Magento® U

Student Guide

Fundamentals of Magento 2 Development
Unit Four

magento.com/training/overview

v.1.0

Magento® U

training@magento.com

magento.com/training/overview

magento.com/training/catalog/certification

Unit Four Contents

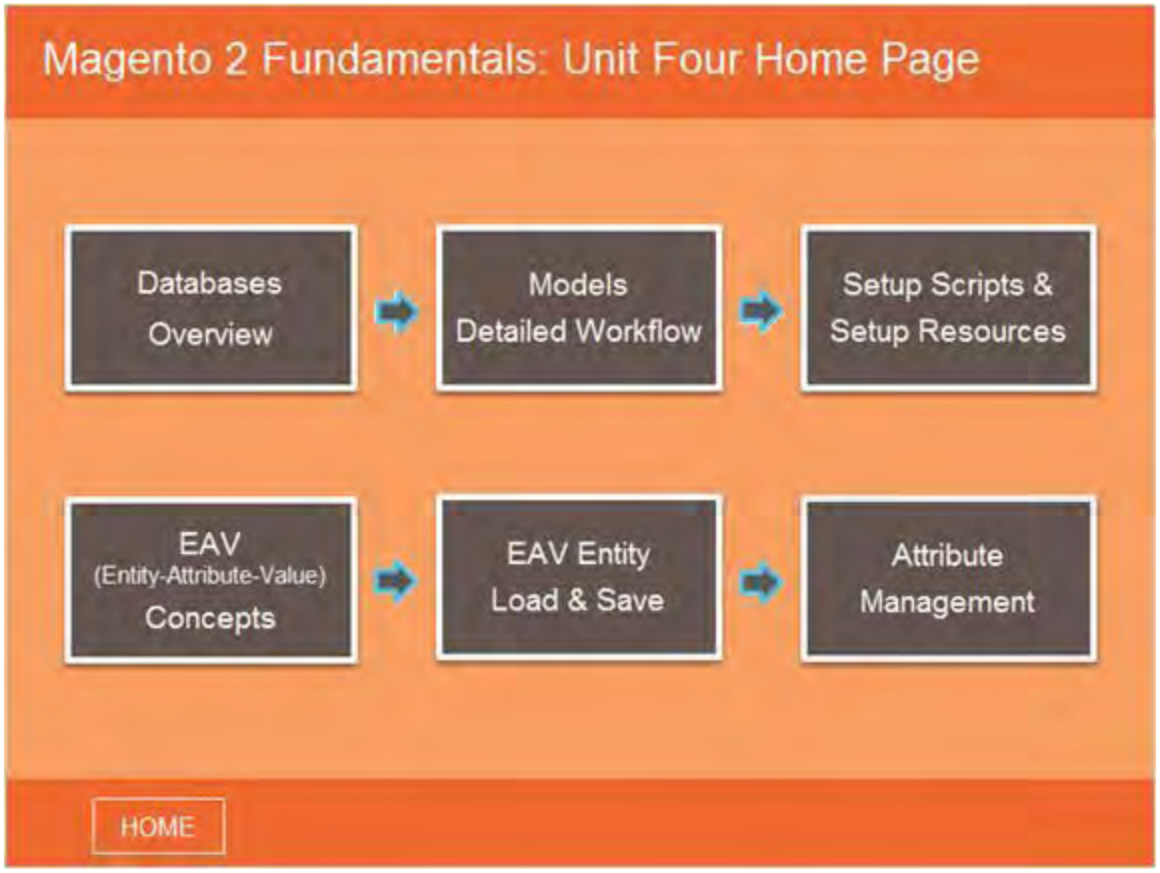
About This Guide

This guide uses the following symbols in the notes that follow the slides.

Symbol	Indicates…
✎	A note, tip, or other information brought to your attention.
❗	Important information that you need to know.
📄	A cross-reference to another document or website.
💡	Best practice recommended by Magento

1. Fundamentals of Magento 2 Development: Unit Four

1.1 Home Page


Magento 2 Fundamentals: Unit Four Home Page

Notes:

Unit Four of the Magento 2 Fundamentals course contains six modules.

The suggested flow of the course is indicated by the arrows. However, you are free to access any of the modules, at any time, by simply clicking the Home button on the bottom of each slide.

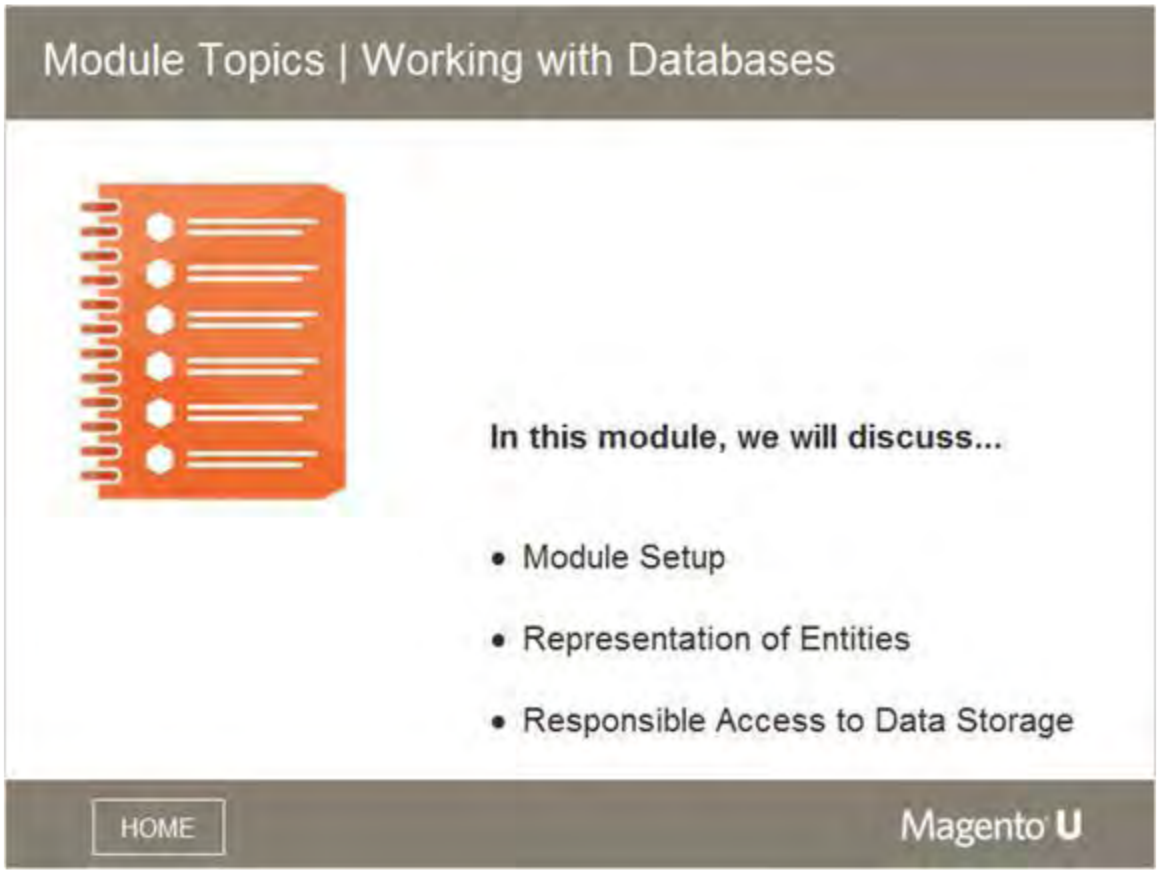
2. Databases Overview

2.1 Databases Overview

Notes:

In this module, we present an overview of how databases work within Magento 2. The database functionality has not changed much from Magento 1 except for enhancements that make the features easier to use.

2.2 Module Topics | Working with Databases



Module Topics | Working with Databases

In this module, we will discuss...

- Module Setup
- Representation of Entities
- Responsible Access to Data Storage

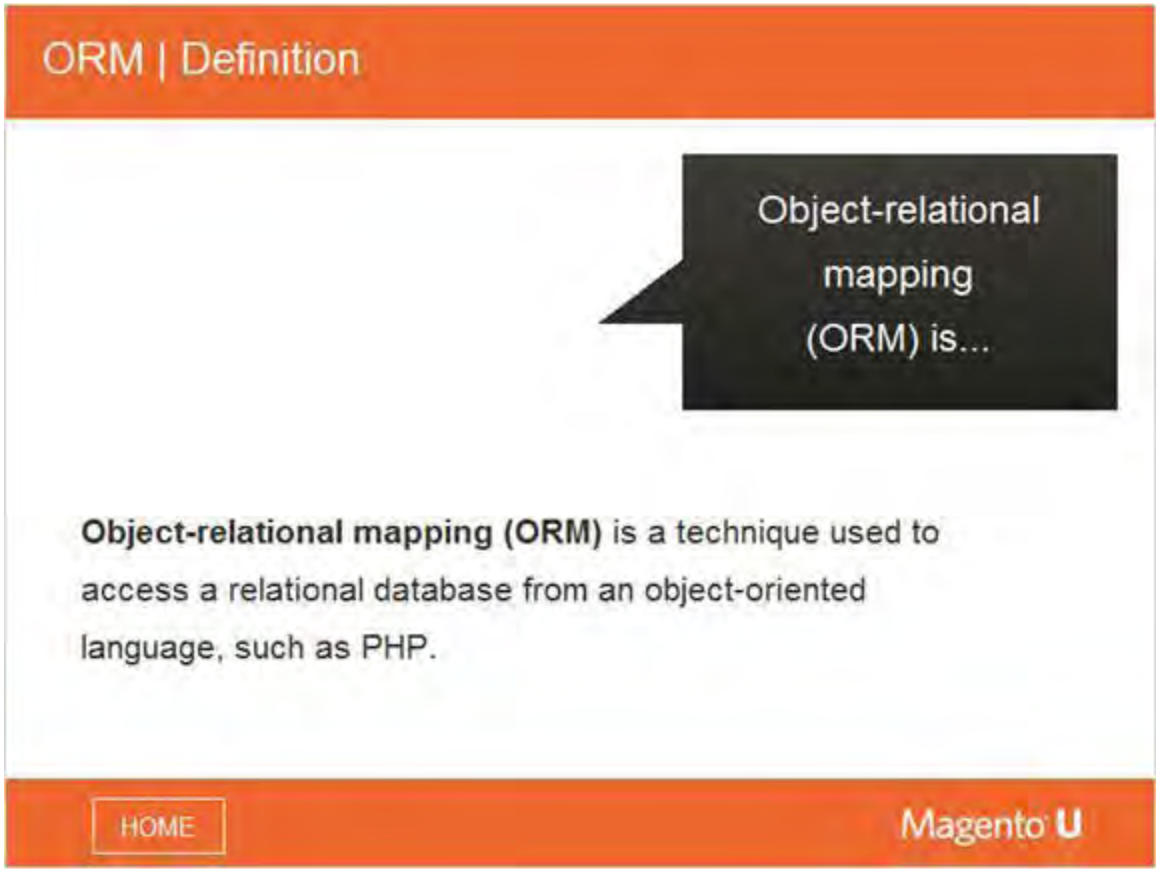
HOME
Magento U

Notes:

In this module, we will discuss:

- Module Setup
- Representation of Entities
- Responsible Access to Data Storage

2.3 ORM | Definition


ORM | Definition

Object-relational mapping (ORM) is...

Object-relational mapping (ORM) is a technique used to access a relational database from an object-oriented language, such as PHP.

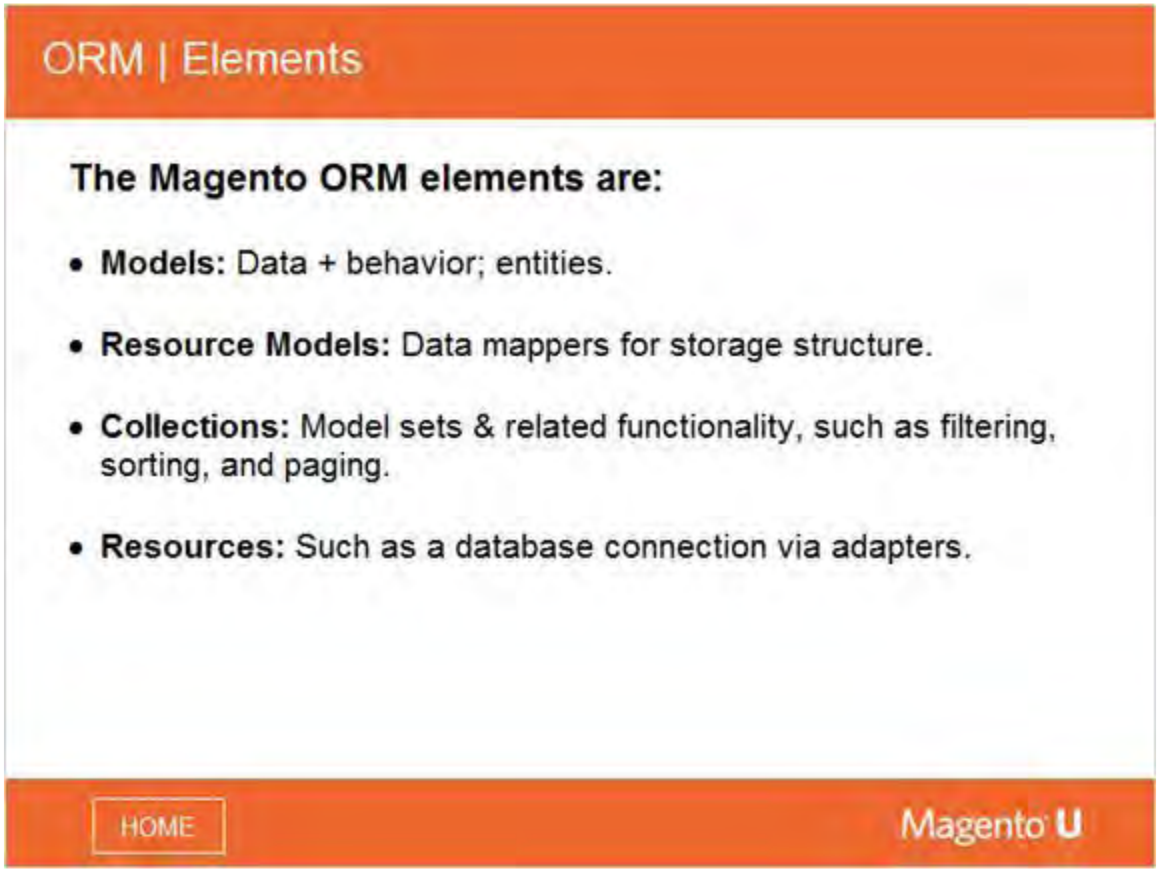
HOME
Magento U

Notes:

Object- relational mapping (ORM) is a technique used to persist objects from an object-oriented language, such as PHP, in a relational database.

The Magento ORM is built around models, resource models, and resource collections.

2.4 ORM | Elements


ORM | Elements

The Magento ORM elements are:

- **Models:** Data + behavior; entities.
- **Resource Models:** Data mappers for storage structure.
- **Collections:** Model sets & related functionality, such as filtering, sorting, and paging.
- **Resources:** Such as a database connection via adapters.

HOME
Magento U

Notes:

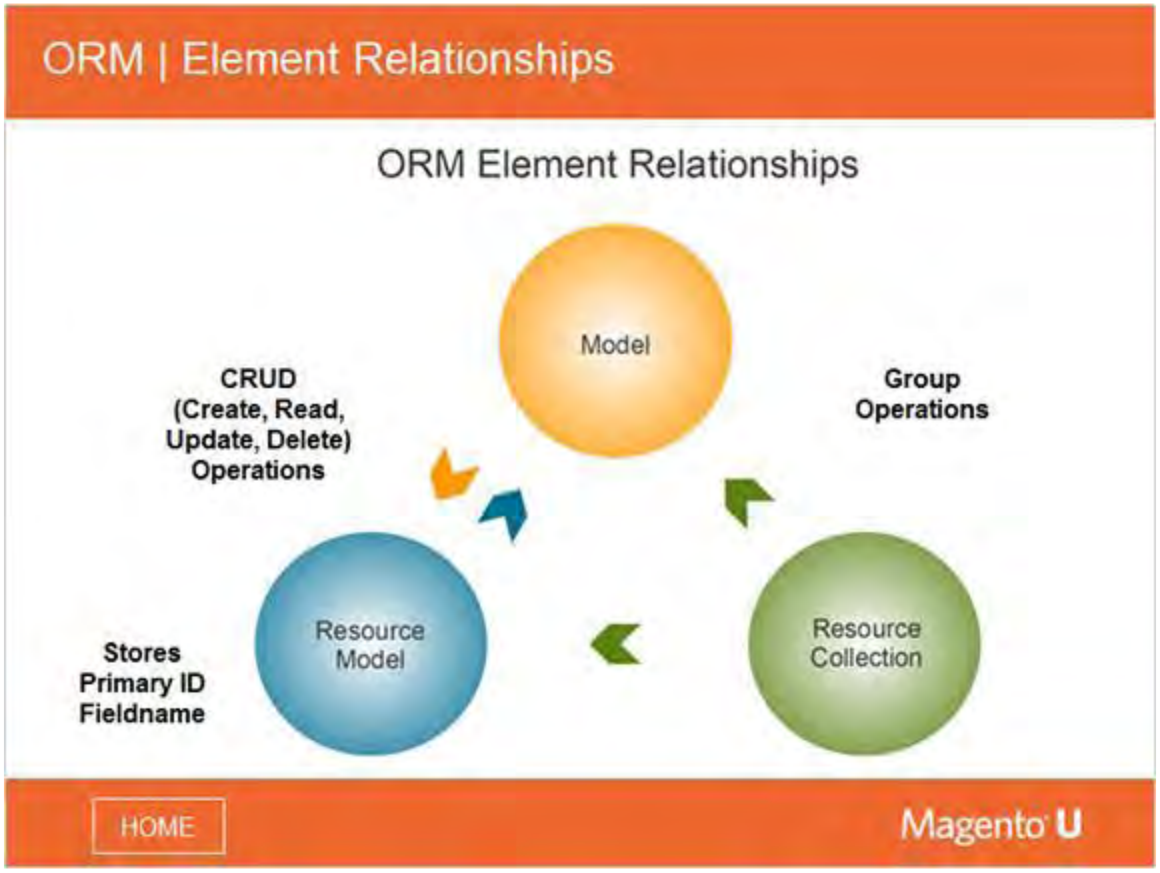
The Magento ORM elements are:

- **Models:** Are data and behavior, representing entities.
- **Resource Models:** Are data mappers for the storage structure.
- **Collections:** Encapsulate sets of models and related functionality, such as filtering, sorting, and paging.
- **Resources:** Include database connections via adapters.

The ORM gives you the capability to create, load, update, and delete data to a database.

A collection in Magento is a class that implements both the `IteratorAggregate` and the `Countable` PHP5 SPL interfaces. Collections are widely used in Magento to store a set of objects of a specific type.

2.5 ORM | Element Relationships


ORM | Element Relationships
ORM Element Relationships
Model
CRUD (Create, Read, Update, Delete) Operations
Group Operations
Stores Primary ID Fieldname
Resource Model
Resource Collection
HOME
Magento U

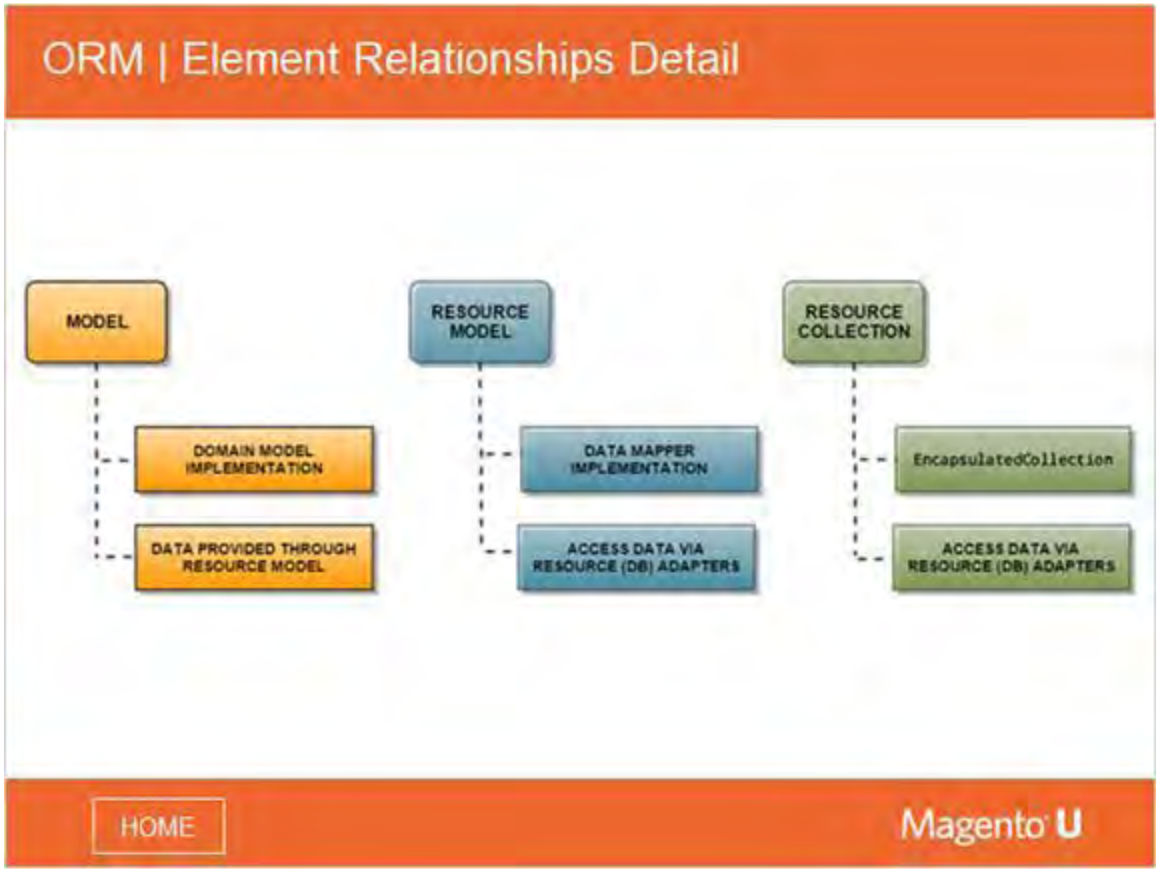
Notes:

The Magento ORM concept requires that only models contain and manipulate data.

Magento has a very flexible, highly abstract, concept of what a model is. Anything that is represented by a database table is most likely going to be a model.

Note: "CRUD" stands for **"**Create, Read, Update, Delete."

2.6 ORM | Element Relationships Detail


ORM | Element Relationships Detail

MODEL
DOMAIN MODEL IMPLEMENTATION
DATA PROVIDED THROUGH RESOURCE MODEL
RESOURCE MODEL
DATA MAPPER IMPLEMENTATION
ACCESS DATA VIA RESOURCE (DB) ADAPTERS
RESOURCE COLLECTION
EncapsulatedCollection
ACCESS DATA VIA RESOURCE (DB) ADAPTERS
HOME
Magento U

Notes:

Models encapsulate storage independent business logic. The models generally don't know where data is persisted. If they require access to the database, they delegate to the resource model.

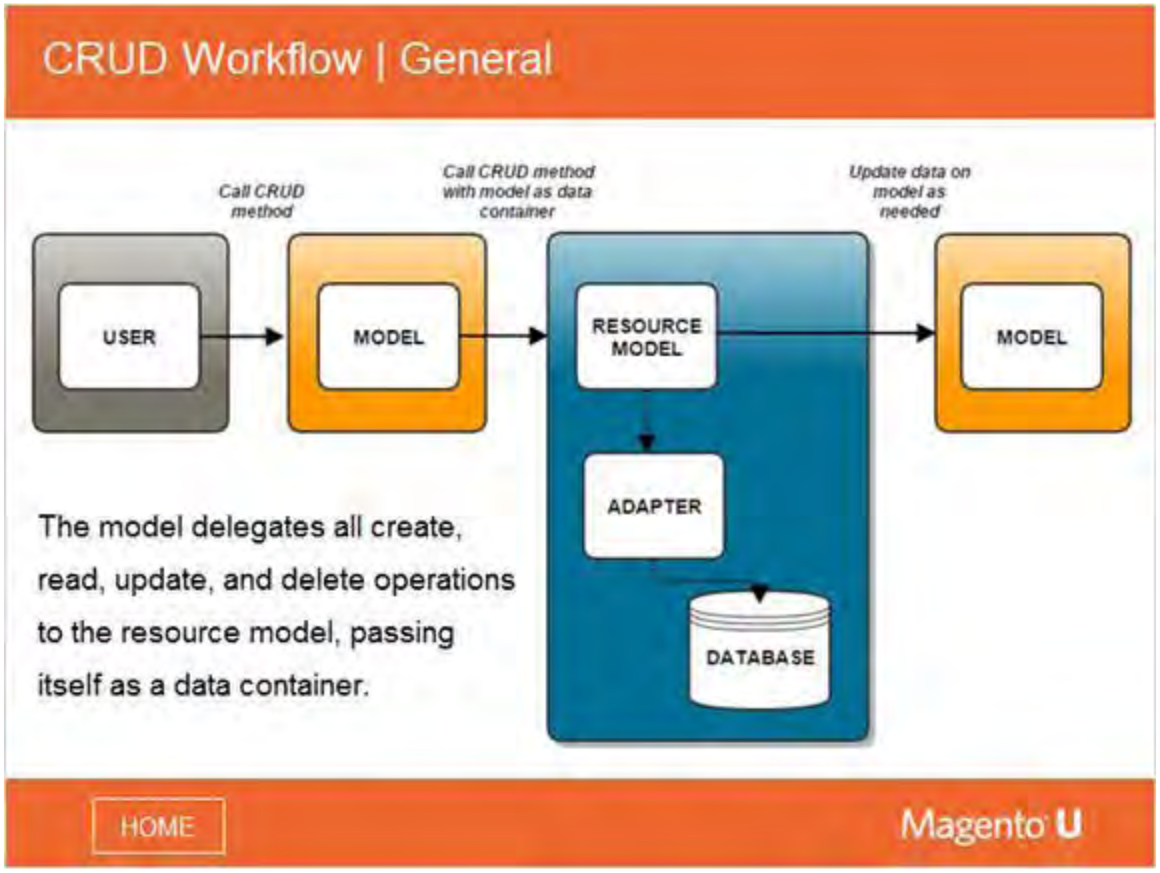
Resource Models encapsulate storage related logic. The model code delegates all storage layer related actions to the resource model. The resource model then uses the DB adapter to access the storage layer.

The benefits this approach offers are:

- Decoupling of the business logic and the storage layer.
- Decoupling of the storage schema from the DB driver implementation.

Resource Collections represent a list of models of a specific type. This approach is used for working with multiple records.

2.7 CRUD Workflow | General



CRUD Workflow | General

Call CRUD method
Call CRUD method with model as data container
Update data on model as needed
USER
MODEL
RESOURCE MODEL
MODEL
ADAPTER
DATABASE
HOME
Magento U

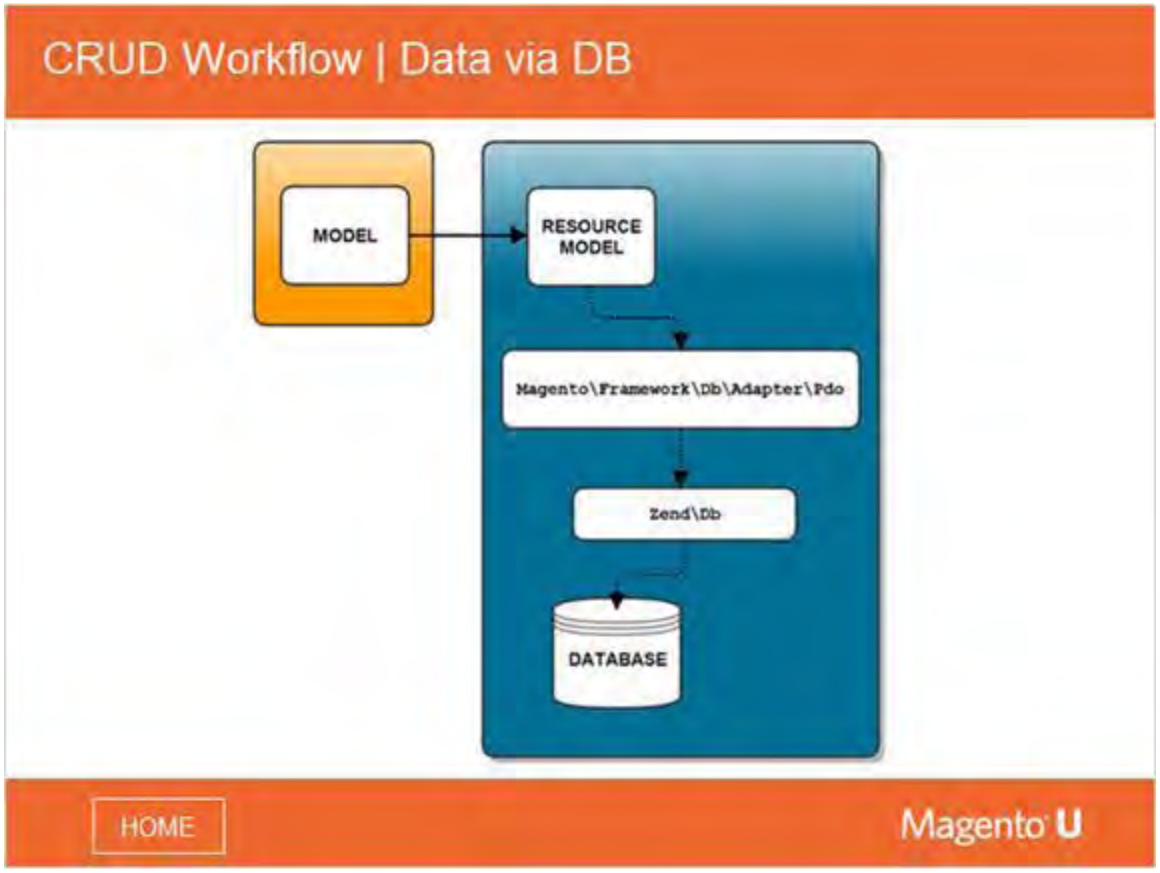
The model delegates all create, read, update, and delete operations to the resource model, passing itself as a data container.

Notes:

The model delegates all create, read, update, and delete operations to the resource model, passing itself as a data container.

The model has no direct access to the database, only to the resource models. So if you need some special data from the database, you would have to create the appropriate resource model.

2.8 CRUD Workflow | Data via DB

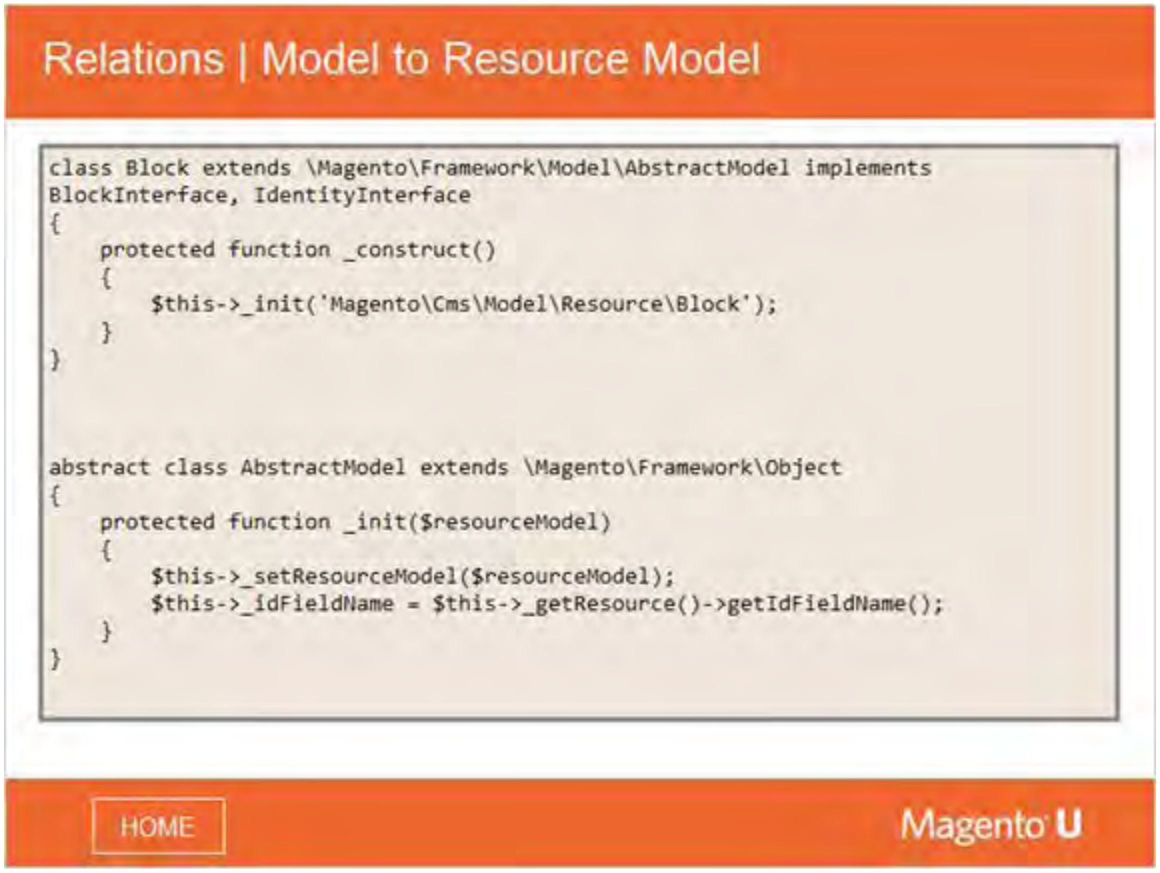


Notes:

There have been a couple of cosmetic changes, but this workflow, from model to resource model to database, is basically the same as in Magento 1.

It is implemented by the use of `\Magento\Framework\Db\Adapter\Pdo` and `Zend\Db`.

2.9 Relations | Model to Resource Model



Relations | Model to Resource Model

```
class Block extends \Magento\Framework\Model\AbstractModel implements
BlockInterface, IdentityInterface
{
    protected function _construct()
    {
        $this->_init('Magento\Cms\Model\Resource\Block');
    }
}


abstract class AbstractModel extends \Magento\Framework\Object
{
    protected function _init($resourceModel)
    {
        $this->_setResourceModel($resourceModel);
        $this->_idFieldName = $this->_getResource()->getIdFieldName();
    }
}
```

Notes:

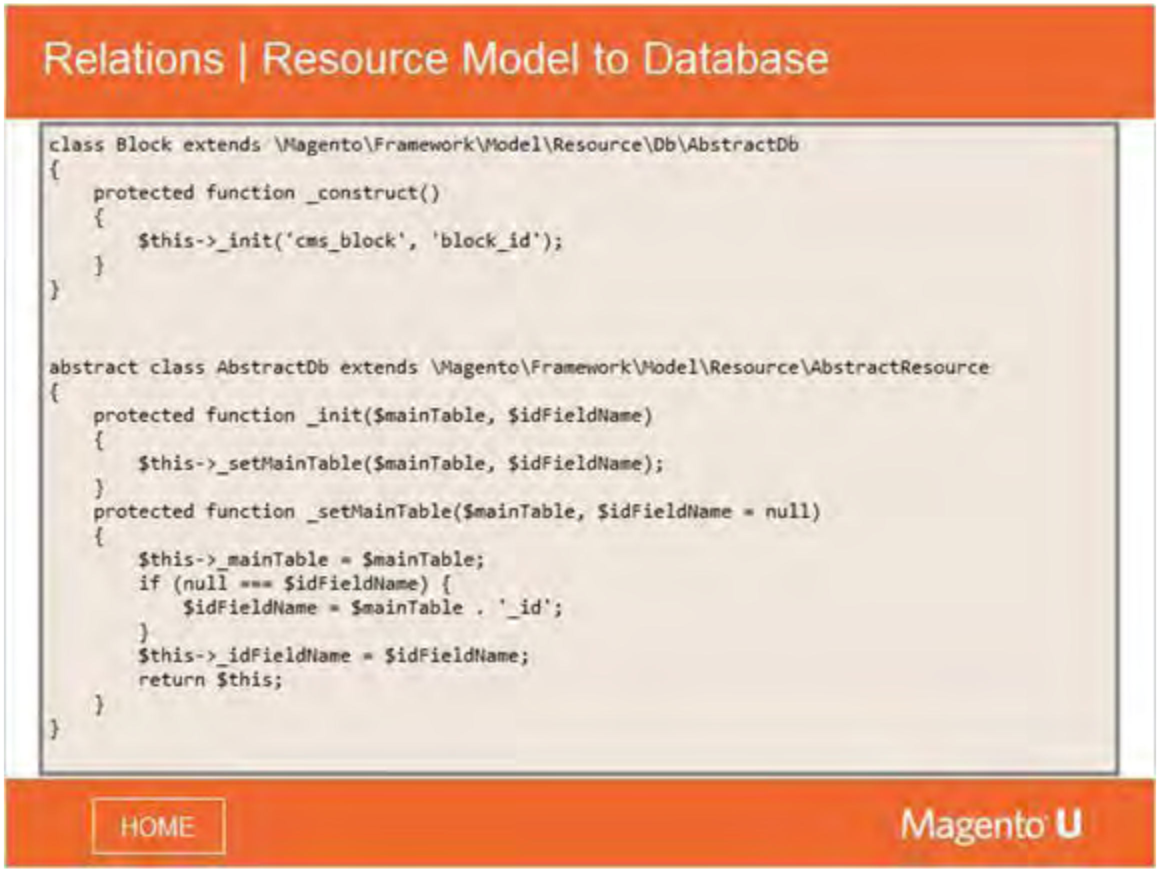
When creating a new model that interfaces with the database, the model needs to know which resource model to use. This is why the class name of the resource model is specified using the `_init()` method in the protected `_construct()` method.

To create the model, you just create a class and extend it from `\Magento\Framework\Model\Resource\AbstractResource`. With Magento 2, you no longer have to declare the model, resource model, define adapters, and more in the configuration. So the process is much simpler than in Magento 1.

So you just create the model class and extend it from the Magento framework `AbstractModel`.

Note that you call `_init` from a single underscore construct method, not the real double underscore constructor.

2.10 Relations | Resource Model to Database



Relations | Resource Model to Database

```
class Block extends \Magento\Framework\Model\Resource\Db\AbstractDb
{
    protected function _construct()
    {
        $this->_init('cms_block', 'block_id');
    }
}


abstract class AbstractDb extends \Magento\Framework\Model\Resource\AbstractResource
{
    protected function _init($mainTable, $idFieldName)
    {
        $this->_setMainTable($mainTable, $idFieldName);
    }
    protected function _setMainTable($mainTable, $idFieldName = null)
    {
        $this->_mainTable = $mainTable;
        if (null === $idFieldName) {
            $idFieldName = $mainTable . '_id';
        }
        $this->_idFieldName = $idFieldName;
        return $this;
    }
}
```

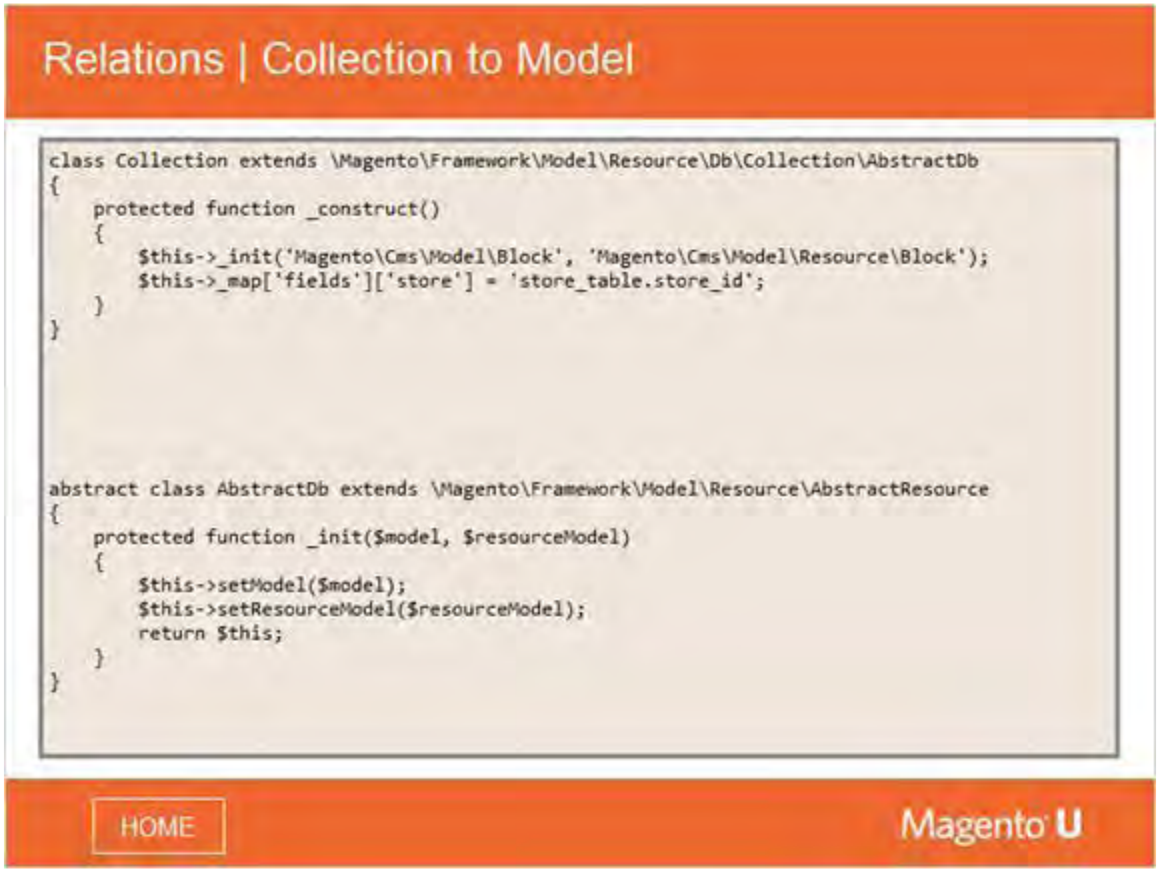
Notes:

A resource model extends `\Magento\Framework\Model\Resource\Db\AbstractDb`.

Again, note the single underscore construct method. Do not confuse it with the real constructor.

You need to define the table name and primary key field name for the resource model because it needs to know where to save the model state. They are specified by calling the `_init()` method in the `protected _construct()` method.

2.11 Relations | Collection to Model



```
class Collection extends \Magento\Framework\Model\Resource\Db\Collection\AbstractDb
{
    protected function _construct()
    {
        $this->_init('Magento\Cms\Model\Block', 'Magento\Cms\Model\Resource\Block');
        $this->_map['fields']['store'] = 'store_table.store_id';
    }
}
```

```
abstract class AbstractDb extends \Magento\Framework\Model\Resource\AbstractResource
{
    protected function _init($model, $resourceModel)
    {
        $this->setModel($model);
        $this->setResourceModel($resourceModel);
        return $this;
    }
}
```

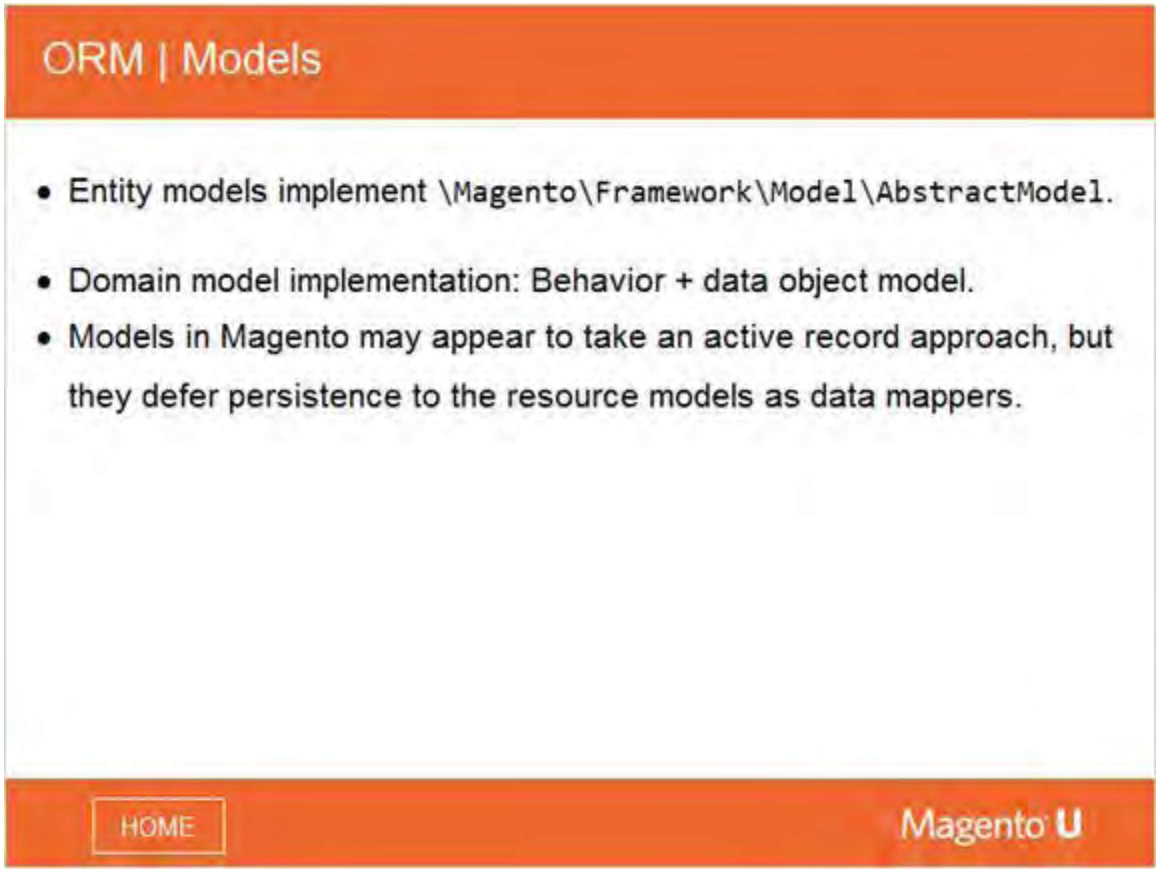
Notes:

The collection class extends `Magento\Framework\Model\Resource\Db\Collection\AbstractCollection.`

When creating a resource collection, you need to specify which model it corresponds to, so that it can instantiate the appropriate classes after loading a list of records. It also needs to know the matching resource model to be able to access the database. This is why the class names of the model and the resource model are specified using the `_init()` method in the `protected _construct()` method.

The remaining code you see on this slide is optional.

2.12 ORM | Models


ORM | Models

- Entity models implement `\Magento\Framework\Model\AbstractModel`.
- Domain model implementation: Behavior + data object model.
- Models in Magento may appear to take an active record approach, but they defer persistence to the resource models as data mappers.

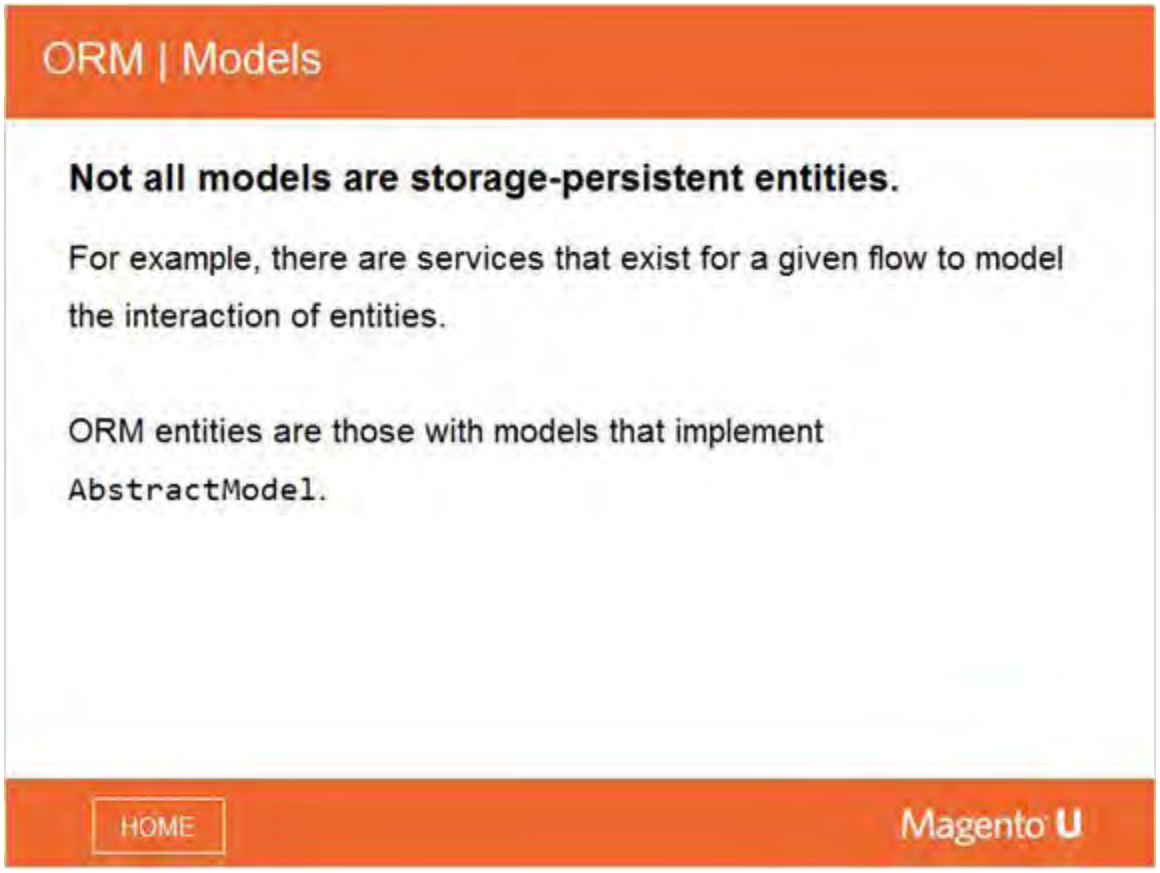
HOME
Magento U

Notes:

The base class for Magento models to extend is `\Magento\Framework\Model\AbstractModel`.

In Magento 2, each model that uses the storage layer has one corresponding resource model and one corresponding resource collection. It might appear that models take an active record approach because they expose, save, load, and delete methods, but in fact they defer persistence to the resource models as data mappers.

2.13 ORM | Models



ORM | Models

Not all models are storage-persistent entities.

For example, there are services that exist for a given flow to model the interaction of entities.

ORM entities are those with models that implement `AbstractModel`.

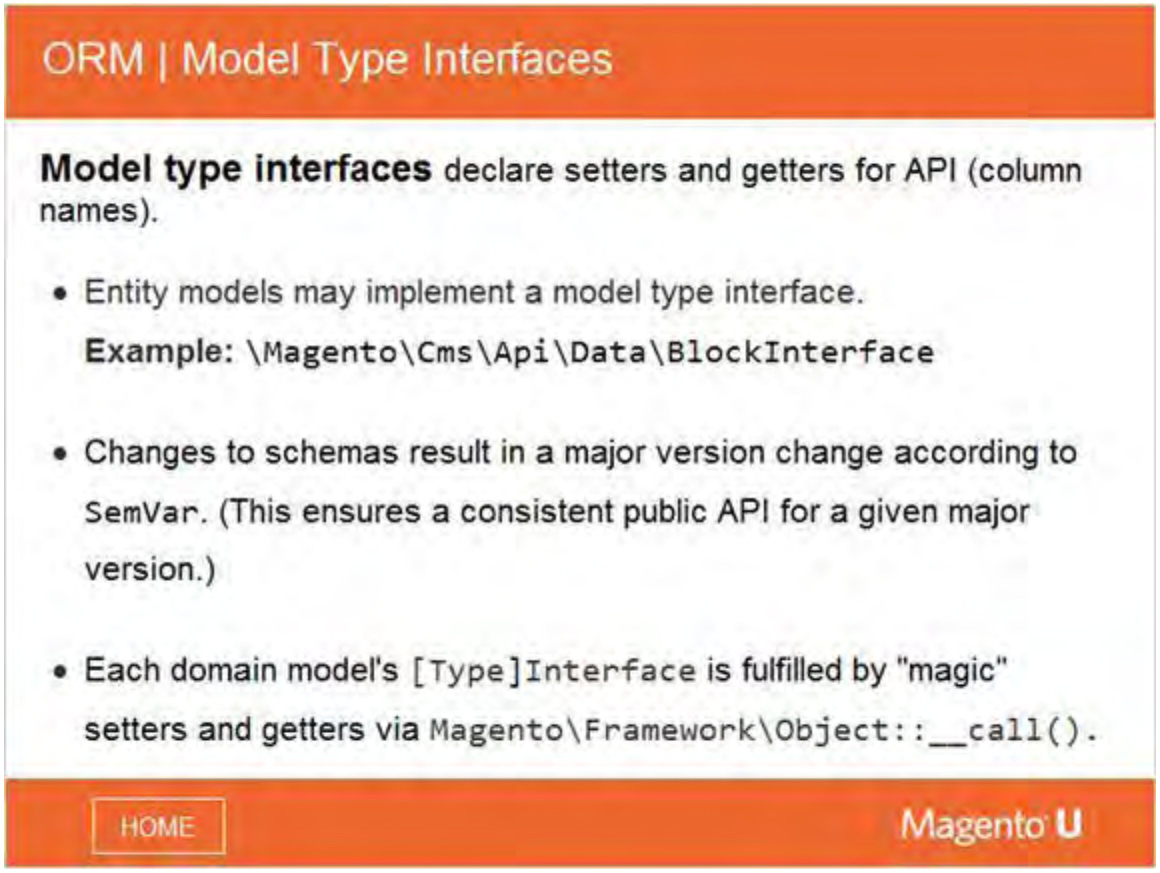
HOME
Magento U

Notes:

Models that are not storage persistent are somewhat of a legacy from Magento 1, and the fact that you had to place many classes under the Model directory.

