\_idFieldName = 'entity\_id';

protected function \_construct()

{

$this->\_init(MailingSignUp::class, MailingSignUpResource::class);

}

protected function \_initSelect()

{

$this->getSelect()

->from(['main\_table' => $this->getMainTable()])

->join('sales\_order',

'main\_table.sales\_order\_id = sales\_order.entity\_id',

[

'customer\_id',

'grand\_total'

])->join('customer\_entity',

'sales\_order.customer\_id = customer\_entity.entity\_id',

[

'email'

]);

return $this;

}

}

Now, whether you load an individual instance or a collection of the model, the instance will contain 3 new properties, customer\_id, grand\_total, and email, which can be accessed using the normal getData functions, for example, $mailingSignUp->getGrandTotal(). In the contexts which this entity will be used, it may be more clear to access this data by another name. This can be accomplished by explicitly defining the getter functions in the model class, like so:

<?php

namespace MyVendor\MyModule\Model;

use MyVendor\MyModule\Model\ResourceModel\MailingSignUp as MailingSignUpResource;

use Magento\Framework\Model\AbstractModel;

class MailingSignUp extends AbstractModel

{

public function \_construct()

{

$this->\_init(MailingSignUpResource::class);

}

public function getSaleAmount()

{

return $this->getData('grand\_total');

}

}

Now, the getSaleAmount function will retrieve the grand total of the associated sale retrieved by the join.