In Magento 2, you can place classes under any folder. True models are those that represent entities and which extend `AbstractModel`.

2.14 ORM | Model Type Interfaces



ORM | Model Type Interfaces

Model type interfaces declare setters and getters for API (column names).

- Entity models may implement a model type interface.

  Example: `\Magento\Cms\Api\Data\BlockInterface`

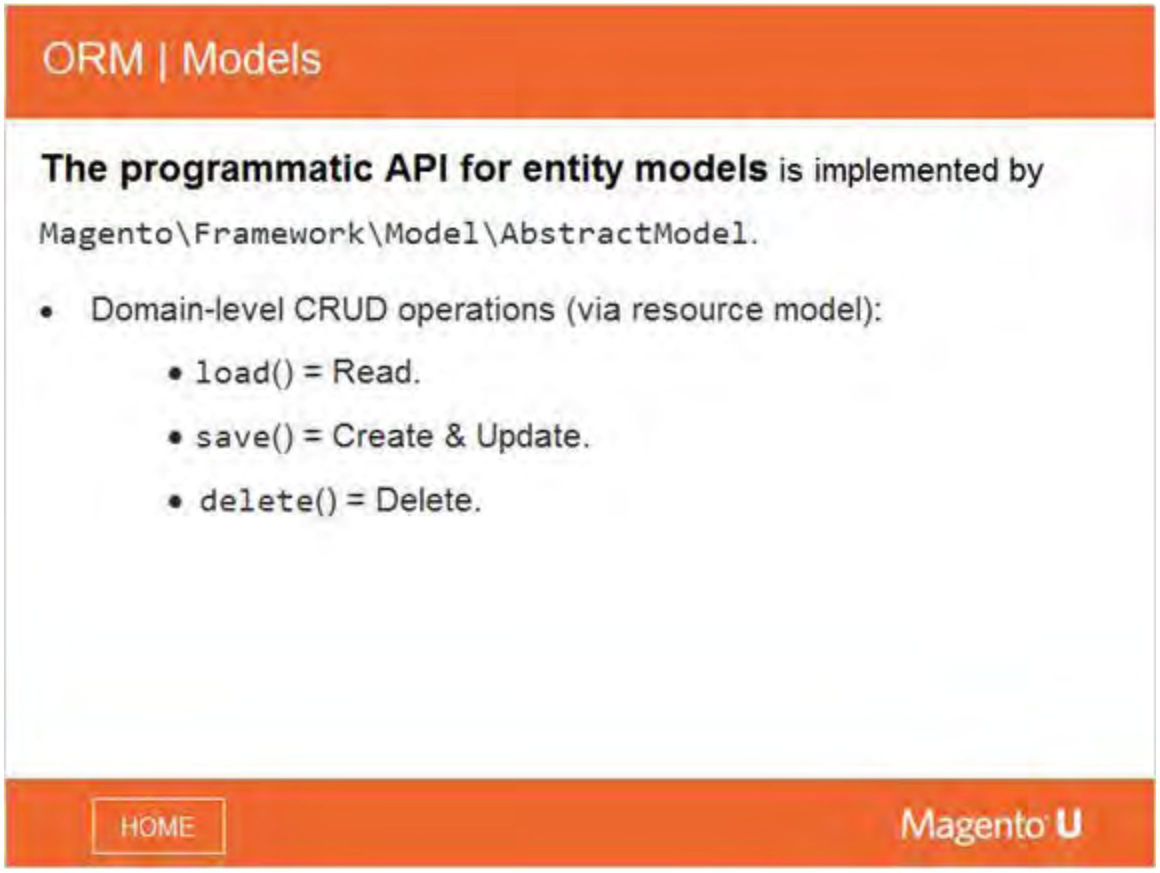
- Changes to schemas result in a major version change according to `SemVar`. (This ensures a consistent public API for a given major version.)

- Each domain model's `[Type]Interface` is fulfilled by "magic" setters and getters via `Magento\Framework\Object::__call()`.

Notes:

The API will be covered in more detail later in the course. For now, it is important just to note that you create an API layer in order to make it easier for developers to work with the application, without worrying about the workings behind it.

2.15 ORM | Models



ORM | Models

The programmatic API for entity models is implemented by `Magento\Framework\Model\AbstractModel`.

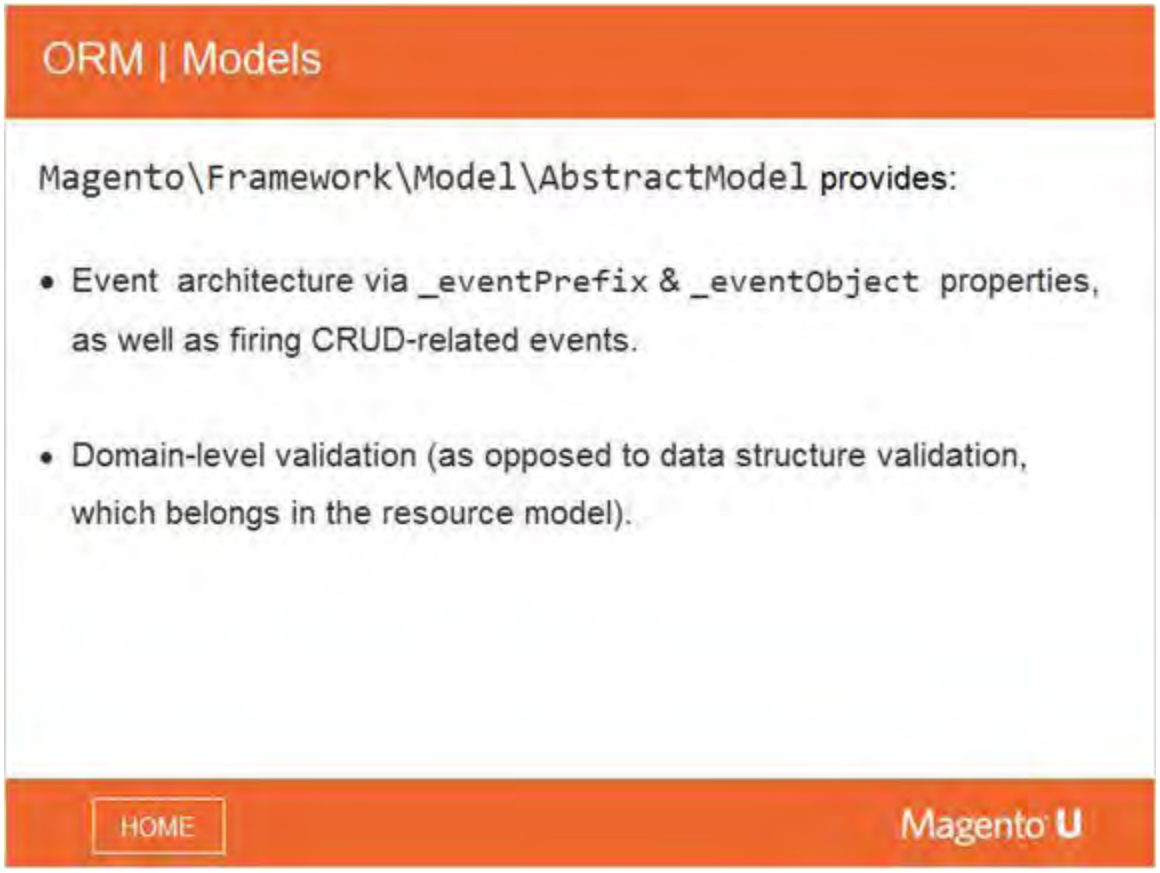
- Domain-level CRUD operations (via resource model):
  - `load()` = Read.
  - `save()` = Create & Update.
  - `delete()` = Delete.

HOME
Magento U

Notes:

The methods for CRUD operations have not changed in Magento 2 – you still use `load(), save()`, and `delete()`.

2.16 ORM | Models



ORM | Models

`Magento\Framework\Model\AbstractModel` provides:

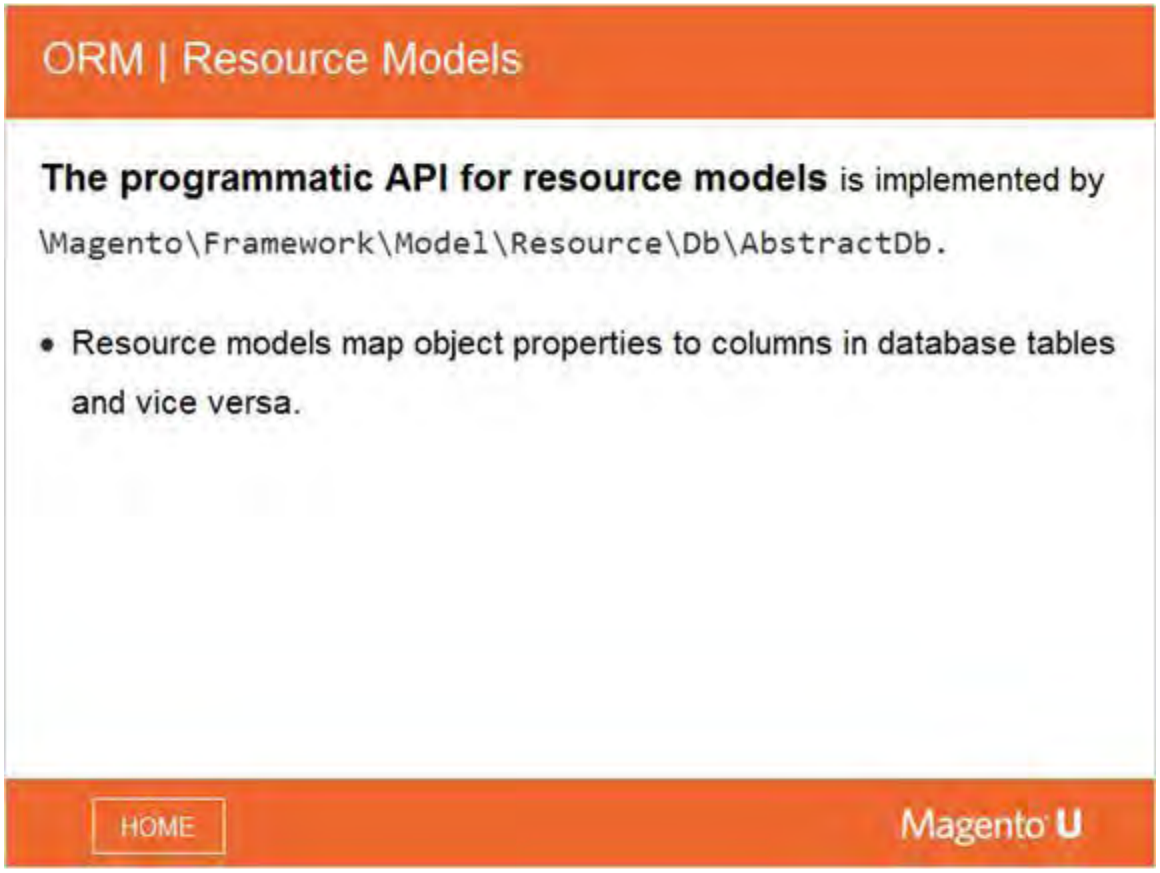
- Event architecture via `_eventPrefix` & `_eventObject` properties, as well as firing CRUD-related events.
- Domain-level validation (as opposed to data structure validation, which belongs in the resource model).

HOME
Magento U

Notes:

Just as in Magento 1, Magento 2 models inherit automatically fired events, which use the `_eventPrefix` and `_eventObject` properties to customize the event name to the specific model type being operated on.

2.17 ORM | Resource Models



ORM | Resource Models

The programmatic API for resource models is implemented by `\Magento\Framework\Model\Resource\Db\AbstractDb.`

- Resource models map object properties to columns in database tables and vice versa.

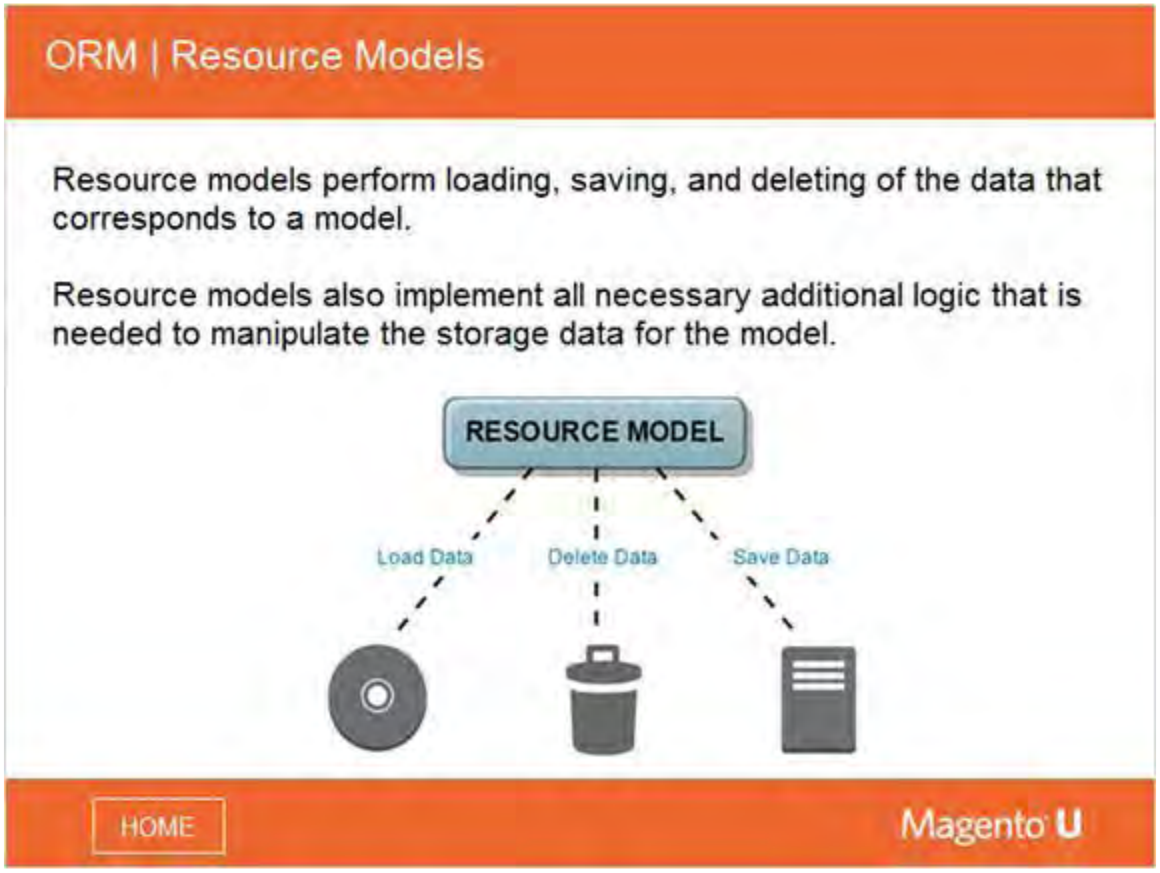
HOME
Magento U

Notes:

Again, we will discuss APIs in more detail in the next unit.

The purpose of this slide is to point out that the API you program against for resource models is implemented by \Magento`\Framework\Model\Resource\Db\AbstractDb.`

2.18 ORM | Resource Models



ORM | Resource Models

Resource models perform loading, saving, and deleting of the data that corresponds to a model.

Resource models also implement all necessary additional logic that is needed to manipulate the storage data for the model.

RESOURCE MODEL
Load Data
Delete Data
Save Data
HOME
Magento U

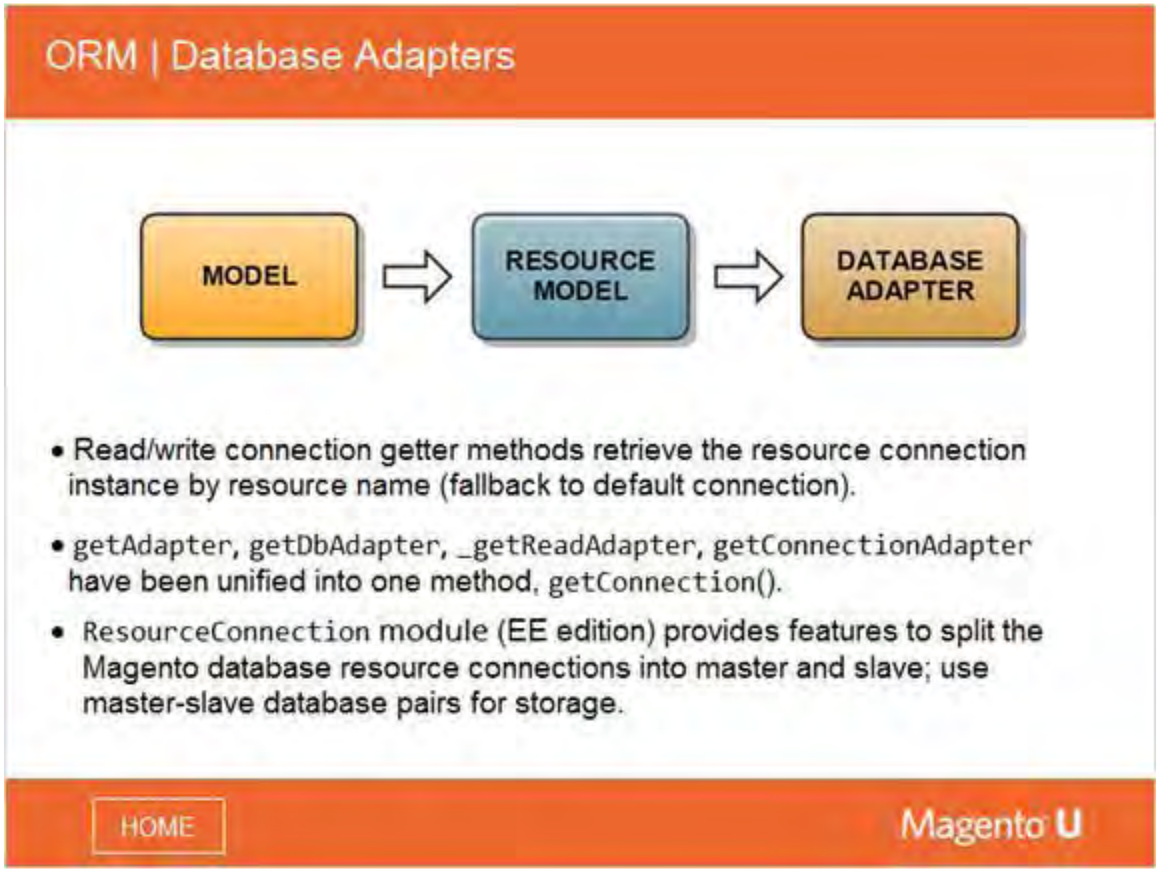
Notes:

We can consider resource models to be a service layer between models and the database storage.

They help us to work with the database; that is, to get, save, and update data. Basically, any model can be connected to a specific resource model that implements access to the database storage layer.

This is unchanged from Magento 1.

2.19 ORM | Database Adapters



ORM | Database Adapters

MODEL
RESOURCE MODEL
DATABASE ADAPTER

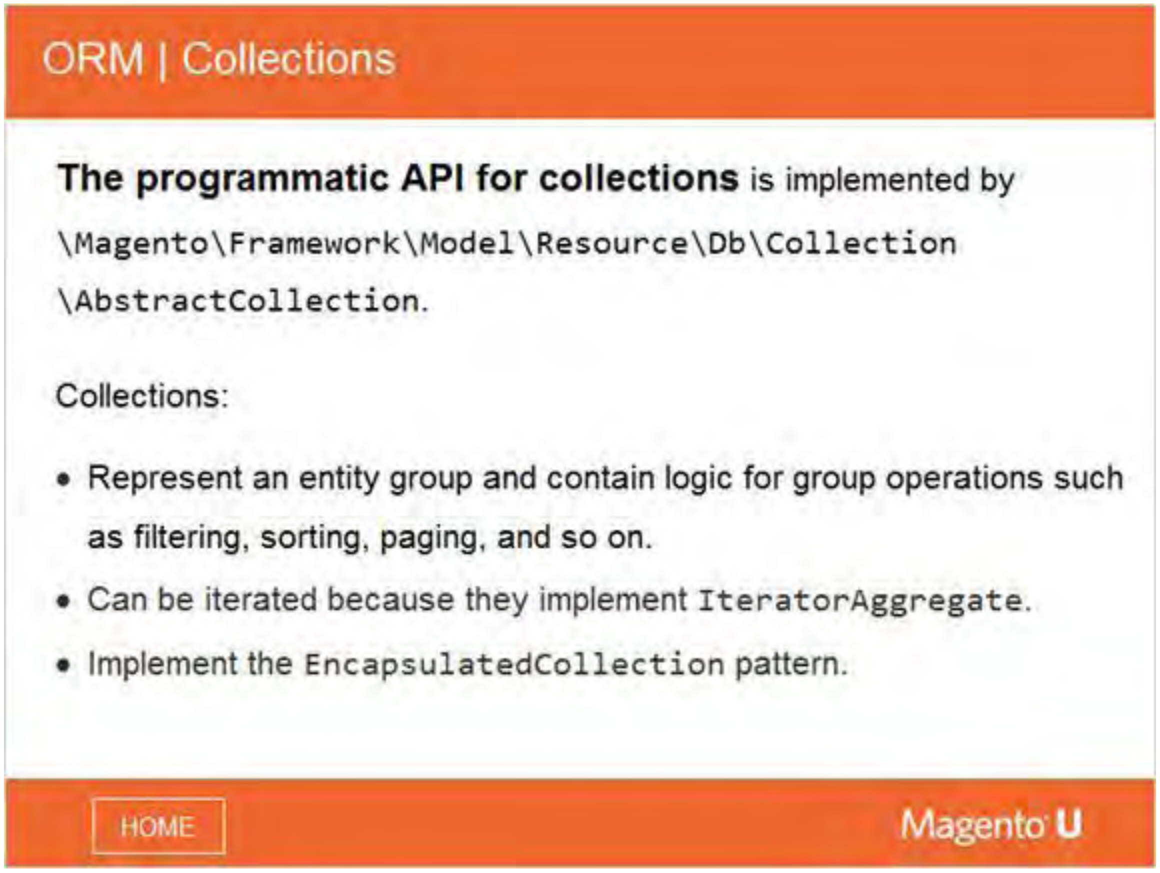
- Read/write connection getter methods retrieve the resource connection instance by resource name (fallback to default connection).
- `getAdapter`, `getDbAdapter`, `_getReadAdapter`, `getConnectionAdapter` have been unified into one method, `getConnection()`.
- `ResourceConnection` module (EE edition) provides features to split the Magento database resource connections into master and slave; use master-slave database pairs for storage.

Notes:

This diagram reminds you how a model, resource model, and database adapter are connected.

- All read/write connection getter methods retrieve the resource connection instance by resource name with a fallback to the default connection.
- The Magento 1 getAdapter, getDbAdapter, _getReadAdapter, getConnectionAdapter that retrieved database connection instances have been unified to one method: getConnection(). Old variables with the adapter keyword in their name are renamed to connection.
- The ResourceConnection module from the Enterprise Edition provides features to use a master-slave database replication setup for load balancing purposes.

2.20 ORM | Collections



ORM | Collections

The programmatic API for collections is implemented by `\Magento\Framework\Model\Resource\Db\Collection\AbstractCollection`.

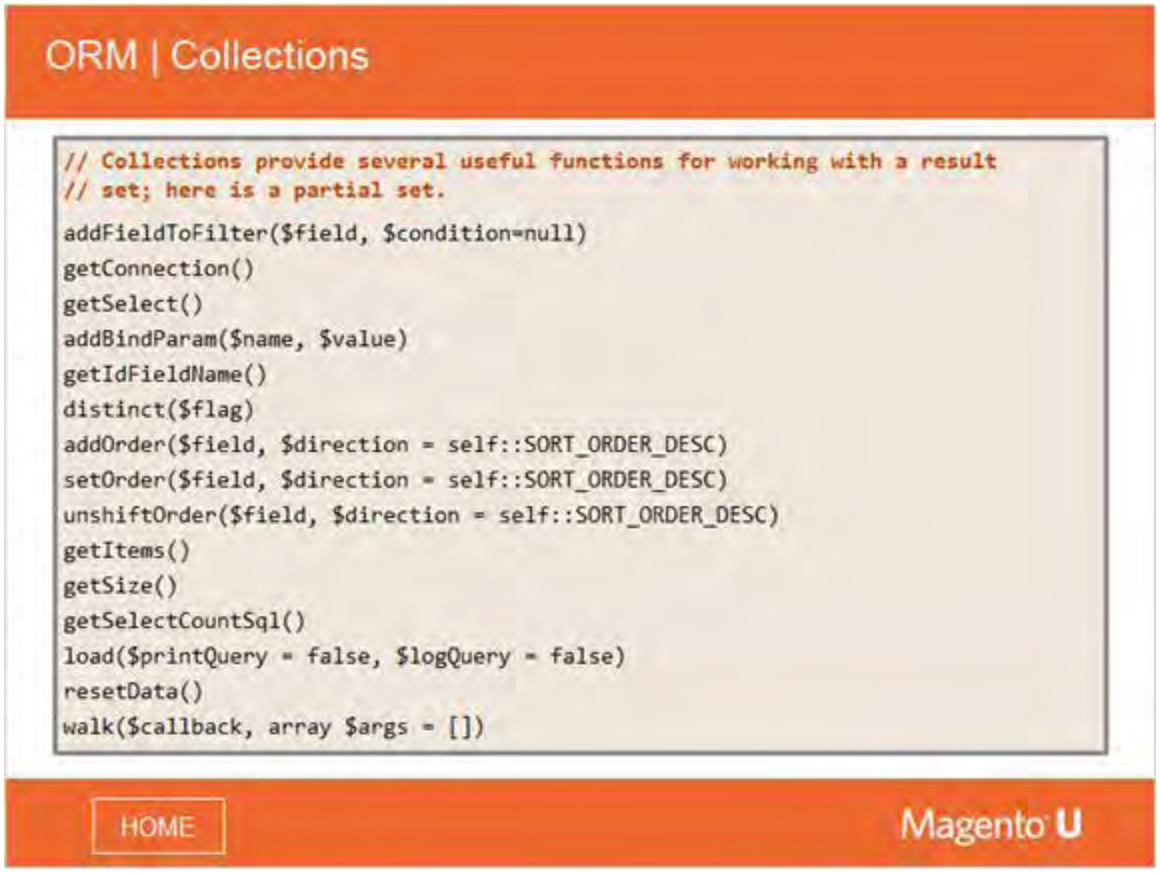
Collections:

- Represent an entity group and contain logic for group operations such as filtering, sorting, paging, and so on.
- Can be iterated because they implement `IteratorAggregate`.
- Implement the `EncapsulatedCollection` pattern.

Notes:

The API you program against for collections is implemented by \Magento`\Framework\Model\Resource\Db\Collection\AbstractCollection.`

2.21 ORM | Collections


ORM | Collections

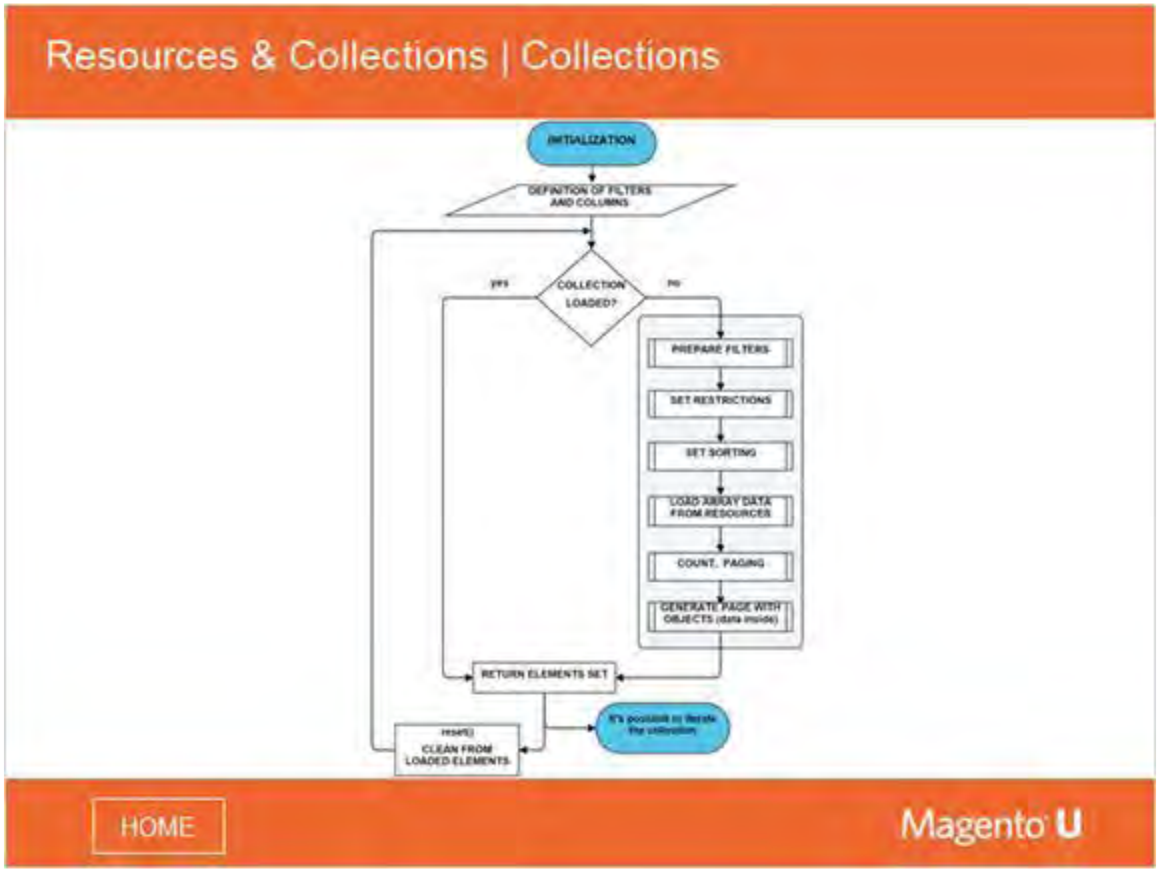
```
// Collections provide several useful functions for working with a result
// set; here is a partial set.
addFieldToFilter($field, $condition=null)
getConnection()
getSelect()
addBindParam($name, $value)
getIdFieldName()
distinct($flag)
addOrder($field, $direction = self::SORT_ORDER_DESC)
setOrder($field, $direction = self::SORT_ORDER_DESC)
unshiftOrder($field, $direction = self::SORT_ORDER_DESC)
getItems()
getSize()
getSelectCountSql()
load($printQuery = false, $logQuery = false)
resetData()
walk($callback, array $args = [])
```

Notes:

Collections provide several useful functions for defining and working with a result set.

As you can see, the set of methods exposed by collections has not changed much from Magento 1.

2.22 Resources & Collections | Collections


Resources & Collections | Collections
INITIALIZATION
DEFINITION OF FILTERS AND COLUMNS
COLLECTION LOADED?
yes
no
PREPARE FILTERS
SET RESTRICTIONS
SET SORTING
LOAD ARRAY DATA FROM RESOURCES
COUNT, PAGING
GENERATE PAGE WITH OBJECTS (data inside)
RETURN ELEMENTS SET
It's possible to iterate the collection
reset()
CLEAN FROM LOADED ELEMENTS
HOME
Magento U

Notes:

This diagram shows how a collection is loaded. It has not changed much since Magento 1.

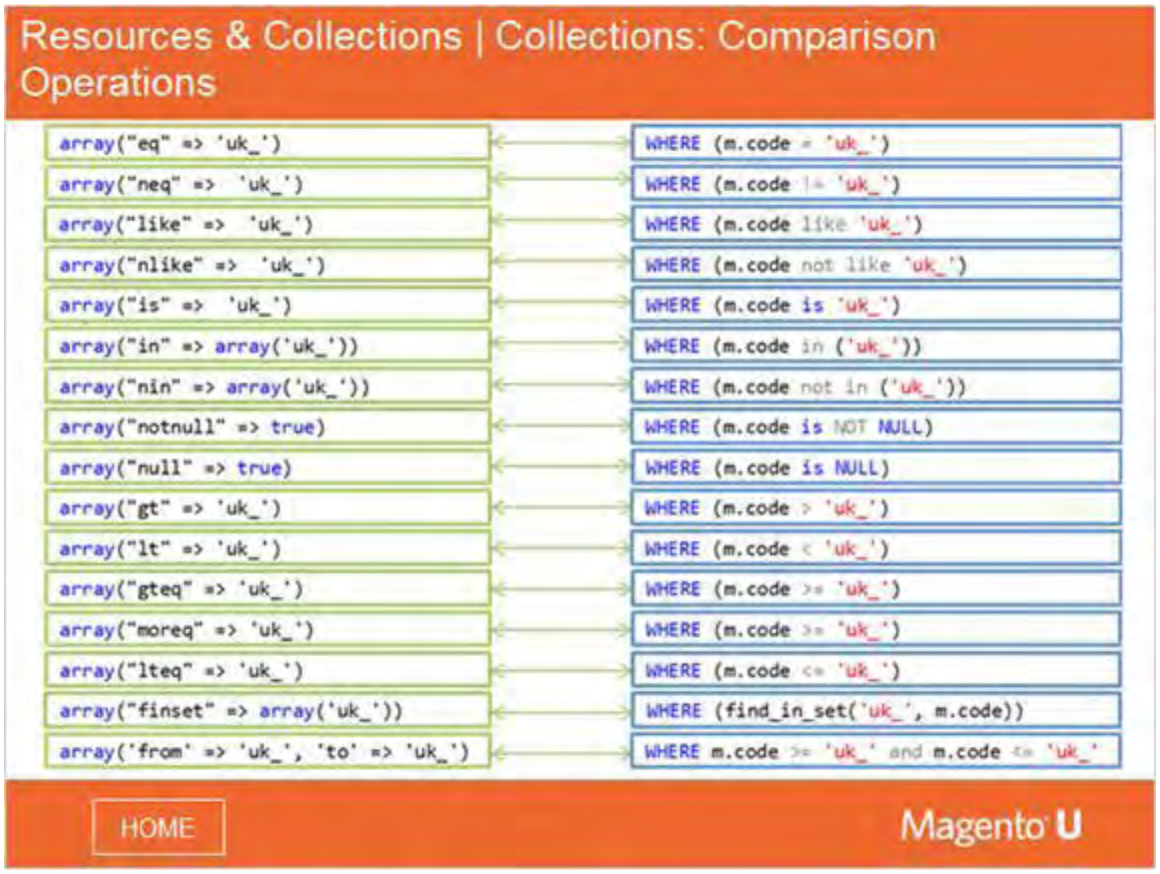
A resource collection is needed to create a set of model instances and operate on them. Resource collections lazy load the record set they represent. The first time the item list is accessed, a collection will load automatically. After that a collection will not load again, even if the `load()` method is explicitly called.

Collections are very close to the database layer. Their task is to retrieve a set of rows from the database, then iterate over them, and for each row, instantiate the matching model class.

It solves these main issues:

- Provides a container for storing collections of objects.
- Prevents unnecessary loading of data.
- Stores all objects during a session.
- Provides an interface for filtering and sorting entities of a specific kind.

2.23 Resources & Collections | Collections: Comparison Operations



Resources & Collections | Collections: Comparison Operations

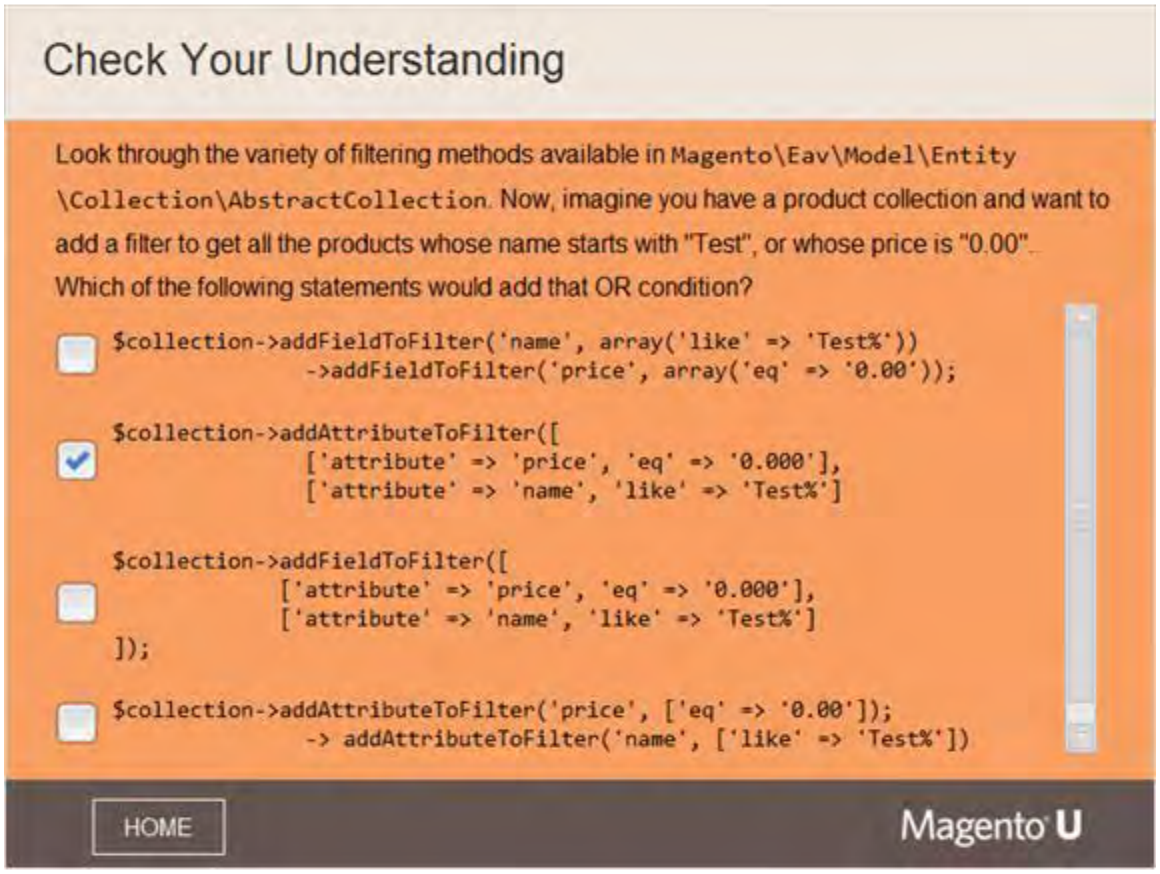
`array("eq" => 'uk_')`	`WHERE (m.code = 'uk_')`
`array("neq" => 'uk_')`	`WHERE (m.code != 'uk_')`
`array("like" => 'uk_')`	`WHERE (m.code like 'uk_')`
`array("nlike" => 'uk_')`	`WHERE (m.code not like 'uk_')`
`array("is" => 'uk_')`	`WHERE (m.code is 'uk_')`
`array("in" => array('uk_'))`	`WHERE (m.code in ('uk_'))`
`array("nin" => array('uk_'))`	`WHERE (m.code not in ('uk_'))`
`array("notnull" => true)`	`WHERE (m.code is NOT NULL)`
`array("null" => true)`	`WHERE (m.code is NULL)`
`array("gt" => 'uk_')`	`WHERE (m.code > 'uk_')`
`array("lt" => 'uk_')`	`WHERE (m.code < 'uk_')`
`array("gteq" => 'uk_')`	`WHERE (m.code >= 'uk_')`
`array("moreq" => 'uk_')`	`WHERE (m.code >= 'uk_')`
`array("lteq" => 'uk_')`	`WHERE (m.code <= 'uk_')`
`array("finset" => array('uk_'))`	`WHERE (find_in_set('uk_', m.code))`
`array('from' => 'uk_', 'to' => 'uk_')`	`WHERE m.code >= 'uk_' and m.code <= 'uk_'`

Notes:

This is a good reference slide, to remind you how collection filter operators can be specified and how they map to SQL `WHERE` conditions.

2.24 Check Your Understanding

(Multiple Response, 10 points, 1 attempt permitted)



Check Your Understanding

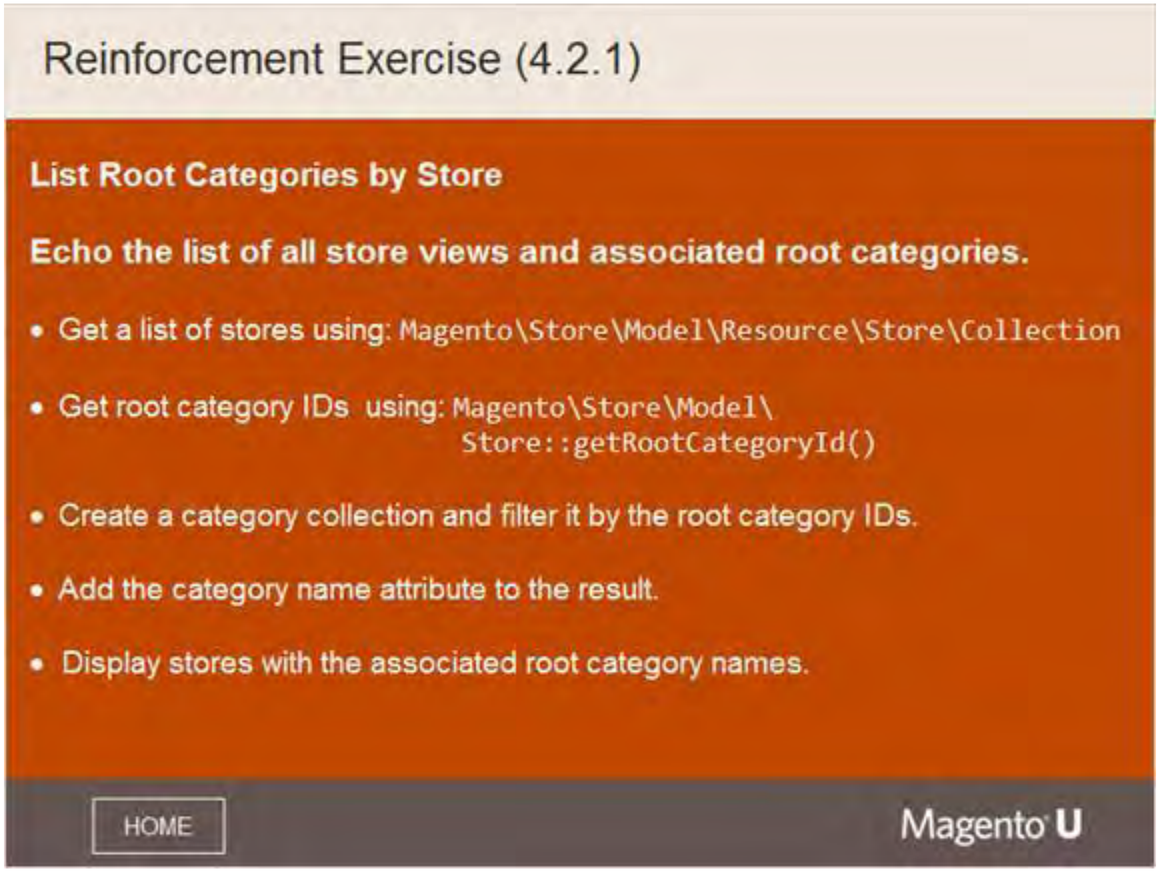
Look through the variety of filtering methods available in `Magento\Eav\Model\Entity\Collection\AbstractCollection`. Now, imagine you have a product collection and want to add a filter to get all the products whose name starts with "Test", or whose price is "0.00". Which of the following statements would add that OR condition?

- [ ]
```
$collection->addFieldToFilter('name', array('like' => 'Test%'))
           ->addFieldToFilter('price', array('eq' => '0.00'));
```
- [✓]
```
$collection->addAttributeToFilter([
           ['attribute' => 'price', 'eq' => '0.000'],
           ['attribute' => 'name', 'like' => 'Test%']
```
- [ ]
```
$collection->addFieldToFilter([
          ['attribute' => 'price', 'eq' => '0.000'],
          ['attribute' => 'name', 'like' => 'Test%']
]);
```
- [ ]
```
$collection->addAttributeToFilter('price', ['eq' => '0.00']);
           -> addAttributeToFilter('name', ['like' => 'Test%'])
```

Correct	Choice
	$collection->addFieldToFilter('name', array('like' => 'Test%')) ->addFieldToFilter('price', array('eq' => '0.00'));
X	$collection->addAttributeToFilter([['attribute' => 'price', 'eq' => '0.000'], ['attribute' => 'name', 'like' => 'Test%']
	$collection->addFieldToFilter([['attribute' => 'price', 'eq' => '0.000'], ['attribute' => 'name', 'like' => 'Test%']]);
	$collection->addAttributeToFilter('price', ['eq' => '0.00']); -> addAttributeToFilter('name', ['like' => 'Test%'])

The correct answer is B.

2.25 Reinforcement Exercise (4.2.1)



Reinforcement Exercise (4.2.1)

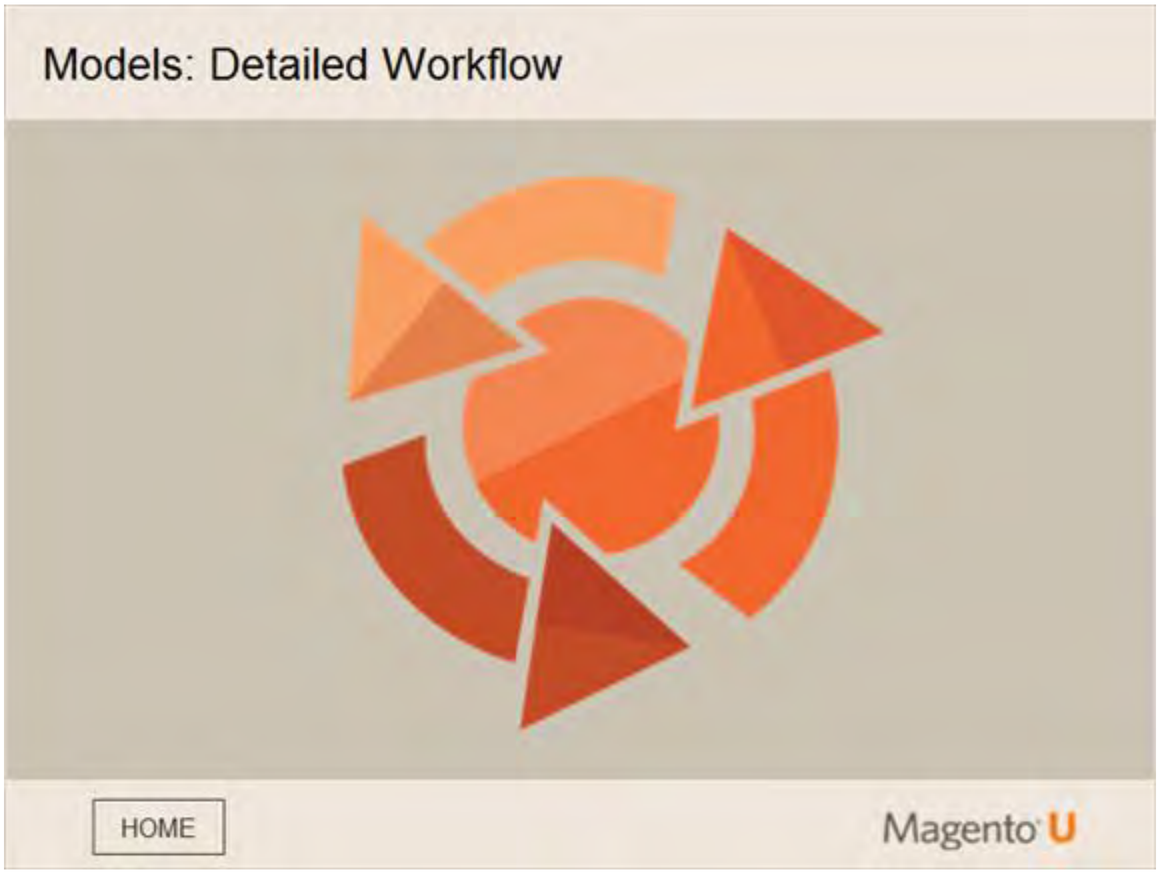
List Root Categories by Store

Echo the list of all store views and associated root categories.

- Get a list of stores using: `Magento\Store\Model\Resource\Store\Collection`
- Get root category IDs using: `Magento\Store\Model\Store::getRootCategoryId()`
- Create a category collection and filter it by the root category IDs.
- Add the category name attribute to the result.
- Display stores with the associated root category names.

3. Models Detailed Workflow

3.1 Models: Detailed Workflow

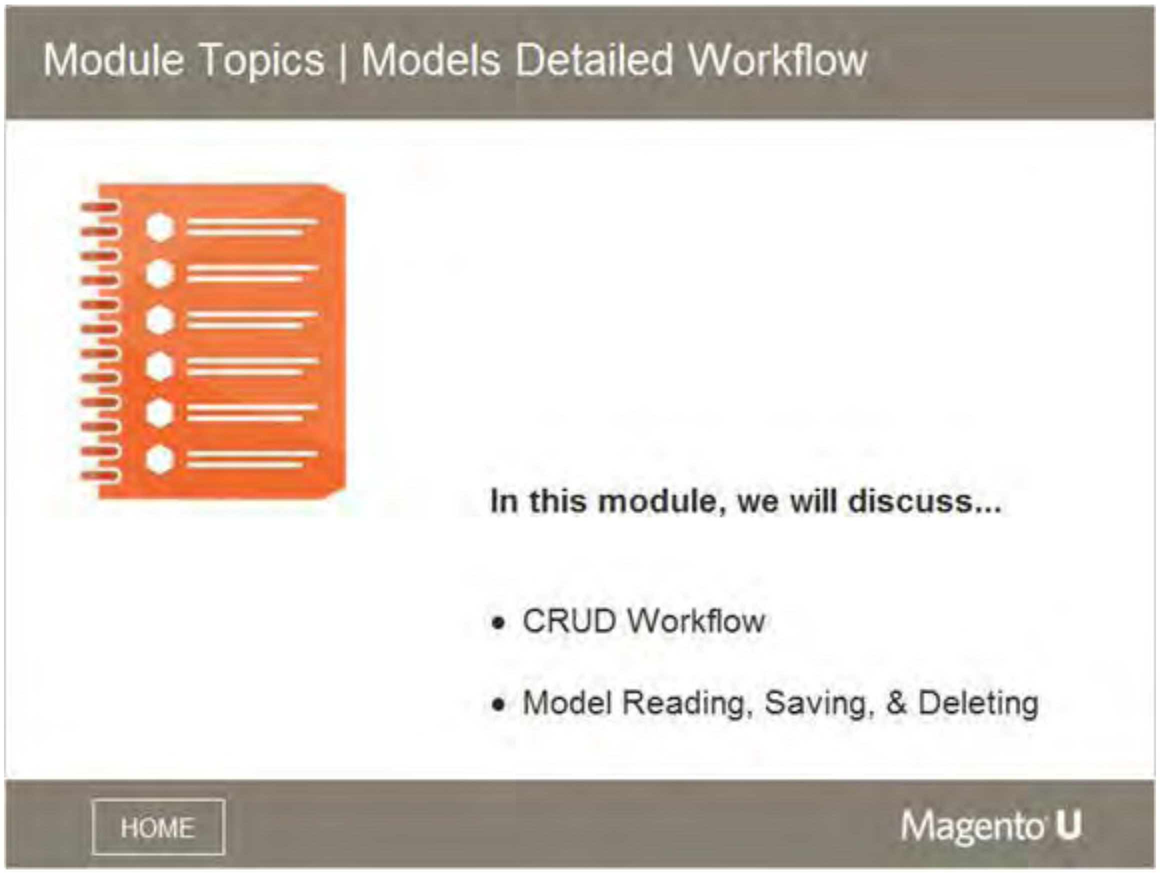


Models: Detailed Workflow

Notes:

We will now examine models and their related workflow in more detail.

3.2 Module Topics | Models Detailed Workflow



Module Topics | Models Detailed Workflow

In this module, we will discuss...

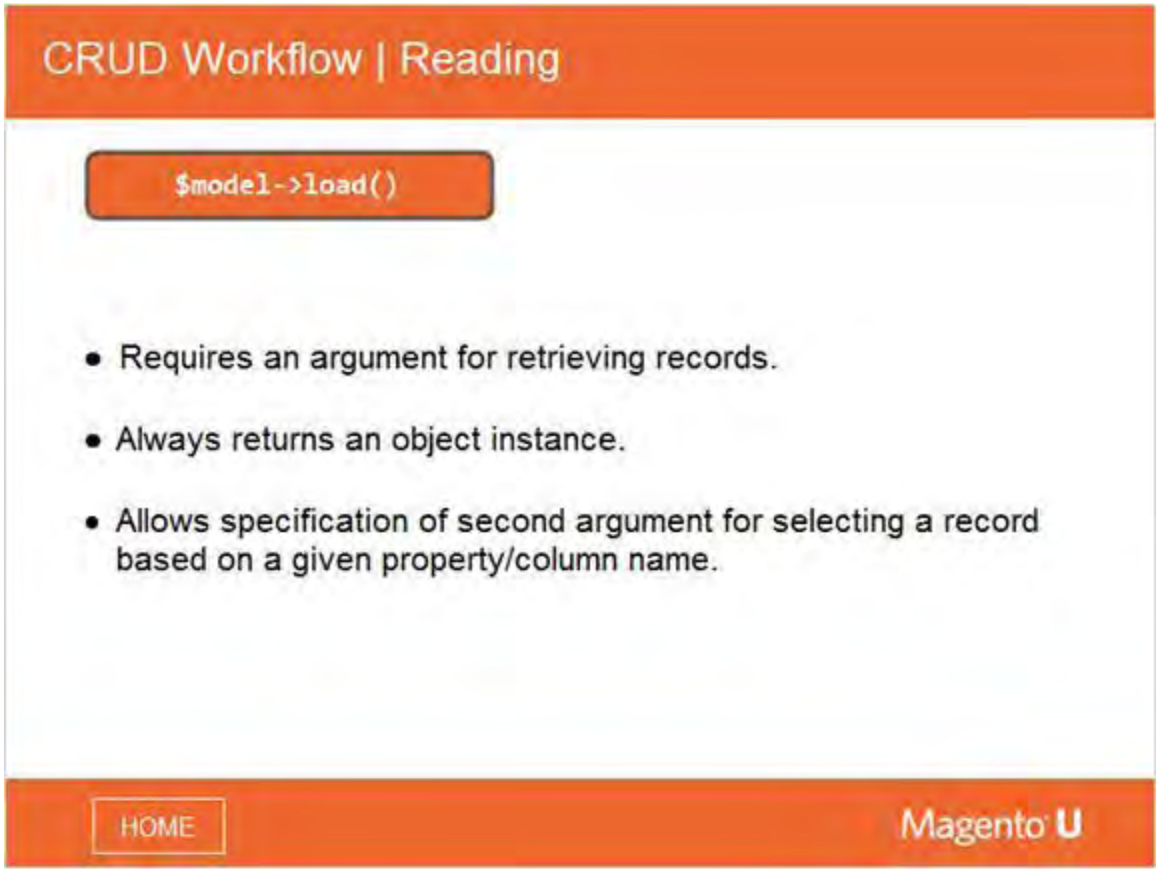
- CRUD Workflow
- Model Reading, Saving, & Deleting

Notes:

In this module, we will discuss:

- The CRUD workflow, which consists of...
- Model reading, saving, and deleting

3.3 CRUD Workflow | Reading



CRUD Workflow | Reading

`$model->load()`

- Requires an argument for retrieving records.
- Always returns an object instance.
- Allows specification of second argument for selecting a record based on a given property/column name.

HOME
Magento U

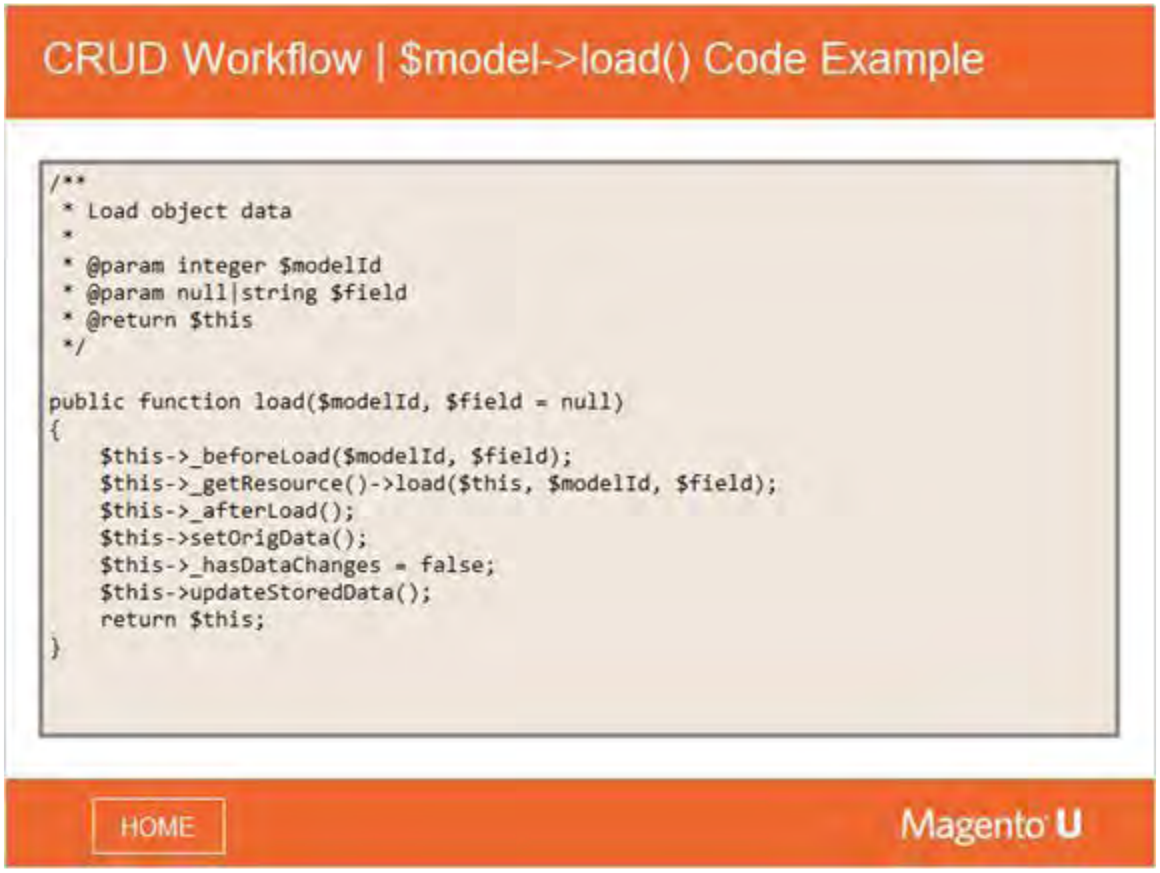
Notes:

How do the different model functions like `load()` work in Magento 2? The same way as in Magento 1. The key is to get the loaded data from the resource model.

When passed a single parameter, the `load()` method will return a record whose id field (which is set in the model's resource) matches that passed in.

The second parameter may be used to specify a field that matched against the first value instead of the id field. However, categories and products can be loaded using `loadByAttribute($attribute, $value, $additionalAttributes = '*')`, a method which you may find helpful.

3.4 CRUD Workflow | $model->load() Code Example



CRUD Workflow | $model->load() Code Example

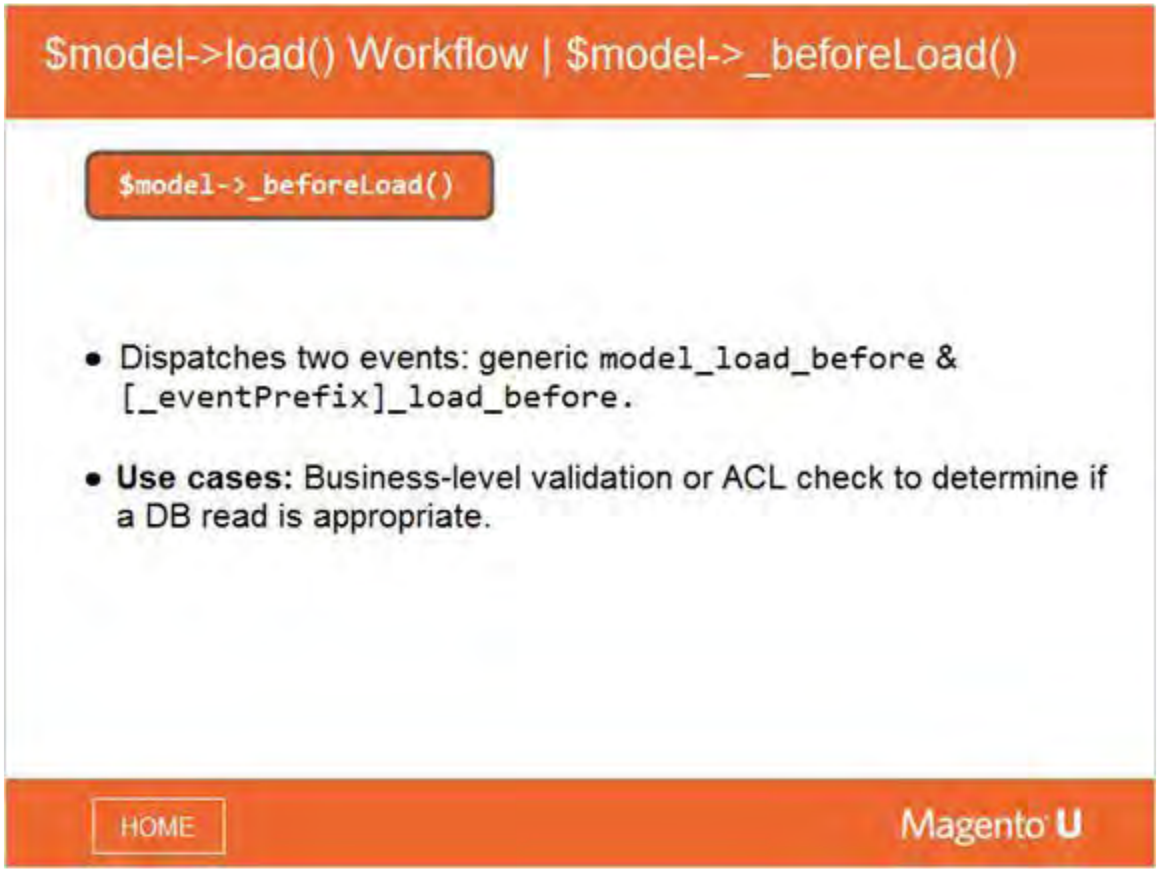
```
/**
 * Load object data
 *
 * @param integer $modelId
 * @param null|string $field
 * @return $this
 */

public function load($modelId, $field = null)
{
    $this->_beforeLoad($modelId, $field);
    $this->_getResource()->load($this, $modelId, $field);
    $this->_afterLoad();
    $this->setOrigData();
    $this->_hasDataChanges = false;
    $this->updateStoredData();
    return $this;
}
```

Notes:

Note that the `Magento\Framework\Model\AbstractModel::load()` method provides you with functionality that can impact the loading process. This is possible due to the `_beforeLoad` and `_afterLoad` methods.

The method `setOrigData()` simply copies the model data into the protected property `_origData` in order to track changes.

3.5 $model->load() Workflow | $model->_beforeLoad()



$model->load() Workflow | $model->_beforeLoad()

`$model->_beforeLoad()`

- Dispatches two events: generic `model_load_before` & `[_eventPrefix]_load_before`.
- **Use cases:** Business-level validation or ACL check to determine if a DB read is appropriate.

HOME
Magento U

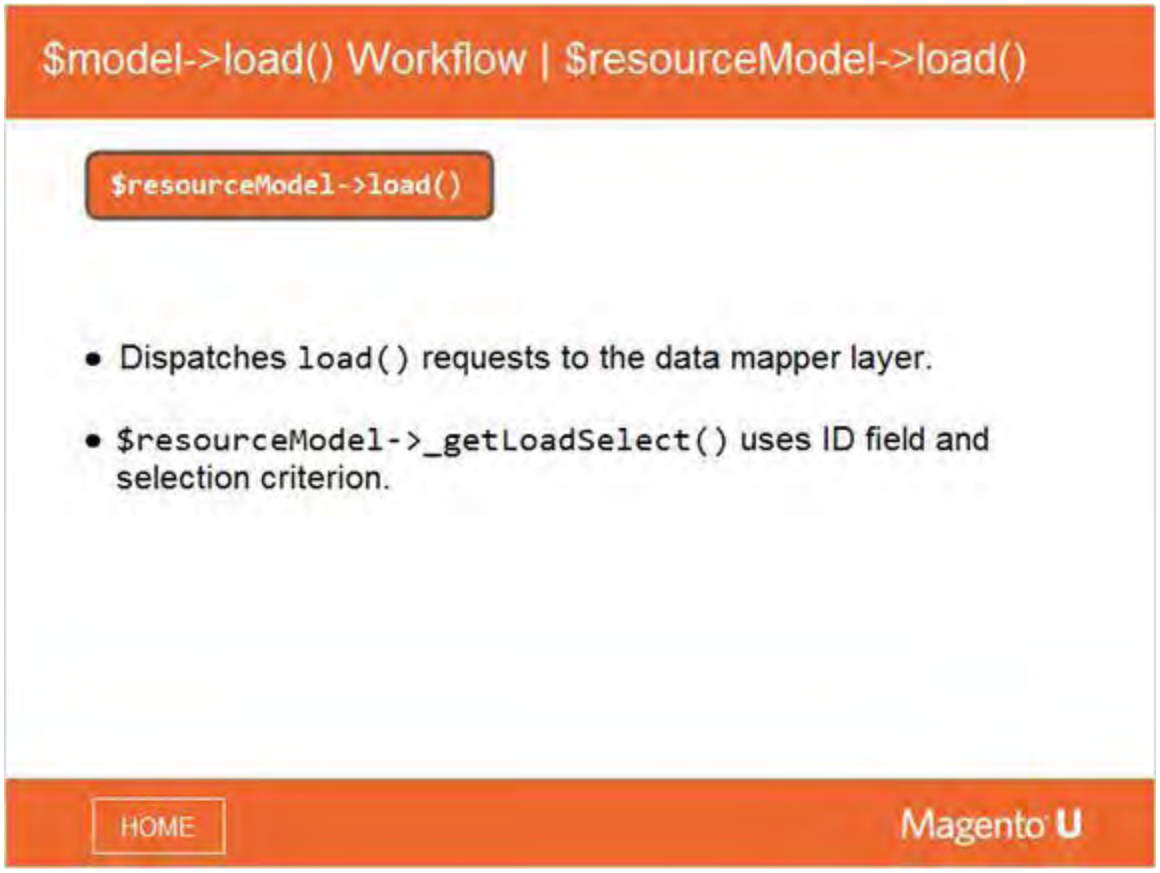
Notes:

The method `_beforeLoad` is used for processing an object directly before loading the data.

Note that models are passed in a `load_before` event dispatch, which then in turn passes them on to the listeners.

For more detail, look at `Magento\Framework\Model\AbstractModel:: _beforeLoad($modelId, $field = null)`.

3.6 $model->load() Workflow | $resourceModel->load()



$model->load() Workflow | $resourceModel->load()

`$resourceModel->load()`

- Dispatches `load()` requests to the data mapper layer.
- `$resourceModel->_getLoadSelect()` uses ID field and selection criterion.

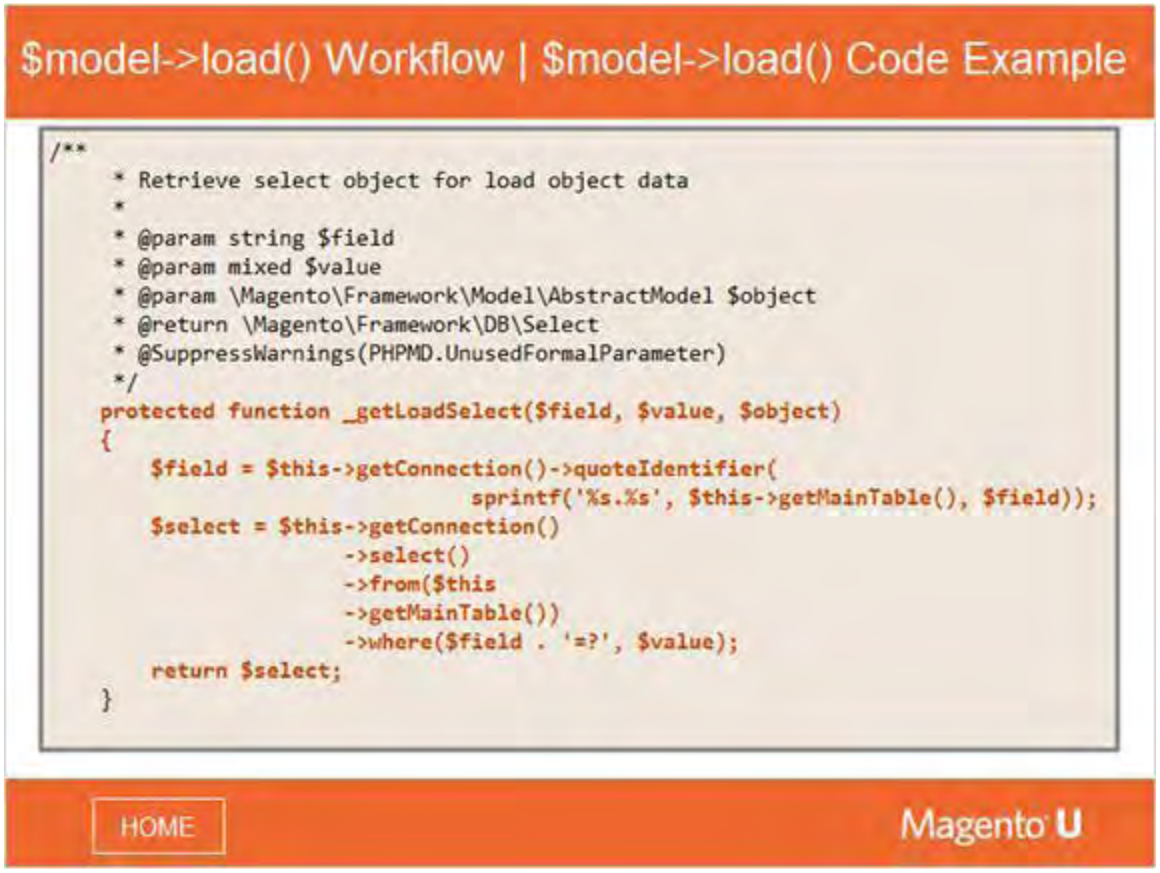
HOME
Magento U

Notes:

This method is provided by the `Magento\Framework\Model\Resource\Db\AbstractDb` class and requires a model object and a `$value` in order to load data from the database.

The optional parameter `$field` can be omitted, which is common. In this case, the default `$idFieldName` will be used.

3.7 $model->load() Workflow | $model->load() Code Example



$model->load() Workflow | $model->load() Code Example

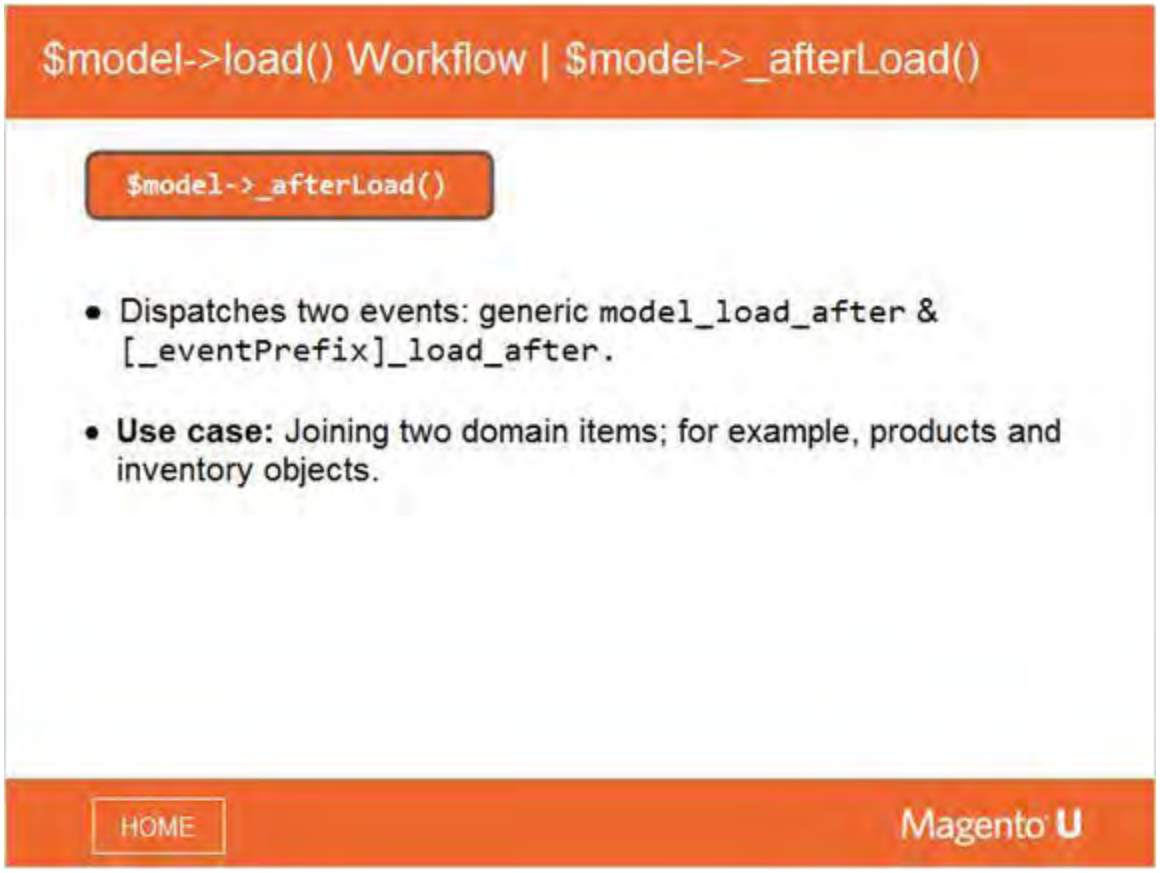
```
/**
     * Retrieve select object for load object data
     *
     * @param string $field
     * @param mixed $value
     * @param \Magento\Framework\Model\AbstractModel $object
     * @return \Magento\Framework\DB\Select
     * @SuppressWarnings(PHPMD.UnusedFormalParameter)
     */
    protected function _getLoadSelect($field, $value, $object)
    {
        $field = $this->getConnection()->quoteIdentifier(
                                  sprintf('%s.%s', $this->getMainTable(), $field));
        $select = $this->getConnection()
                        ->select()
                        ->from($this
                        ->getMainTable())
                        ->where($field . '=?', $value);
        return $select;
    }
```

Notes:

This code is from a resource model.

`The method _getLoadSelect` is used to bind an SQL query to retrieve the relevant data. All connection differentiation in Magento is unified because semantically it is the same connection, meaning that the difference between read and write adapters that was present in Magento 1 has been eliminated.

3.8 $model->load() Workflow | $model->_afterLoad()



$model->load() Workflow | $model->_afterLoad()

`$model->_afterLoad()`

- Dispatches two events: generic `model_load_after` & `[_eventPrefix]_load_after`.
- **Use case:** Joining two domain items; for example, products and inventory objects.

HOME
Magento U

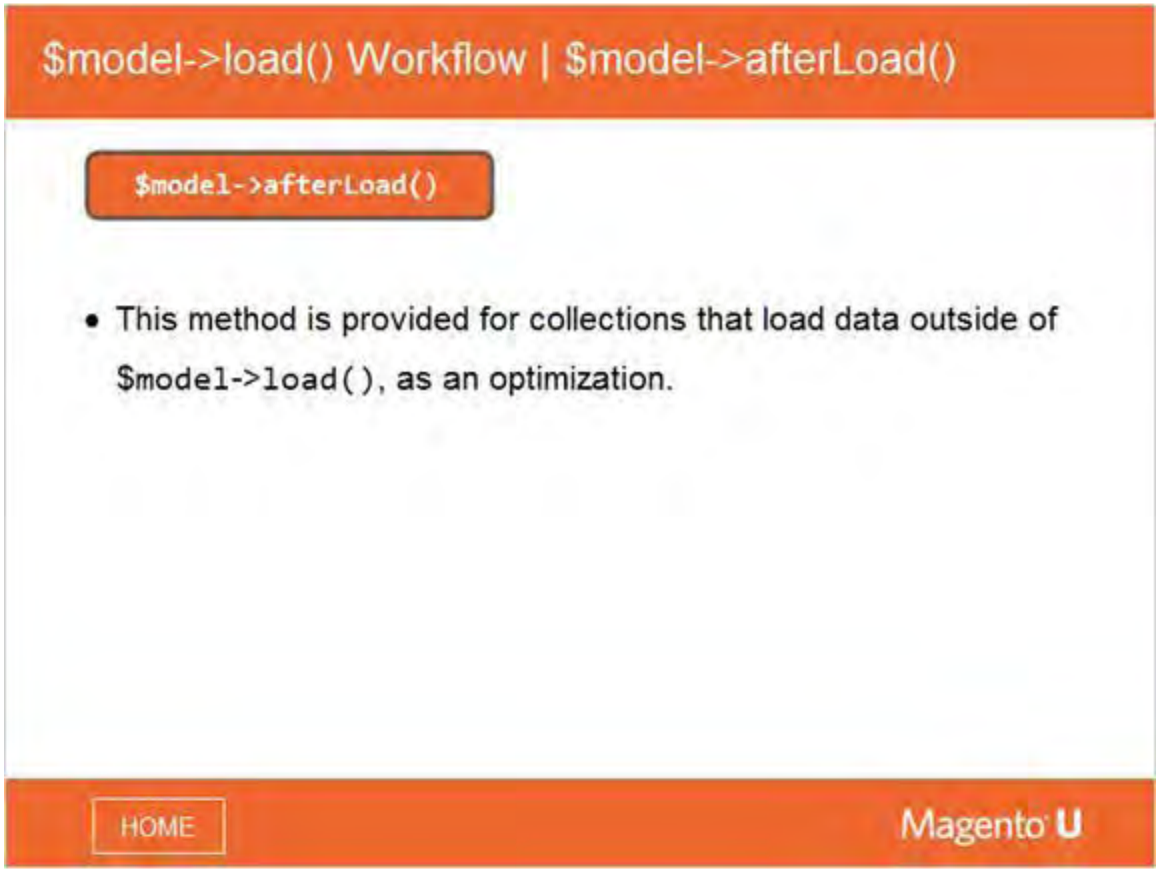
Notes:

The `_afterLoad` method is used for processing an object directly after loading the data. Every time a Magento model is loaded, you can intercept it by observing the events `model_load_before` and `model_load_after`, which carry a simple data transfer object. The model instance itself is under the key “object”.

As you can see on the slide, there also is a more specific event, which uses a entity specific “prefix”. In Magento, most of the core models redefine the internal variable “`_eventPrefix`”, so one can observe CRUD operations for specific models.

To determine the prefix used for a model, you need to check the model source code.

3.9 $model->load() Workflow | $model->afterLoad()



$model->load() Workflow | $model->afterLoad()

$model->afterLoad()

- This method is provided for collections that load data outside of `$model->load()`, as an optimization.

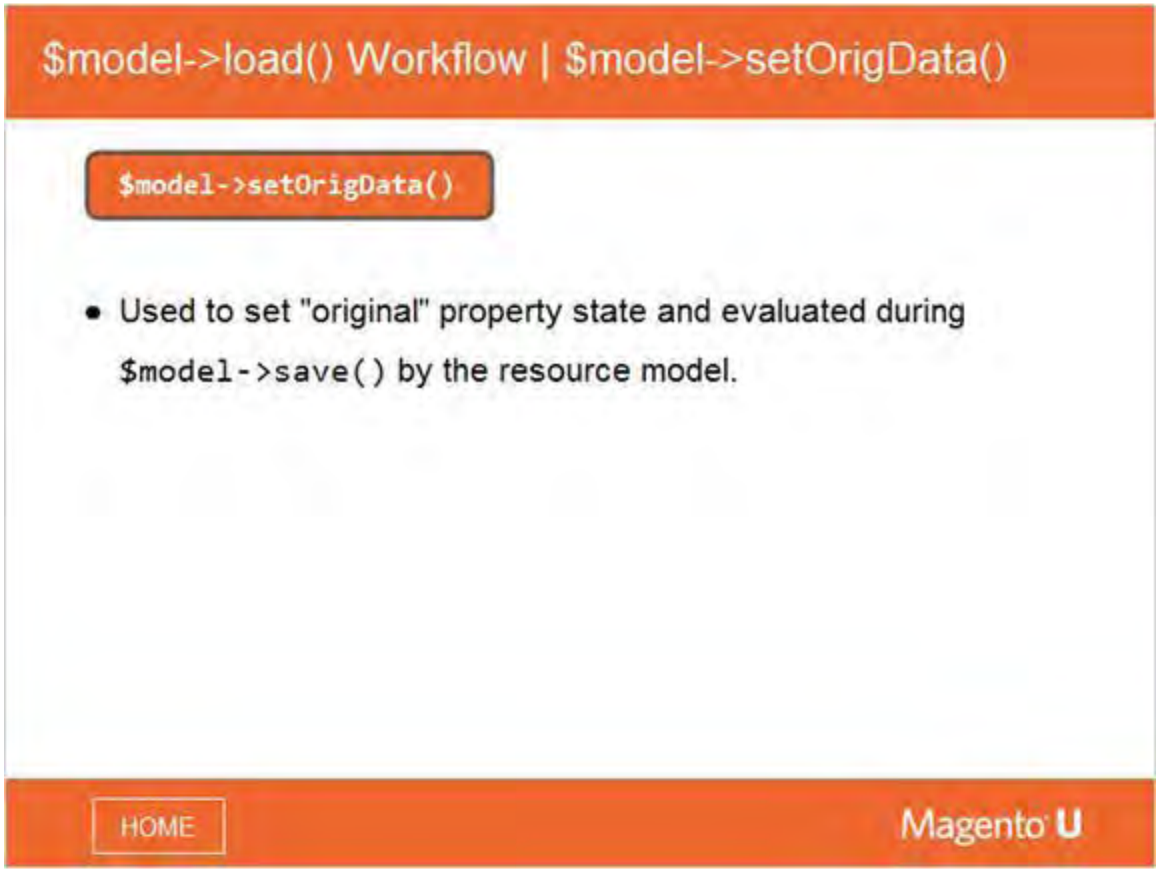
HOME
Magento U

Notes:

Be aware that events like `model_load_before` and `model_load_after` will not be dispatched while loading a collection. This is because a collection does not load each item individually but instead loads models in the collection at once.

`core_collection_abstract_load_before and core_collection_abstract_load_after` will be dispatched once for the complete collection, along with specific events like `*_load_before` and `*_load_after`.

3.10 $model->load() Workflow | $model->afterLoad()



$model->load() Workflow | $model->setOrigData()

`$model->setOrigData()`

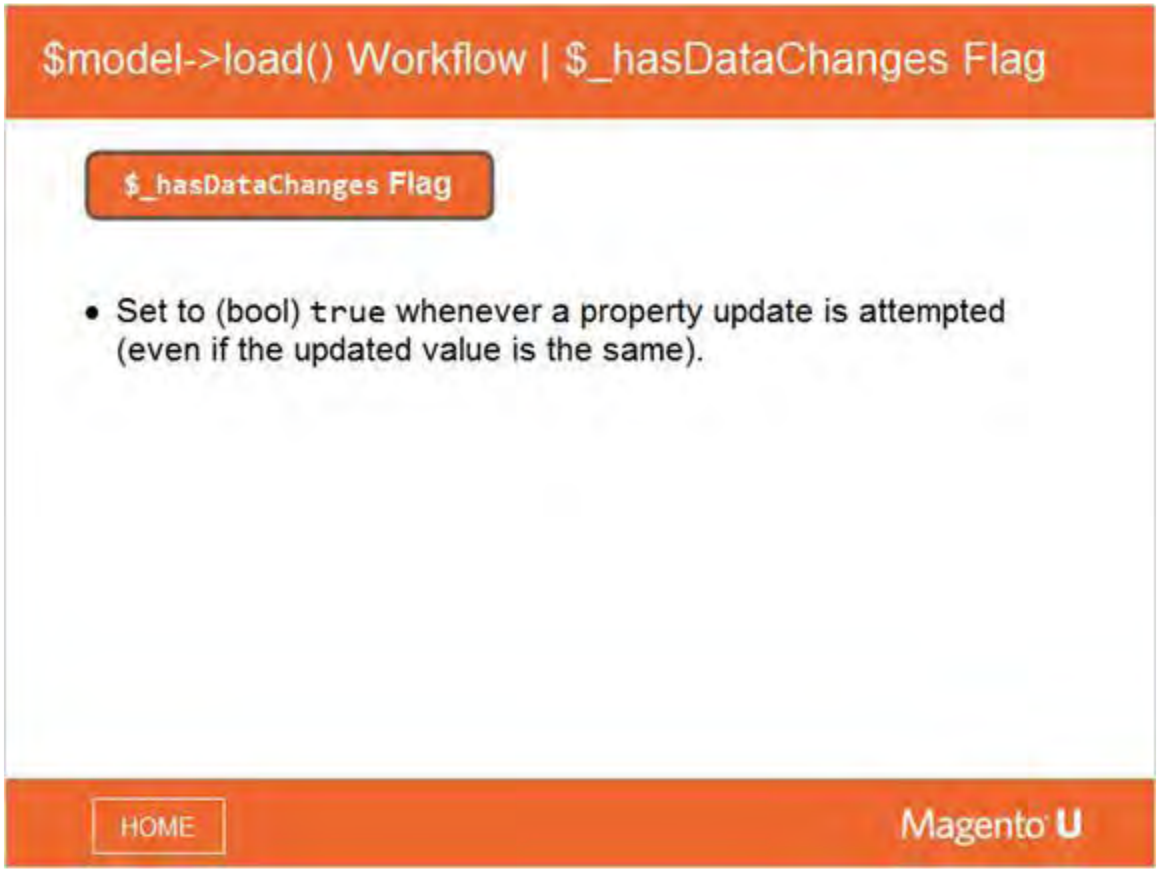
- Used to set "original" property state and evaluated during `$model->save()` by the resource model.

HOME
Magento U

Notes:

Since models inherit from `Magento\Framework\Model\AbstractModel`, you can easily get the original loaded data using the `public function getOrigData($key=null)` method.

3.11 $model->load() Workflow | $_hasDataChanges Flag



$model->load() Workflow | $_hasDataChanges Flag

$_hasDataChanges Flag

- Set to (bool) `true` whenever a property update is attempted (even if the updated value is the same).

HOME
Magento U

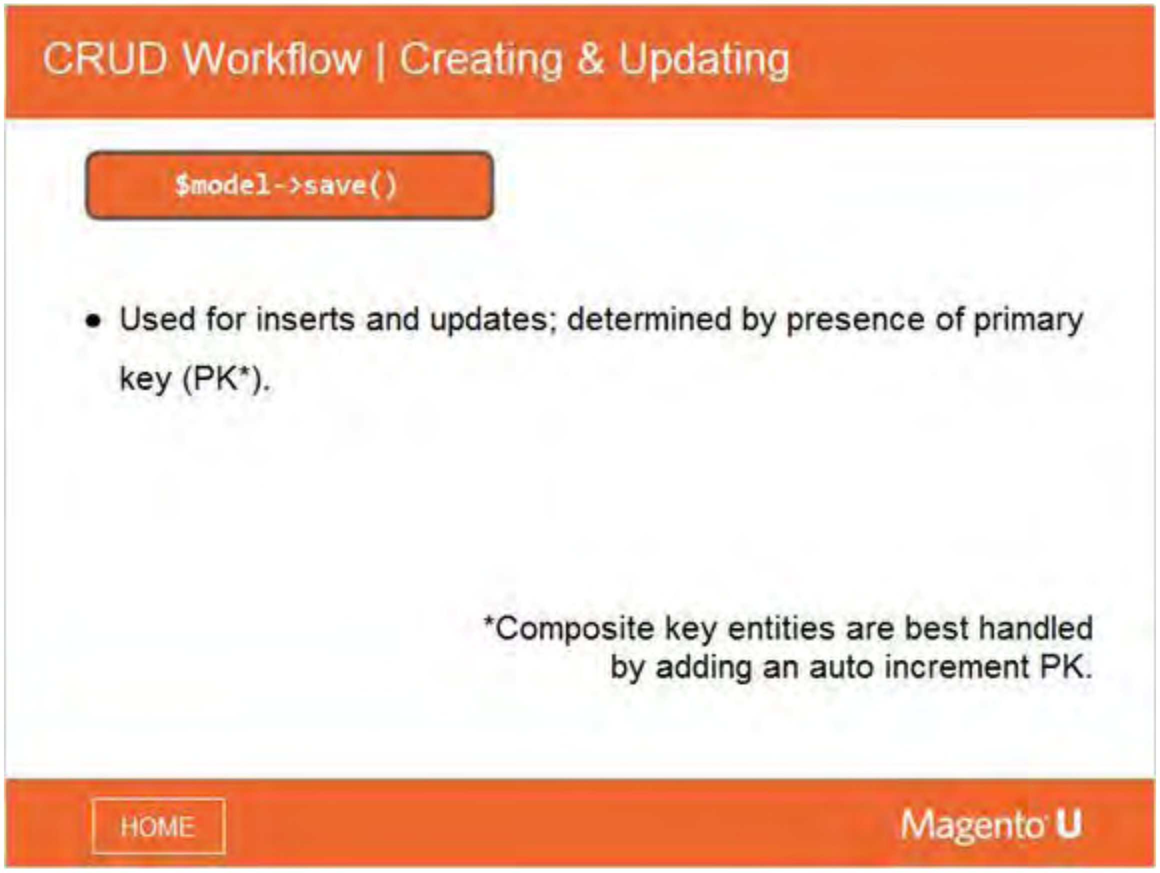
Notes:

Another helpful method that models expose with public visibility is `hasDataChanges()`.

By using this method you can check if the model data was changed since the last time the model was loaded or saved.

Resource models have a similar method, the protected function `isModified()`, which is a simple wrapper for the model `hasDataChanges()` method.

3.12 CRUD Workflow | Creating & Updating



CRUD Workflow | Creating & Updating

`$model->save()`

- Used for inserts and updates; determined by presence of primary key (PK*).

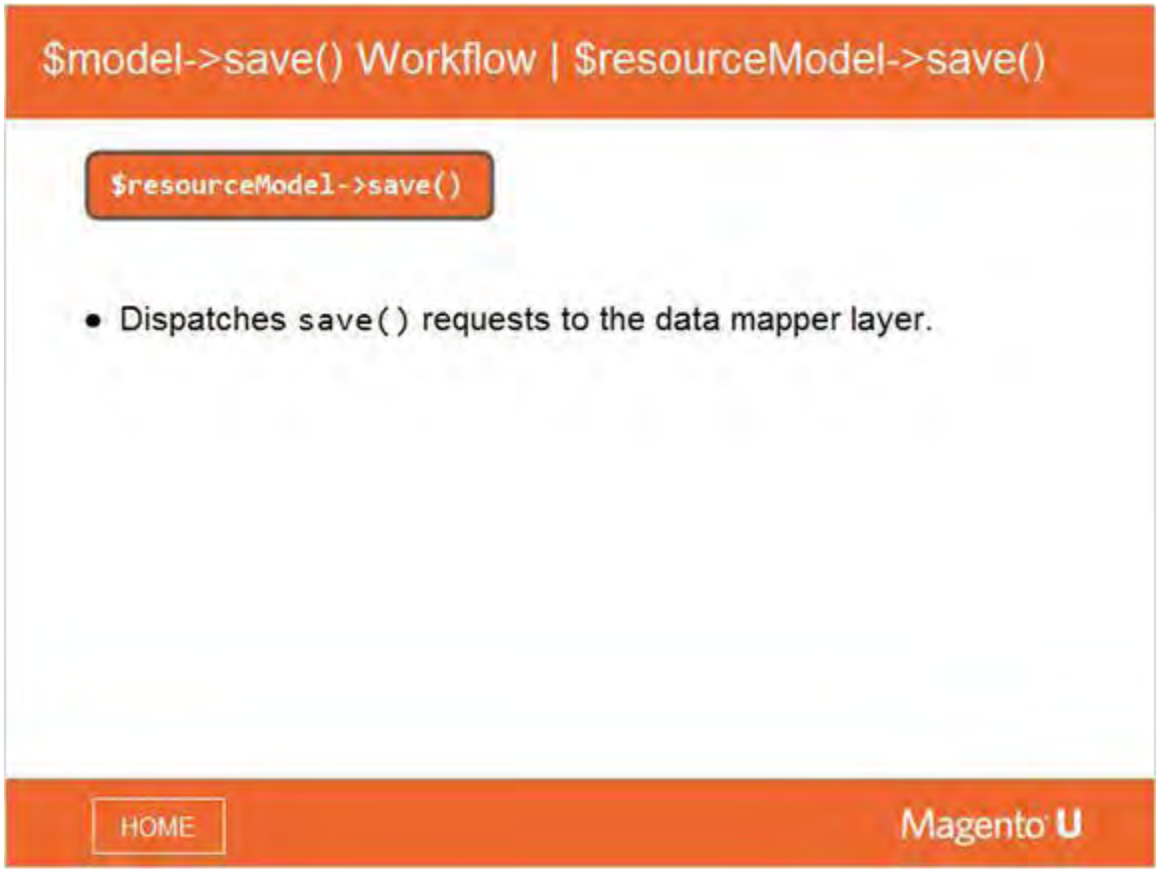
*Composite key entities are best handled by adding an auto increment PK.

HOME
Magento U

Notes:

The `save()` method will allow you to either `INSERT` a new model into the database or `UPDATE` an existing one.

3.13 $model->save() Workflow | $resourceModel->save()



$model->save() Workflow | $resourceModel->save()

`$resourceModel->save()`

- Dispatches `save()` requests to the data mapper layer.

HOME
Magento U

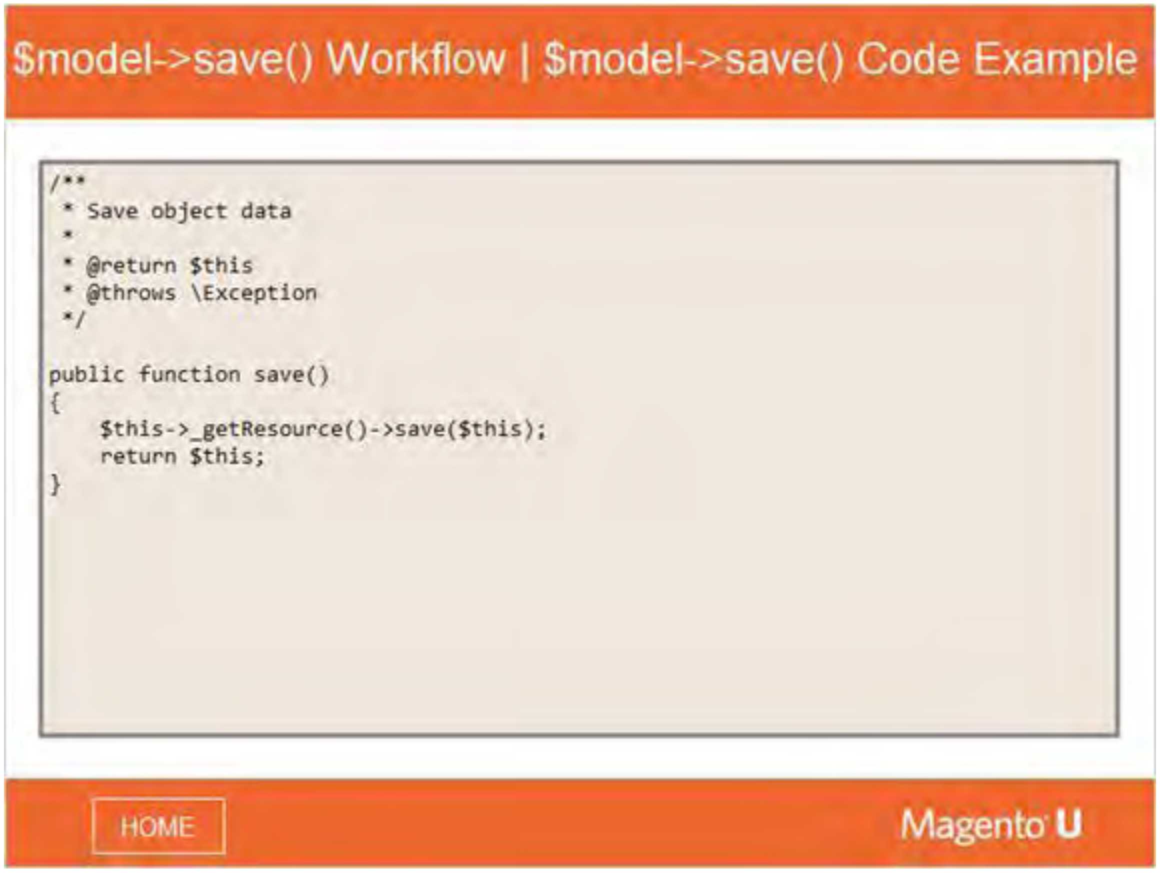
Notes:

The essence of the resource model `save()` method is straightforward:

- Open a transaction.
- Delegate the saving to a data mapper.
- Commit the transaction.

All the rest is error handling or skipping the save process if the object wasn't changed.

3.14 $model->save() Workflow | $model->save() Code Example

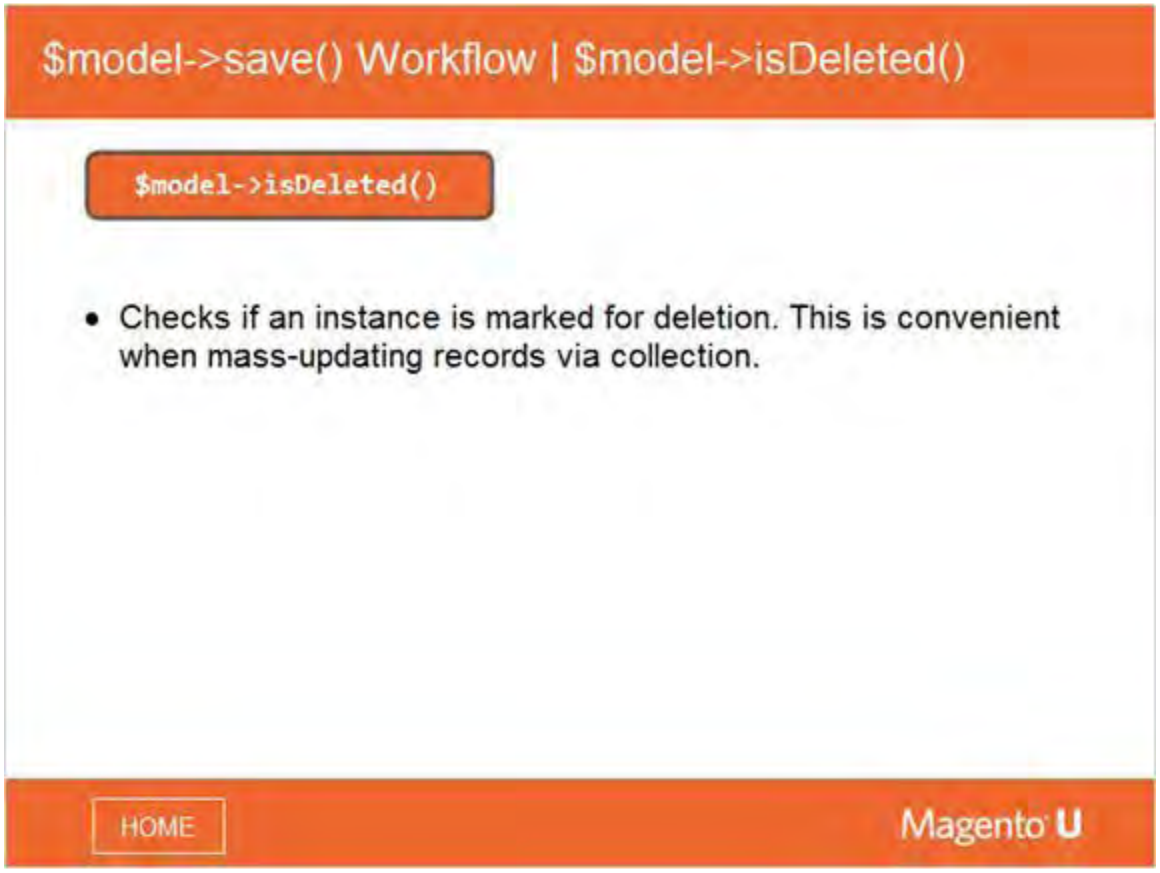

$model->save() Workflow | $model->save() Code Example

```
/**
 * Save object data
 *
 * @return $this
 * @throws \Exception
 */

public function save()
{
    $this->_getResource()->save($this);
    return $this;
}
```

Notes:

The model's `save()` method itself is pretty simple -- all the logic is delegated to the resource model.

3.15 $model->save() Workflow | $model->isDeleted()


$model->save() Workflow | $model->isDeleted()

`$model->isDeleted()`

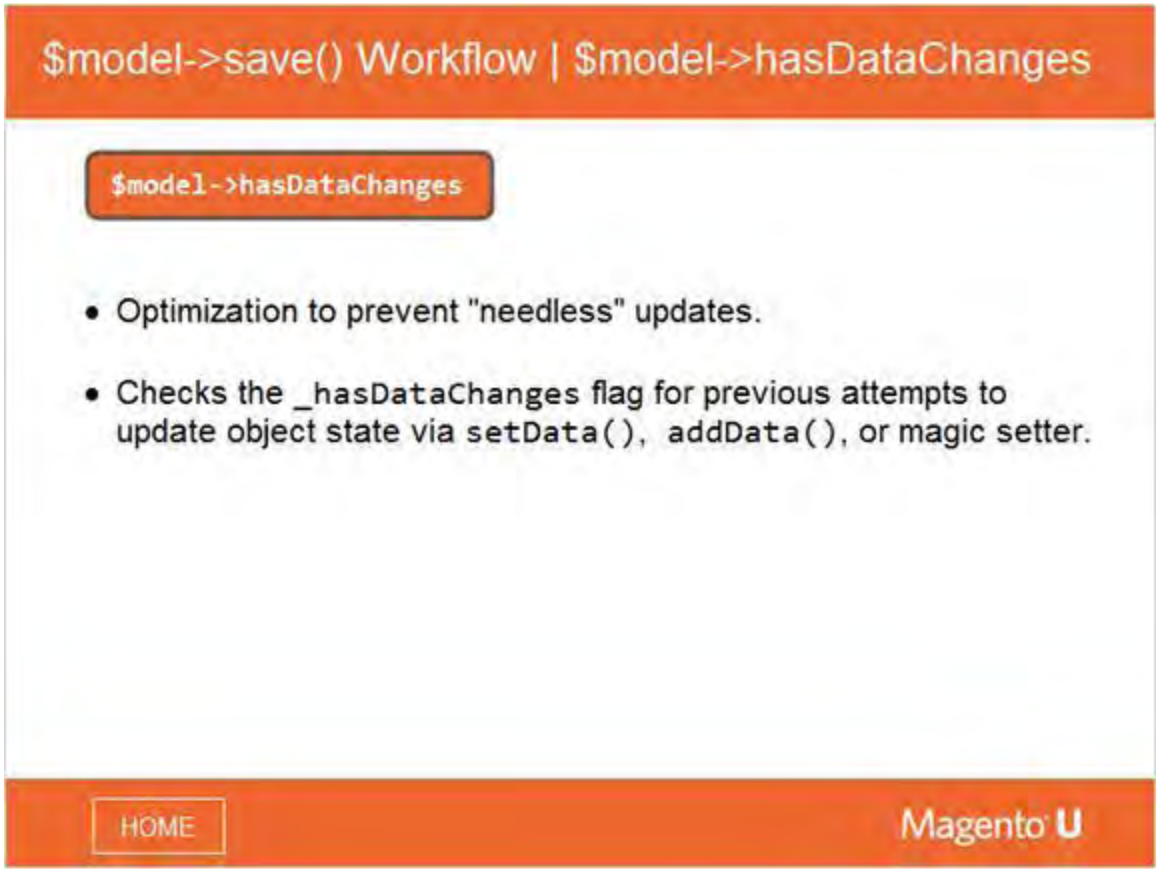
- Checks if an instance is marked for deletion. This is convenient when mass-updating records via collection.

HOME
Magento U

Notes:

Actually the method does both: If the `$isDeleted` parameter is specified, it sets the `_isDeleted` flag value; otherwise, it returns the current flag value.

3.16 $model->save() Workflow | $model->isDeleted()



$model->save() Workflow | $model->hasDataChanges

$model->hasDataChanges

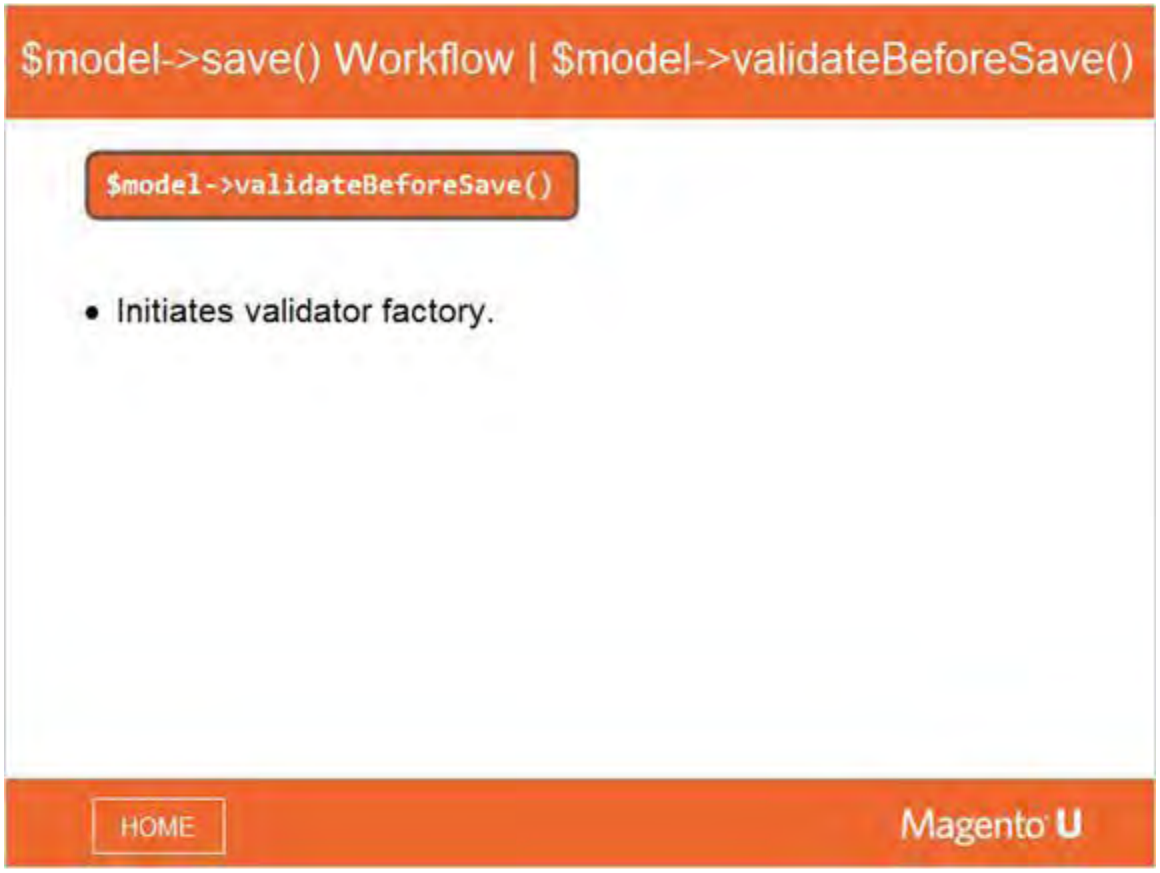
- Optimization to prevent "needless" updates.
- Checks the `_hasDataChanges` flag for previous attempts to update object state via `setData()`, `addData()`, or magic setter.

HOME
Magento U

Notes:

This slide is self-explanatory.

3.17 $model->save() Workflow | $model->isDeleted()



$model->save() Workflow | $model->validateBeforeSave()

`$model->validateBeforeSave()`

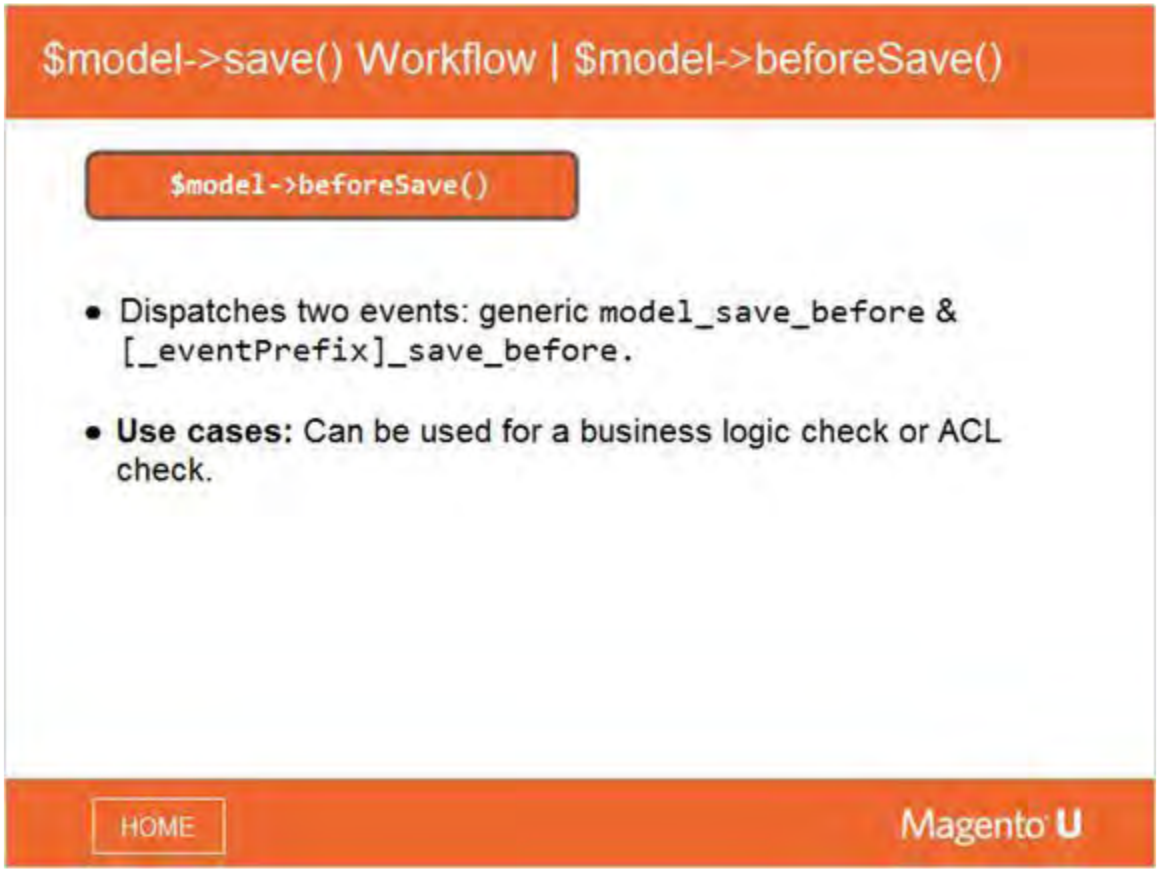
- Initiates validator factory.

HOME
Magento U

Notes:

The method `validateBeforeSave()` uses a validator, which contains all the rules to validate the current model.

3.18 $model->save() Workflow | $model->beforeSave()



$model->save() Workflow | $model->beforeSave()

`$model->beforeSave()`

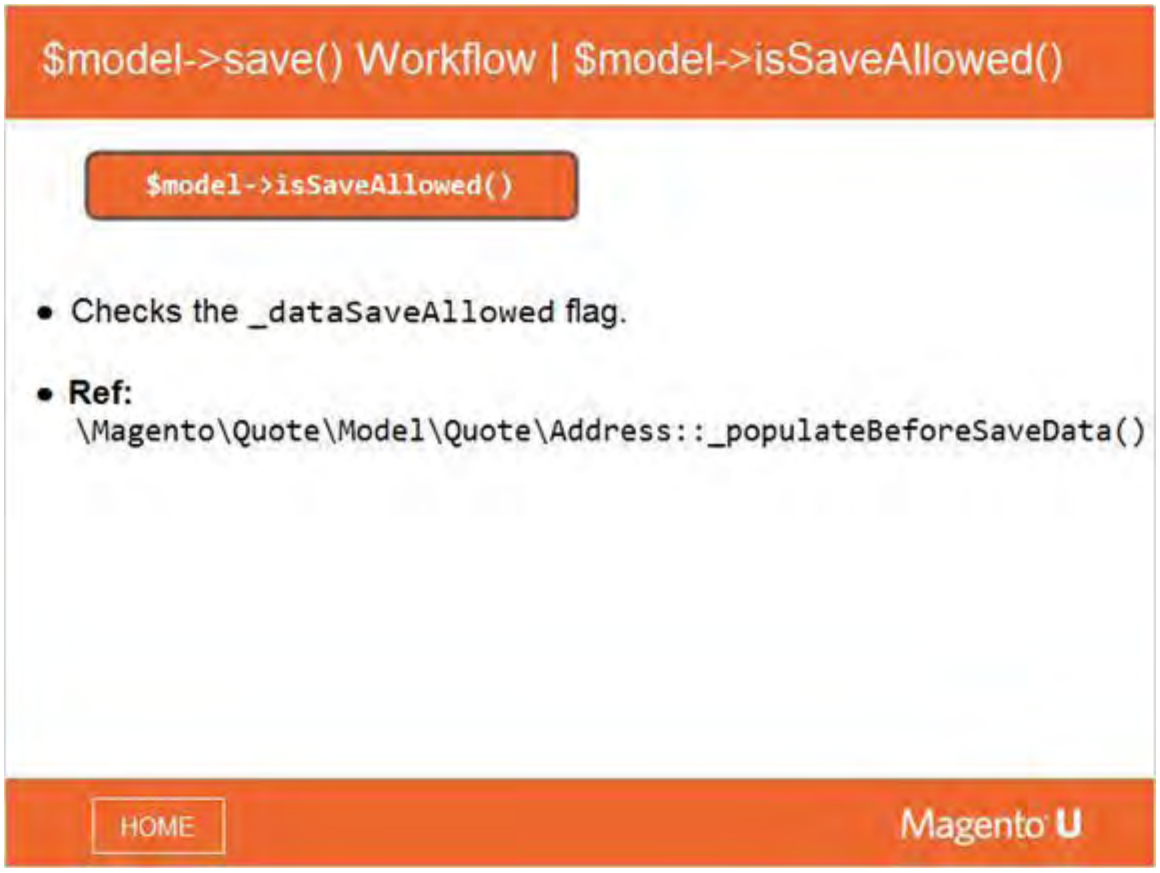
- Dispatches two events: generic `model_save_before` & `[_eventPrefix]_save_before`.
- **Use cases:** Can be used for a business logic check or ACL check.

HOME
Magento U

Notes:

The `save()` method also checks the object state (whether it is new or not) using the `isObjectNew($flag = null)` method. This method helps to detect if the object was created in this transaction.

3.19 $model->save() Workflow | $model->isSaveAllowed()



$model->save() Workflow | $model->isSaveAllowed()

`$model->isSaveAllowed()`

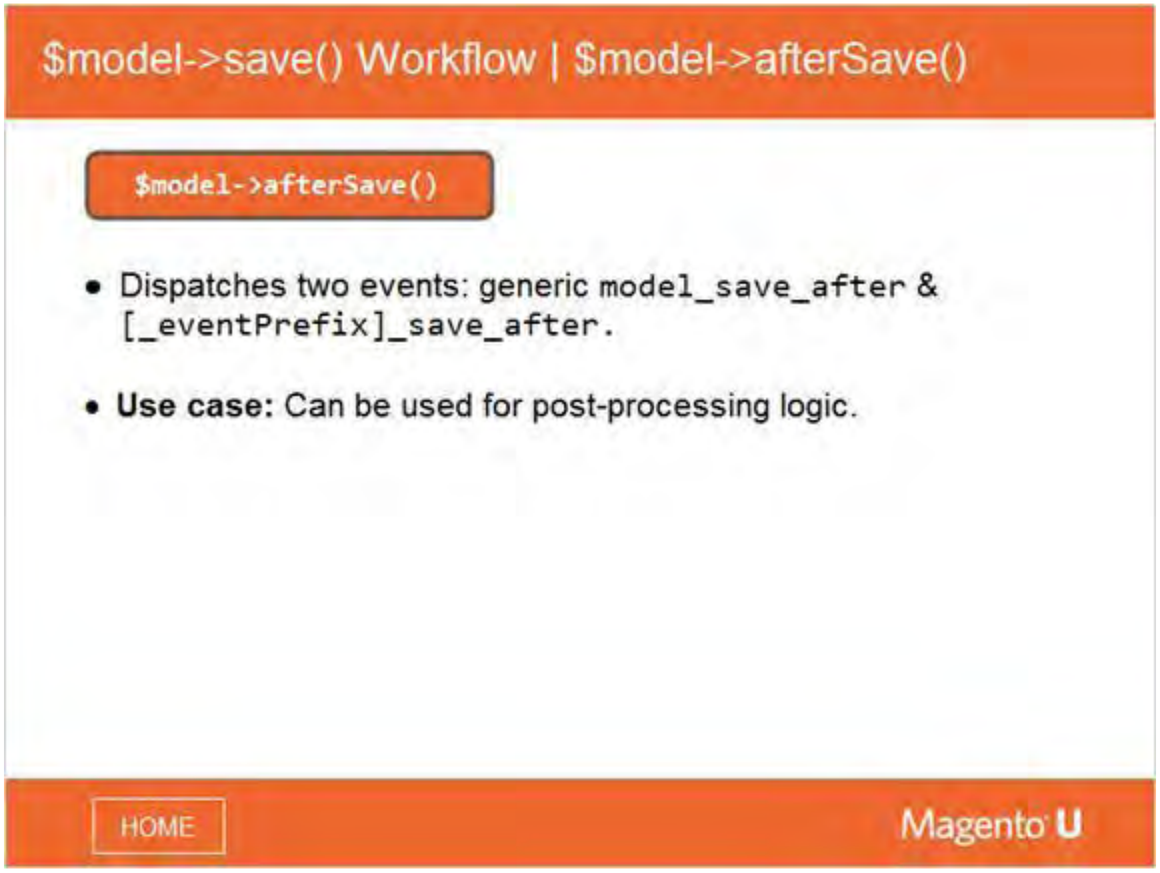
- Checks the `_dataSaveAllowed` flag.
- **Ref:** `\Magento\Quote\Model\Quote\Address::_populateBeforeSaveData()`

HOME
Magento U

Notes:

In the resource model `save()` method, the `_dataSaveAllowed` flag can stop saving after the `beforeSave()` method was called. This can be helpful in case some data validation failed.

3.20 $model->save() Workflow | $model->afterSave()



$model->save() Workflow | $model->afterSave()

$model->afterSave()

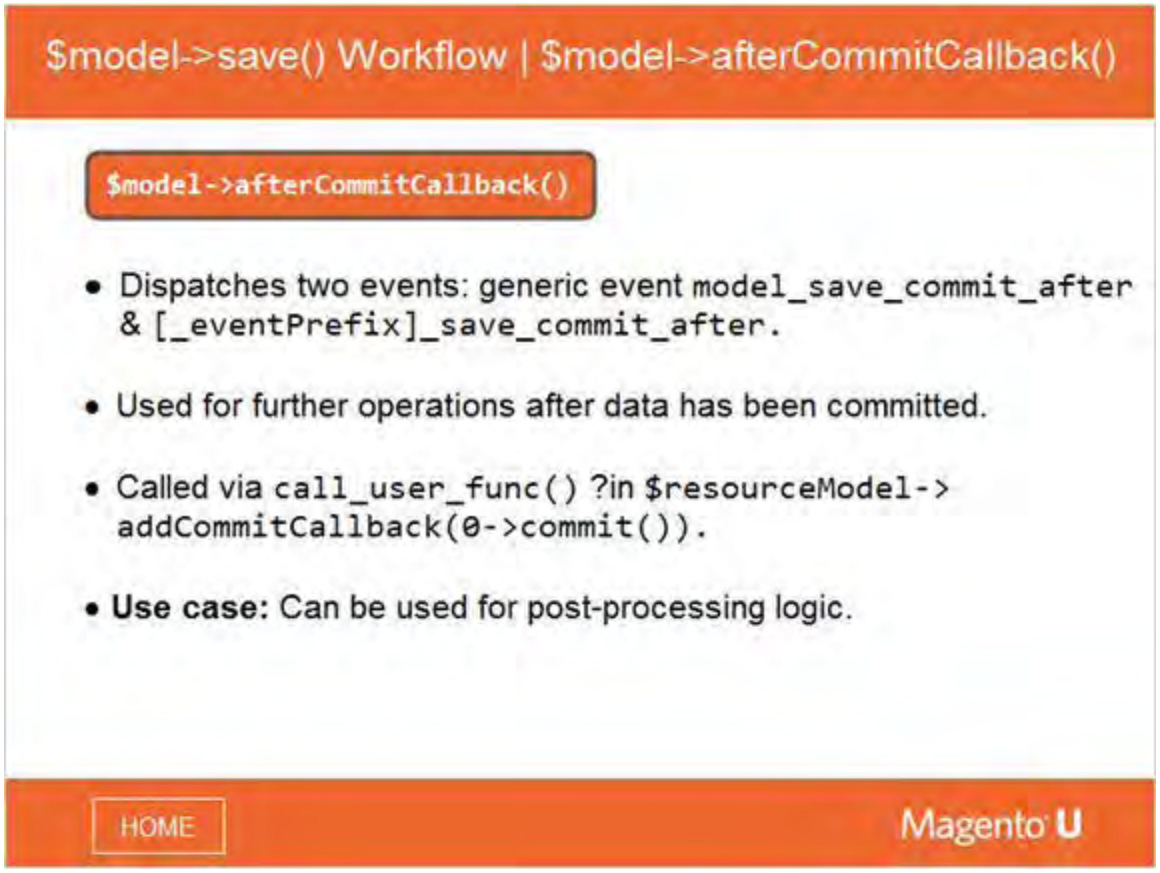
- Dispatches two events: generic `model_save_after` & `[_eventPrefix]_save_after`.
- **Use case:** Can be used for post-processing logic.

HOME
Magento U

Notes:

The slide is self-explanatory.

3.21 $model->save() Workflow | $model->afterCommitCallback()



$model->save() Workflow | $model->afterCommitCallback()

`$model->afterCommitCallback()`

- Dispatches two events: generic event `model_save_commit_after` & `[_eventPrefix]_save_commit_after`.
- Used for further operations after data has been committed.
- Called via `call_user_func()` ?in `$resourceModel->addCommitCallback(0->commit())`.
- **Use case:** Can be used for post-processing logic.

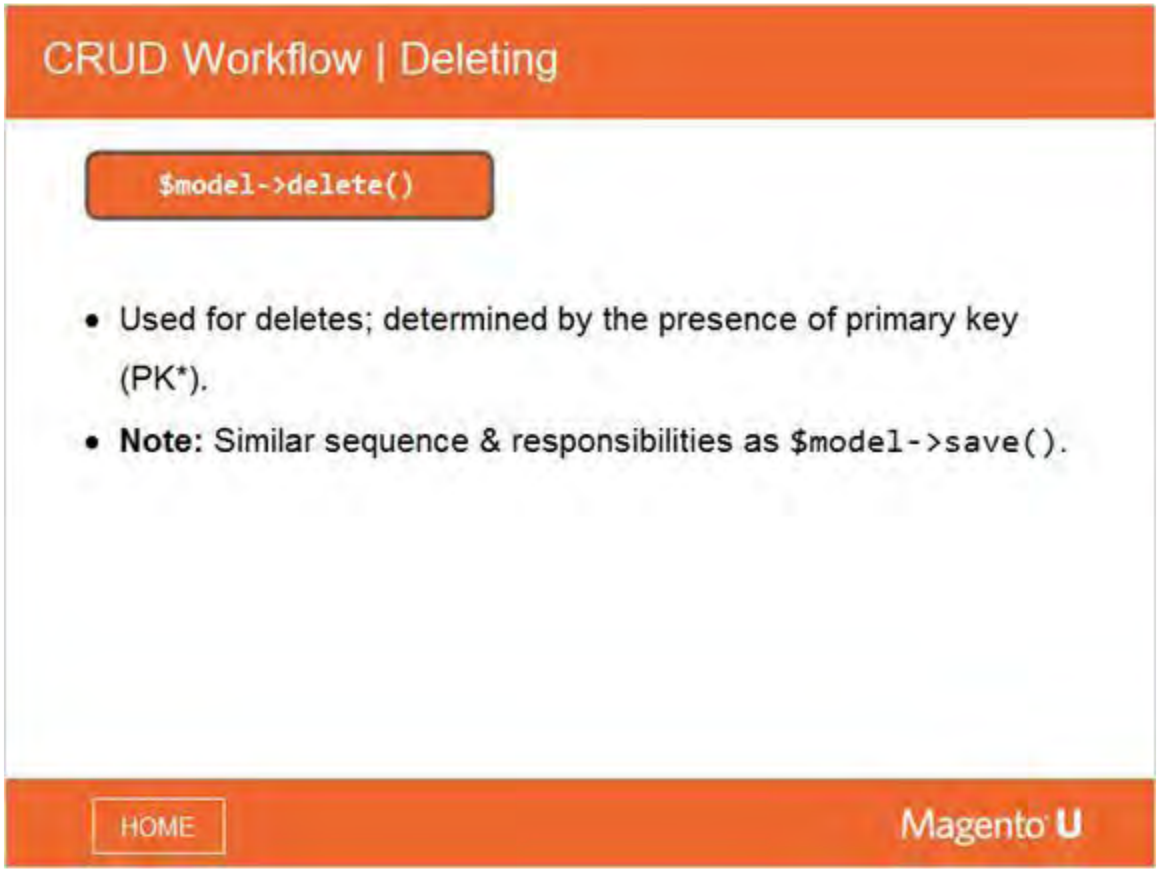
HOME

Magento U

Notes:

The slide is self-explanatory.

3.22 CRUD Workflow | Deleting



CRUD Workflow | Deleting

`$model->delete()`

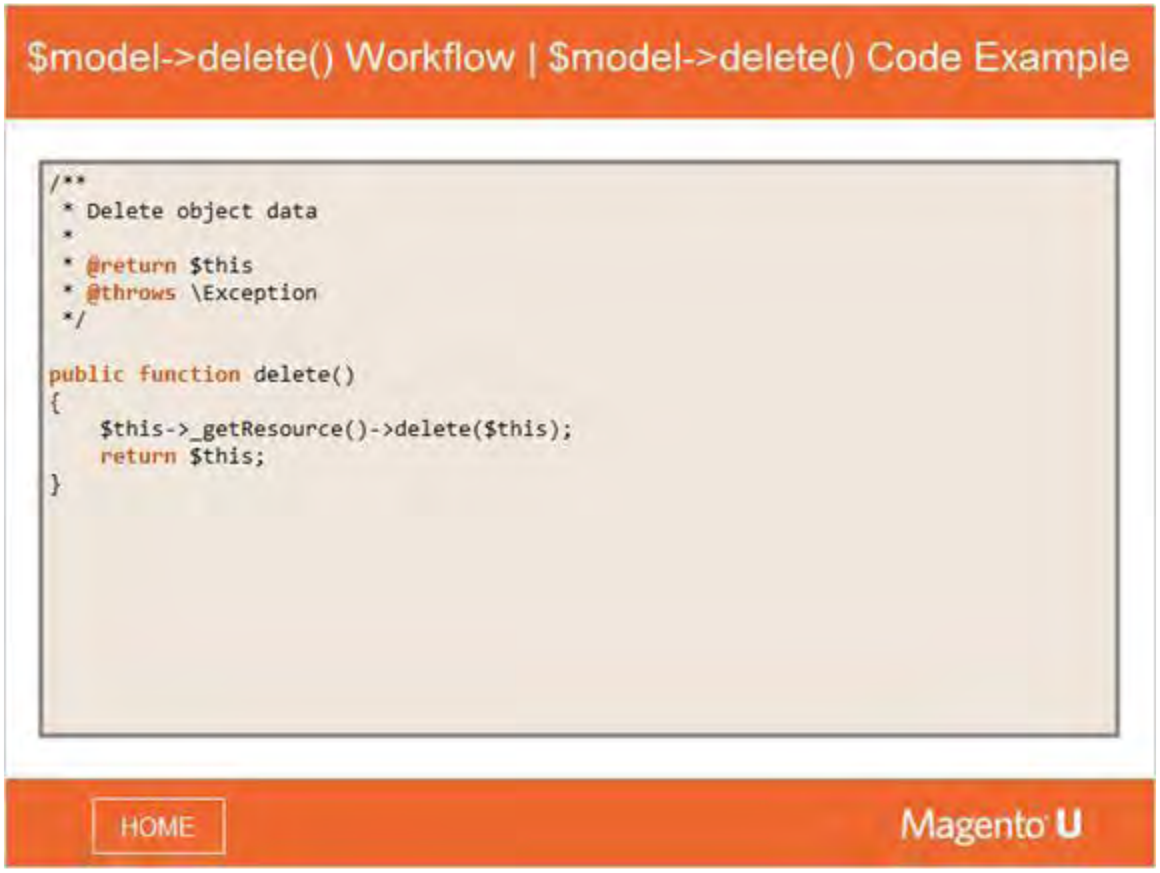
- Used for deletes; determined by the presence of primary key (PK*).
- **Note:** Similar sequence & responsibilities as `$model->save()`.

HOME
Magento U

Notes:

The slide is self-explanatory.

3.23 $model->delete() Workflow | $model->delete() Code Example



$model->delete() Workflow | $model->delete() Code Example

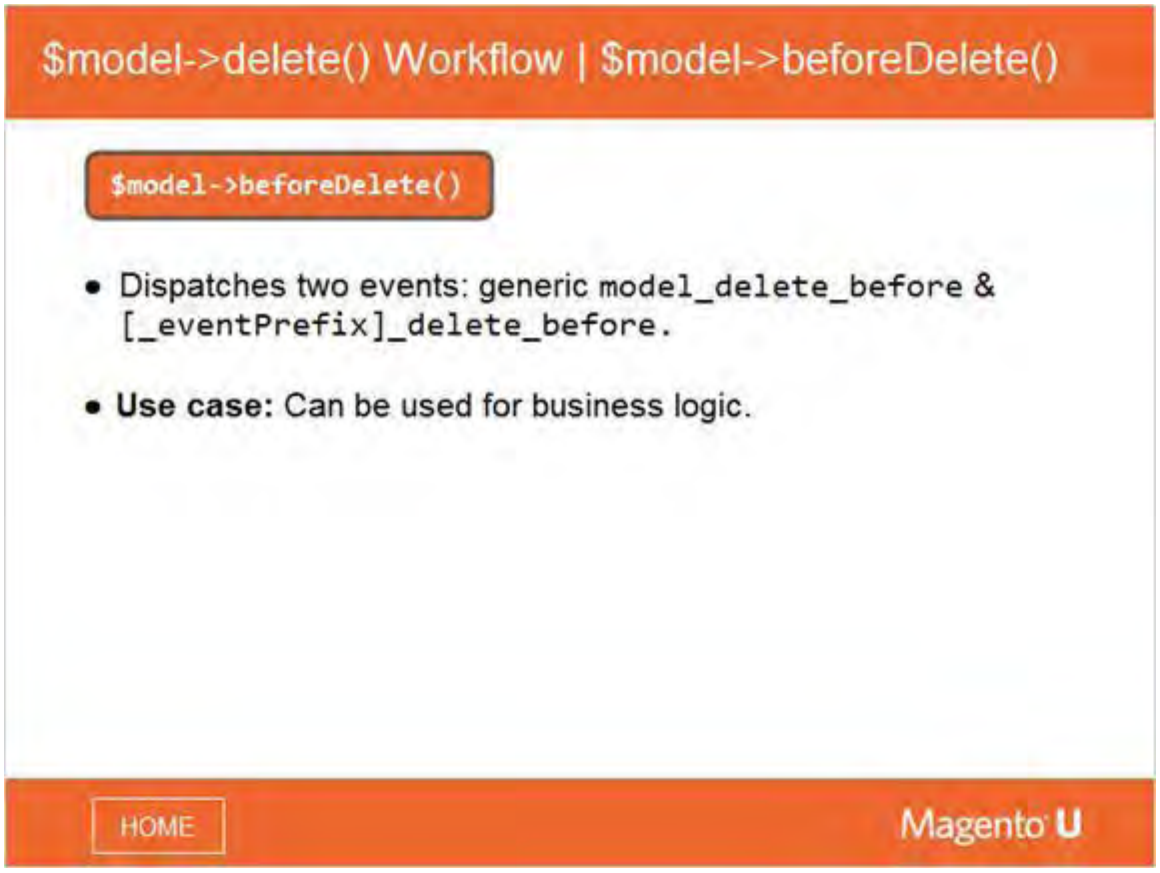
```
/**
 * Delete object data
 *
 * @return $this
 * @throws \Exception
 */

public function delete()
{
    $this->_getResource()->delete($this);
    return $this;
}
```

Notes:

And again, the model delegates all the work to the resource model, which in turn:

- Starts a transaction.
- Dispatches before and after delete methods.
- Cleans the model cache.
- Directly deletes the data from the database.
- Commits the transaction if there were no errors.

3.24 $model->delete() Workflow | $model->beforeDelete()


$model->delete() Workflow | $model->beforeDelete()

$model->beforeDelete()

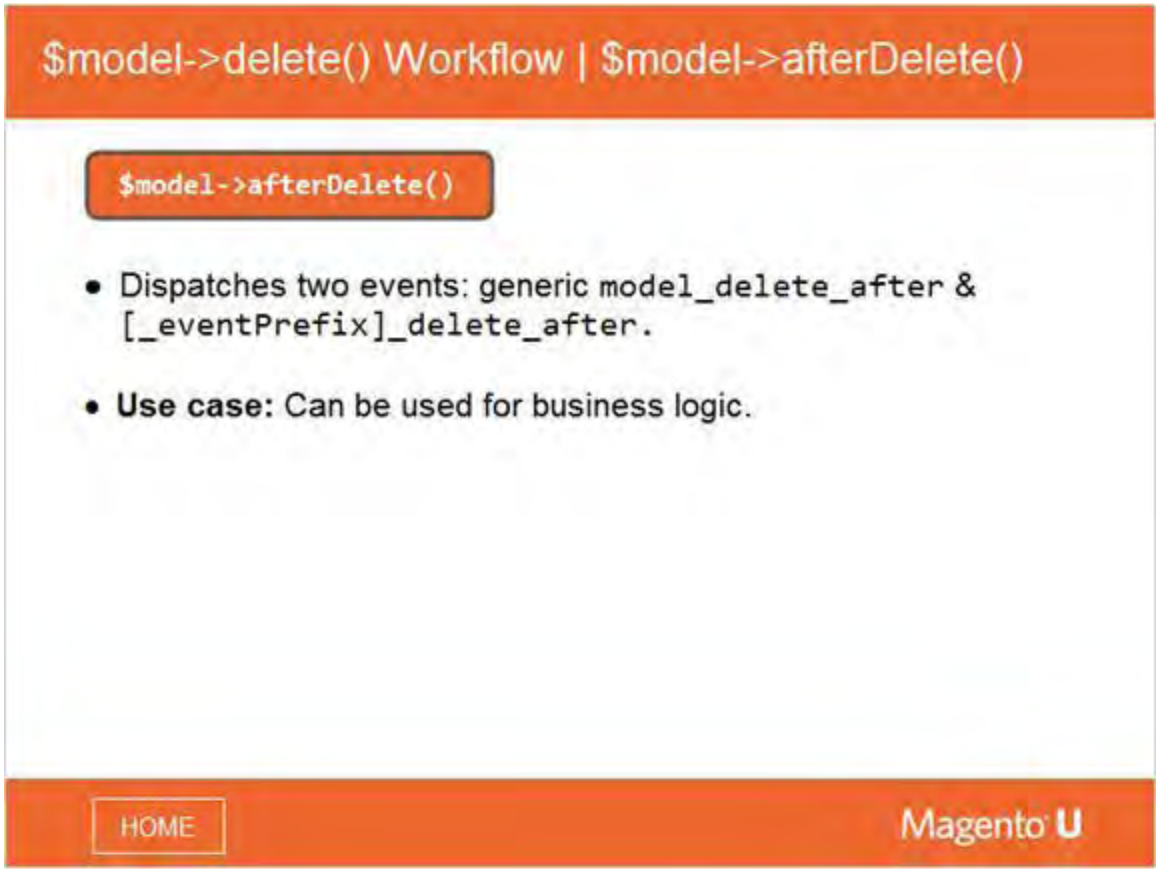
- Dispatches two events: generic `model_delete_before` & `[_eventPrefix]_delete_before`.
- **Use case:** Can be used for business logic.

HOME
Magento U

Notes:

The events are also used to check if the delete action is allowed for the current area, and to clear the model object-related cache.

3.25 $model->delete() Workflow | $model->afterDelete()


$model->delete() Workflow | $model->afterDelete()

`$model->afterDelete()`

- Dispatches two events: generic `model_delete_after` & `[_eventPrefix]_delete_after`.
- **Use case:** Can be used for business logic.

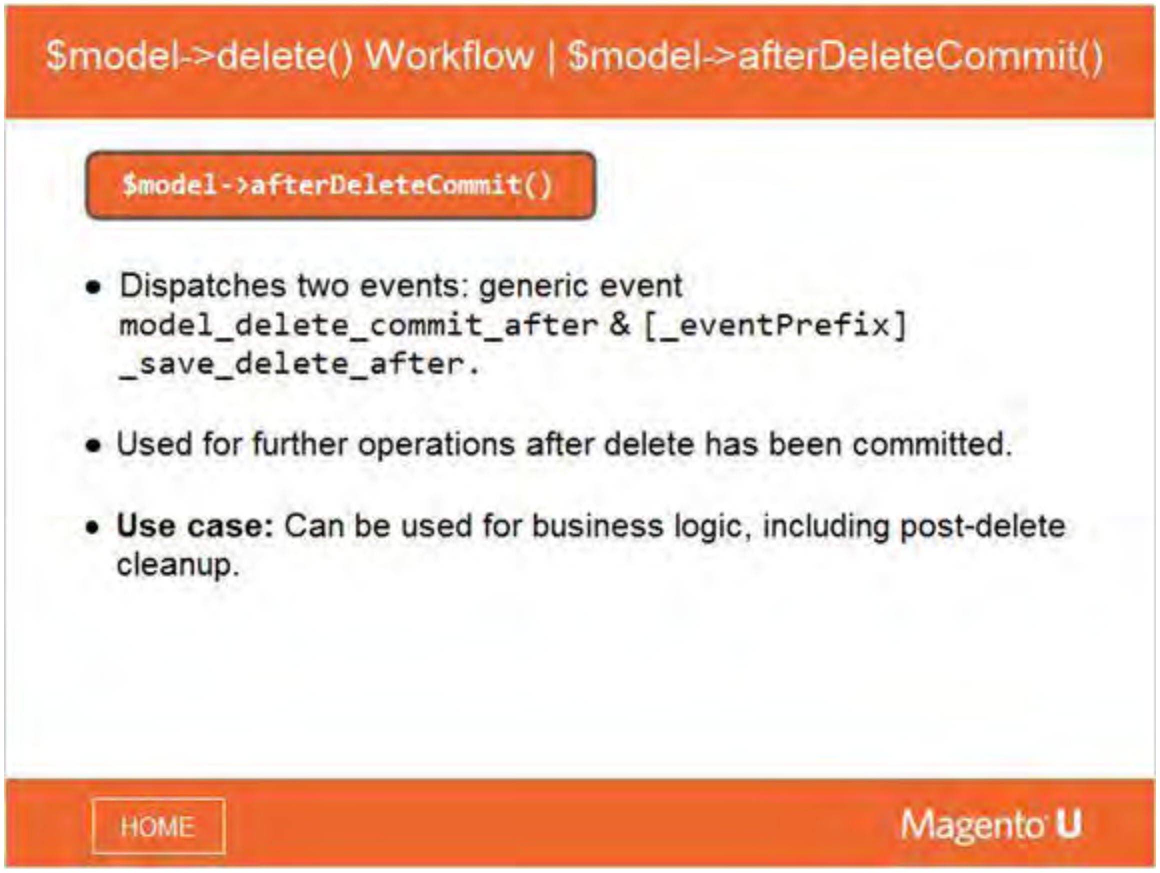
HOME
Magento U

Notes:

The delete after events are used for processing the object after its data has been deleted.

It also dispatches the `clean_cache_by_tags` event.

3.26 $model->delete() Workflow | $model->afterDeleteCommit()



$model->delete() Workflow | $model->afterDeleteCommit()

`$model->afterDeleteCommit()`

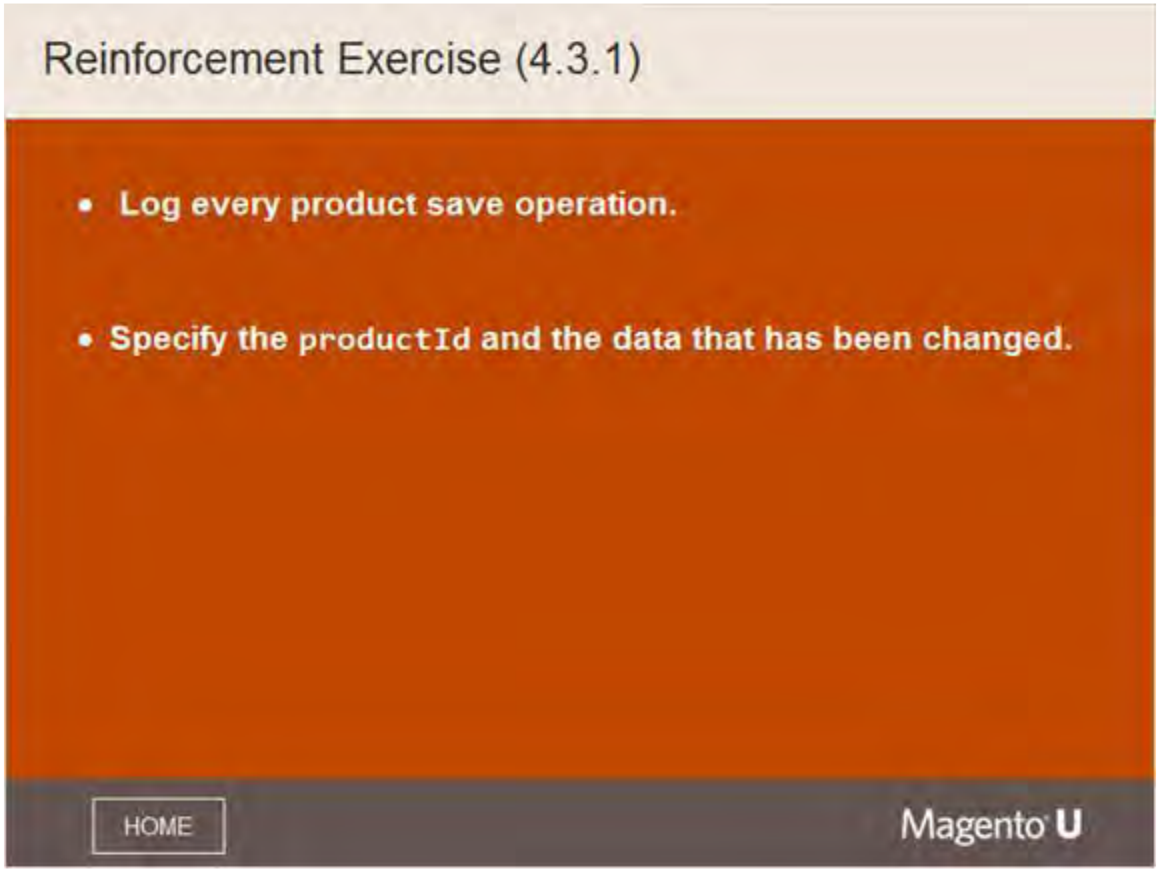
- Dispatches two events: generic event `model_delete_commit_after` & `[_eventPrefix]_save_delete_after`.
- Used for further operations after delete has been committed.
- **Use case:** Can be used for business logic, including post-delete cleanup.

HOME
Magento U

Notes:

The slide is self-explanatory.

3.27 Reinforcement Exercise (4.3.1)



Reinforcement Exercise (4.3.1)

- Log every product save operation.
- Specify the `productId` and the data that has been changed.

HOME
Magento U

4. Setup Scripts & Setup Resources

4.1 Setup Scripts & Setup Resources

Setup Scripts & Setup Resources

Setup Resources & Install / Upgrade Scripts

Notes:

We will now examine setup scripts and resources in detail.

4.2 Module Topics | Setup Scripts & Setup Resources

Module Topics | Setup Scripts & Setup Resources

In this module, we will discuss...

- Creating New Entities
- Loading & Saving an Entity from a Database
- Classes & Database Tables

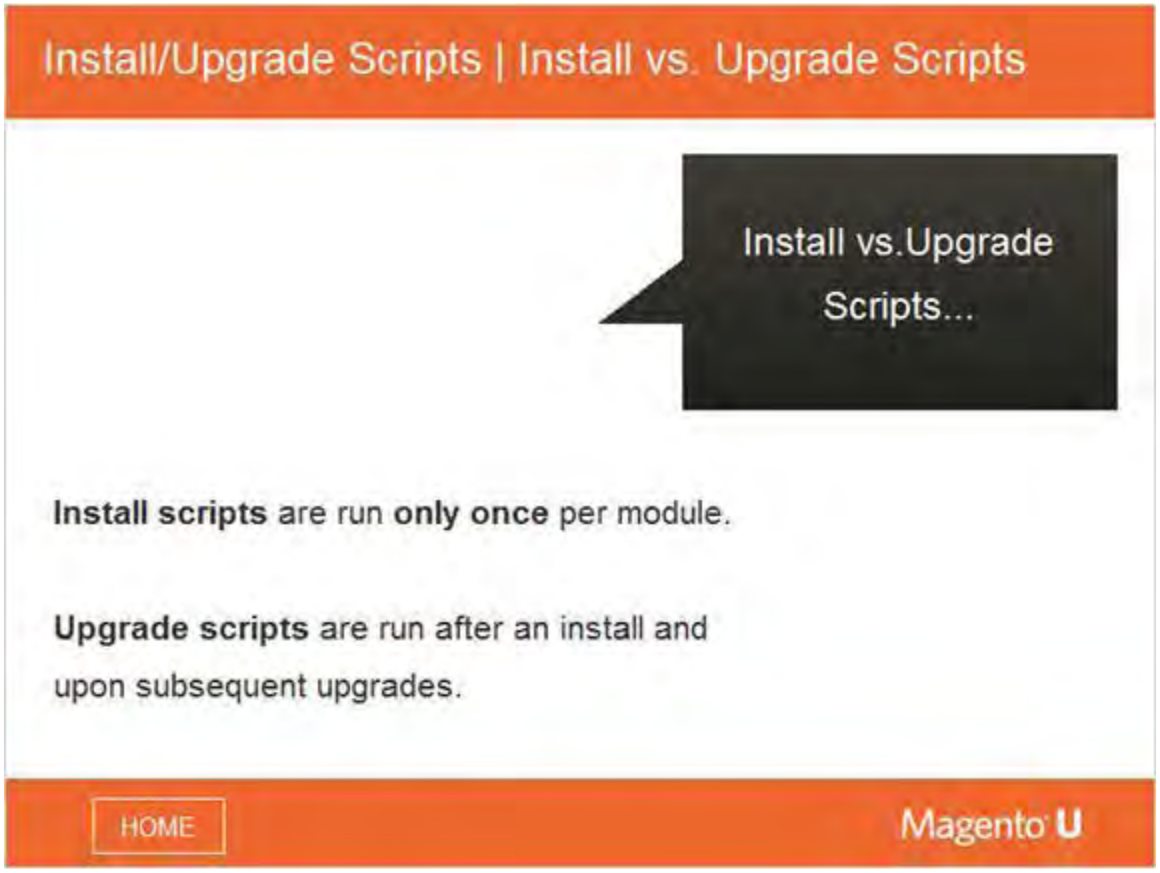
HOME
Magento U

Notes:

In this module, we will discuss:

- Creating new entities
- Loading & saving an entity from a database
- Classes & database tables

4.3 Install/Upgrade Scripts | Install vs. Upgrade Scripts



Install/Upgrade Scripts | Install vs. Upgrade Scripts

Install vs.Upgrade Scripts...

Install scripts are run **only once** per module.

Upgrade scripts are run after an install and upon subsequent upgrades.

HOME
Magento U

Notes:

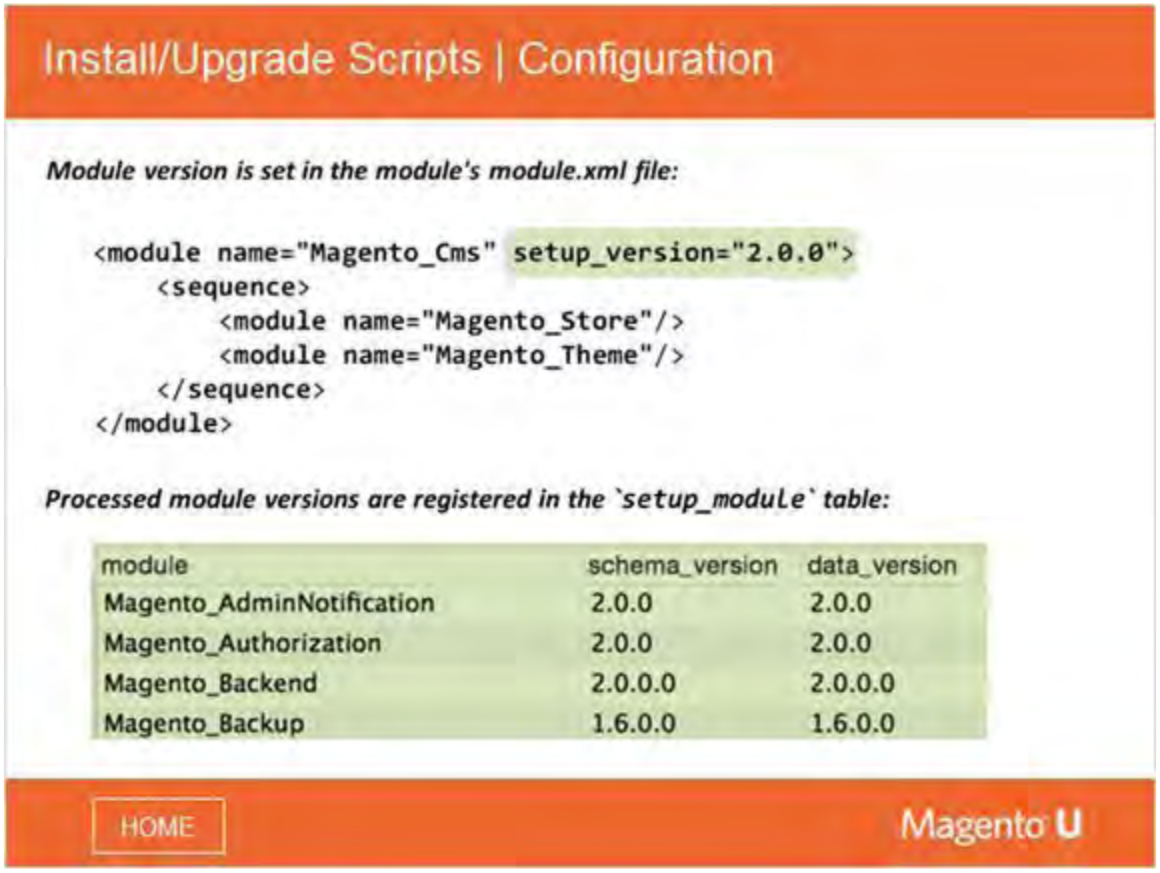
Setup scripts are a powerful tool for Magento database customization and for setting configuration settings.

Magento uses install and upgrade scripts to:

- Allow modules to create and modify the required database resources automatically.
- Update tables on the production server.
- Allow other developers to upgrade their system automatically.

Once Magento runs an ***install*** script for a module, it will never run another install script for that module again. To make further adjustments, you need to use upgrade scripts.

4.4 Install/Upgrade Scripts | Configuration



Install/Upgrade Scripts | Configuration

Module version is set in the module's module.xml file:

```
<module name="Magento_Cms" setup_version="2.0.0">
    <sequence>
        <module name="Magento_Store"/>
        <module name="Magento_Theme"/>
    </sequence>
</module>
```

Processed module versions are registered in the `setup_module` table:

module	schema_version	data_version
Magento_AdminNotification	2.0.0	2.0.0
Magento_Authorization	2.0.0	2.0.0
Magento_Backend	2.0.0.0	2.0.0.0
Magento_Backup	1.6.0.0	1.6.0.0

HOME
Magento U

Notes:

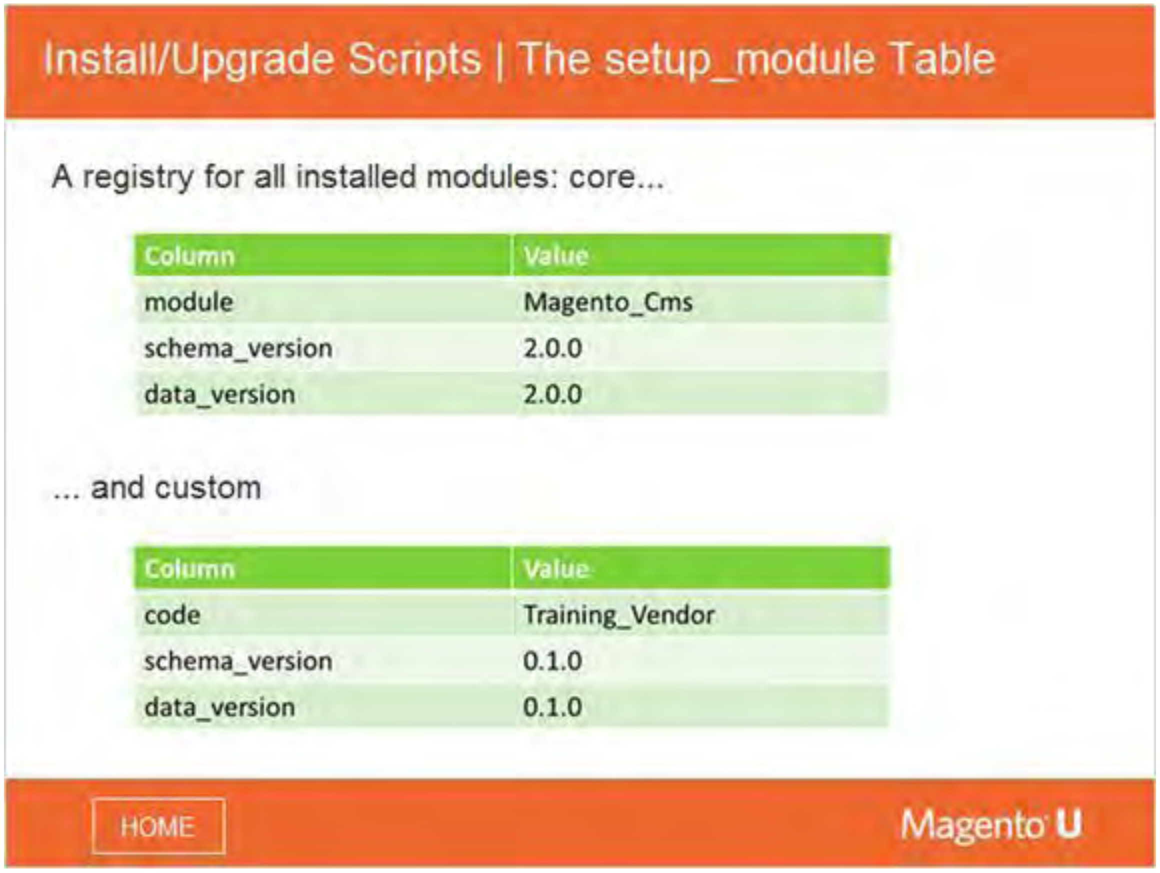
For setup scripts to be evaluated for a module, a setup version needs to be declared.

To install a module:

1. Disable the cache under System > Cache Management.
2. In the base directory, run `bin/magento setup:upgrade`.
3. In the base directory, run `bin/magento module:enable --clear-static-content Module_Name`.
4. Under Stores > Configuration > Advanced > Advanced, check that the module is present, where Module_Name is the name of the module that you are enabling.

The module is now installed. The module version is used to determine which setup scripts to apply.

4.5 Install/Upgrade Scripts | The setup_module Table



Install/Upgrade Scripts | The setup_module Table

A registry for all installed modules: core...

Column	Value
module	Magento_Cms
schema_version	2.0.0
data_version	2.0.0

... and custom

Column	Value
code	Training_Vendor
schema_version	0.1.0
data_version	0.1.0

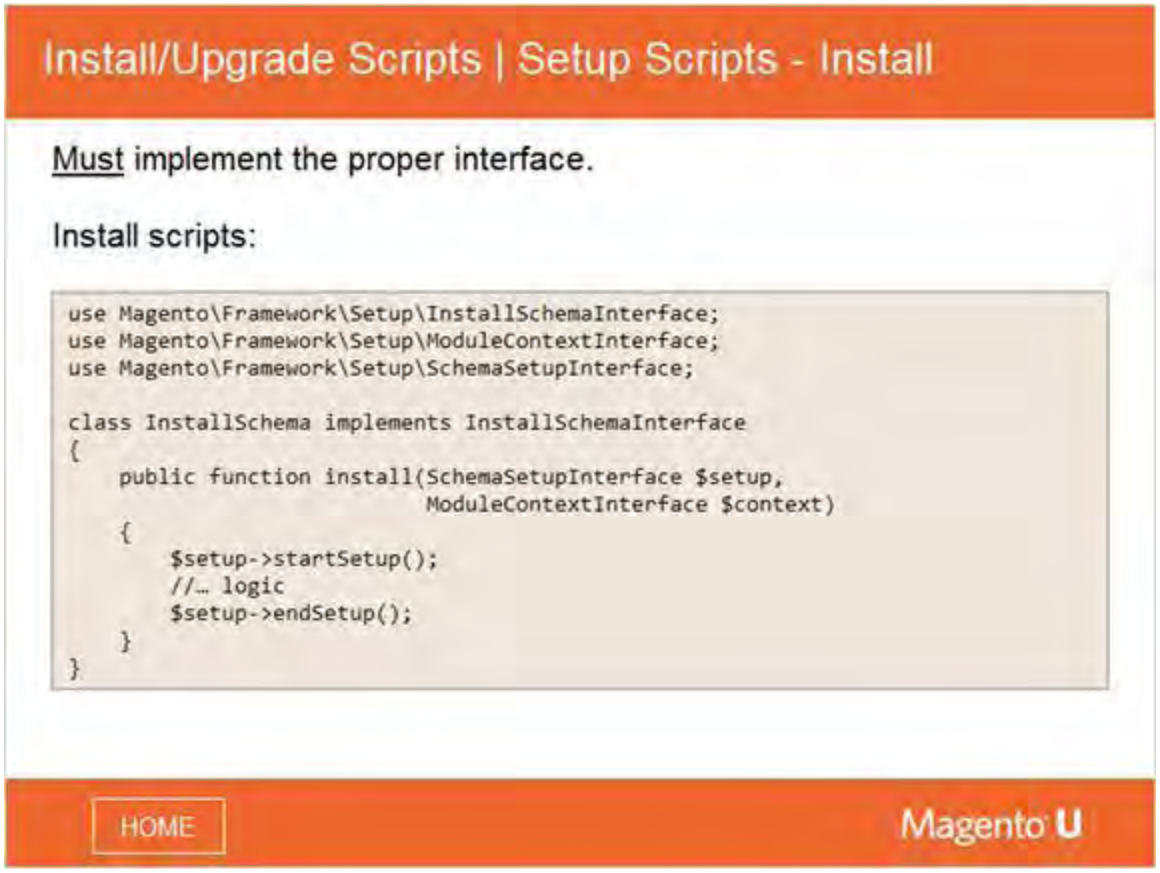
HOME
Magento U

Notes:

Each installed module is recorded with the last known version in the `setup_module` database table.

Note: The schema version and the `data_version` are always the same, unless an error occurs while running a setup script.

4.6 Install/Upgrade Scripts | Setup Scripts - Install



Install/Upgrade Scripts | Setup Scripts - Install

Must implement the proper interface.

Install scripts:

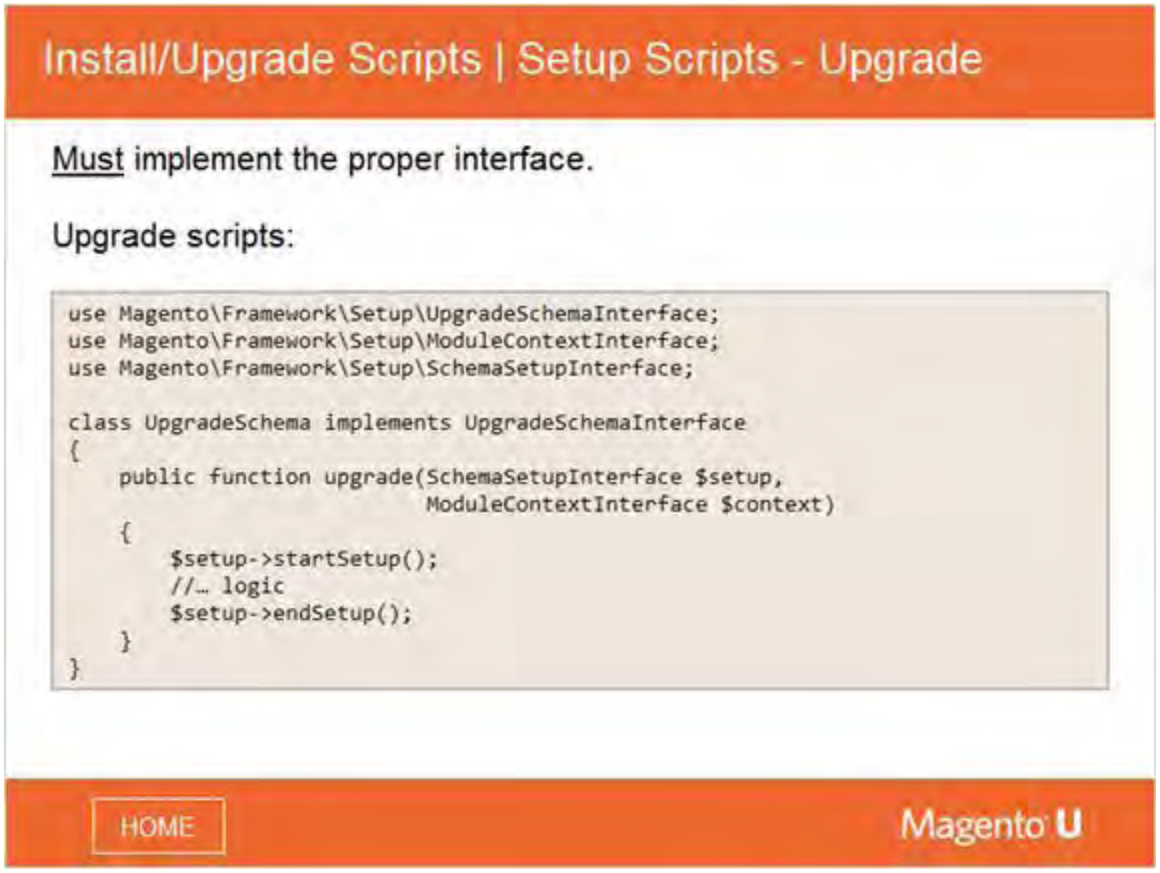
```
use Magento\Framework\Setup\InstallSchemaInterface;
use Magento\Framework\Setup\ModuleContextInterface;
use Magento\Framework\Setup\SchemaSetupInterface;

class InstallSchema implements InstallSchemaInterface
{
    public function install(SchemaSetupInterface $setup,
                            ModuleContextInterface $context)
    {
        $setup->startSetup();
        //… logic
        $setup->endSetup();
    }
}
```

Notes:

This is the method that will contain any `CREATE TABLE` or other SQL code that needs to be run to initialize a new module.

The syntax is the same as in Magento 1.

4.7 Install/Upgrade Scripts | Setup Scripts - Upgrade



Install/Upgrade Scripts | Setup Scripts - Upgrade

Must implement the proper interface.

Upgrade scripts:

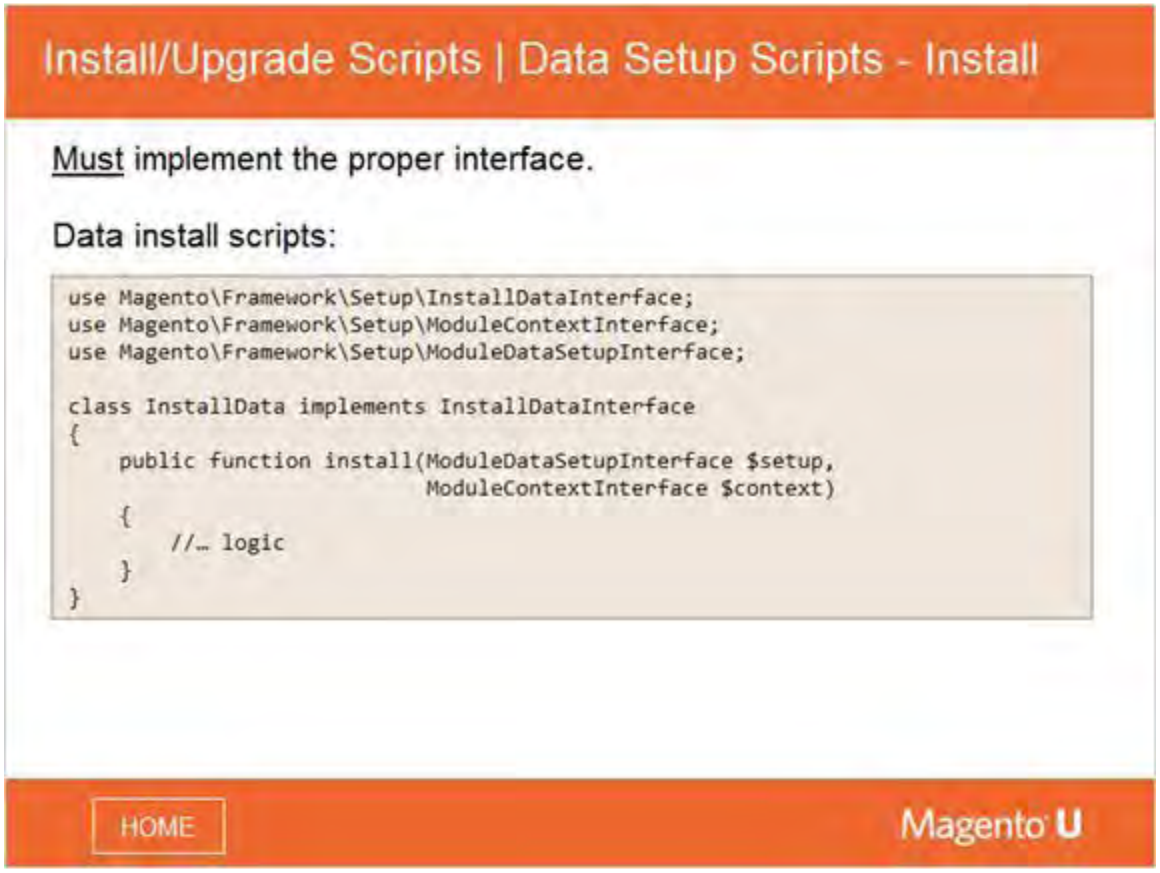
```
use Magento\Framework\Setup\UpgradeSchemaInterface;
use Magento\Framework\Setup\ModuleContextInterface;
use Magento\Framework\Setup\SchemaSetupInterface;

class UpgradeSchema implements UpgradeSchemaInterface
{
    public function upgrade(SchemaSetupInterface $setup,
                            ModuleContextInterface $context)
    {
        $setup->startSetup();
        //… logic
        $setup->endSetup();
    }
}
```

Notes:

Magento’s setup supports a simple versioning scheme that will let you automatically run scripts to upgrade your modules.

Once Magento runs an install script for a module, it will never run another install for that module again (short of manually deleting the reference in the setup_module table). Instead, you’ll need to create an upgrade script. Upgrade scripts are very similar to install scripts, with a few key differences.

4.8 Install/Upgrade Scripts | Data Setup Scripts - Install



Install/Upgrade Scripts | Data Setup Scripts - Install

Must implement the proper interface.

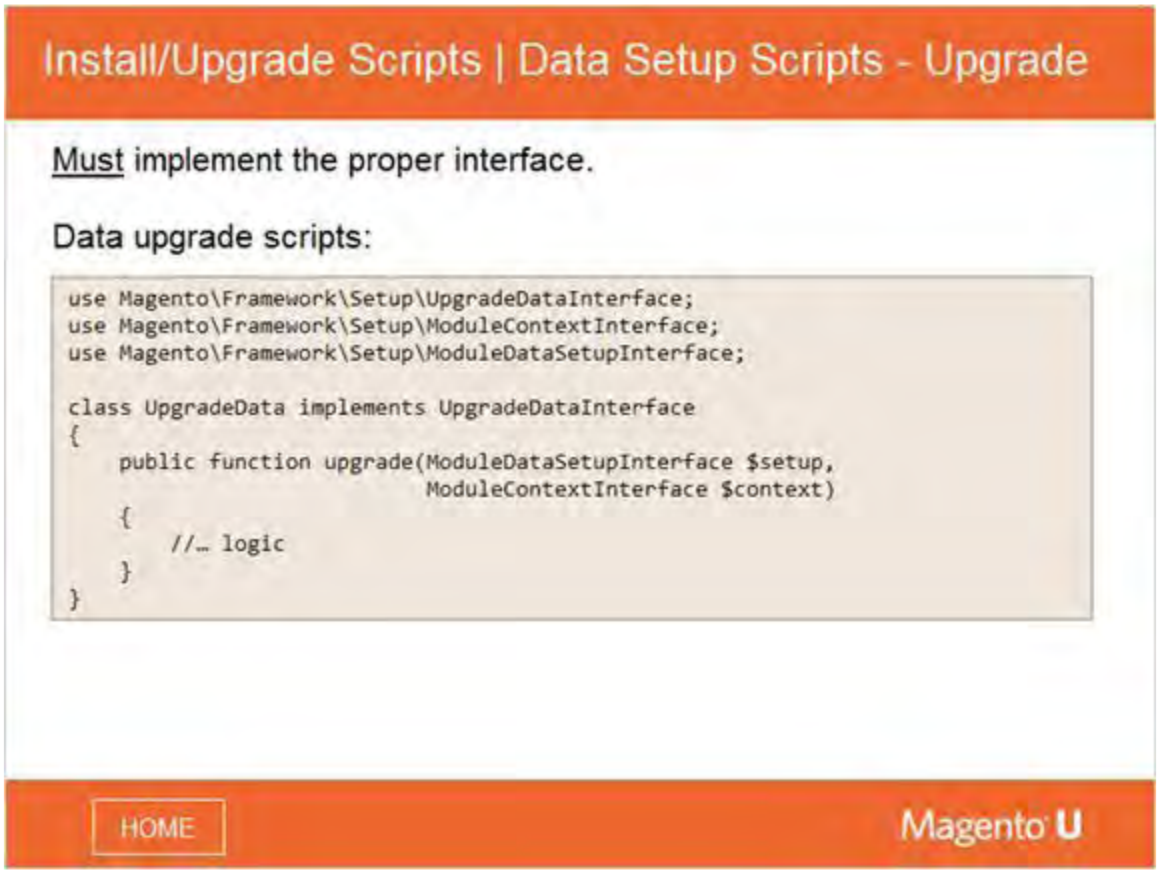
Data install scripts:

```
use Magento\Framework\Setup\InstallDataInterface;
use Magento\Framework\Setup\ModuleContextInterface;
use Magento\Framework\Setup\ModuleDataSetupInterface;

class InstallData implements InstallDataInterface
{
    public function install(ModuleDataSetupInterface $setup,
                            ModuleContextInterface $context)
    {
        //… logic
    }
}
```

Notes:

You can split the database structure modifications from the data insertion by using data setup scripts. The data setup scripts will work in about the same way as your schema setup scripts and have the same versioning conventions.

4.9 Install/Upgrade Scripts | Data Setup Scripts - Upgrade



Install/Upgrade Scripts | Data Setup Scripts - Upgrade

Must implement the proper interface.

Data upgrade scripts:

```
use Magento\Framework\Setup\UpgradeDataInterface;
use Magento\Framework\Setup\ModuleContextInterface;
use Magento\Framework\Setup\ModuleDataSetupInterface;

class UpgradeData implements UpgradeDataInterface
{
    public function upgrade(ModuleDataSetupInterface $setup,
                            ModuleContextInterface $context)
    {
        //… logic
    }
}
```

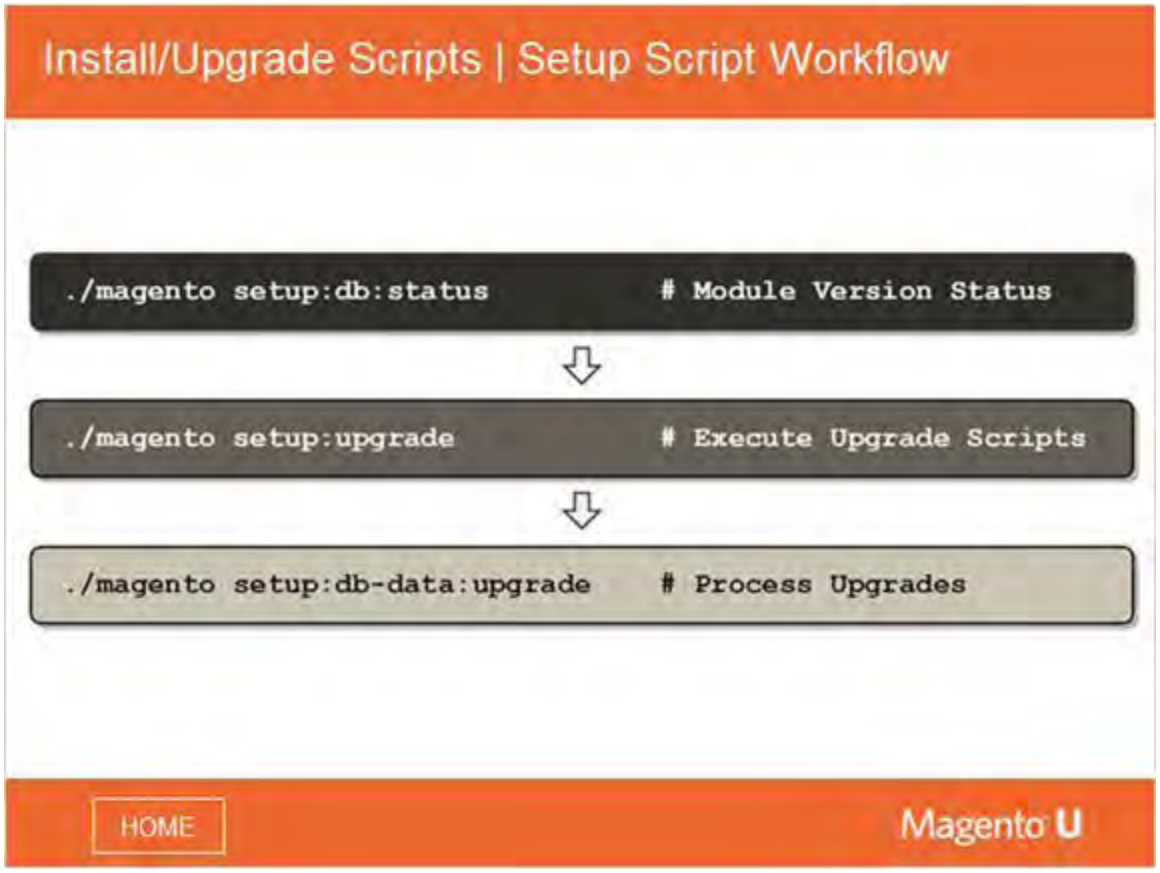
Notes:

In addition to the setup resource model, the setup application provides the module context to every install or upgrade script.

The context provides such information as current module version, so the install or upgrade class can base its logic on this information.

```
if (version_compare($context->getVersion(), '1.0.0', '<')) {
```

4.10 Install/Upgrade Scripts | Setup Script Workflow



Install/Upgrade Scripts | Setup Script Workflow

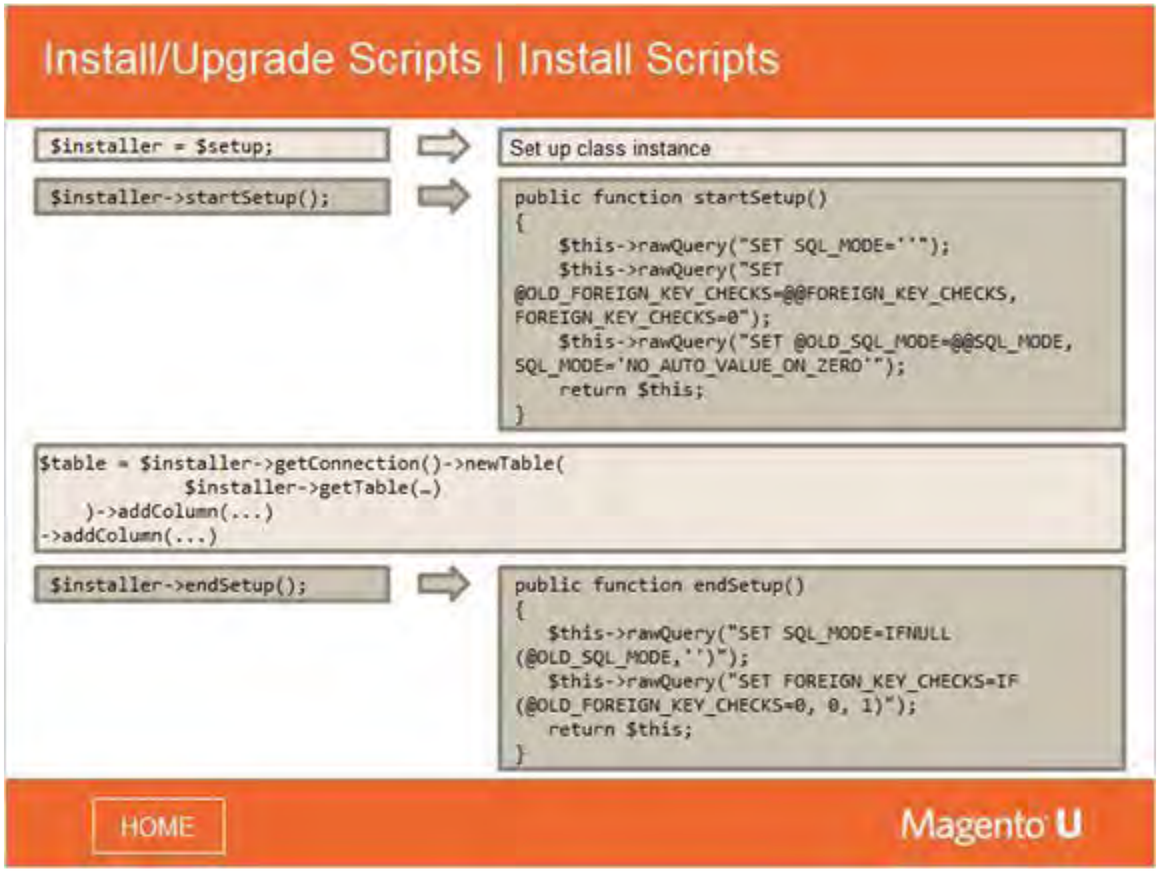
```
./magento setup:db:status              # Module Version Status
./magento setup:upgrade                # Execute Upgrade Scripts
./magento setup:db-data:upgrade        # Process Upgrades
```

Notes:

Setup scripts are only processed when:

- A module's setup version is declared.
- The configuration cache has been refreshed or is turned off.
- The upgrade command is run in the base directory `bin/magento setup:upgrade`.

4.11 Install/Upgrade Scripts | Install Scripts


Install/Upgrade Scripts | Install Scripts

```
$installer = $setup;
```

Set up class instance

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

```
public function startSetup()
{
    $this->rawQuery("SET SQL_MODE=''");
    $this->rawQuery("SET
@OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0");
    $this->rawQuery("SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO'");
    return $this;
}
```

```
$table = $installer->getConnection()->newTable(
            $installer->getTable(…)
    )->addColumn(...)
->addColumn(...)
```

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

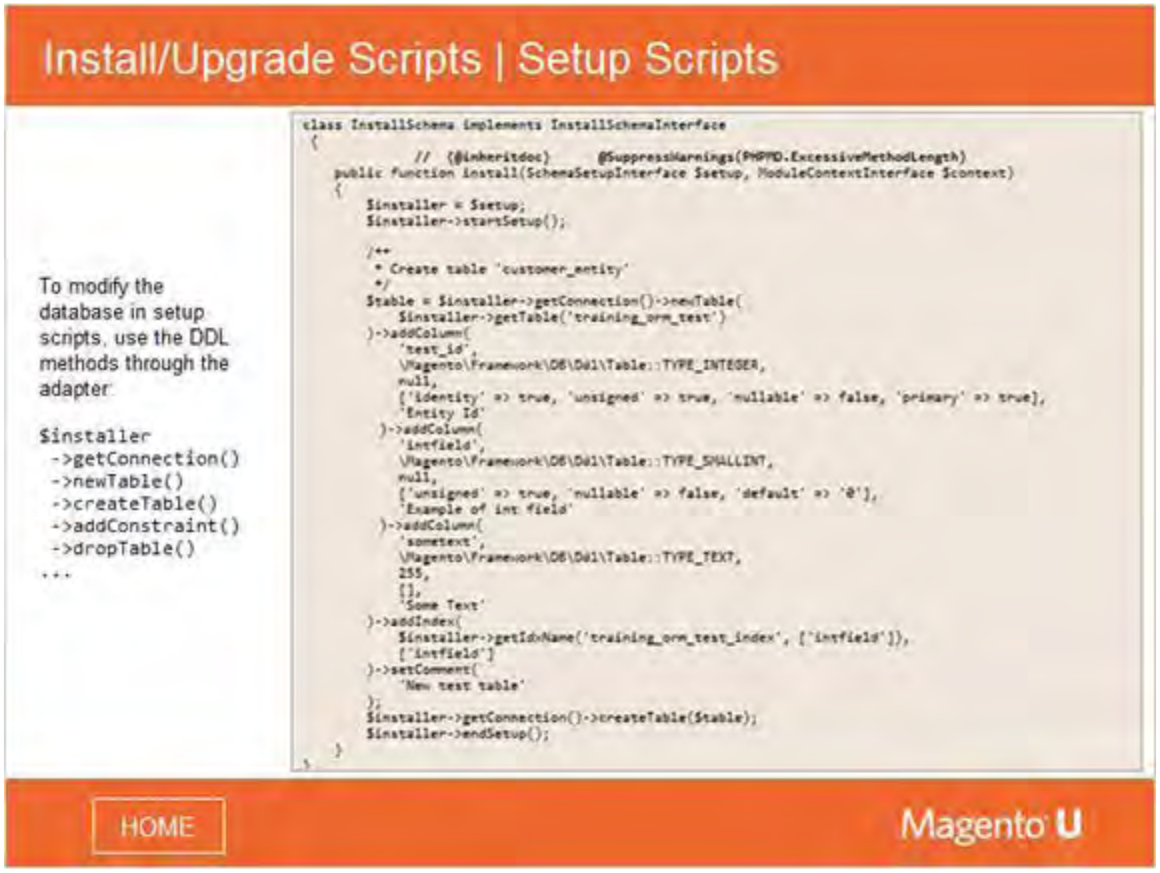
```
public function endSetup()
{
   $this->rawQuery("SET SQL_MODE=IFNULL
(@OLD_SQL_MODE,'')");
   $this->rawQuery("SET FOREIGN_KEY_CHECKS=IF
(@OLD_FOREIGN_KEY_CHECKS=0, 0, 1)");
   return $this;
}
```

Notes:

Every setup method should start with `$installer->startSetup()` and end with `$installer->endSetup()`. Some database-specific preparations will be executed.

Even if your setup script doesn't change the database structure, **it still is a best practice** to wrap your logic with these two methods in case additional functionality is added to Magento in the future.

4.12 Install/Upgrade Scripts | Setup Scripts



Install/Upgrade Scripts | Setup Scripts

To modify the database in setup scripts, use the DDL methods through the adapter:

```
$installer
  ->getConnection()
  ->newTable()
  ->createTable()
  ->addConstraint()
  ->dropTable()
...
```

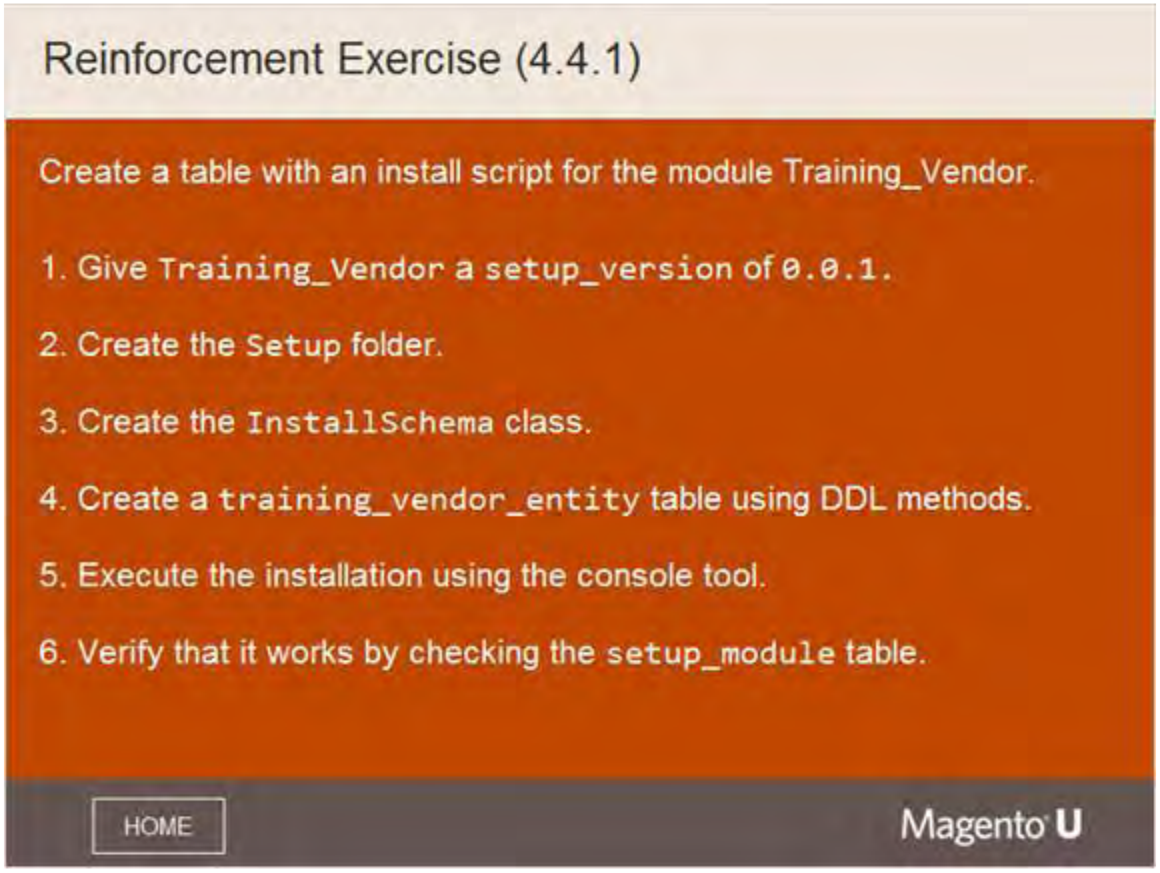
```
class InstallSchema implements InstallSchemaInterface
 {
            //  {@inheritdoc}     @SuppressWarnings(PHPMD.ExcessiveMethodLength)
    public function install(SchemaSetupInterface $setup, ModuleContextInterface $context)
    {
        $installer = $setup;
        $installer->startSetup();

        /**
         * Create table 'customer_entity'
         */
        $table = $installer->getConnection()->newTable(
            $installer->getTable('training_orm_test')
        )->addColumn(
            'test_id',
            \Magento\Framework\DB\Ddl\Table::TYPE_INTEGER,
            null,
            ['identity' => true, 'unsigned' => true, 'nullable' => false, 'primary' => true],
            'Entity Id'
         )->addColumn(
            'intfield',
            \Magento\Framework\DB\Ddl\Table::TYPE_SMALLINT,
            null,
            ['unsigned' => true, 'nullable' => false, 'default' => '0'],
            'Example of int field'
         )->addColumn(
            'sometext',
            \Magento\Framework\DB\Ddl\Table::TYPE_TEXT,
            255,
            [],
            'Some Text'
        )->addIndex(
            $installer->getIdxName('training_orm_test_index', ['intfield']),
            ['intfield']
        )->setComment(
            'New test table'
        );
        $installer->getConnection()->createTable($table);
        $installer->endSetup();
    }
}
```

Notes:

There are two ways to perform required changes to a database:

- **Plain SQL:** The `run()` method accepts a string containing the (multiline) SQL needed to set up your database table(s). This approach can be used in RDBMS-specific setup scripts.
- **Using DDL operations through the adapter:** The database adapter instance that is available through `$installer->getConnection()` provides all necessary methods to manipulate the database in an RDBMS transparent way.

4.13 Reinforcement Exercise (4.4.1)

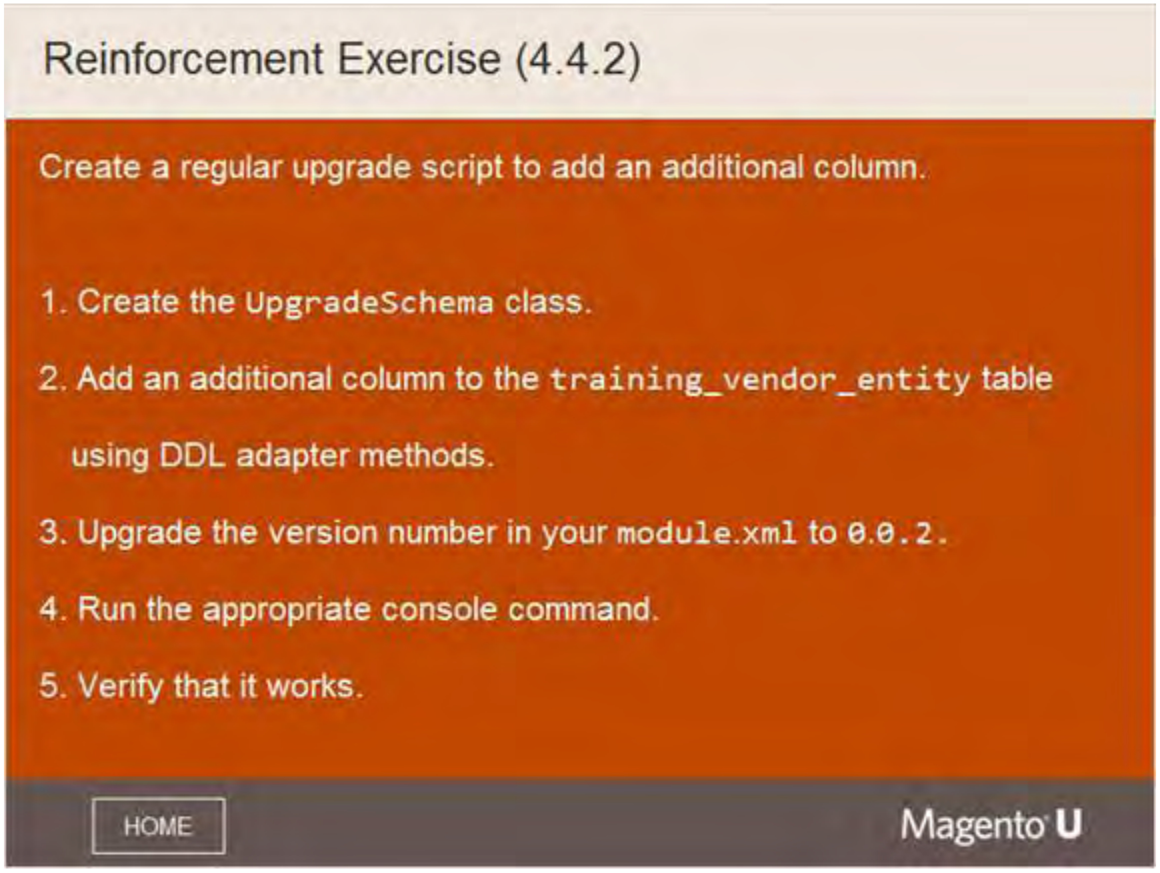


Reinforcement Exercise (4.4.1)

Create a table with an install script for the module Training_Vendor.

1. Give `Training_Vendor` a `setup_version` of `0.0.1`.
2. Create the `Setup` folder.
3. Create the `InstallSchema` class.
4. Create a `training_vendor_entity` table using DDL methods.
5. Execute the installation using the console tool.
6. Verify that it works by checking the `setup_module` table.

4.14 Reinforcement Exercise (4.4.2)



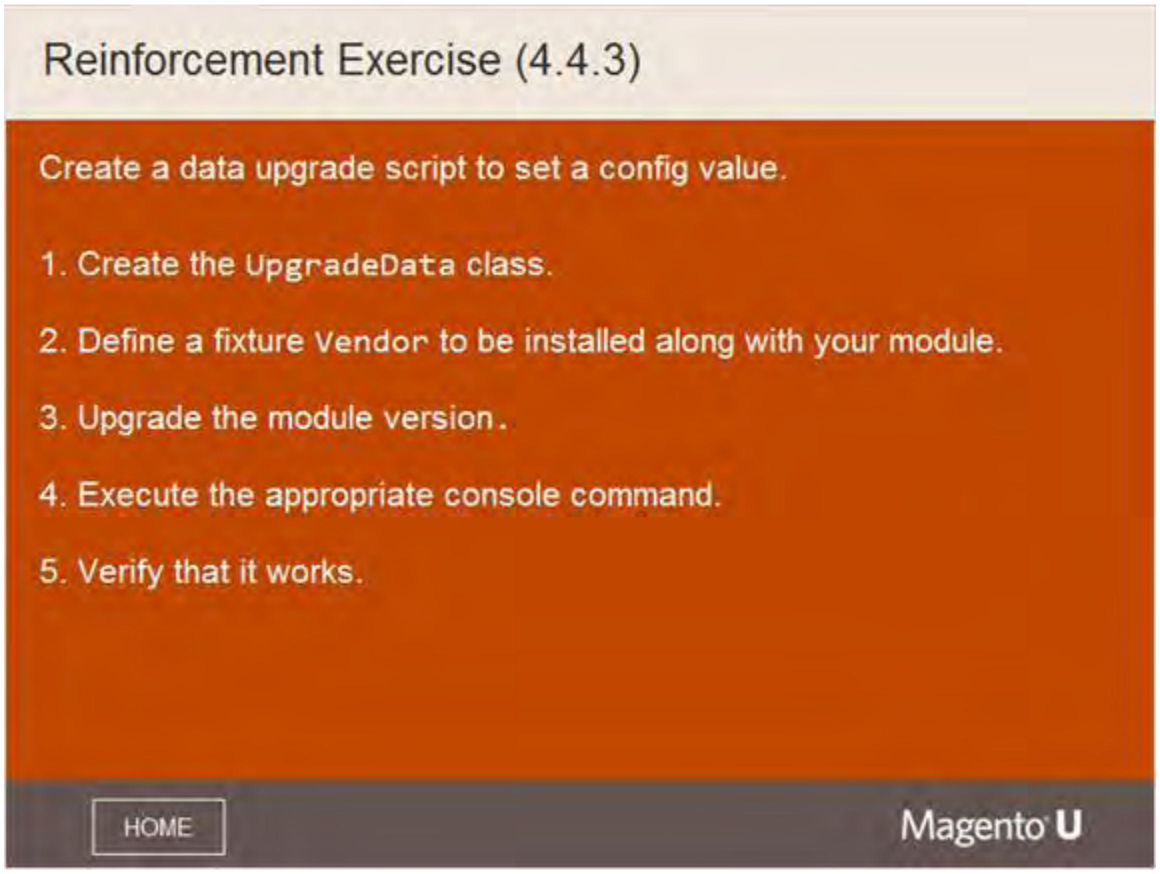
Reinforcement Exercise (4.4.2)

Create a regular upgrade script to add an additional column.

1. Create the `UpgradeSchema` class.
2. Add an additional column to the `training_vendor_entity` table using DDL adapter methods.
3. Upgrade the version number in your `module.xml` to `0.0.2`.
4. Run the appropriate console command.
5. Verify that it works.

HOME
Magento U

4.15 Reinforcement Exercise (4.4.3)



Reinforcement Exercise (4.4.3)

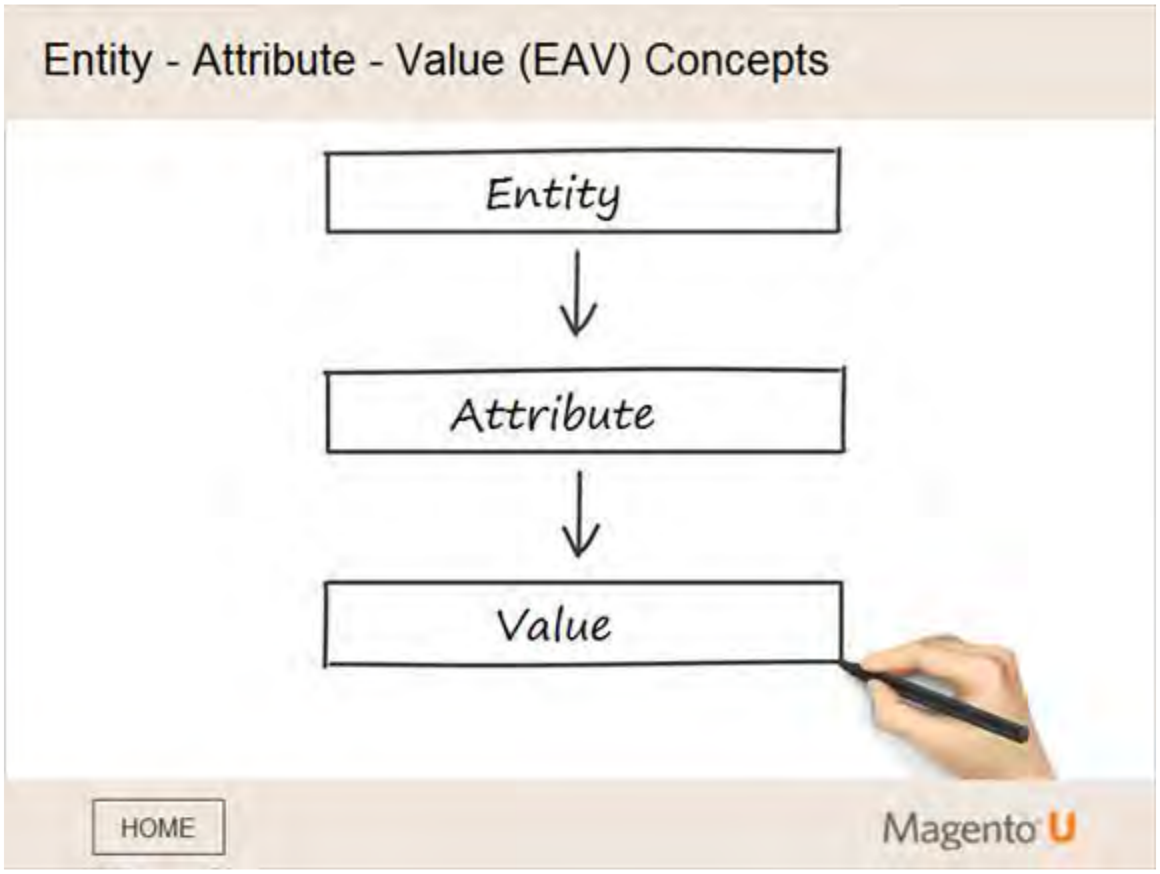
Create a data upgrade script to set a config value.

1. Create the `UpgradeData` class.
2. Define a fixture `Vendor` to be installed along with your module.
3. Upgrade the module version.
4. Execute the appropriate console command.
5. Verify that it works.

HOME
Magento U

5. Entity Attribute Value

5.1 Entity - Attribute - Value (EAV) Concepts

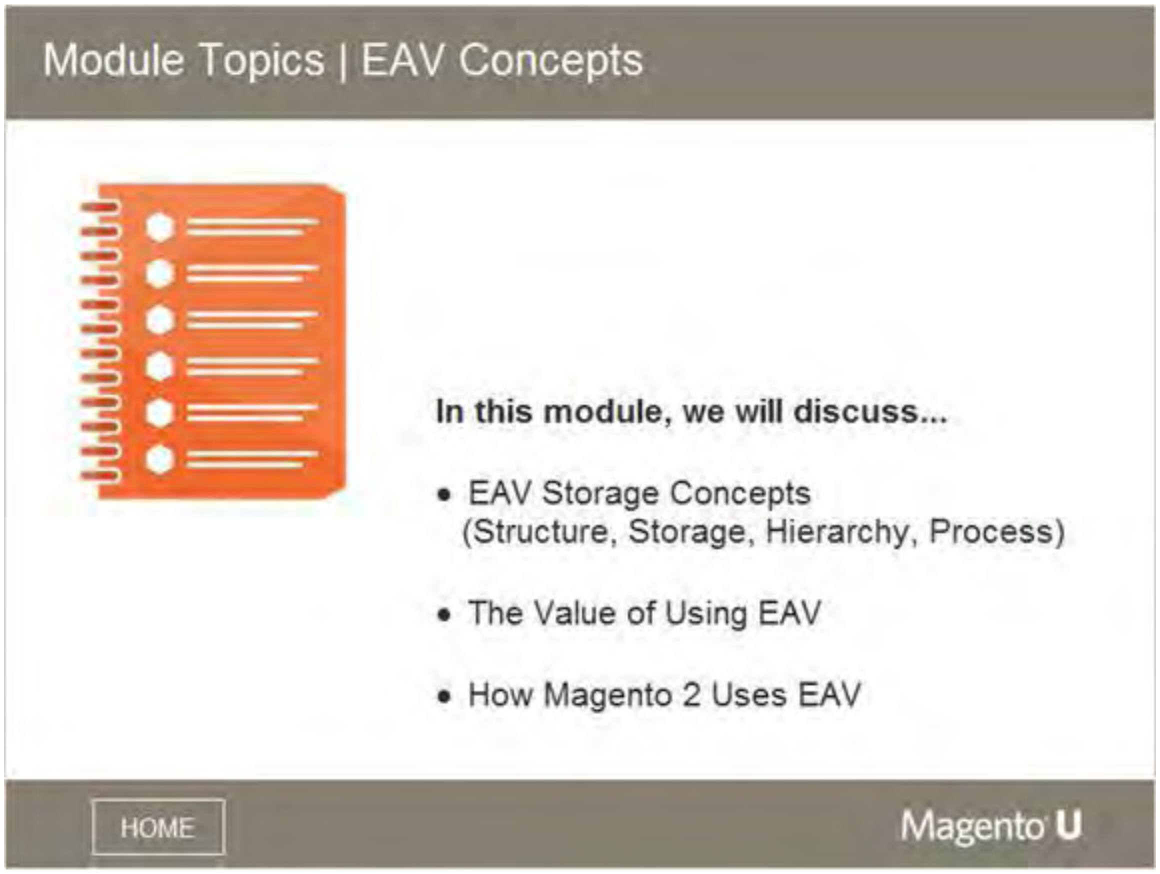


Notes:

We now shift to a discussion of Entity - Attribute - Value concepts in Magento 2.

This module covers the EAV database storage approach -- which tables are part of the EAV concept and how these tables are related to each other. This module also describes the PHP classes that make up the EAV implementation and how they interact.

5.2 Module Topics | EAV Concepts



Module Topics | EAV Concepts

In this module, we will discuss...

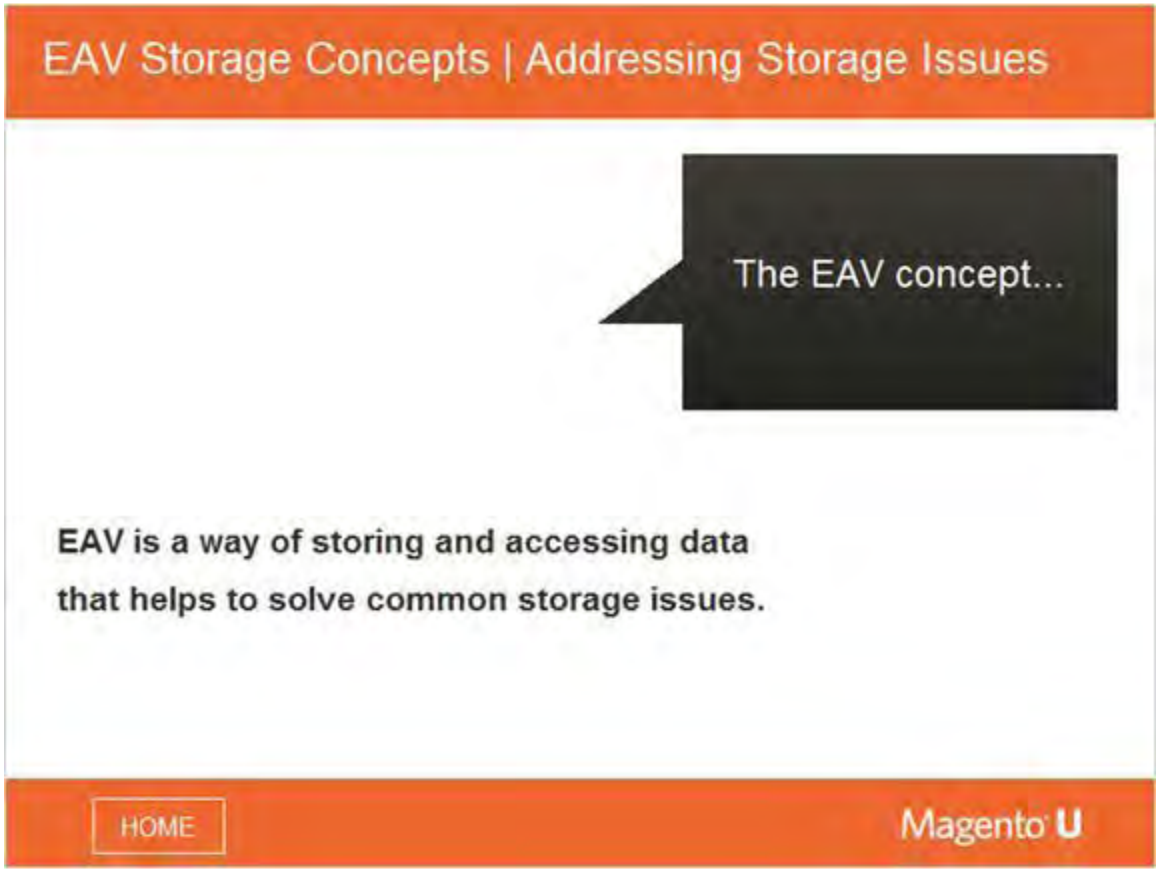
- EAV Storage Concepts (Structure, Storage, Hierarchy, Process)
- The Value of Using EAV
- How Magento 2 Uses EAV

Notes:

In this module, we will discuss:

- EAV storage concepts: The table structure, storage concepts, class hierarchy, and the process.
- The value of using EAV.
- How Magento 2 uses EAV.

5.3 EAV Storage Concepts | Addressing Storage Issues



EAV Storage Concepts | Addressing Storage Issues

The EAV concept...

EAV is a way of storing and accessing data that helps to solve common storage issues.

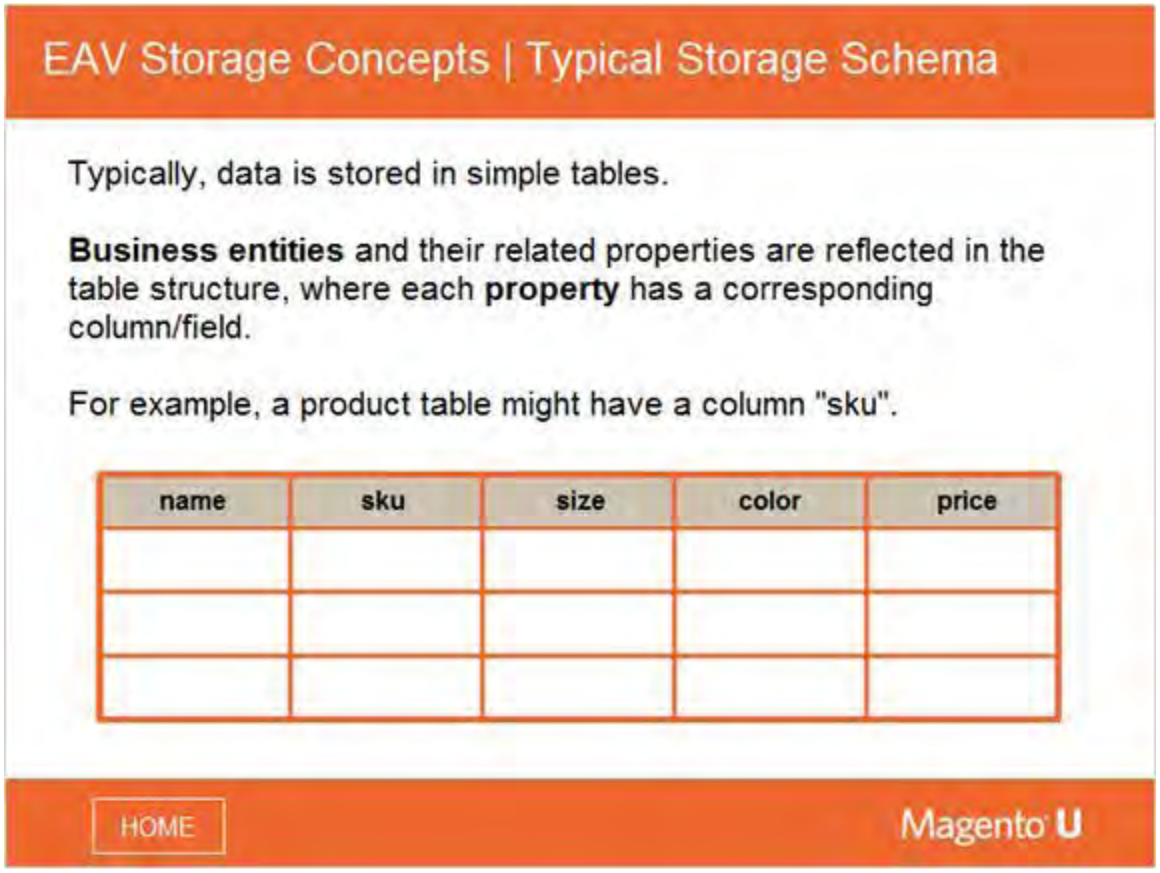
HOME
Magento U

Notes:

Magento engineers have implemented an object attribute to the table column mapping model to accommodate a wide range of requirements for the display and description of products, meeting the needs of each website as well as multiple ranges of storefronts.

In practice, this means that the Magento EAV model spreads out the entity and attribute descriptions over a number of tables. This also provides the flexibility to expand available attributes for products (and other entities) to meet future website expansion.

5.4 EAV Storage Concepts | Typical Storage Schema



EAV Storage Concepts | Typical Storage Schema

Typically, data is stored in simple tables.

Business entities and their related properties are reflected in the table structure, where each **property** has a corresponding column/field.

For example, a product table might have a column "sku".

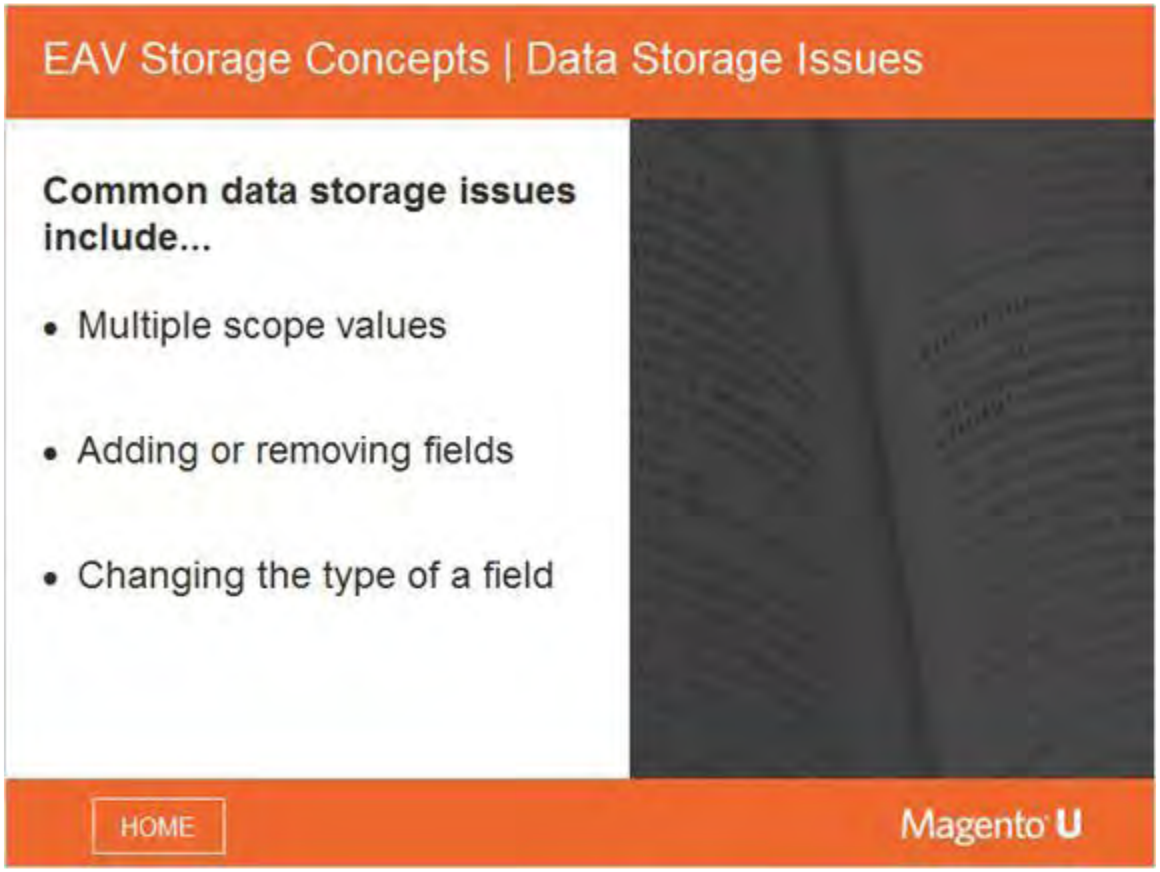
name	sku	size	color	price

HOME
Magento U

Notes:

Typically we store data in plain tables like the one shown. Business entities and their properties (for example, products and their SKUs) are reflected in the table structure, where each property has a corresponding column.

5.5 EAV Storage Concepts | Data Storage Issues



EAV Storage Concepts | Data Storage Issues

Common data storage issues include...

- Multiple scope values
- Adding or removing fields
- Changing the type of a field

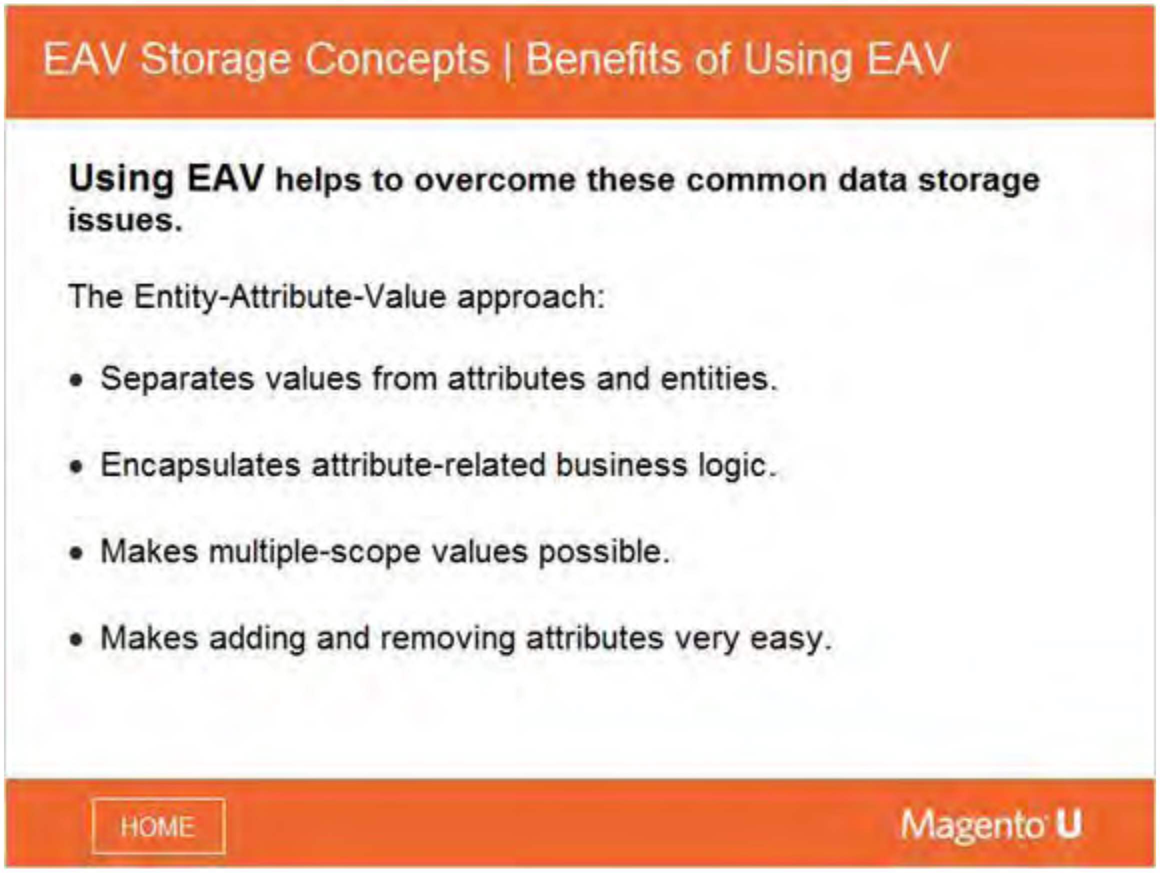
HOME
Magento U

Notes:

Some of the common data storage issues that confront developers are:

- **Multiple scope values:** It is cumbersome to represent different values for multiple scopes (global, website, store view) when using a flat table structure.
- **Adding/removing new fields:** With a flat table, adding an attribute means altering the table to add a new column.
- **Changing the type of fields:** Changing the field type (int, text, varchar, and so on) also means altering the table.

5.6 EAV Storage Concepts | Benefits of Using EAV


HOME
Magento U

EAV Storage Concepts | Benefits of Using EAV

Using EAV helps to overcome these common data storage issues.

The Entity-Attribute-Value approach:

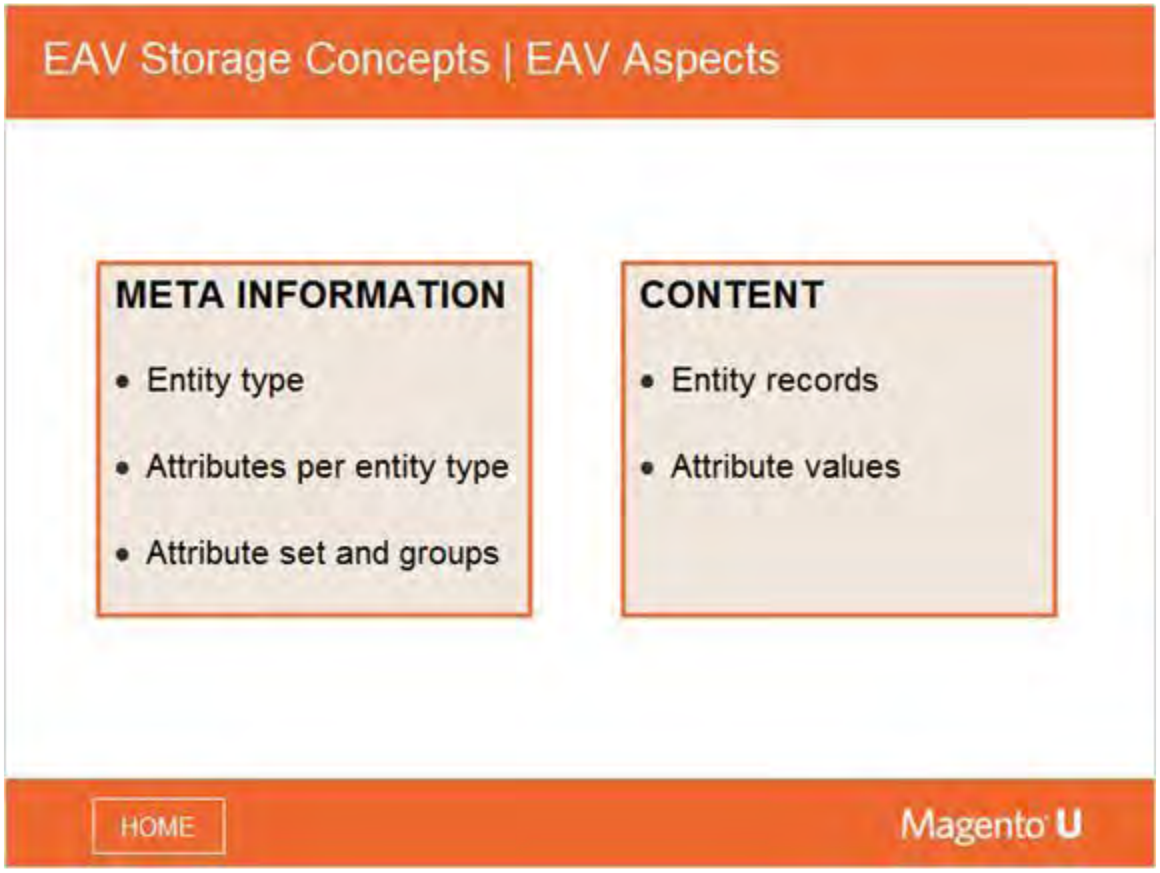
- Separates values from attributes and entities.
- Encapsulates attribute-related business logic.
- Makes multiple-scope values possible.
- Makes adding and removing attributes very easy.

Notes:

EAV helps to overcome these common data storage issues because it:

- Separates values from attributes and entities.
- Encapsulates attribute-related business logic.
- Makes multiple-scope values easier.
- Makes adding and removing attributes very easy.

5.7 EAV Storage Concepts | EAV Aspects



EAV Storage Concepts | EAV Aspects

META INFORMATION

- Entity type
- Attributes per entity type
- Attribute set and groups

CONTENT

- Entity records
- Attribute values

HOME
Magento U

Notes:

One aspect of EAV is managing relationships -- that is, **meta information** -- about available entities.

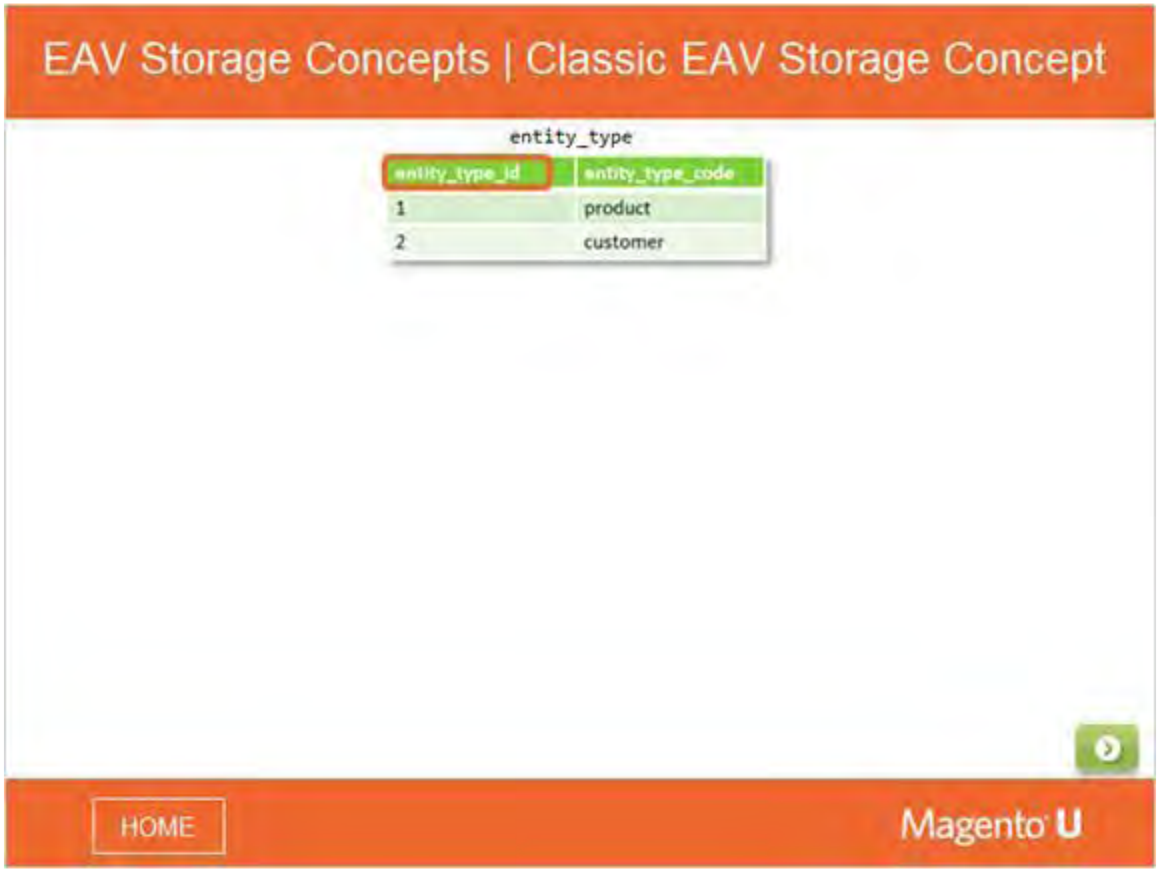
The meta information includes:

- Entity type
- Attributes per entity type
- Attribute set and groups

The other aspect is managing content – how to store:

- Entity values
- Attribute values

5.8 EAV Storage Concepts | Classic EAV Storage Concept


EAV Storage Concepts | Classic EAV Storage Concept

entity_type

entity_type_id	entity_type_code
1	product
2	customer

Notes:

This slide shows the table relationships of the four main tables of a classic EAV implementation.

First, we have the entity type. This table has one record for each entity type that is available in the system; for example products, categories, and customers. Each entity type is identified by a unique `entity_type_id` and a unique human readable `entity_type_code`. Besides specifying what entities exist, there is no additional information about the entities available so far.

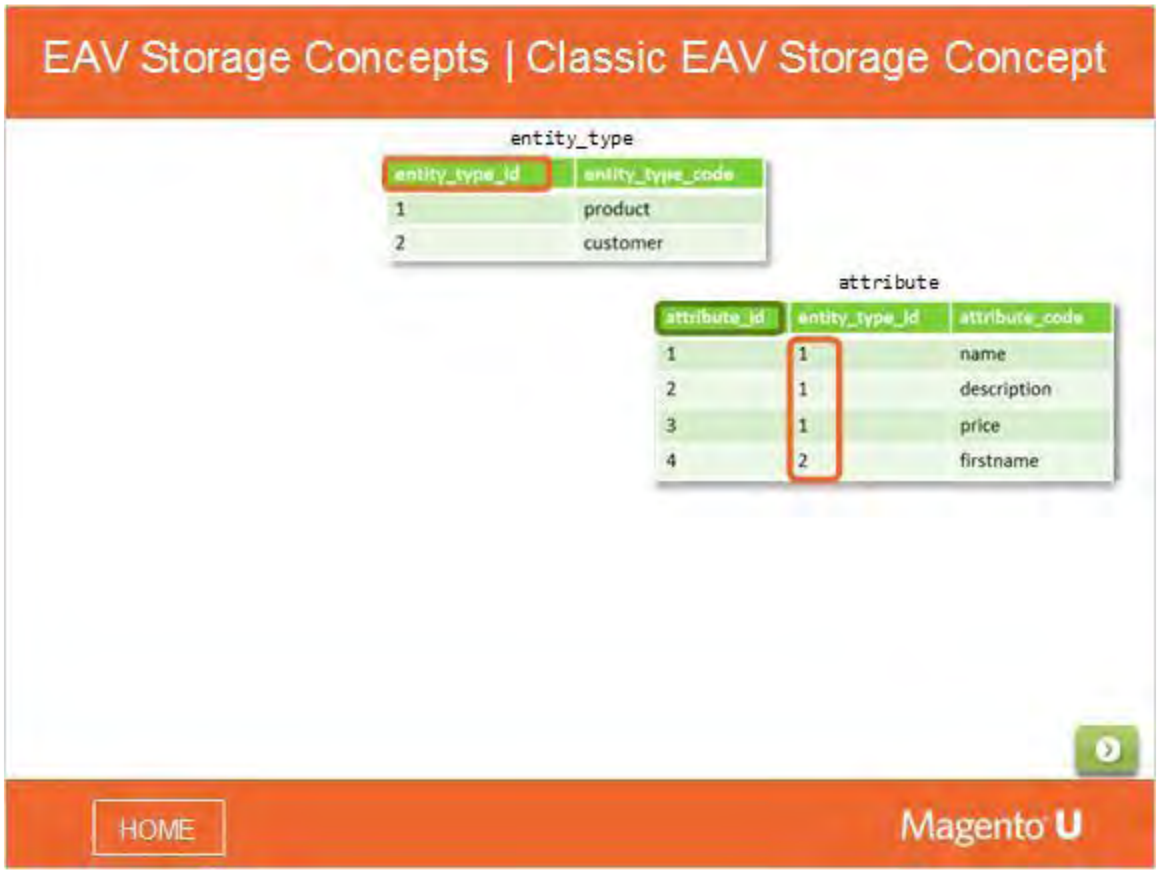
To store further information about entities, an attribute table is part of the system. Each record in the attribute table is linked to a specific entity type via the `entity_type_id` column. Each attribute is identified by a unique `attribute_id`, and a human readable `attribute_code`. The `attribute_code` has to be unique within a given `entity_type`. Given the example on the slide, we now know that products have a name, a description, and a price, and customers have a first name. But there still are no actual entities, only the meta information about entities.

To store actual entities, a content table is needed. To create an entity, a record is added to an entity table. A unique `entity_id` identifies each entity, and every record is linked to a specific `entity_type` record through a foreign key constraint on the `entity_type_id`. You must agree that there still is too little information stored about each of the entities to be useful, as so far they only consist of a numeric entity ID.

To store the actual attribute values, another table is needed: `attribute_value`. Each attribute value is linked to an entity and to the attribute. Now, we can determine that the `firstname` attribute value for the customer with the `entity ID 3` is `John`. Even though this storage schema is not completely normalized, it does not contain a lot of duplication. But it has a number of issues:

- The tables grow very large.
- Information about different entity types are all mixed together in the tables.
- MySQL is unable to create an efficient index on the `attribute_value.value` column, thus making queries by attribute value extremely slow.
- For humans it is hard to read the table.

attribute (Slide Layer)



EAV Storage Concepts | Classic EAV Storage Concept

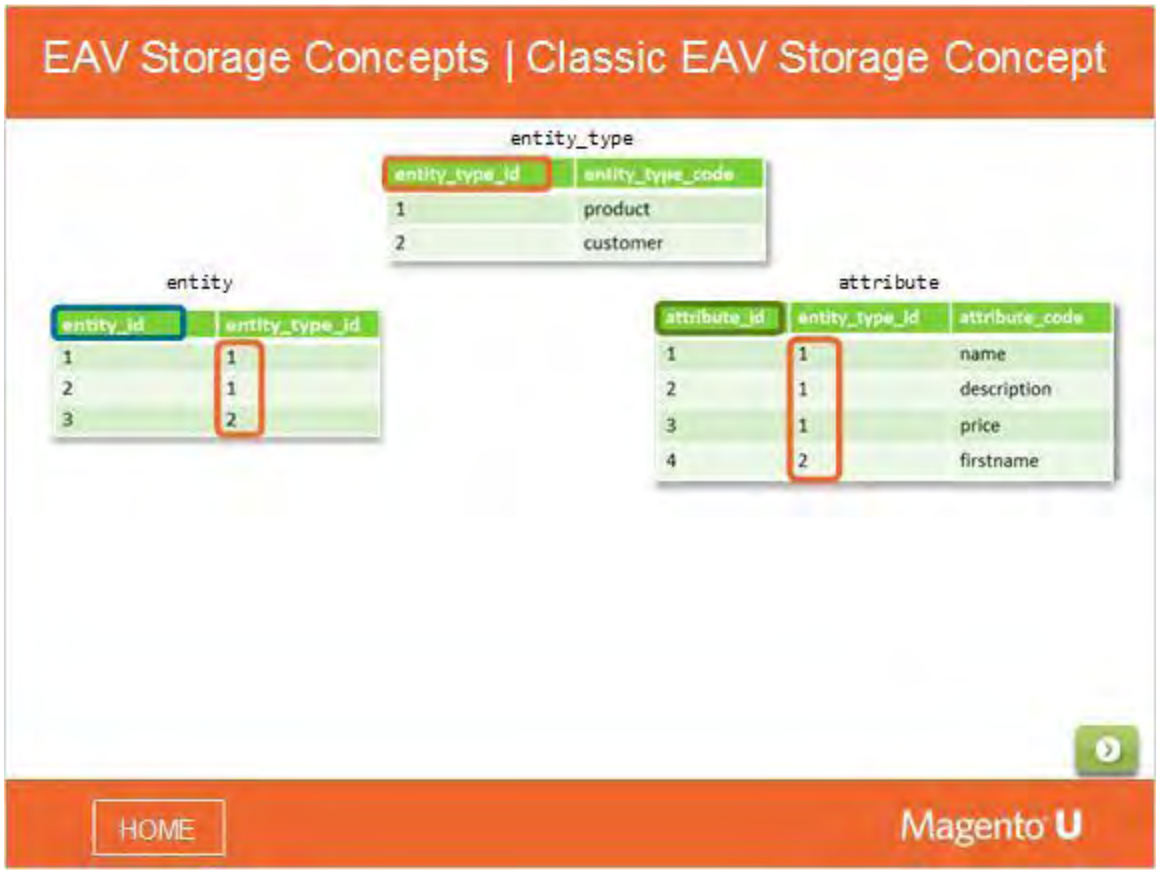
entity_type

entity_type_id	entity_type_code
1	product
2	customer

attribute

attribute_id	entity_type_id	attribute_code
1	1	name
2	1	description
3	1	price
4	2	firstname

entity (Slide Layer)



EAV Storage Concepts | Classic EAV Storage Concept

entity_type

entity_type_id	entity_type_code
1	product
2	customer

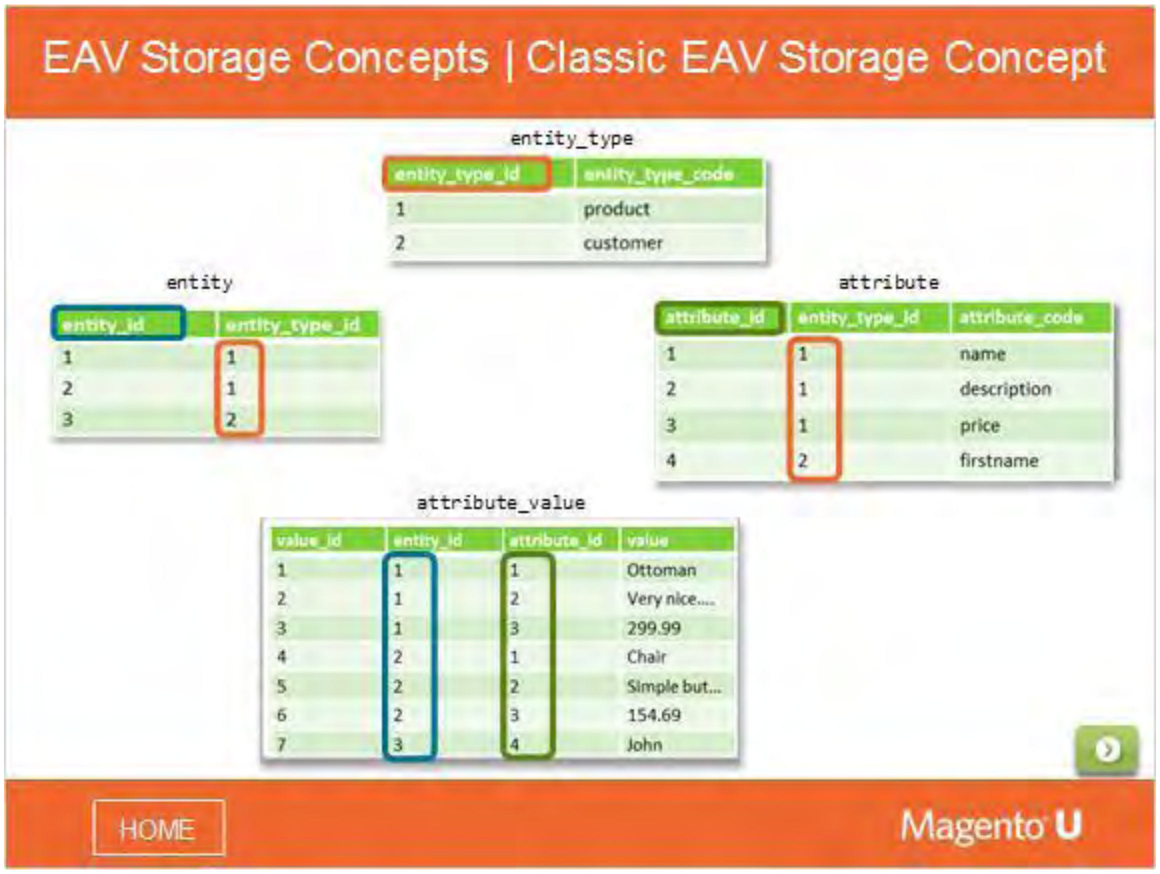
entity

entity_id	entity_type_id
1	1
2	1
3	2

attribute

attribute_id	entity_type_id	attribute_code
1	1	name
2	1	description
3	1	price
4	2	firstname

attribute value (Slide Layer)



EAV Storage Concepts | Classic EAV Storage Concept

entity_type

entity_type_id	entity_type_code
1	product
2	customer

entity

entity_id	entity_type_id
1	1
2	1
3	2

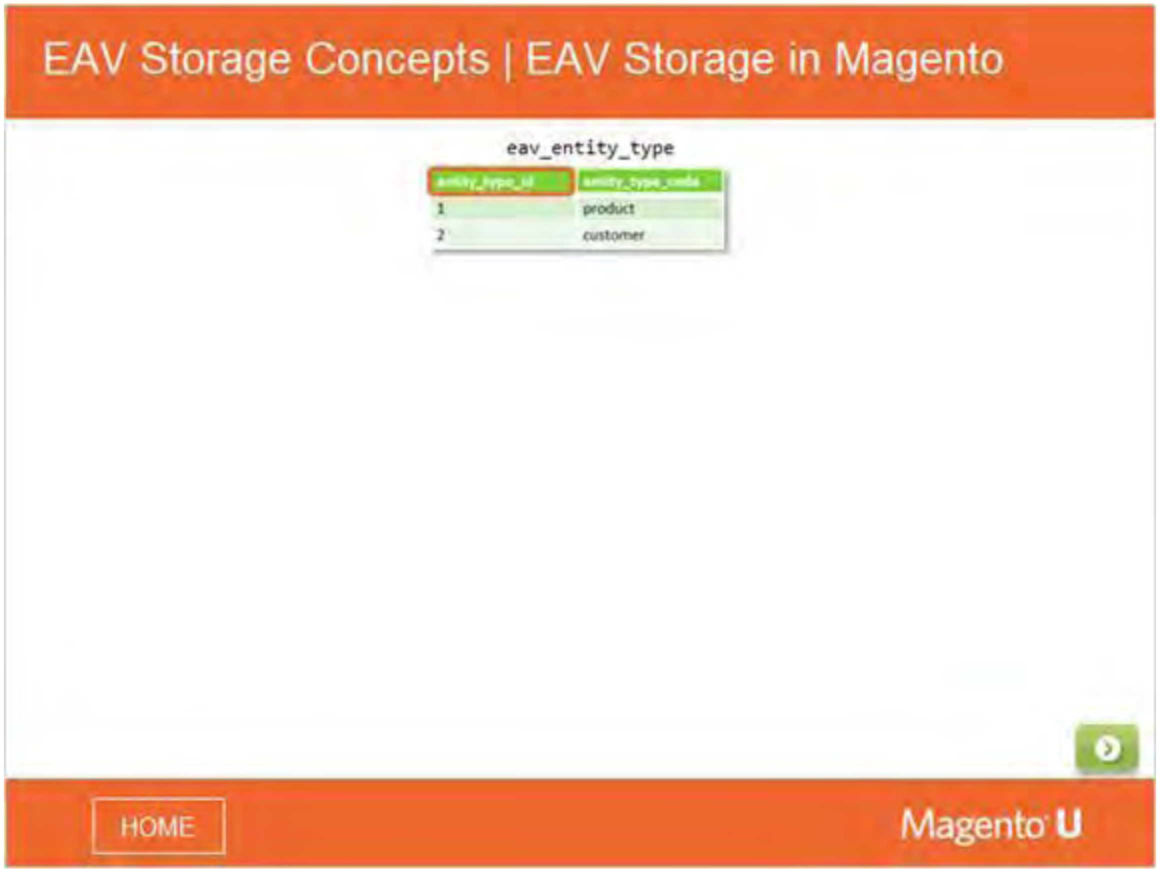
attribute

attribute_id	entity_type_id	attribute_code
1	1	name
2	1	description
3	1	price
4	2	firstname

attribute_value

value_id	entity_id	attribute_id	value
1	1	1	Ottoman
2	1	2	Very nice....
3	1	3	299.99
4	2	1	Chair
5	2	2	Simple but...
6	2	3	154.69
7	3	4	John

5.9 EAV Storage Concepts | EAV Storage in Magento



EAV Storage Concepts | EAV Storage in Magento

eav_entity_type

entity_type_id	entity_type_code
1	product
2	customer

Notes:

To resolve these issues, Magento engineers introduced some changes to the classic EAV storage model. This slide build walks you through a simplified Magento specific EAV implementation, pointing out the differences to the classic model. Please note that the real tables contain more columns, but in order to understand the EAV structure they are not important. We will discuss those later. It starts again with an `entity_type` table. So far there is no structural difference. Each entity type has an ID and a code.

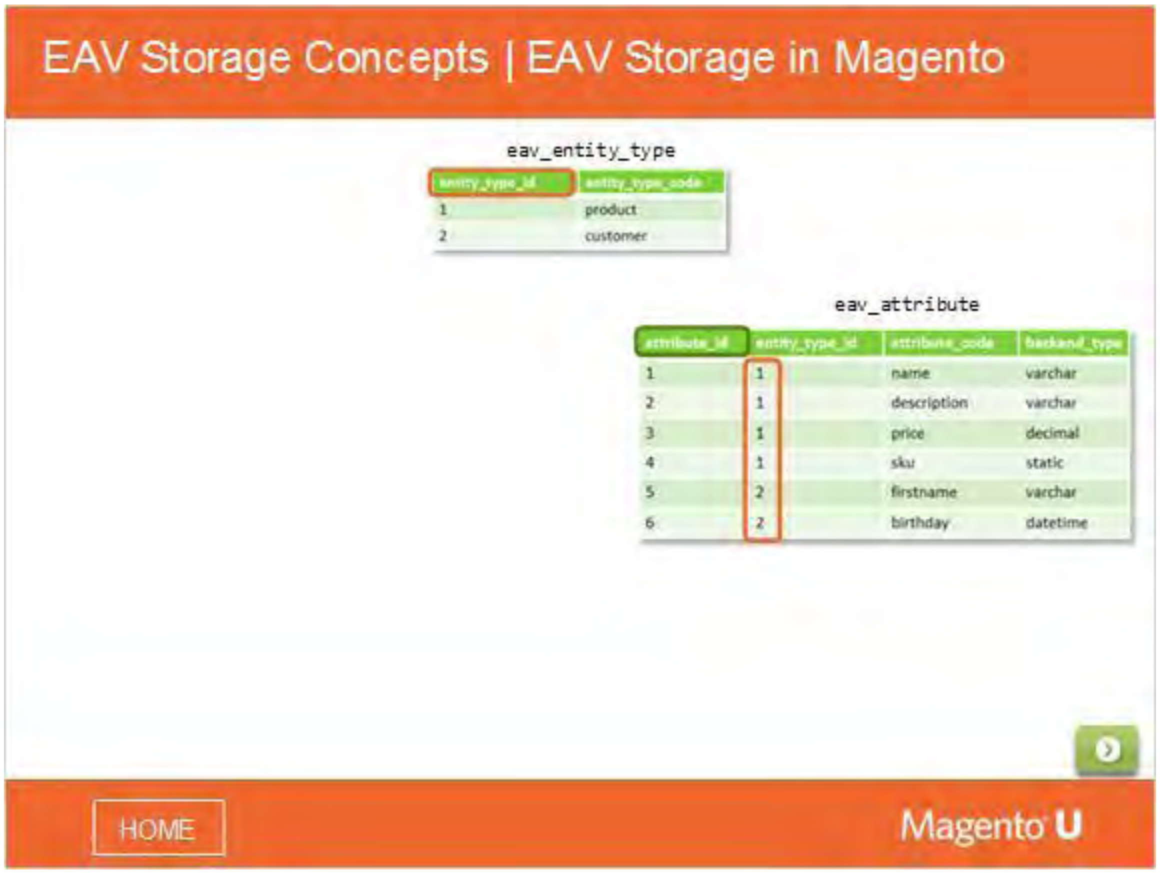
The `eav_attribute` table contains some small differences to the classic model. One of those is that each attribute's data type is recorded in a column called `backend_type`. Now the system knows that the name attribute is a varchar value, price is a decimal and birthday is a datetime value.

For storing the actual entities, Magento introduced the first big change: The entity table is now split by entity type. Each entity type has its own entity table. All product entities are stored in a table called `catalog_product_entity`, customer are stored in a table called customer_entity, and so forth. As a consequence, the number of tables grows, but the size of each table is smaller. Also it is easier for us humans to read the tables, because we now know what entity type we are dealing with when we are looking at a table. Another change Magento introduced is that the values of all attributes with the backend type static are stored in a dedicated column in the entity table. For products, for example, the attribute sku is static, meaning there is a sku column in the product entity table.

For the attribute values, Magento introduced the second big change from the classical EAV structure: The attribute value table is not only split by entity type, but also by backend type. For example, the values of customer attributes with the datetime backend type are stored in the table `customer_entity_datetime`. This means that the number of tables is even larger, but the number of records in each of the tables is a lot lower and the performance during conditional queries is a lot better because the database is able to create efficient indexes on the value columns.

Note that the columns of the attribute value tables are identical. The only difference is the data type of the value column.

eav_attribute (Slide Layer)



EAV Storage Concepts | EAV Storage in Magento

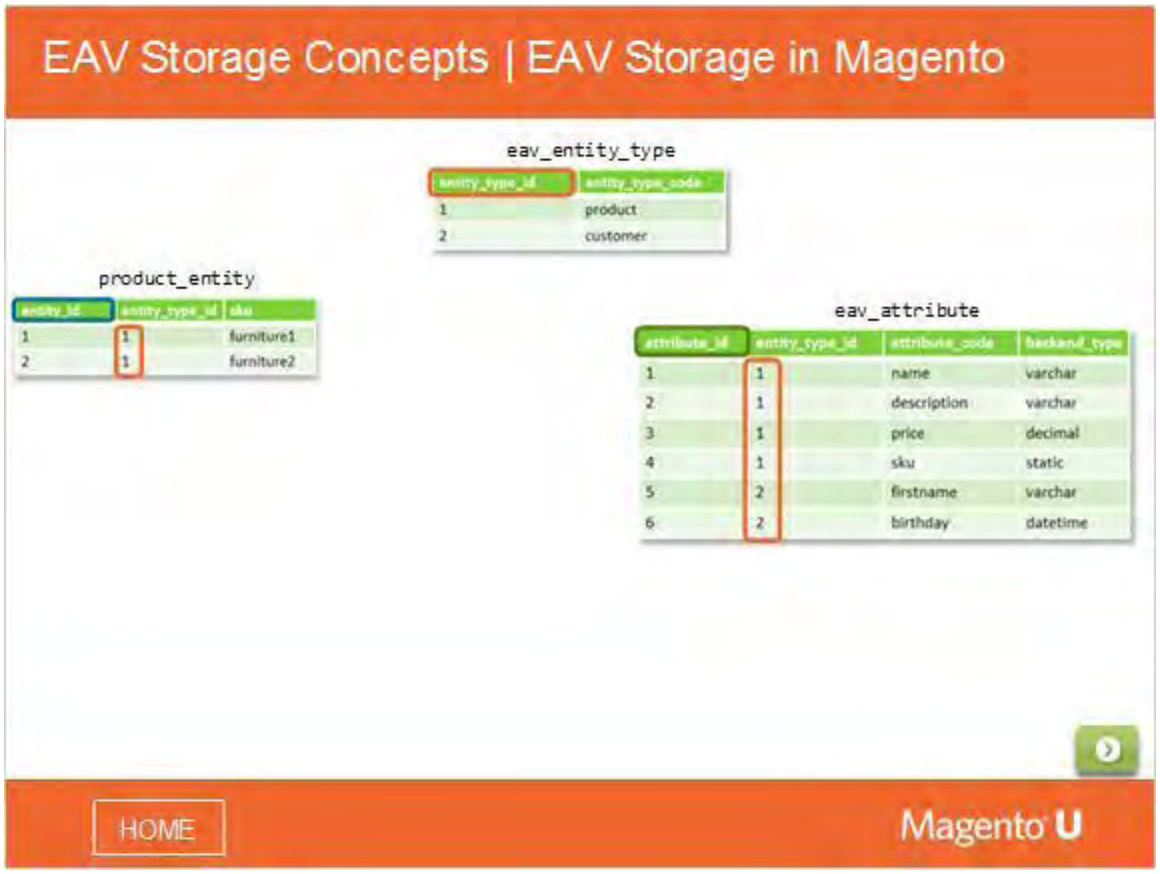
eav_entity_type

entity_type_id	entity_type_code
1	product
2	customer

eav_attribute

attribute_id	entity_type_id	attribute_code	backend_type
1	1	name	varchar
2	1	description	varchar
3	1	price	decimal
4	1	sku	static
5	2	firstname	varchar
6	2	birthday	datetime

product_entity (Slide Layer)



EAV Storage Concepts | EAV Storage in Magento

eav_entity_type

entity_type_id	entity_type_code
1	product
2	customer

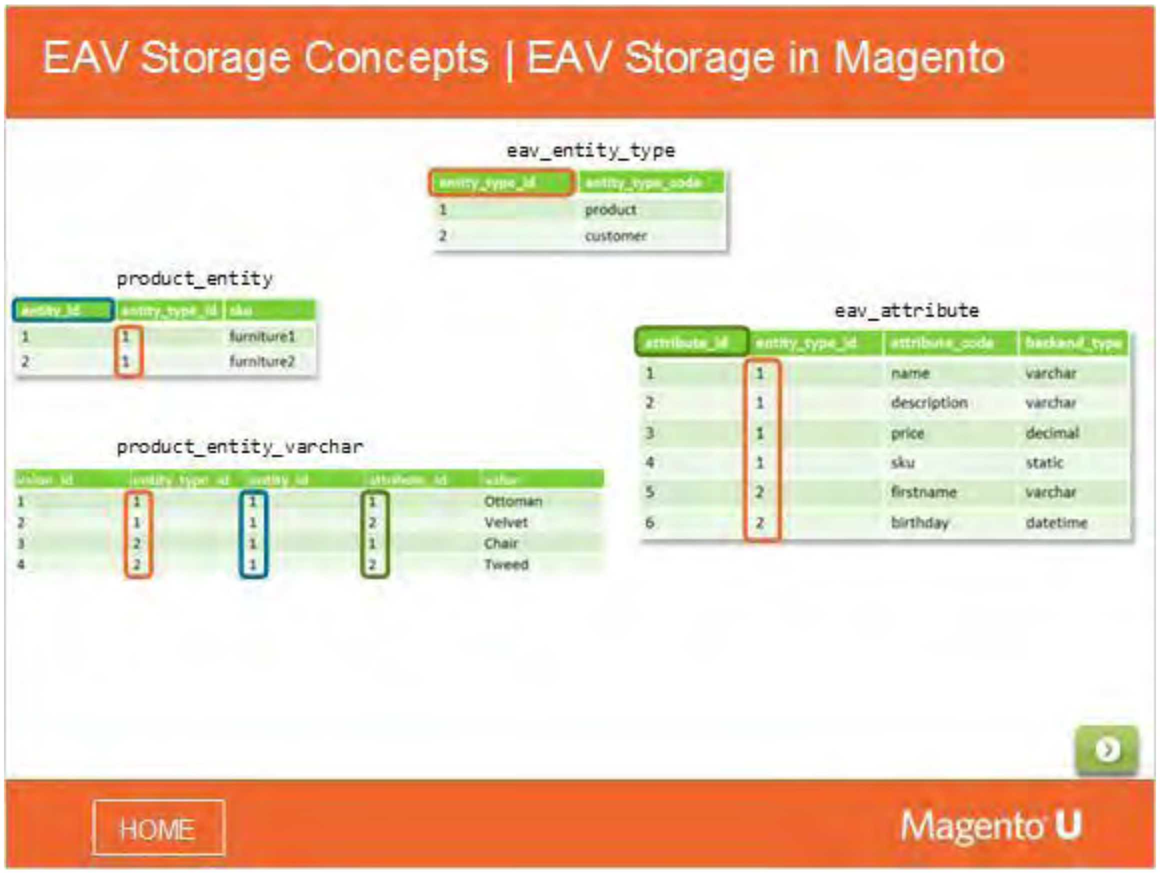
product_entity

entity_id	entity_type_id	sku
1	1	furniture1
2	1	furniture2

eav_attribute

attribute_id	entity_type_id	attribute_code	backend_type
1	1	name	varchar
2	1	description	varchar
3	1	price	decimal
4	1	sku	static
5	2	firstname	varchar
6	2	birthday	datetime

product_entity_varchar (Slide Layer)



EAV Storage Concepts | EAV Storage in Magento

eav_entity_type

entity_type_id	entity_type_code
1	product
2	customer

product_entity

entity_id	entity_type_id	sku
1	1	furniture1
2	1	furniture2

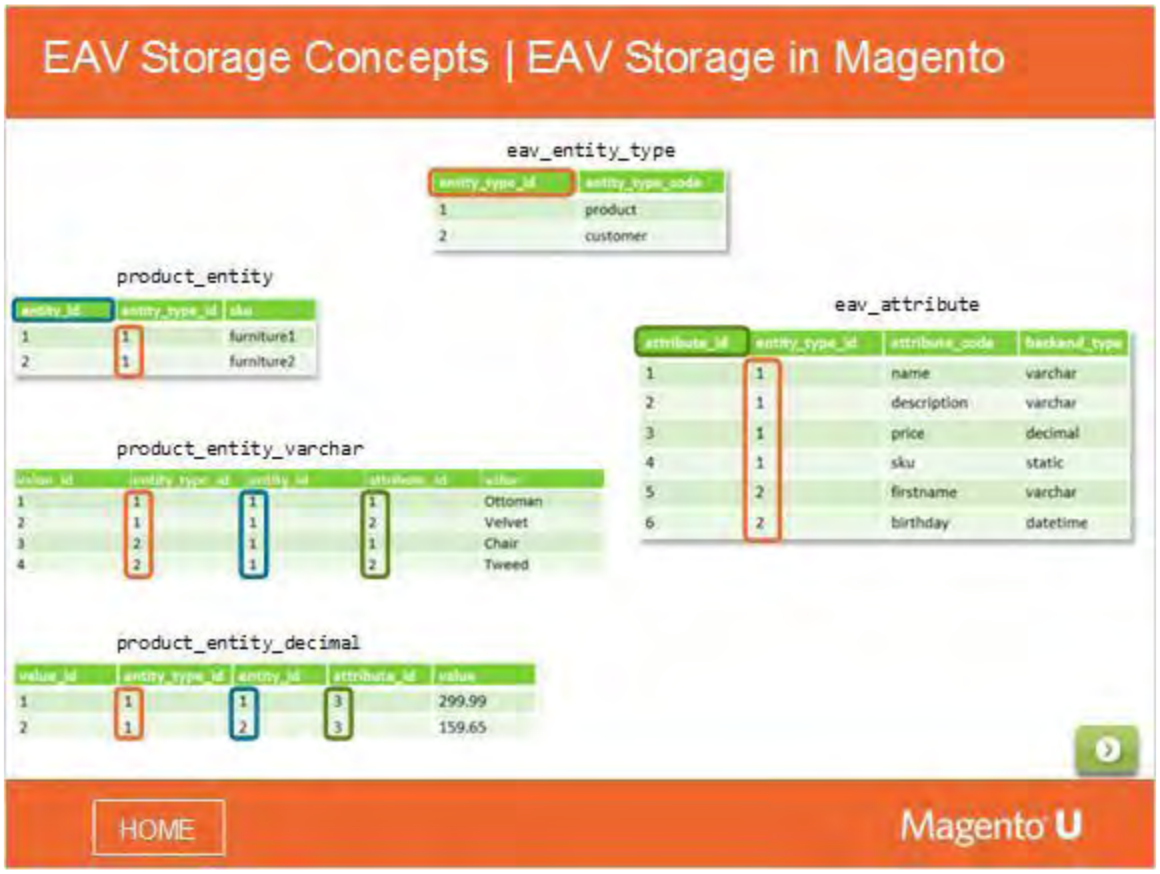
eav_attribute

attribute_id	entity_type_id	attribute_code	backend_type
1	1	name	varchar
2	1	description	varchar
3	1	price	decimal
4	1	sku	static
5	2	firstname	varchar
6	2	birthday	datetime

product_entity_varchar

value_id	entity_type_id	entity_id	attribute_id	value
1	1	1	1	Ottoman
2	1	1	2	Velvet
3	2	1	1	Chair
4	2	1	2	Tweed

product_entity_decimal (Slide Layer)



EAV Storage Concepts | EAV Storage in Magento

eav_entity_type

entity_type_id	entity_type_code
1	product
2	customer

product_entity

entity_id	entity_type_id	sku
1	1	furniture1
2	1	furniture2

eav_attribute

attribute_id	entity_type_id	attribute_code	backend_type
1	1	name	varchar
2	1	description	varchar
3	1	price	decimal
4	1	sku	static
5	2	firstname	varchar
6	2	birthday	datetime

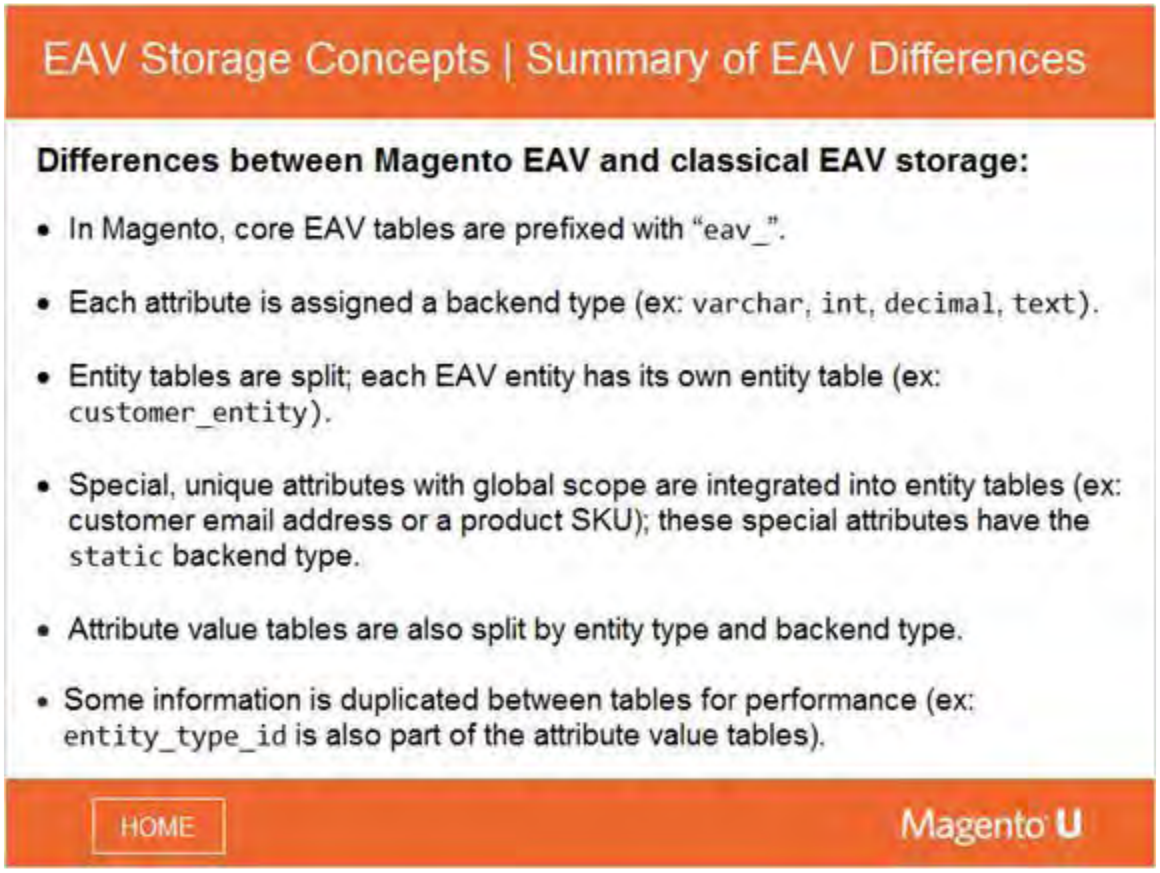
product_entity_varchar

value_id	entity_type_id	entity_id	attribute_id	value
1	1	1	1	Ottoman
2	1	1	2	Velvet
3	2	1	1	Chair
4	2	1	2	Tweed

product_entity_decimal

value_id	entity_type_id	entity_id	attribute_id	value
1	1	1	3	299.99
2	1	2	3	159.65

5.10 EAV Storage Concepts | Summary of EAV Differences



EAV Storage Concepts | Summary of EAV Differences

Differences between Magento EAV and classical EAV storage:

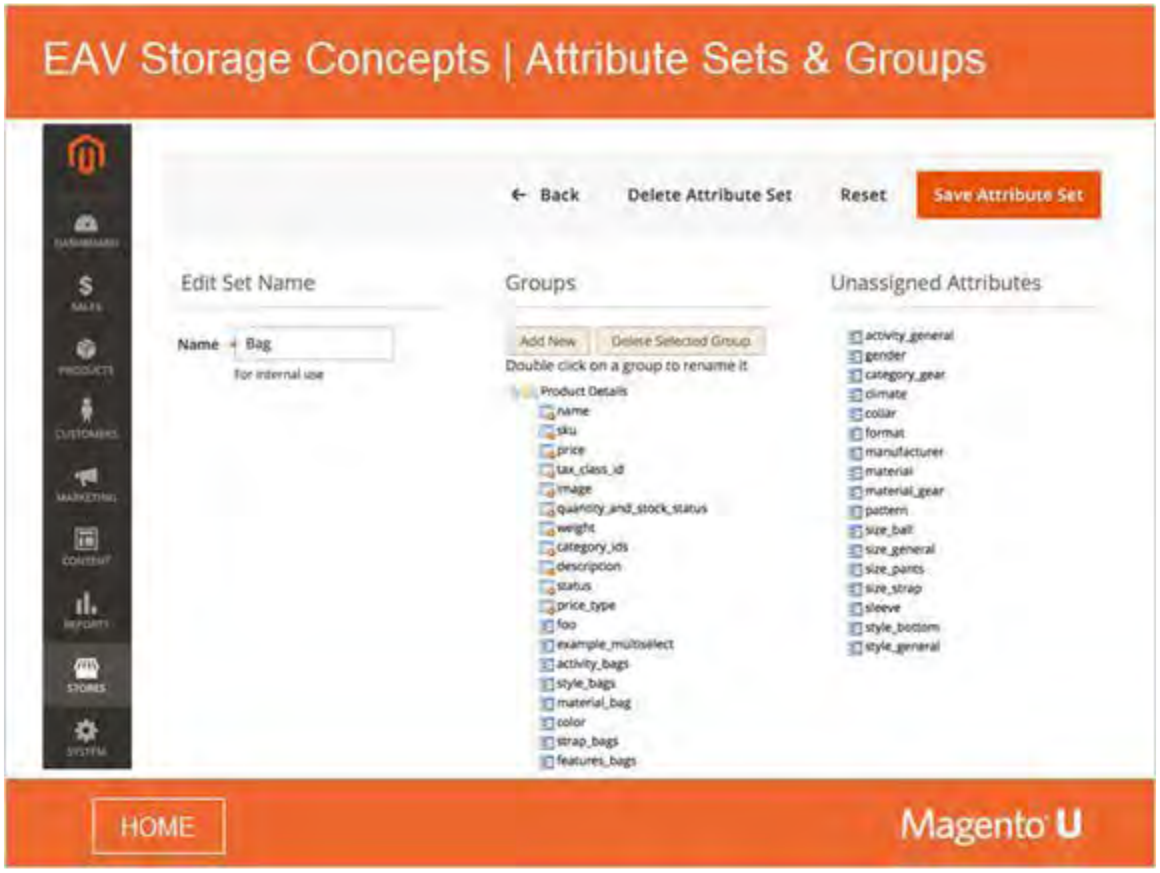
- In Magento, core EAV tables are prefixed with "`eav_`".
- Each attribute is assigned a backend type (ex: `varchar`, `int`, `decimal`, `text`).
- Entity tables are split; each EAV entity has its own entity table (ex: `customer_entity`).
- Special, unique attributes with global scope are integrated into entity tables (ex: customer email address or a product SKU); these special attributes have the `static` backend type.
- Attribute value tables are also split by entity type and backend type.
- Some information is duplicated between tables for performance (ex: `entity_type_id` is also part of the attribute value tables).

Notes:

The following list highlights some of the differences between Magento EAV storage and classical EAV storage:

- In Magento the core EAV tables are prefixed with "`eav_`".
- Each attribute is assigned a backend type. Examples: varchar, int, decimal, text, datetime.
- The entity tables are split; each EAV entity has its own entity table. Examples: `catalog_product_entity`, `catalog_category_entity` and `customer_entity`.
- Special, unique attributes with a global scope are integrated into the entity tables; for example, the customer email address or a product SKU. These special attributes have the backend type static.
- The attribute value tables are also split by entity type and backend type. So customer entities are stored in the EAV entity table `customer_entity`, and the associated attribute values are stored in the EAV value tables `customer_entity_int, customer_entity_varchar, customer_entity_decimal, customer_entity_datetime`, etc.
- For performance reasons some information is duplicated between tables. For example the `entity_type_id` is also part of the attribute value tables.

5.11 EAV Storage Concepts | Attribute Sets & Groups



Edit Set Name

Name * Bag

For internal use

Groups

Double click on a group to rename it

- Product Details
  - name
  - sku
  - price
  - tax_class_id
  - image
  - quantity_and_stock_status
  - weight
  - category_ids
  - description
  - status
  - price_type
  - foo
  - example_multiselect
  - activity_bags
  - style_bags
  - material_bag
  - color
  - strap_bags
  - features_bags

Unassigned Attributes

- activity_general
- gender
- category_gear
- climate
- collar
- format
- manufacturer
- material
- material_gear
- pattern
- size_ball
- size_general
- size_pants
- size_strap
- sleeve
- style_bottom
- style_general

Notes:

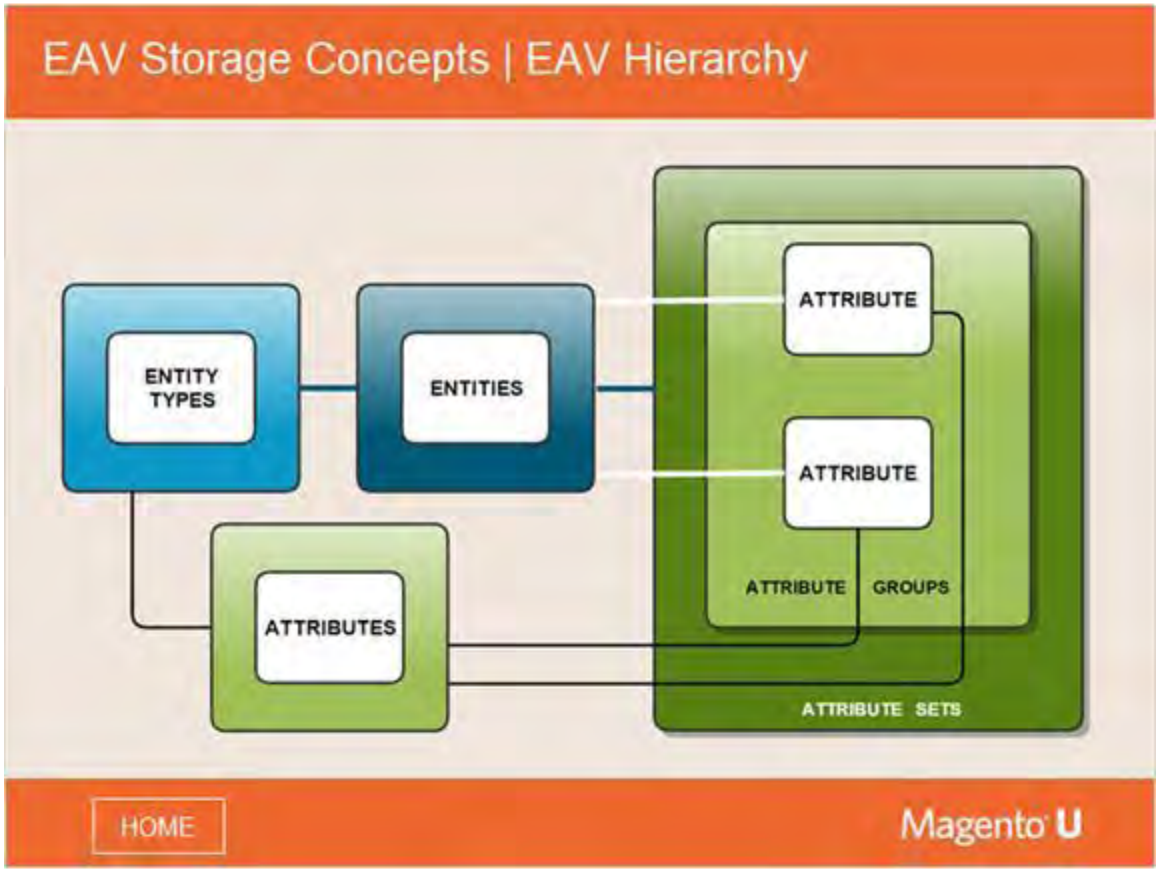
Magento has added the capability to organize product attributes into attribute sets to facilitate different types of products.

Each attribute set has several attribute groups (for example, Prices). An attribute that is associated with an attribute set is always linked to an attribute group. A store owner can create new attribute sets and groups as needed.

The sole purpose of attribute groups is to create clarity in the Admin interface -- there is no functional difference between attribute groups.

Each EAV entity in Magento has at least one attribute set (`Default`), although it is not obvious in the Admin interface (except for products). As a result, each attribute set has at least one attribute group: `General`.

5.12 EAV Storage Concepts | EAV Hierarchy

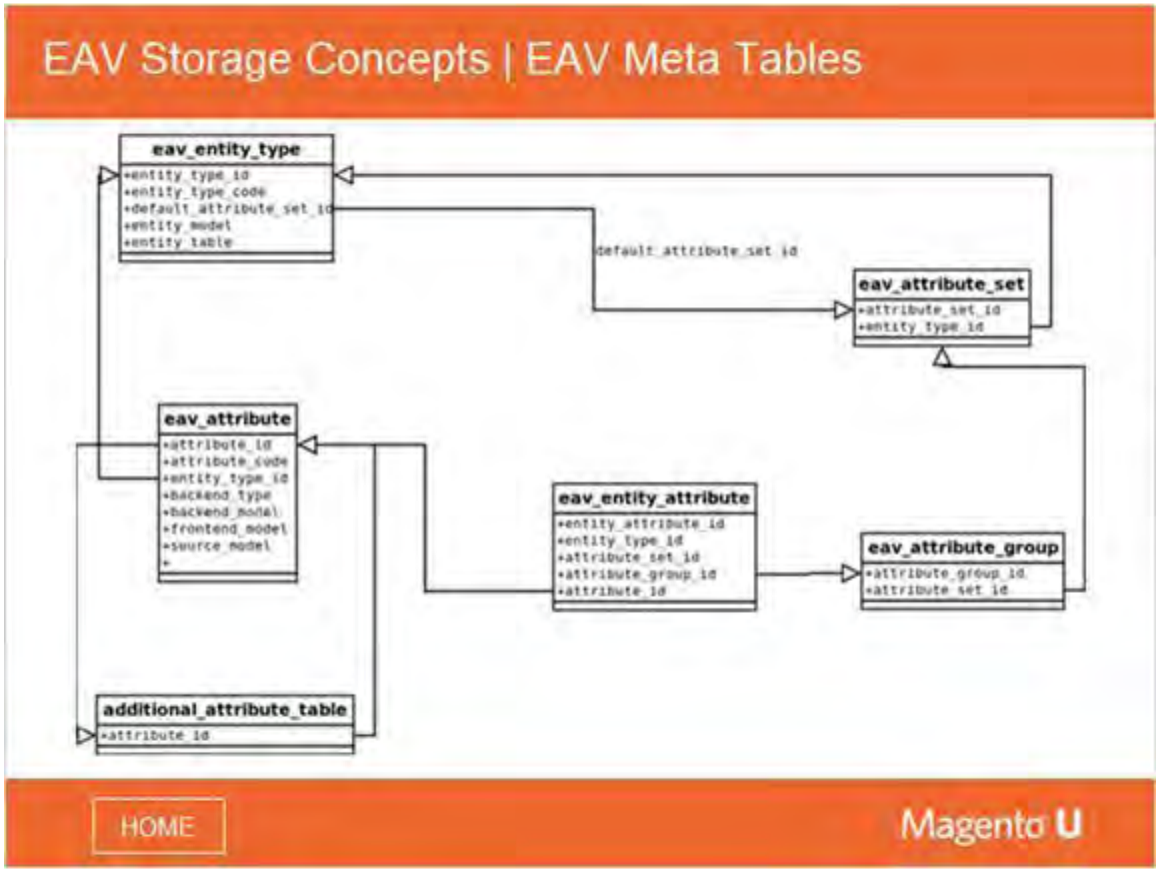


Notes:

Above is a depiction of the EAV hierarchy.

Entity types form the basis of the Magento EAV model. Each entity type has one or more attributes and attribute sets. Each attribute set has one or more attribute groups. Every attribute is associated to exactly one entity type and one or more attribute sets via attribute groups.

5.13 EAV Storage Concepts | EAV Meta Tables



Notes:

The `eav_entity_type` table contains the different entity types.

The `eav_attribute` table contains the available attributes for all entity types.

Some entity types require additional attribute properties. For example, the property `used_in_layered_navigation` is only useful for product attributes, not for customer attributes.

These additional attribute properties are stored in additional attribute tables:

- `catalog_eav_attribute`
- `customer_eav_attribute`

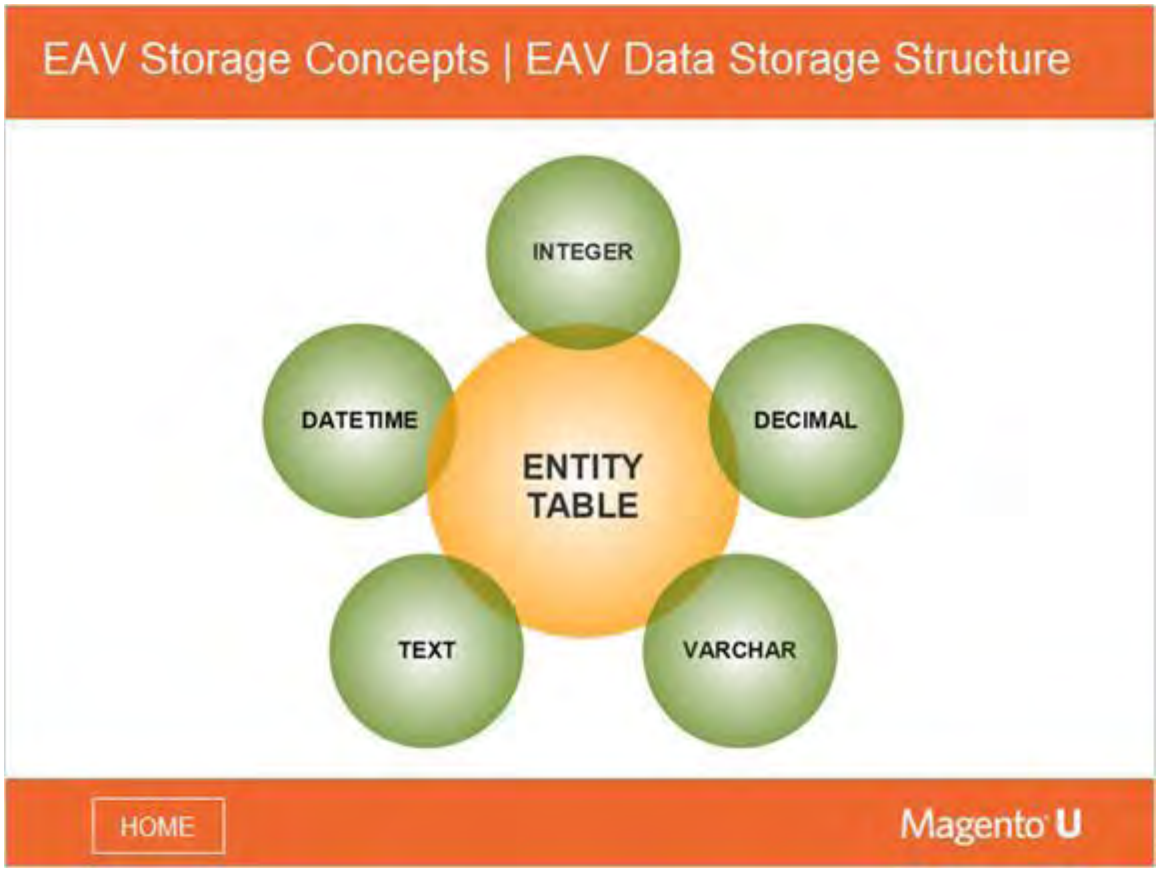
These additional attribute tables are specified in a column of the `eav_entity_type` table.

The `eav_attribute_set` table contains the attribute set declarations. The `eav_attribute_group` contains the attribute groups.

The `eav_entity_attribute` table links all the information needed to render the product backend interface:

- Entity types
- Attribute sets
- Attribute groups
- Attributes

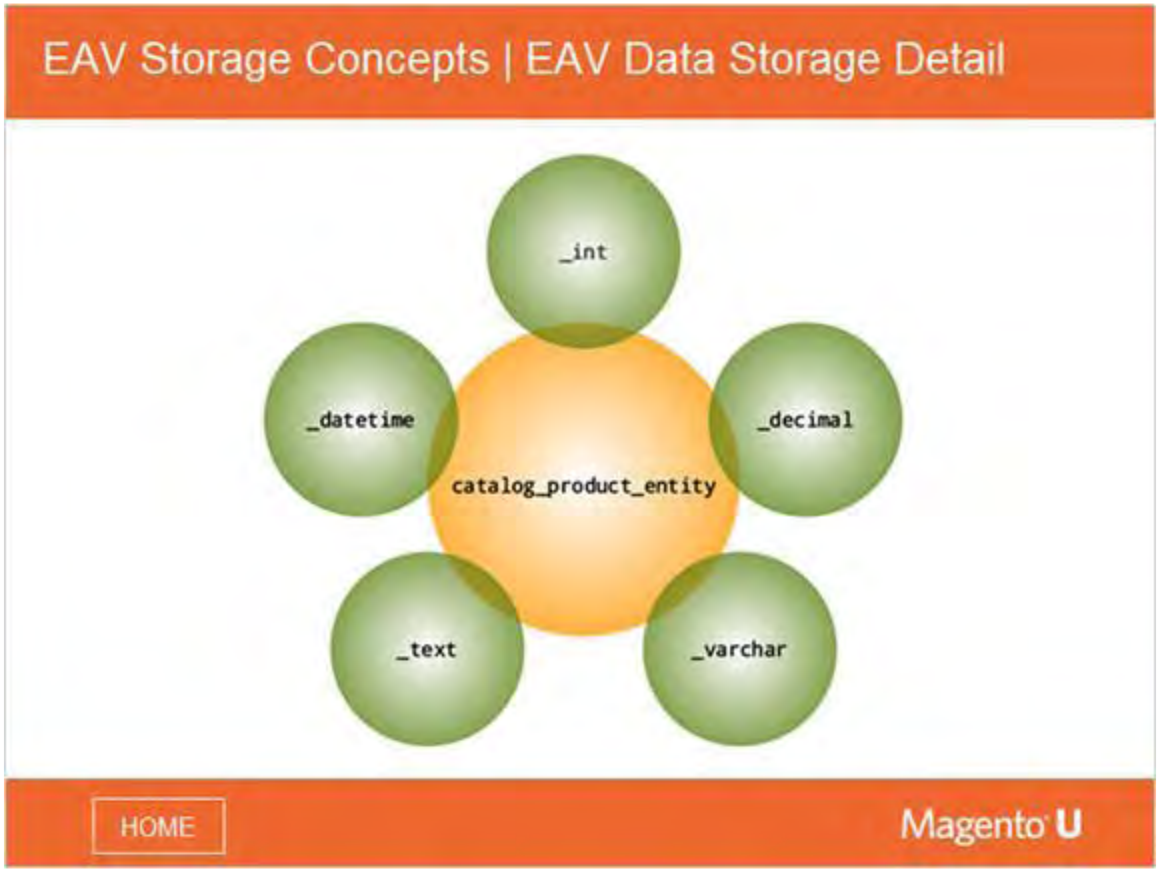
5.14 EAV Storage Concepts | EAV Data Storage Structure



Notes:

Entity attribute values are stored in several related tables. The default table an attribute value is stored in is determined by the attributes `backend_type` property.

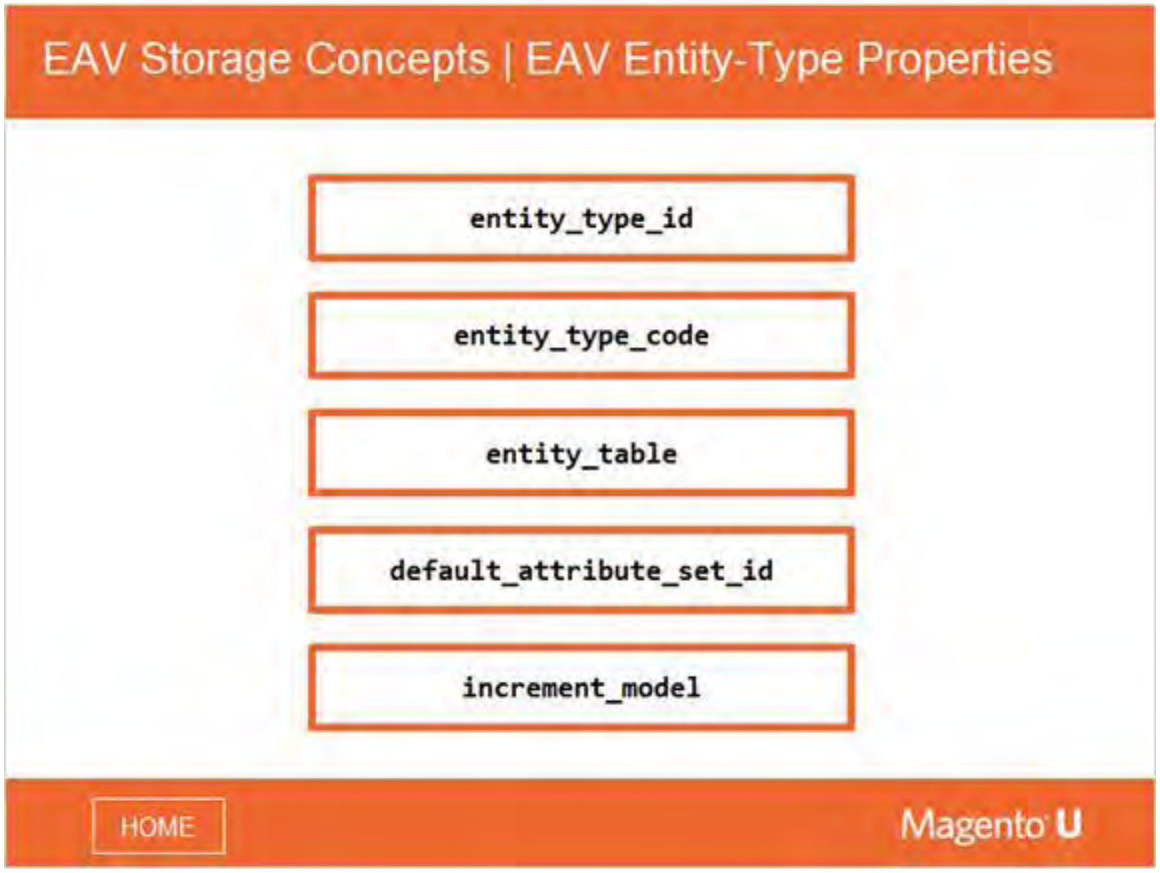
5.15 EAV Storage Concepts | EAV Data Storage Detail



Notes:

The concrete table name for the attribute values is built by appending the backend type to the name of the entity table. Attributes with the backend type `static` are stored in the entity table itself.

5.16 EAV Storage Concepts | EAV Entity-Type Properties



EAV Storage Concepts | EAV Entity-Type Properties

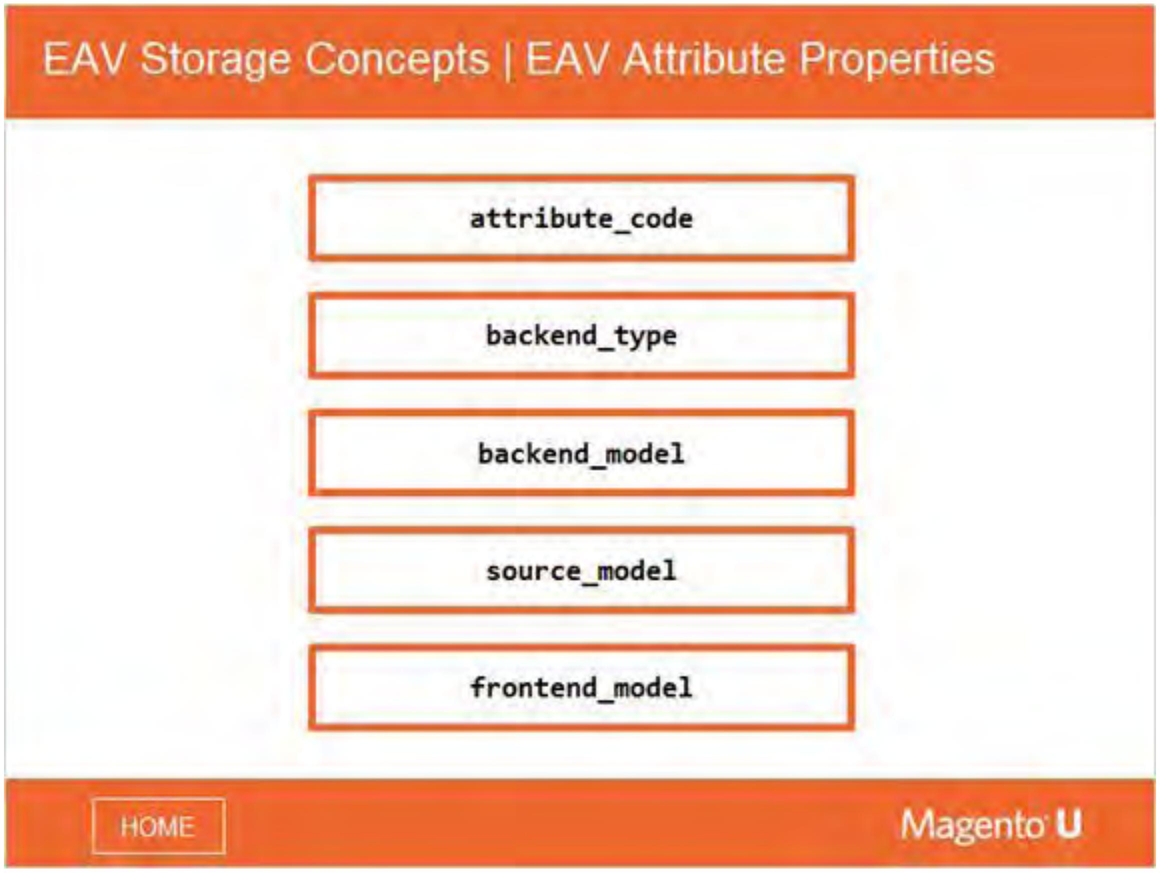
Notes:

All properties of entity types are stored in the table `eav_entity_type`. It is a good idea to be familiar with the main fields of this table when working with EAV entities.

When creating new EAV entities, it is a good practice to study all available properties. The main properties of entity types are:

- **`entity_type_id`:** The unique numeric identifier that is used throughout the system in many places.
- **`entity_type_code`:** The human-readable unique identifier of the entity type.
- **`entity_table`:** The entity table name.
- **`default_attribute_set_id`:** Attribute set ID, which is used when no attribute set is specified for a new model.
- **`increment_model`:** PHP class name that handles increments.

5.17 EAV Storage Concepts | EAV Attribute Properties


EAV Storage Concepts | EAV Attribute Properties

attribute_code

backend_type

backend_model

source_model

frontend_model

HOME
Magento U

Notes:

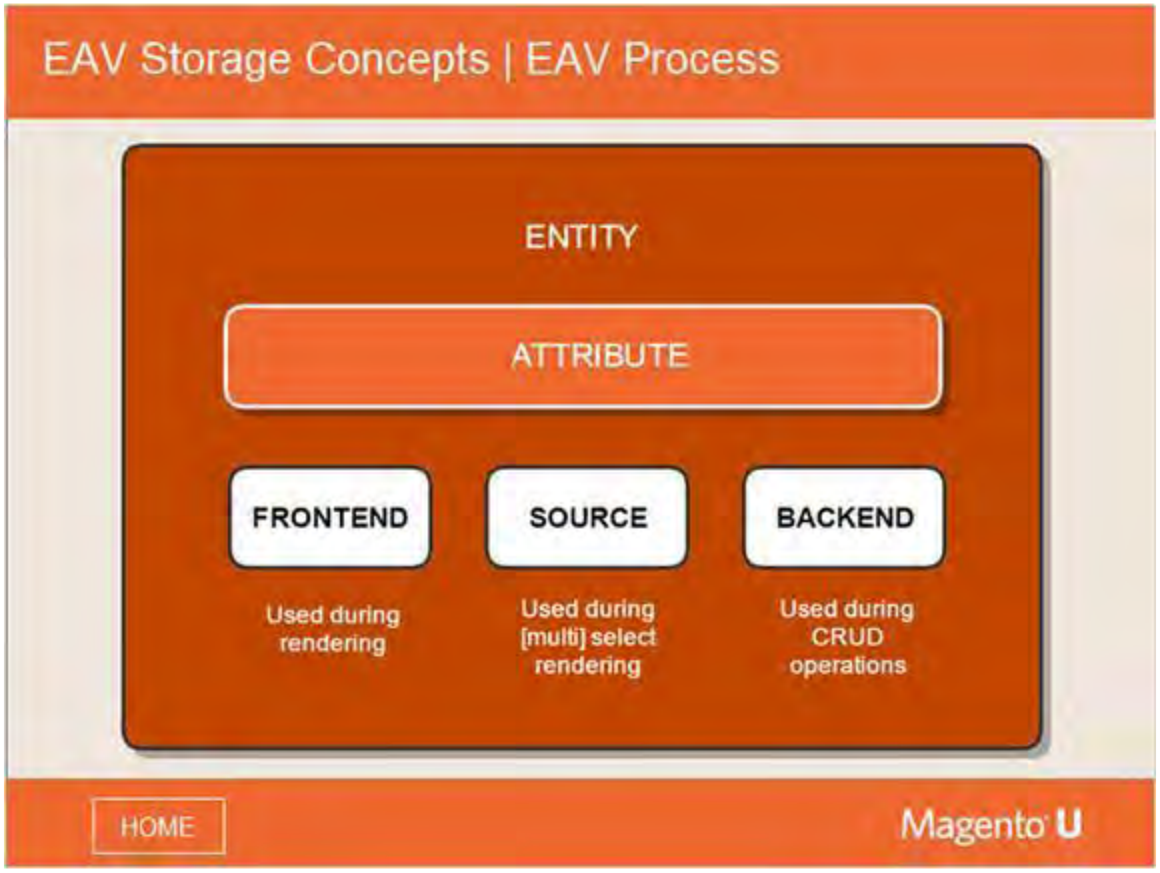
All common properties of EAV attributes are stored in the table `eav_attribute`. You should be familiar with the main fields of this table when working with EAV entities. When creating new EAV attributes, all available properties should be known.

`entity-type-specific` attribute properties are stored in additional attribute tables (for example, `catalog_eav_attribute`).

The main properties of EAV attributes are:

- **`attribute_code:`** The human-readable identifier of the entity type. It must be unique within the scope of the associated entity type.
- **`backend_type`:** The data type of the attribute. Native Magento uses static, varchar, datetime, int, text, and decimal; custom types can also be used.
- **`backend_model:`** Attribute backend models are used to process attribute data before or after CRUD operations.
- `source_model:` Attribute source models are used to provide the options for select and multiselect input-type attributes.
- **`frontend_model:`** Attribute frontend models are used to format attribute values for display.

5.18 EAV Storage Concepts | EAV Process

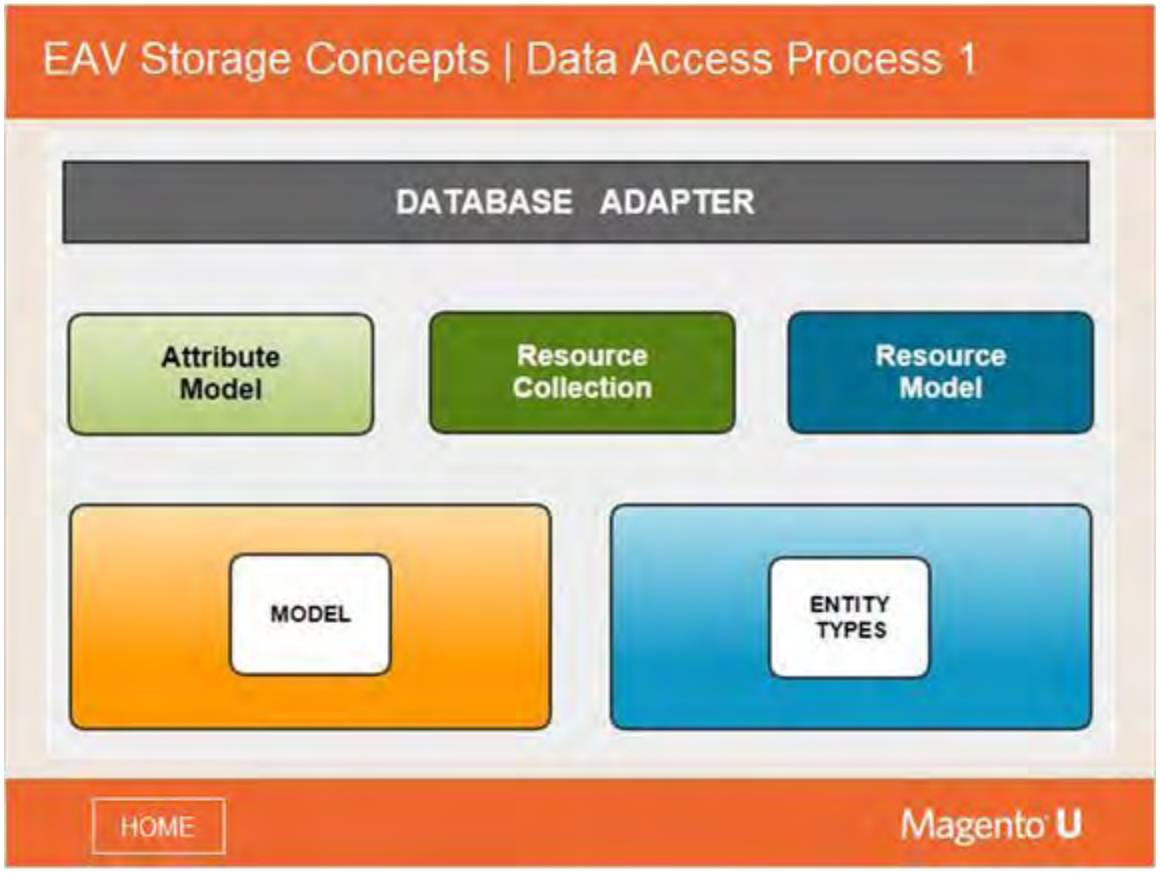

EAV Storage Concepts | EAV Process
ENTITY
ATTRIBUTE
FRONTEND
Used during rendering
SOURCE
Used during [multi] select rendering
BACKEND
Used during CRUD operations
HOME
Magento U

Notes:

When rendering multiselect and select input fields, the source model supplies the list of possible attribute option-IDs and option-labels.

If a multiselect or select attribute option is rendered (for example, on the product detail page), the frontend model uses the source model to render the correct label for the attribute option.

5.19 EAV Storage Concepts | Data Access Process 1

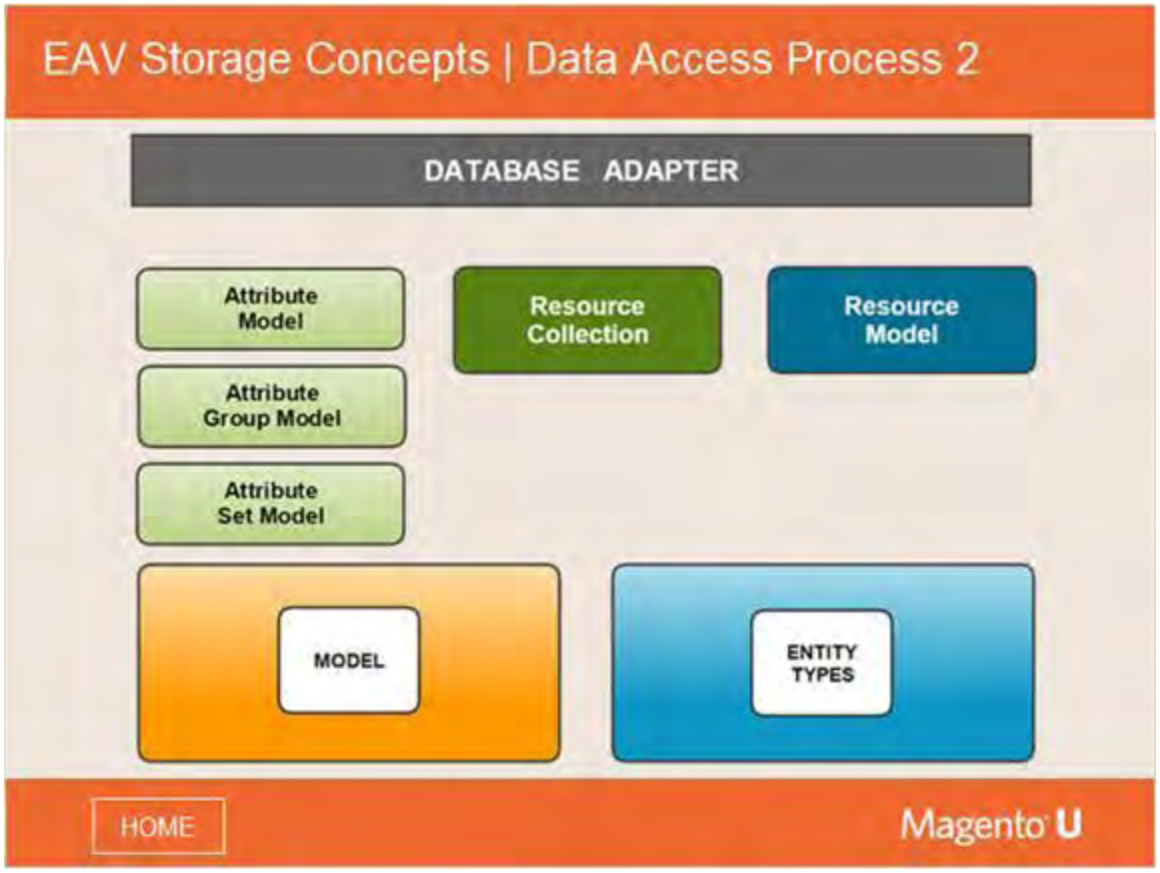


Notes:

Compared to the flat table model, the database access structure of EAV models is more complex.

The entity type model and the attribute models play an important role in gathering all the information needed to load and save an EAV attribute value.

5.20 EAV Storage Concepts | Data Access Process 2

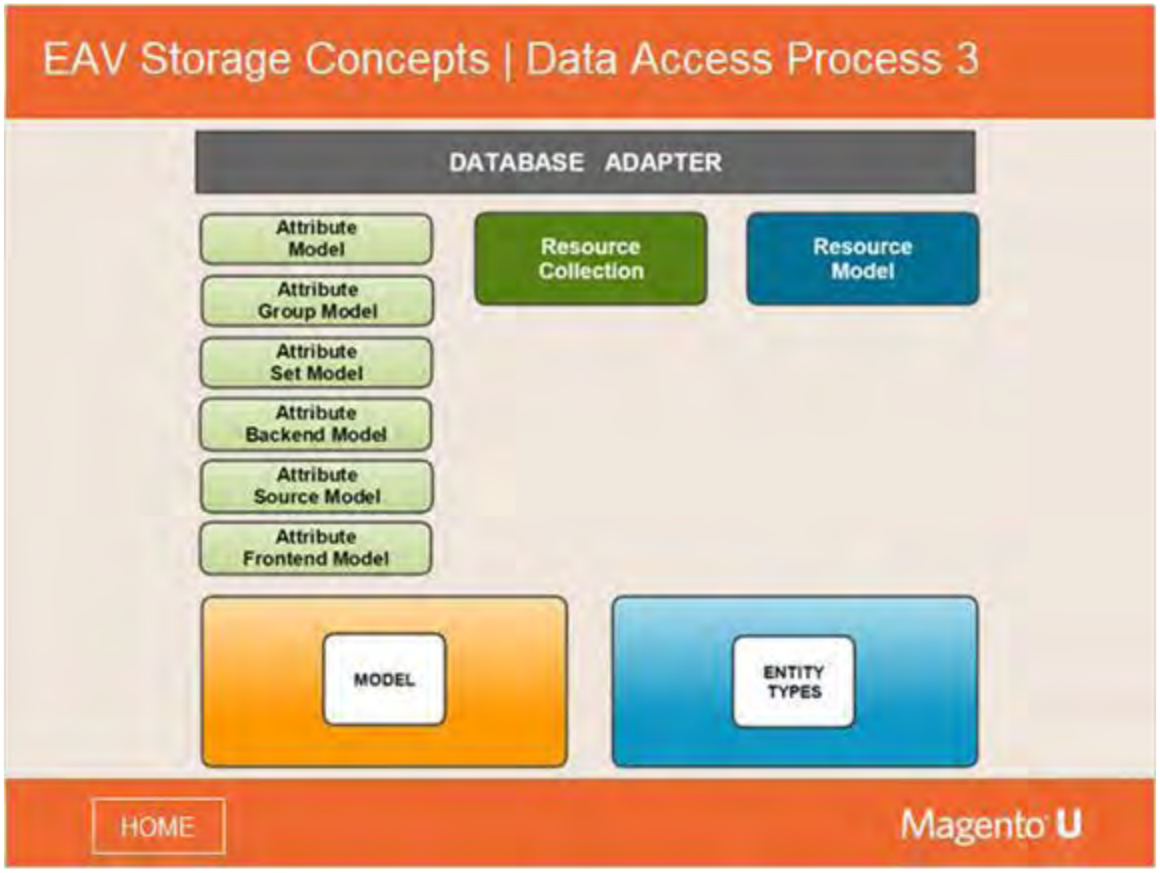


Notes:

When displaying the attributes for EAV entities like products, categories, or customers (for example, in forms in the Adminhtml interface), the attribute set model and group model are used.

The attribute set is also used when loading all attributes associated to a specific product for the frontend detail view.

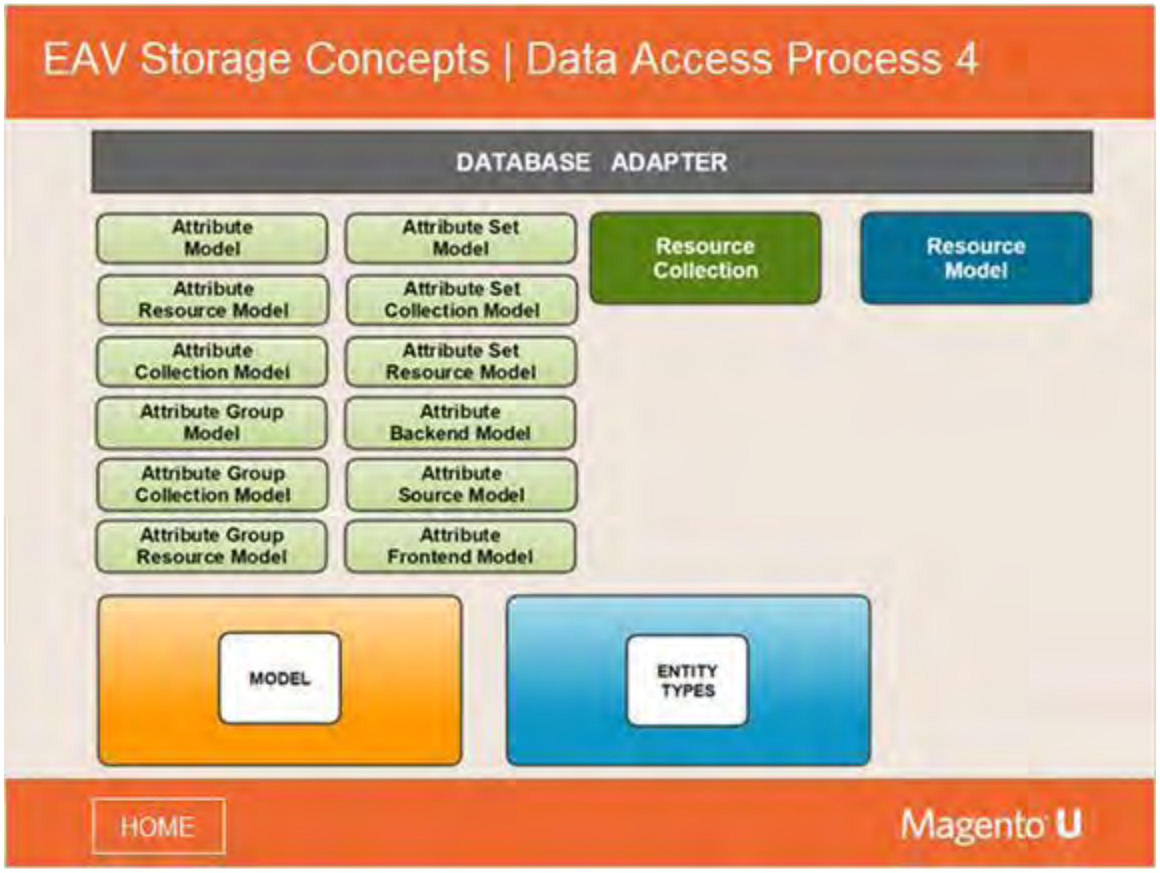
5.21 EAV Storage Concepts | Data Access Process 3



Notes:

During the load, save, and rendering processes, the attribute backend, source, and frontend models are used.

5.22 EAV Storage Concepts | Data Access Process 4



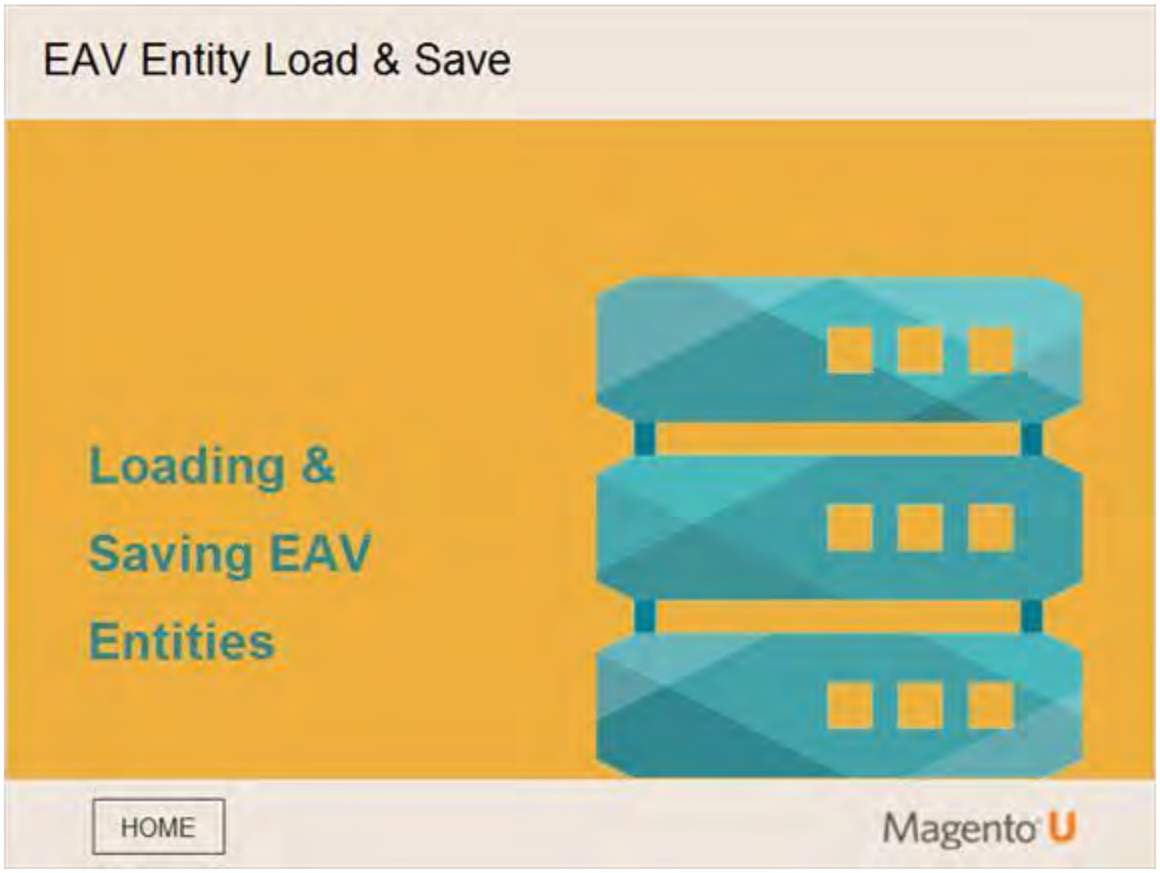
Notes:

Because all the meta information about attributes is also stored in tables, they are represented by model classes with matching resource models and collections.

This gives you a very granular approach when working with EAV entities and attributes. For everyday tasks, these additional classes (besides the entity models) are transparent.

6. EAV Entity Load and Save

6.1 EAV Entity Load & Save



EAV Entity Load & Save

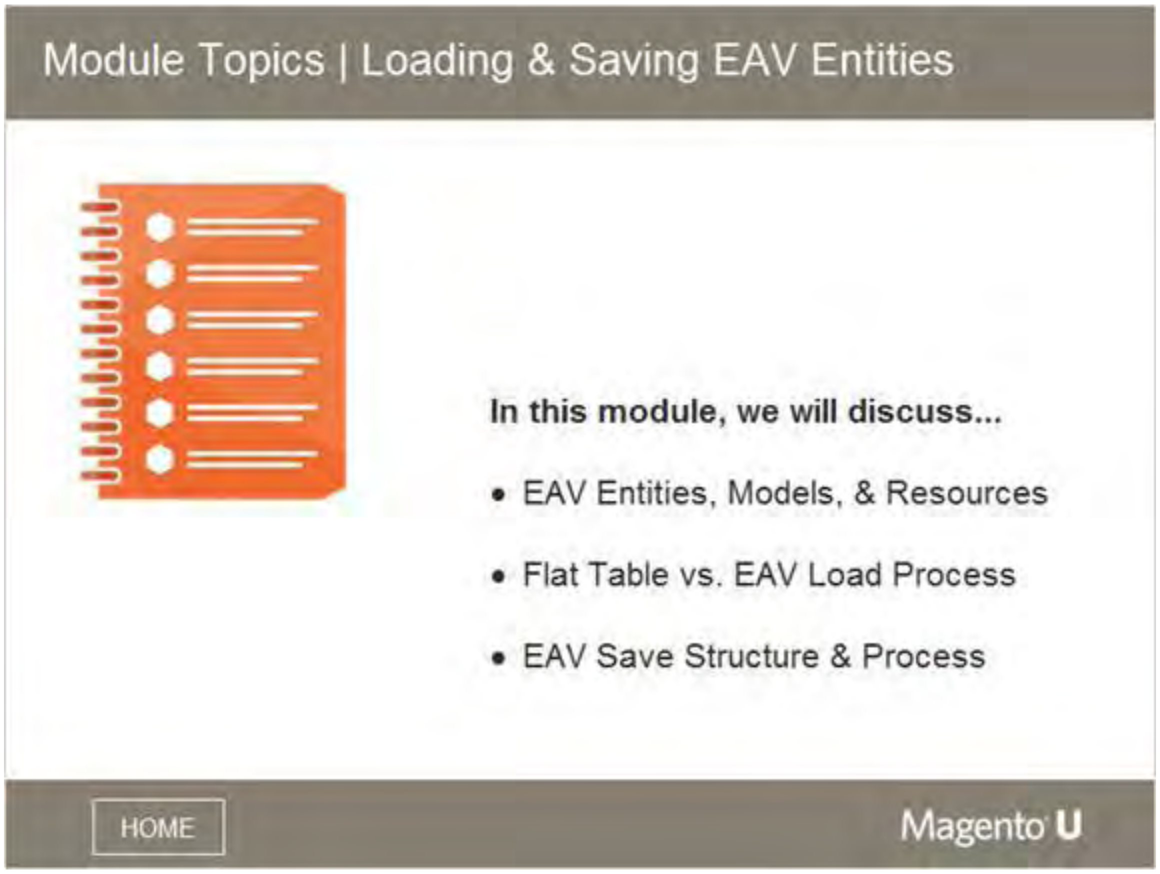
Loading & Saving EAV Entities

Notes:

This module on loading and saving EAV entities covers how the EAV resource model and collections implement CRUD operations, including the SQL queries used to access EAV-related data.

It also covers how Magento 2's EAV loading and saving processes differ from the "regular" load and save processes of flat table resource models.

6.2 Module Topics | Loading & Saving EAV Entities



Module Topics | Loading & Saving EAV Entities

In this module, we will discuss...

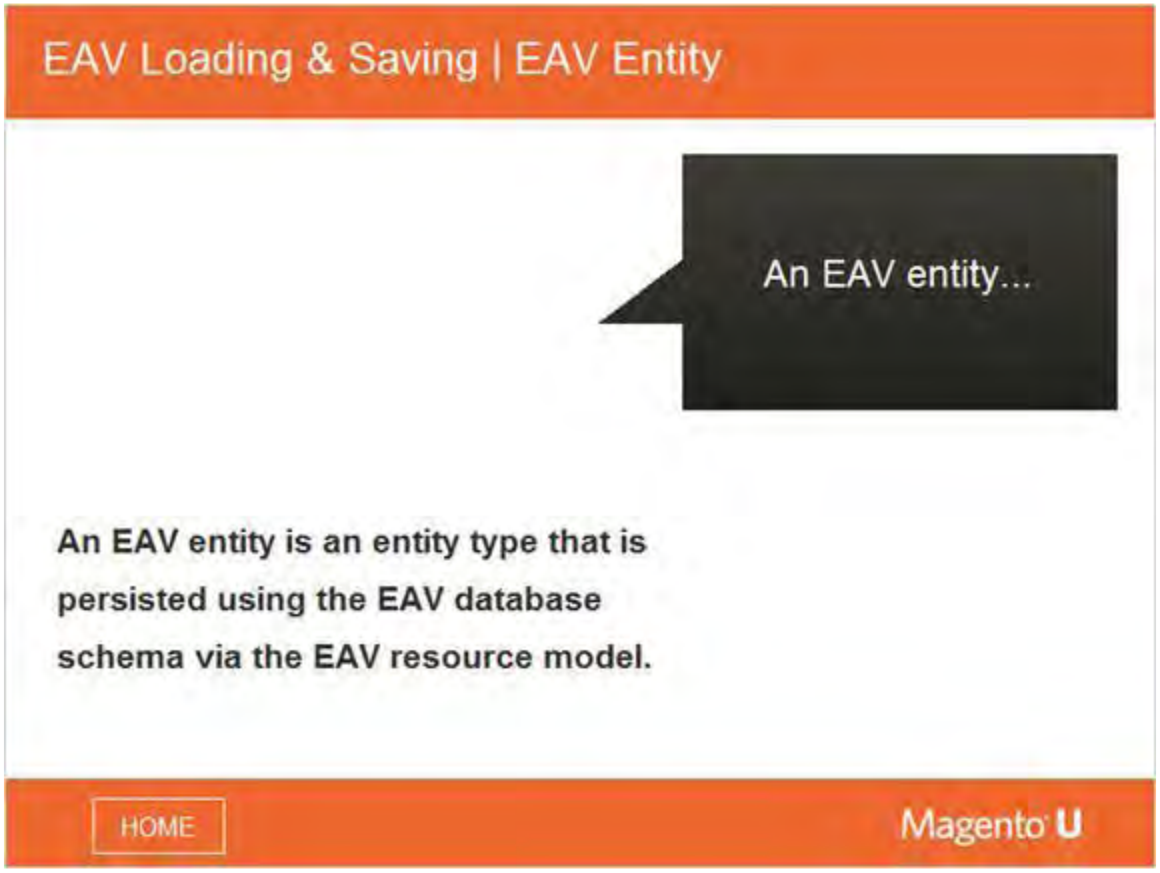
- EAV Entities, Models, & Resources
- Flat Table vs. EAV Load Process
- EAV Save Structure & Process

Notes:

In this module, we will discuss:

- EAV Entities, Models, & Resources
- Loading from Flat Table vs. the EAV Load Process
- The EAV Save Process

6.3 EAV Loading & Saving | EAV Entity



EAV Loading & Saving | EAV Entity

An EAV entity...

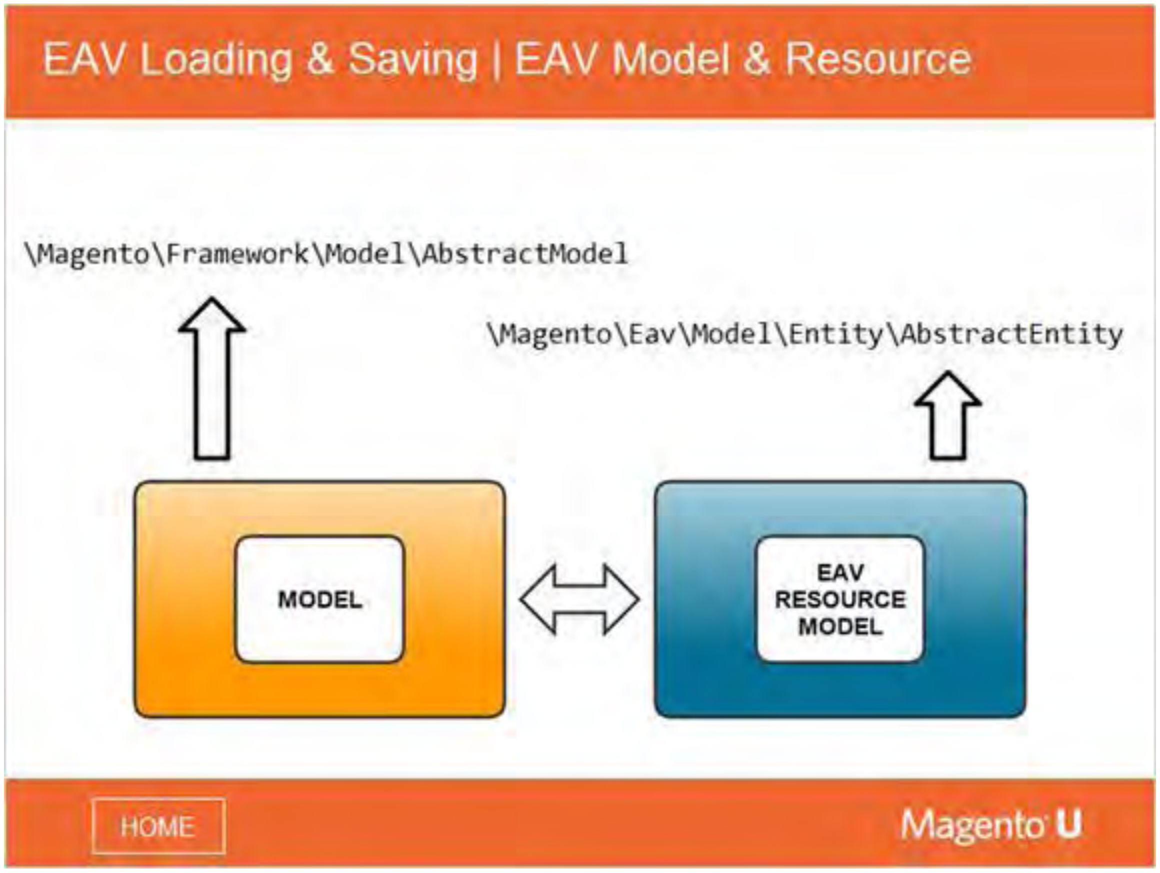
An EAV entity is an entity type that is persisted using the EAV database schema via the EAV resource model.

HOME
Magento U

Notes:

An EAV entity is an entity type that is persisted using the EAV database schema, via the EAV resource model.

6.4 EAV Loading & Saving | EAV Model & Resource

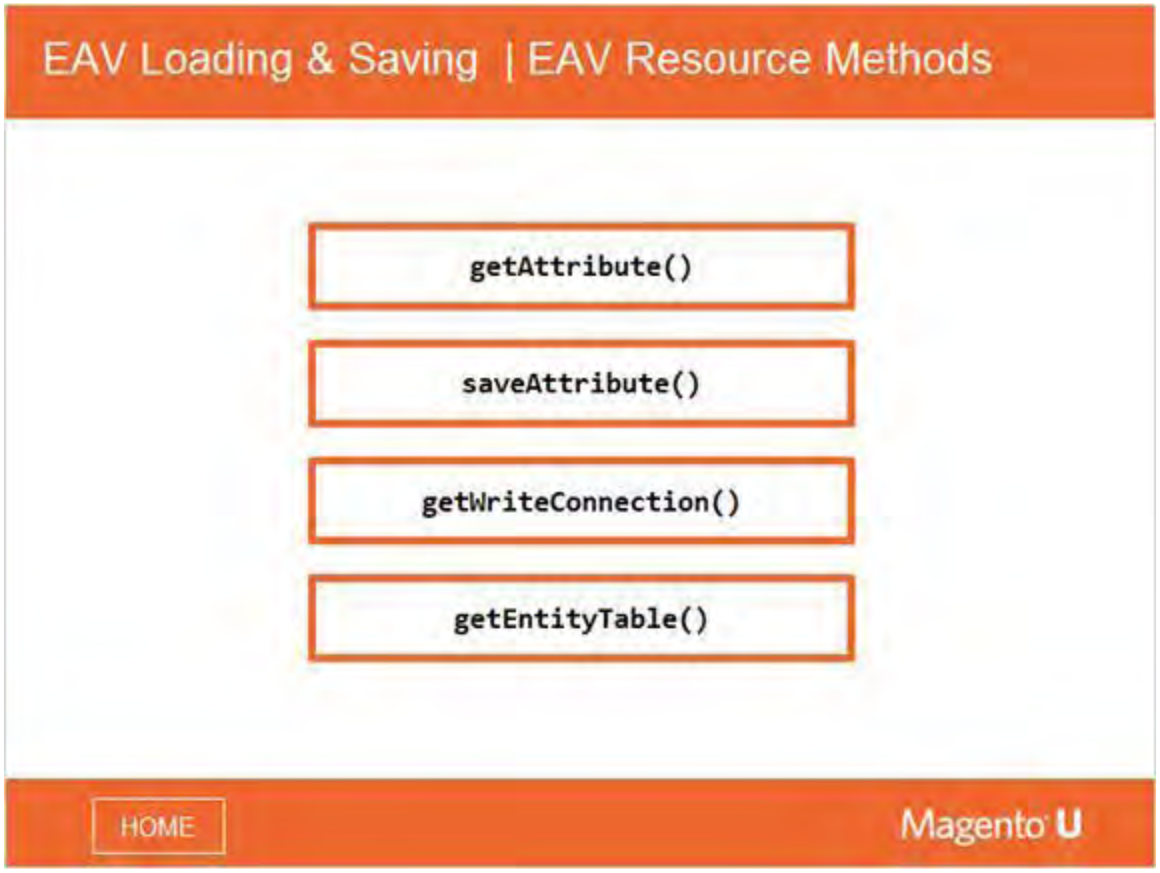

EAV Loading & Saving | EAV Model & Resource
\Magento\Framework\Model\AbstractModel
\Magento\Eav\Model\Entity\AbstractEntity
MODEL
EAV RESOURCE MODEL
HOME
Magento U

Notes:

This diagram demonstrates that:

- EAV based entity models also extend `Magento\Framework\Model\AbstractModel.`
- EAV based resource models extend `Magento\Eav\Model\Entity\AbstractEntity`, rather than `\Magento\Framework\Model\Resource\Db\AbstractDb`.

6.5 EAV Loading & Saving | EAV Resource Methods


EAV Loading & Saving | EAV Resource Methods

getAttribute()

saveAttribute()

getWriteConnection()

getEntityTable()

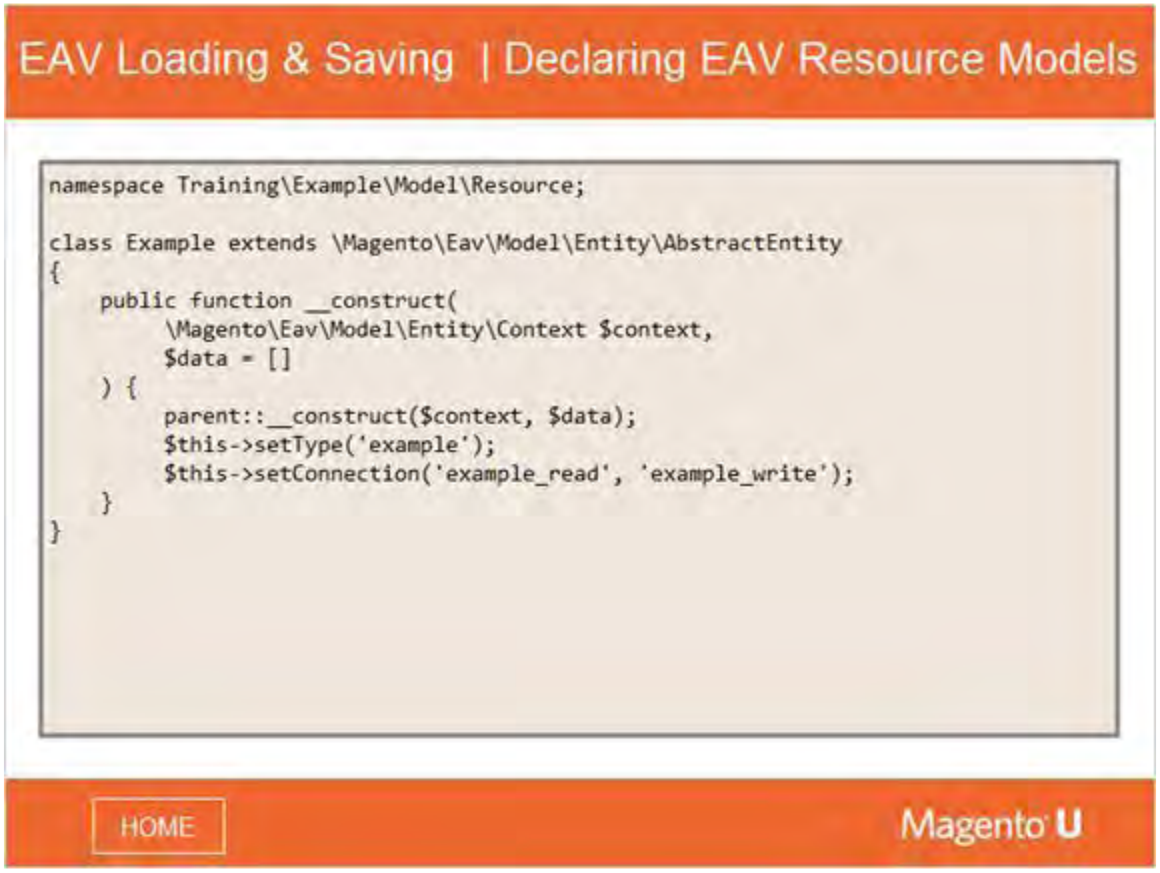
HOME
Magento U

Notes:

The EAV resource model implements additional methods beyond those used in the abstract flat table resource model. These include:

- **`getAttribute():`** Allows you to get an attribute instance by code or numeric ID.
  Example: `$product->getResource()->getAttribute('color');`
- **`saveAttribute():`** Allows you to save an attribute value on an entity, without going through the full entity save process.
  Example: `$product->setWeight(1.99)->getResource()->saveAttribute($product, 'weight');`
- **`getWriteConnection():`** In addition to the `getReadConnection()` method that also exists on flat table resource models, EAV resource models have an interface method to access the write adapter.
- **`getEntityTable():`** In contrast to the `getMainTable()` method of flat table resource models, EAV resource models implement the `getEntityTable()` method instead.

6.6 EAV Loading & Saving | Declaring EAV Resource Models



EAV Loading & Saving | Declaring EAV Resource Models

```
namespace Training\Example\Model\Resource;

class Example extends \Magento\Eav\Model\Entity\AbstractEntity
{
    public function __construct(
        \Magento\Eav\Model\Entity\Context $context,
        $data = []
    ) {
        parent::__construct($context, $data);
        $this->setType('example');
        $this->setConnection('example_read', 'example_write');
    }
}
```

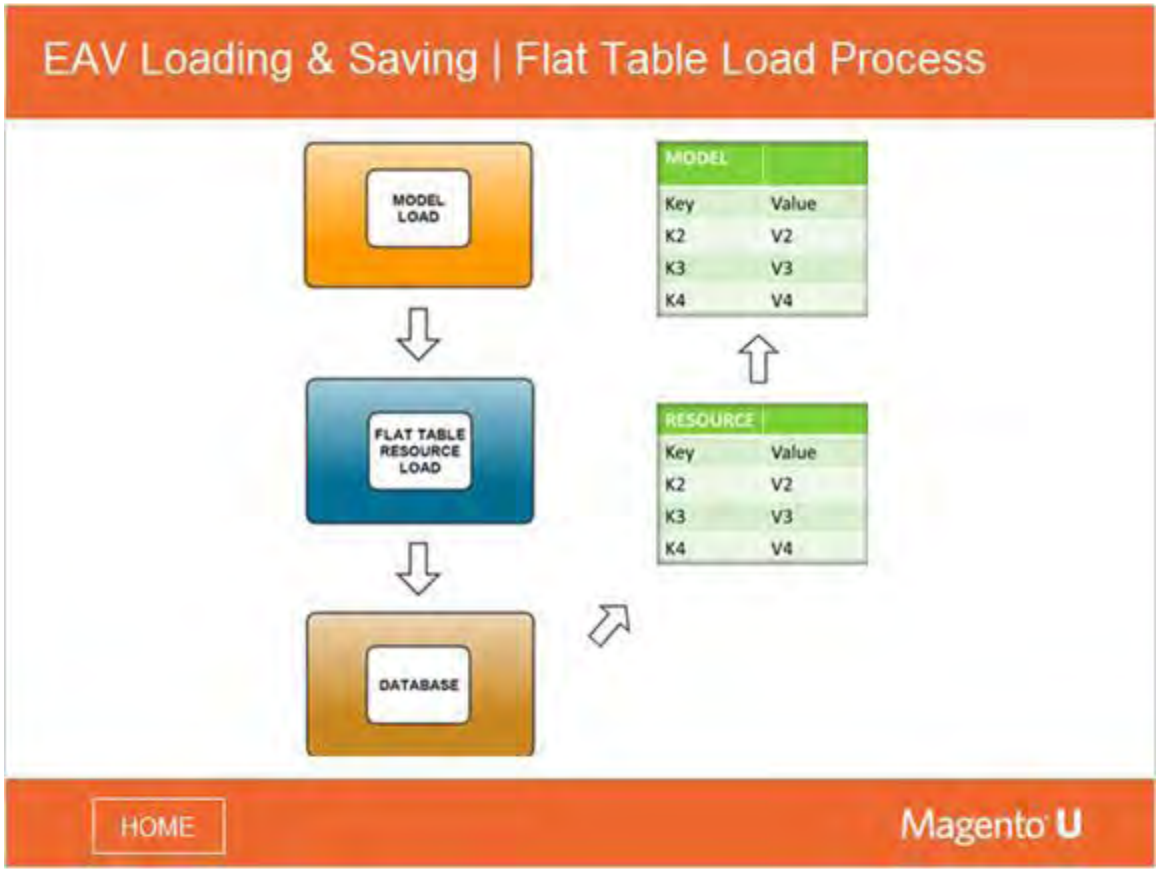
Notes:

There is no difference in declaring EAV based entity models compared to declaring flat table based models; the `_init()` method is called with the class name of the resource model.

On the other hand, declaring an EAV based resource model differs from declaring a flat table based resource model. Instead of initializing the resource model with the main table name and the primary key column, EAV resource models are initialized with the entity type code and the names of the DB read and write resources to use.

All additional information (for example, the entity table name) is read from the EAV meta table `eav_entity_type`.

6.7 EAV Loading & Saving | Flat Table Load Process



EAV Loading & Saving | Flat Table Load Process

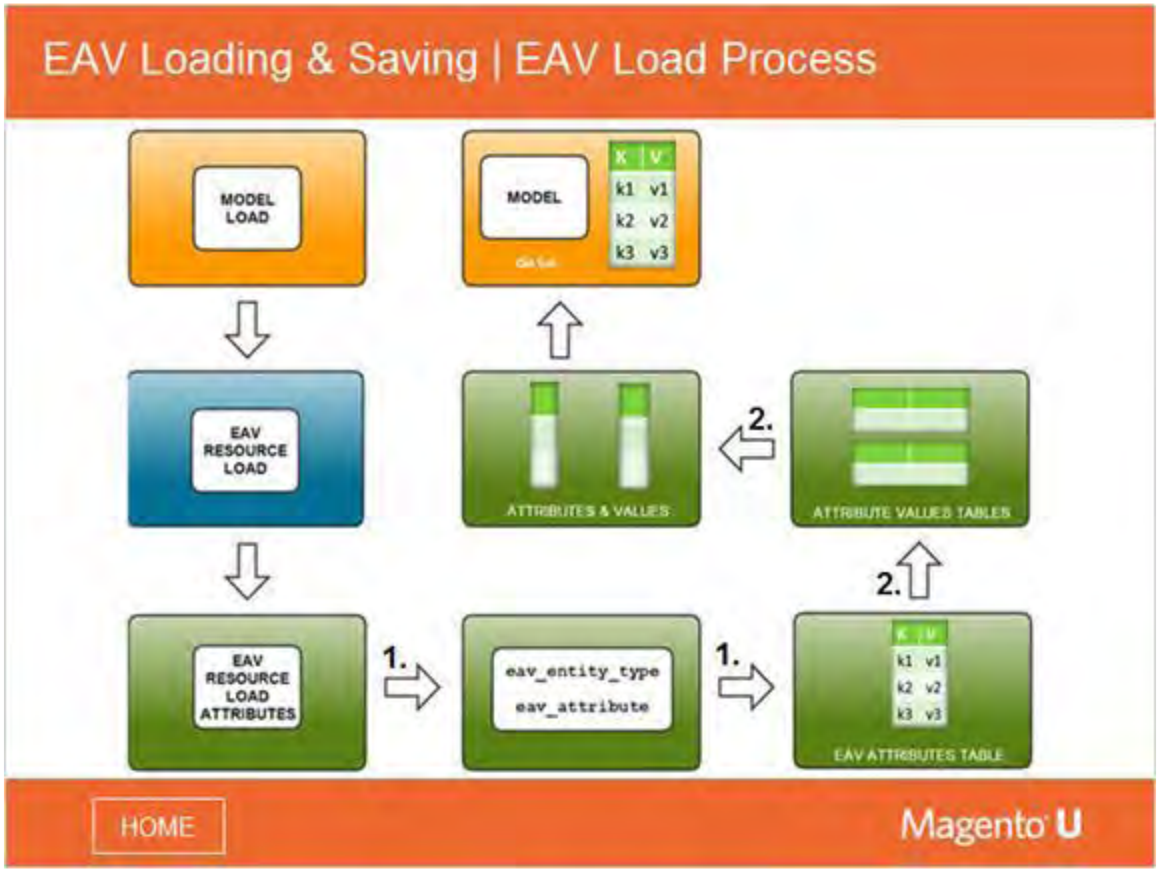
MODEL	
Key	Value
K2	V2
K3	V3
K4	V4

RESOURCE	
Key	Value
K2	V2
K3	V3
K4	V4

Notes:

This diagram shows a schematic of the flat table load process.

6.8 EAV Loading & Saving | EAV Load Process



Notes:

This diagram depicts the EAV load process.

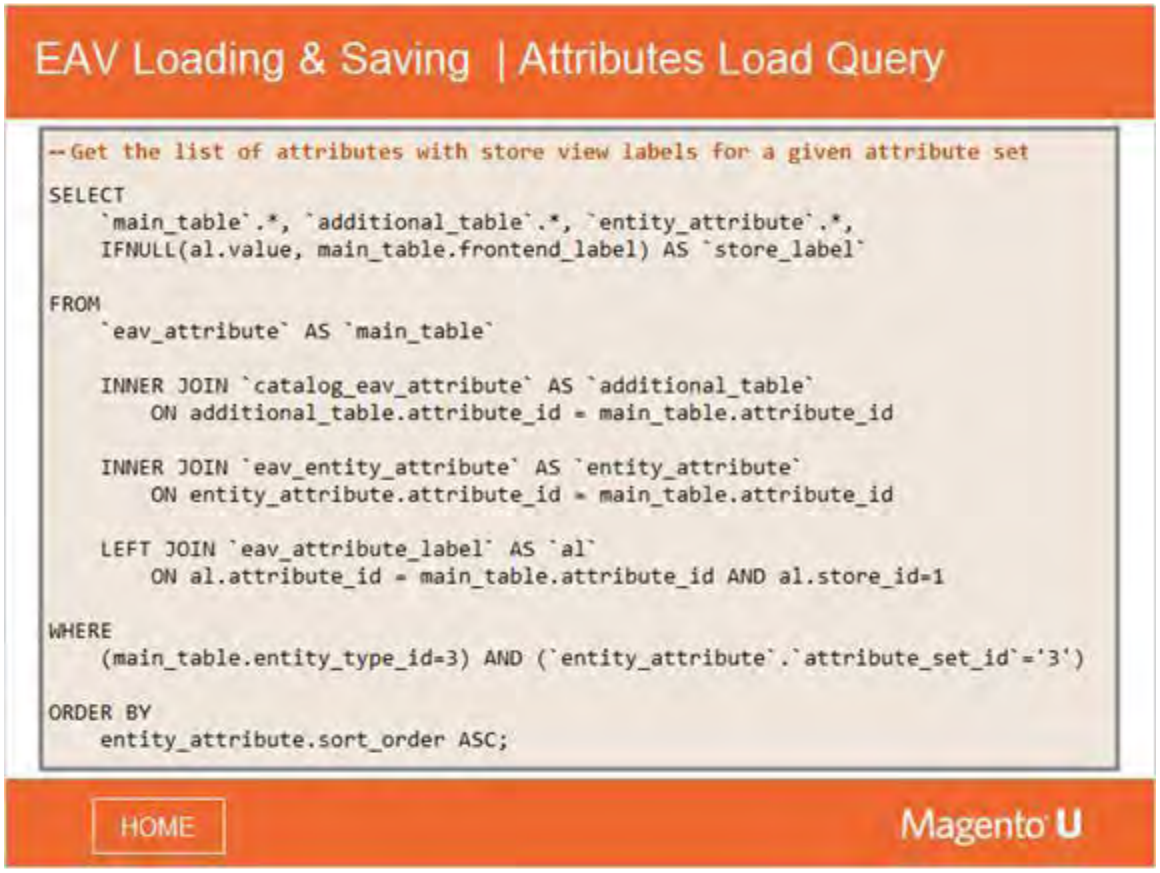
Recall that there are two aspects to the EAV process: managing relationships (that is, meta information about entities and attributes), and managing content (that is, entity and attribute value records).

The two load steps labeled with 1. are related to the meta aspect of EAV. The two steps labeled with 2. refer to the concrete attribute values.

During the loading of EAV entities, the meta information is read first. This includes the attribute labels and the backend types, as well as the information on which attributes to include in the load process.

Then the accomplishing attribute values are read from the value tables, and set on the entity model combined with the attribute codes from the meta information.

6.9 EAV Loading & Saving | Attributes Load Query



EAV Loading & Saving | Attributes Load Query

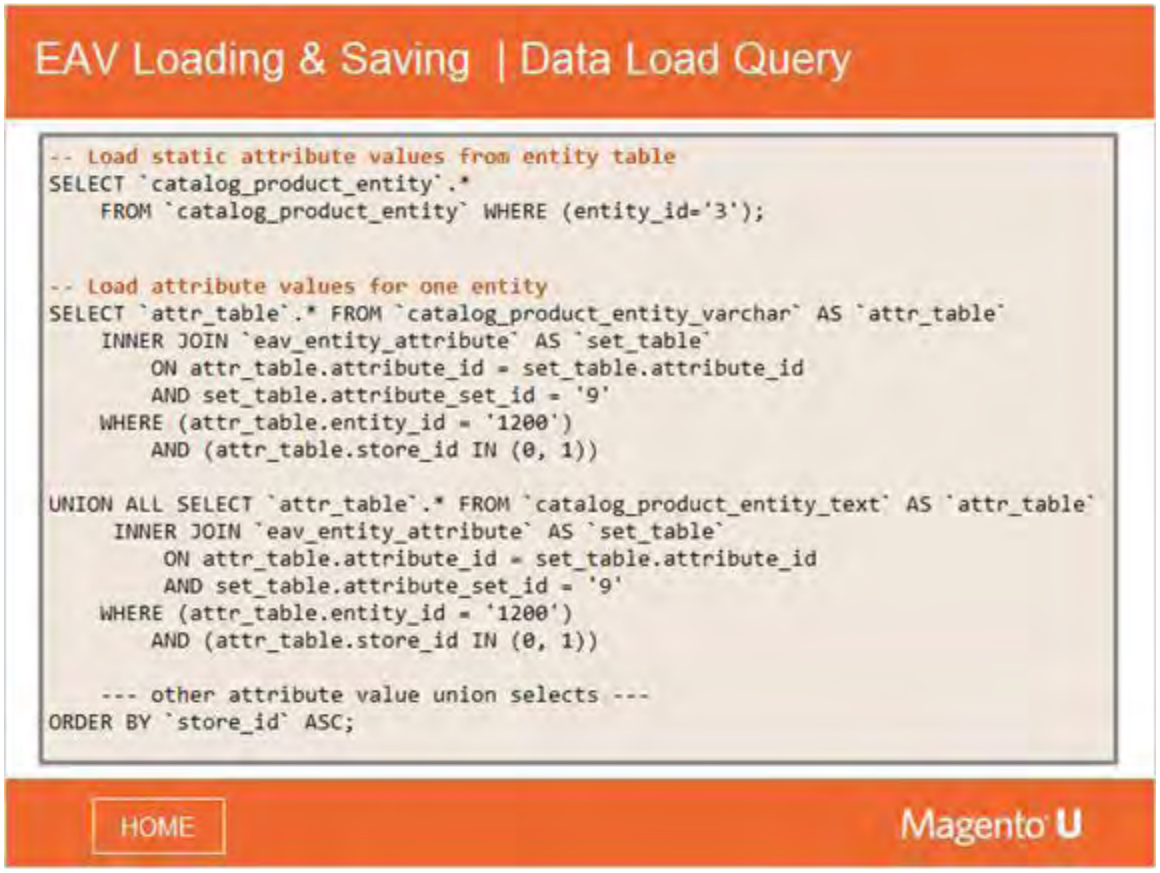
```
--Get the list of attributes with store view labels for a given attribute set

SELECT
    `main_table`.*, `additional_table`.*, `entity_attribute`.*,
    IFNULL(al.value, main_table.frontend_label) AS `store_label`

FROM
    `eav_attribute` AS `main_table`

    INNER JOIN `catalog_eav_attribute` AS `additional_table`
        ON additional_table.attribute_id = main_table.attribute_id

    INNER JOIN `eav_entity_attribute` AS `entity_attribute`
        ON entity_attribute.attribute_id = main_table.attribute_id

    LEFT JOIN `eav_attribute_label` AS `al`
        ON al.attribute_id = main_table.attribute_id AND al.store_id=1

WHERE
    (main_table.entity_type_id=3) AND (`entity_attribute`.`attribute_set_id`='3')

ORDER BY
    entity_attribute.sort_order ASC;
```

Notes:

The code example demonstrates how Magento gets the list of entity attributes with store view labels for a given attribute set.

In this query you can see how Magento implements store view scope values with a fallback to the default scope using SQL. The attribute codes will be combined with the attribute values to populate an entity with data.

6.10 EAV Loading & Saving | Data Load Query



EAV Loading & Saving | Data Load Query

```
-- Load static attribute values from entity table
SELECT `catalog_product_entity`.*
    FROM `catalog_product_entity` WHERE (entity_id='3');

-- Load attribute values for one entity
SELECT `attr_table`.* FROM `catalog_product_entity_varchar` AS `attr_table`
    INNER JOIN `eav_entity_attribute` AS `set_table`
        ON attr_table.attribute_id = set_table.attribute_id
        AND set_table.attribute_set_id = '9'
    WHERE (attr_table.entity_id = '1200')
        AND (attr_table.store_id IN (0, 1))

UNION ALL SELECT `attr_table`.* FROM `catalog_product_entity_text` AS `attr_table`
     INNER JOIN `eav_entity_attribute` AS `set_table`
         ON attr_table.attribute_id = set_table.attribute_id
         AND set_table.attribute_set_id = '9'
    WHERE (attr_table.entity_id = '1200')
        AND (attr_table.store_id IN (0, 1))

    --- other attribute value union selects ---
ORDER BY `store_id` ASC;
```

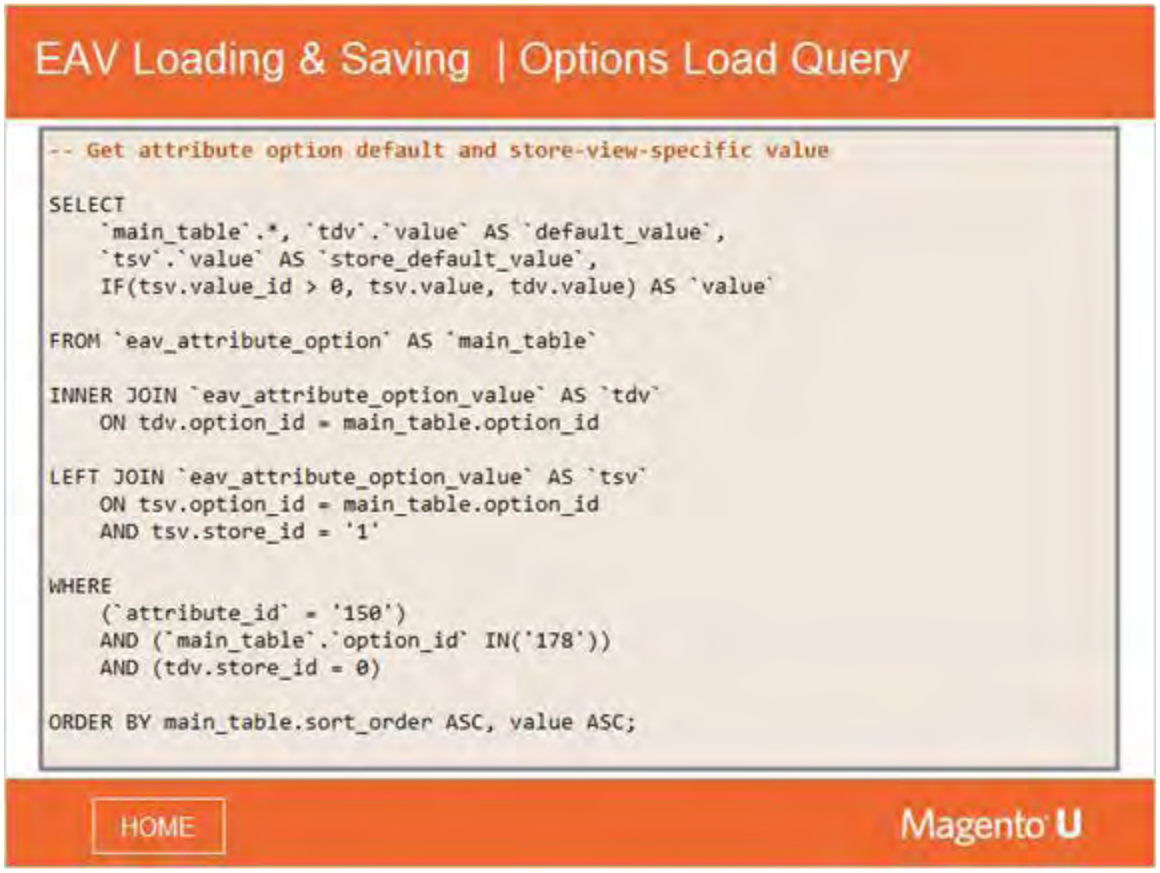
Notes:

Load static attribute values from entity table.
This is the simplest query during the EAV load process. It simply selects all the values for a specific entity from the entity table, in effect loading the attributes with a **static** backend type.

Load attribute values for one entity.
Selecting all attribute values from the entity value tables for a full entity load is done using UNION selects.

This is much faster than joins and is possible because all the attribute value tables have an identical structure, except for the type of the value column. The attribute values will be combined with the attribute code and set on the entity models to populate them.

6.11 EAV Loading & Saving | Options Load Query


EAV Loading & Saving | Options Load Query

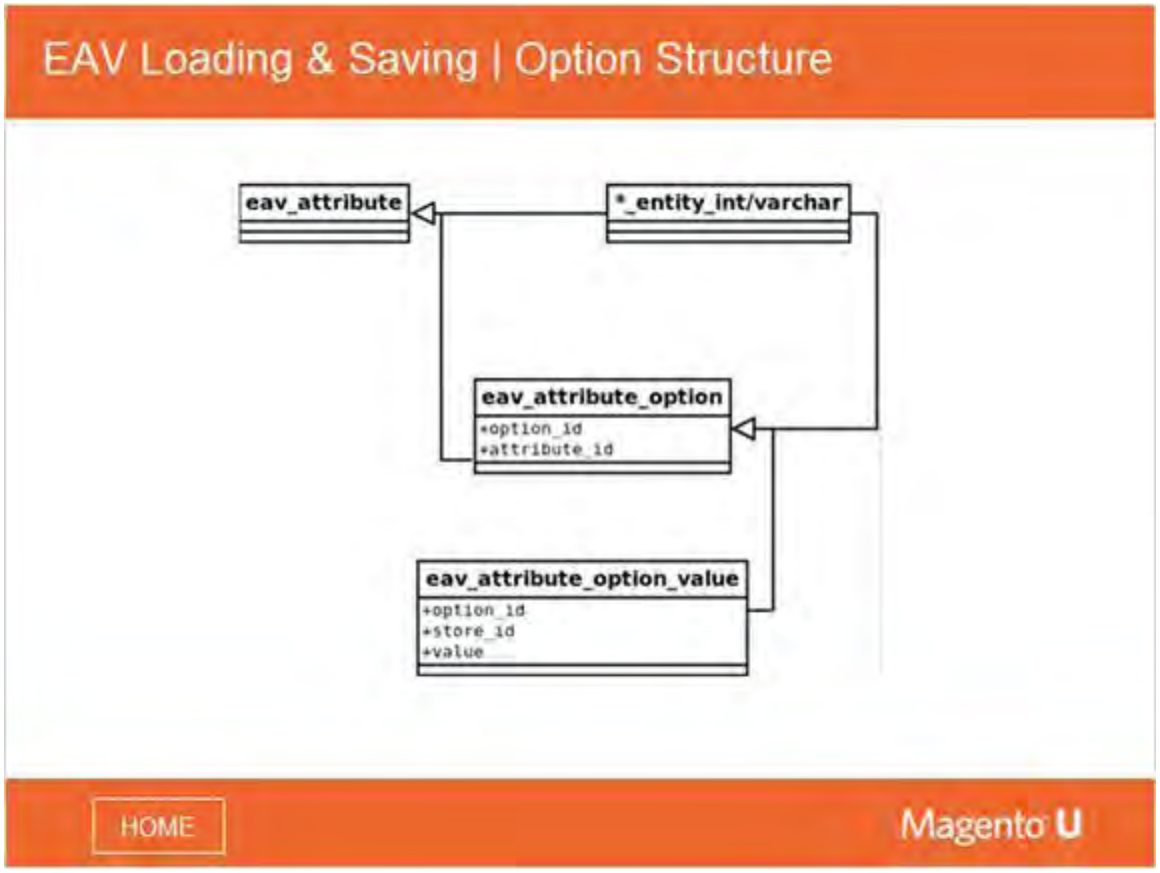
```
-- Get attribute option default and store-view-specific value

SELECT
    `main_table`.*, `tdv`.`value` AS `default_value`,
    `tsv`.`value` AS `store_default_value`,
    IF(tsv.value_id > 0, tsv.value, tdv.value) AS `value`

FROM `eav_attribute_option` AS `main_table`

INNER JOIN `eav_attribute_option_value` AS `tdv`
    ON tdv.option_id = main_table.option_id

LEFT JOIN `eav_attribute_option_value` AS `tsv`
    ON tsv.option_id = main_table.option_id
    AND tsv.store_id = '1'

WHERE
    (`attribute_id` = '150')
    AND (`main_table`.`option_id` IN('178'))
    AND (tdv.store_id = 0)

ORDER BY main_table.sort_order ASC, value ASC;
```

Notes:

For select or multiselect attributes, Magento 2 uses the table source model. This source model loads the option IDs and option values (labels) from the tables `eav_attribute_option` and `eav_attribute_option_value`.

The store-view-specific option values are fetched via a LEFT JOIN on the value table if there is a store-view-specific value; otherwise, it uses the default value for the store ID, "0".

6.12 EAV Loading & Saving | Option Structure

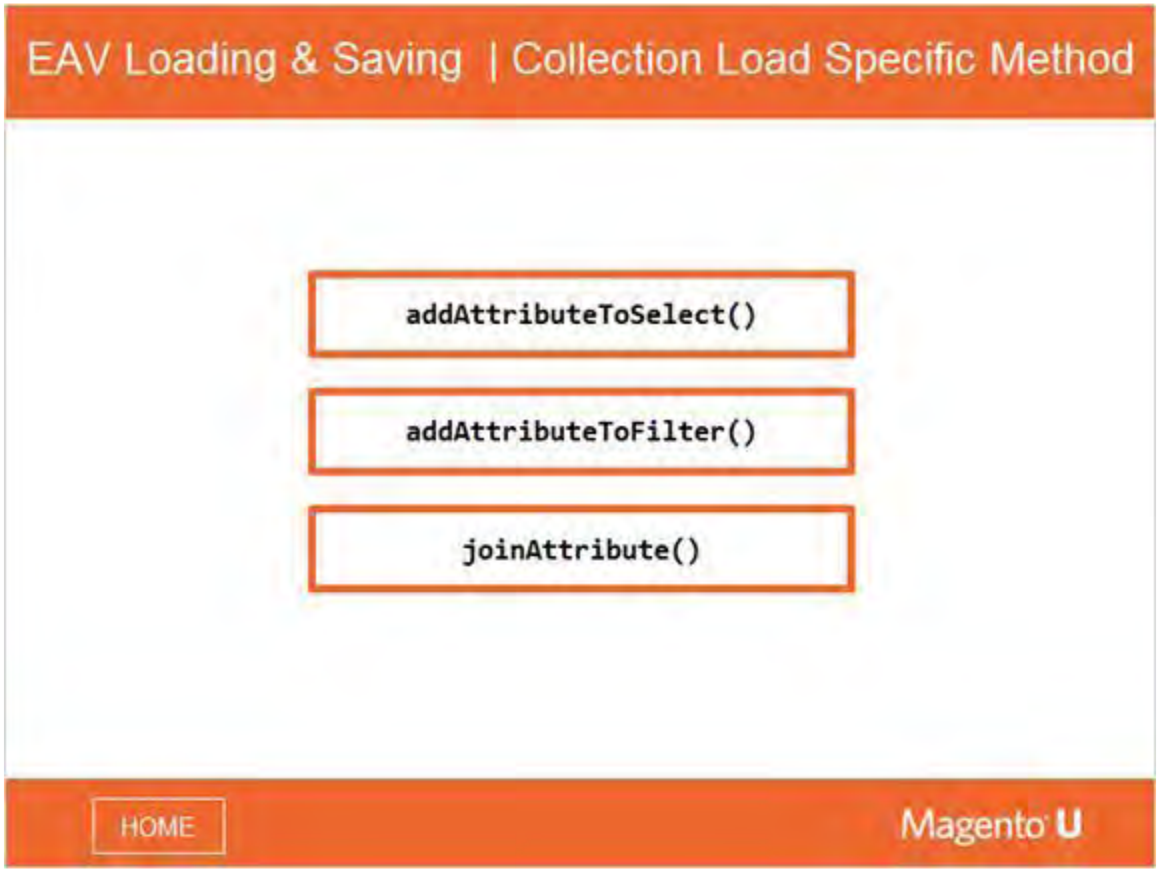


Notes:

This diagram shows the table structure of the EAV options, which is queried using the SQL shown on the previous slide.

The values stored on the entities are the option IDs from the `eav_attribute_option` table. For select attributes, the value is stored in the integer attribute value table. For multiselect, it is stored as a comma separated list of IDs in a varchar attribute value table.

6.13 EAV Loading & Saving | Collection Load Specific Method



EAV Loading & Saving | Collection Load Specific Method

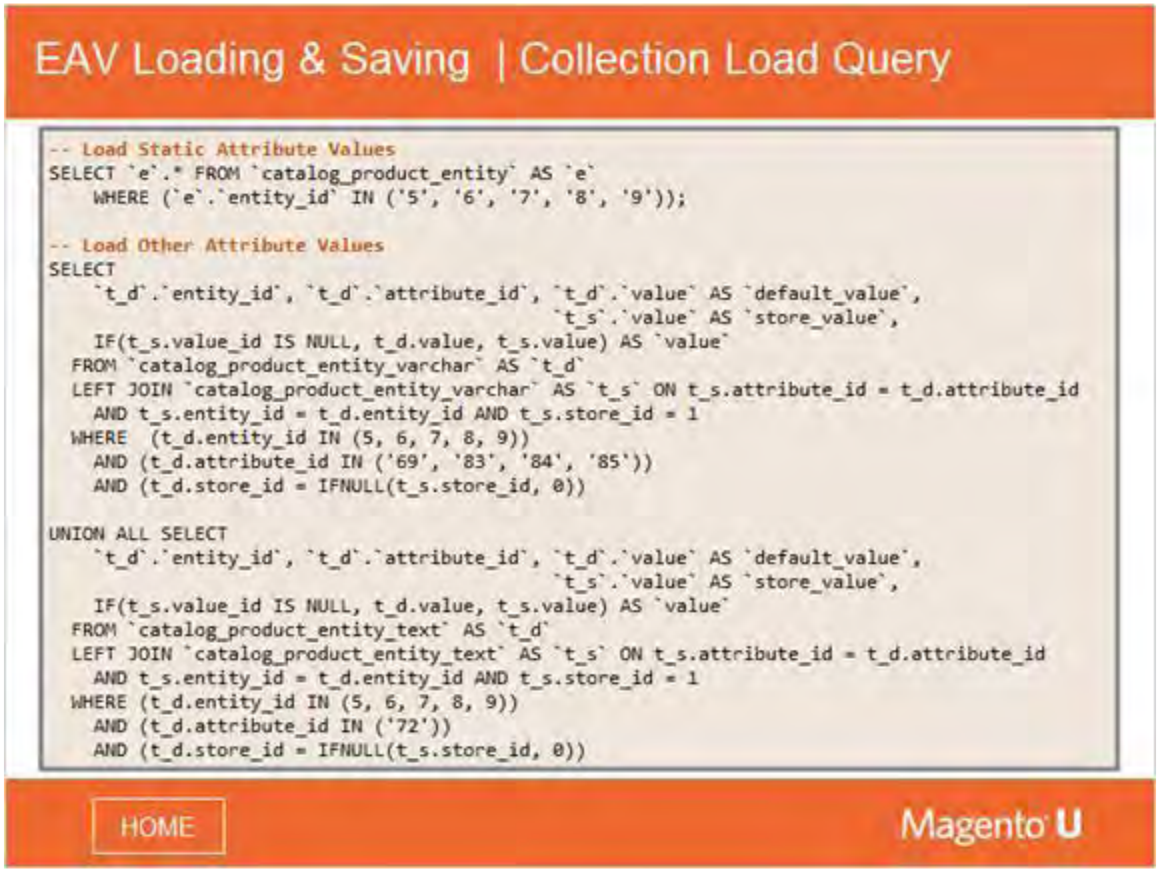
```
addAttributeToSelect()
addAttributeToFilter()
joinAttribute()
```

Notes:

There are several methods on EAV collections you call to influence the load process:

- **`addAttributeToSelect()`:** Adds the attributes value table to select and the attribute value to result.
- **`addAttributeToFilter()`:** Joins the attribute table specific for this attribute and adds a filter condition.
- **`joinAttribute()`:** Joins a value table from an attribute from another entity.

6.14 EAV Loading & Saving | Collection Load Query



EAV Loading & Saving | Collection Load Query

```
-- Load Static Attribute Values
SELECT `e`.* FROM `catalog_product_entity` AS `e`
    WHERE (`e`.`entity_id` IN ('5', '6', '7', '8', '9'));

-- Load Other Attribute Values
SELECT
    `t_d`.`entity_id`, `t_d`.`attribute_id`, `t_d`.`value` AS `default_value`,
                                             `t_s`.`value` AS `store_value`,
    IF(t_s.value_id IS NULL, t_d.value, t_s.value) AS `value`
  FROM `catalog_product_entity_varchar` AS `t_d`
  LEFT JOIN `catalog_product_entity_varchar` AS `t_s` ON t_s.attribute_id = t_d.attribute_id
    AND t_s.entity_id = t_d.entity_id AND t_s.store_id = 1
  WHERE  (t_d.entity_id IN (5, 6, 7, 8, 9))
    AND (t_d.attribute_id IN ('69', '83', '84', '85'))
    AND (t_d.store_id = IFNULL(t_s.store_id, 0))

UNION ALL SELECT
    `t_d`.`entity_id`, `t_d`.`attribute_id`, `t_d`.`value` AS `default_value`,
                                             `t_s`.`value` AS `store_value`,
    IF(t_s.value_id IS NULL, t_d.value, t_s.value) AS `value`
  FROM `catalog_product_entity_text` AS `t_d`
  LEFT JOIN `catalog_product_entity_text` AS `t_s` ON t_s.attribute_id = t_d.attribute_id
    AND t_s.entity_id = t_d.entity_id AND t_s.store_id = 1
  WHERE (t_d.entity_id IN (5, 6, 7, 8, 9))
    AND (t_d.attribute_id IN ('72'))
    AND (t_d.store_id = IFNULL(t_s.store_id, 0))
```

Notes:

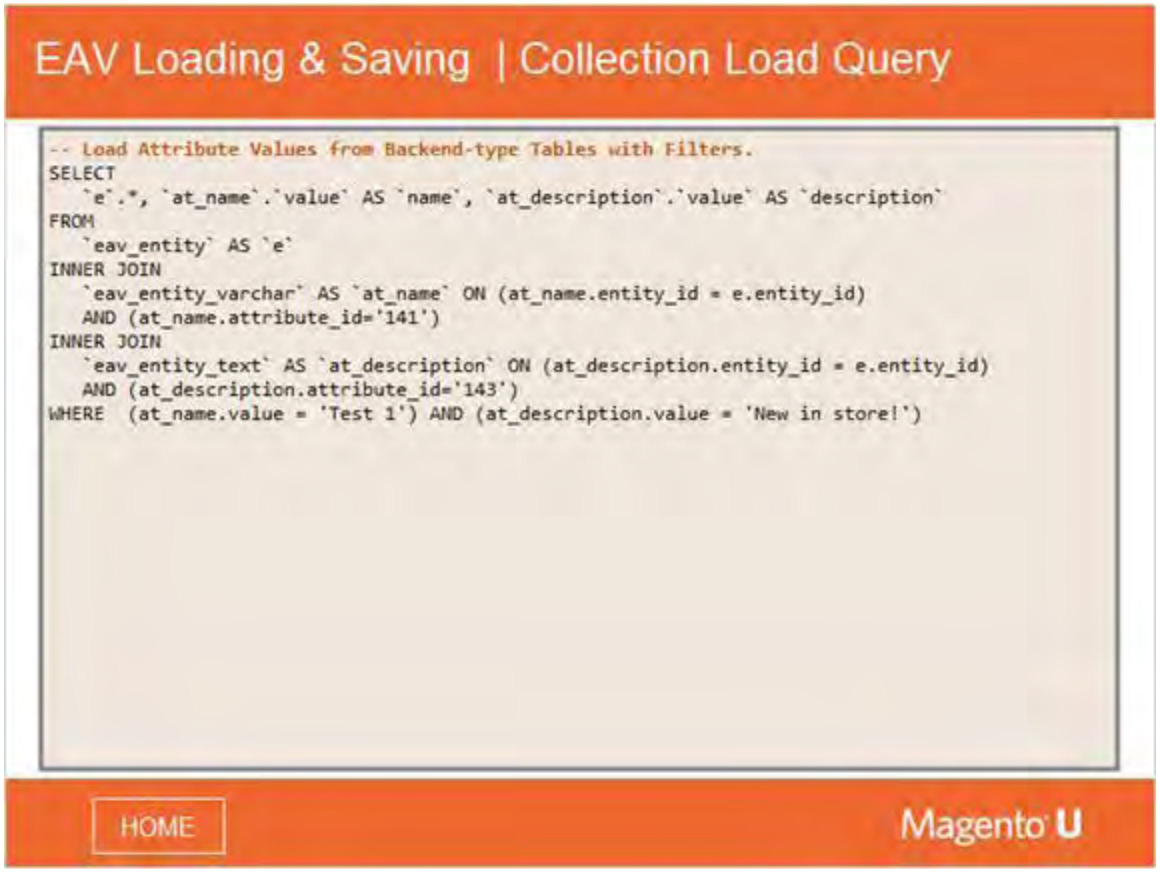
Static attributes are loaded from the entity table using a simple SELECT query.

When an attribute has been added to an EAV collection via `addAttributeToSelect()`, the attribute's backend type value table is added to the select using a UNION SELECT.

Catalog EAV entities with store scope values will LEFT JOIN the attribute value table to fetch the store view value, if it exists.

Note: Adding more attributes with the same backend type will use the same value table.

6.15 EAV Loading & Saving | Collection Load Query



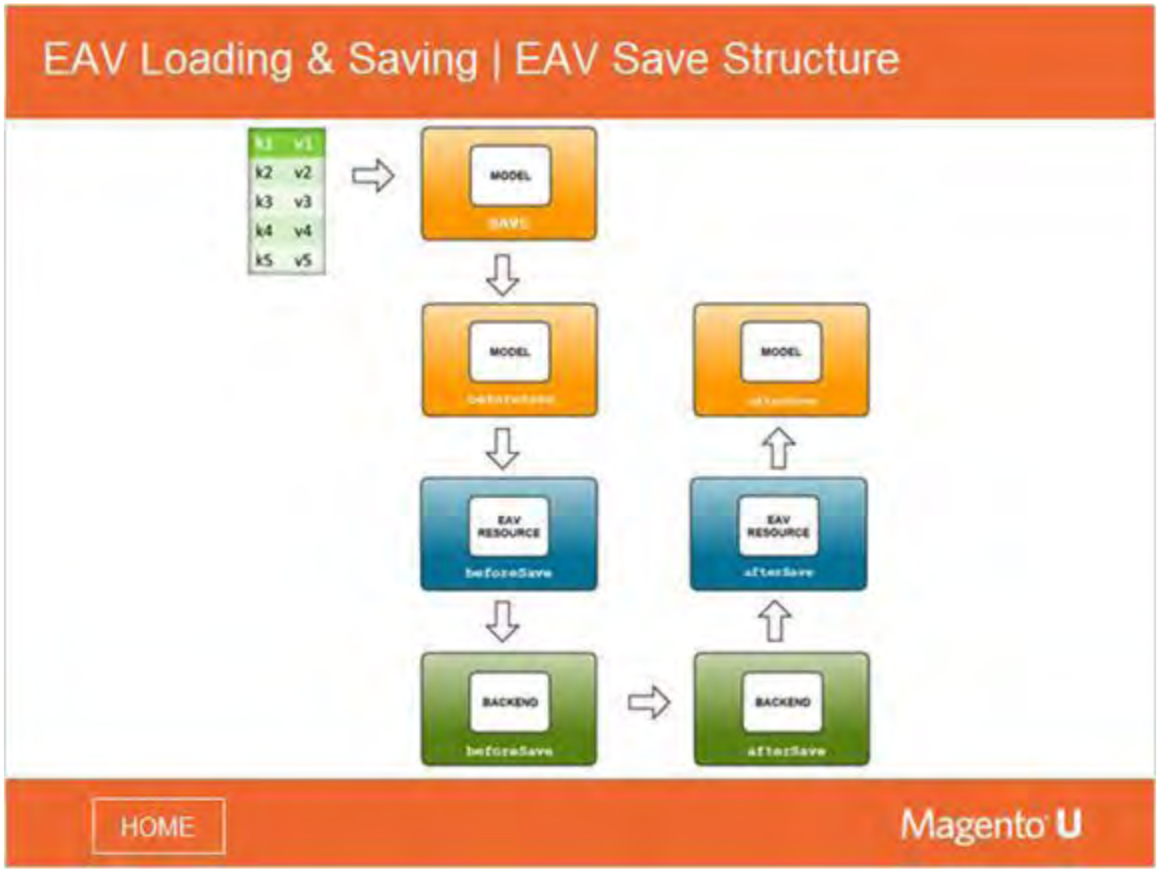
EAV Loading & Saving | Collection Load Query

```
-- Load Attribute Values from Backend-type Tables with Filters.
SELECT
   `e`.*, `at_name`.`value` AS `name`, `at_description`.`value` AS `description`
FROM
   `eav_entity` AS `e`
INNER JOIN
   `eav_entity_varchar` AS `at_name` ON (at_name.entity_id = e.entity_id)
   AND (at_name.attribute_id='141')
INNER JOIN
   `eav_entity_text` AS `at_description` ON (at_description.entity_id = e.entity_id)
   AND (at_description.attribute_id='143')
WHERE  (at_name.value = 'Test 1') AND (at_description.value = 'New in store!')
```

Notes:

This is the continuation of the code example from the previous slide.

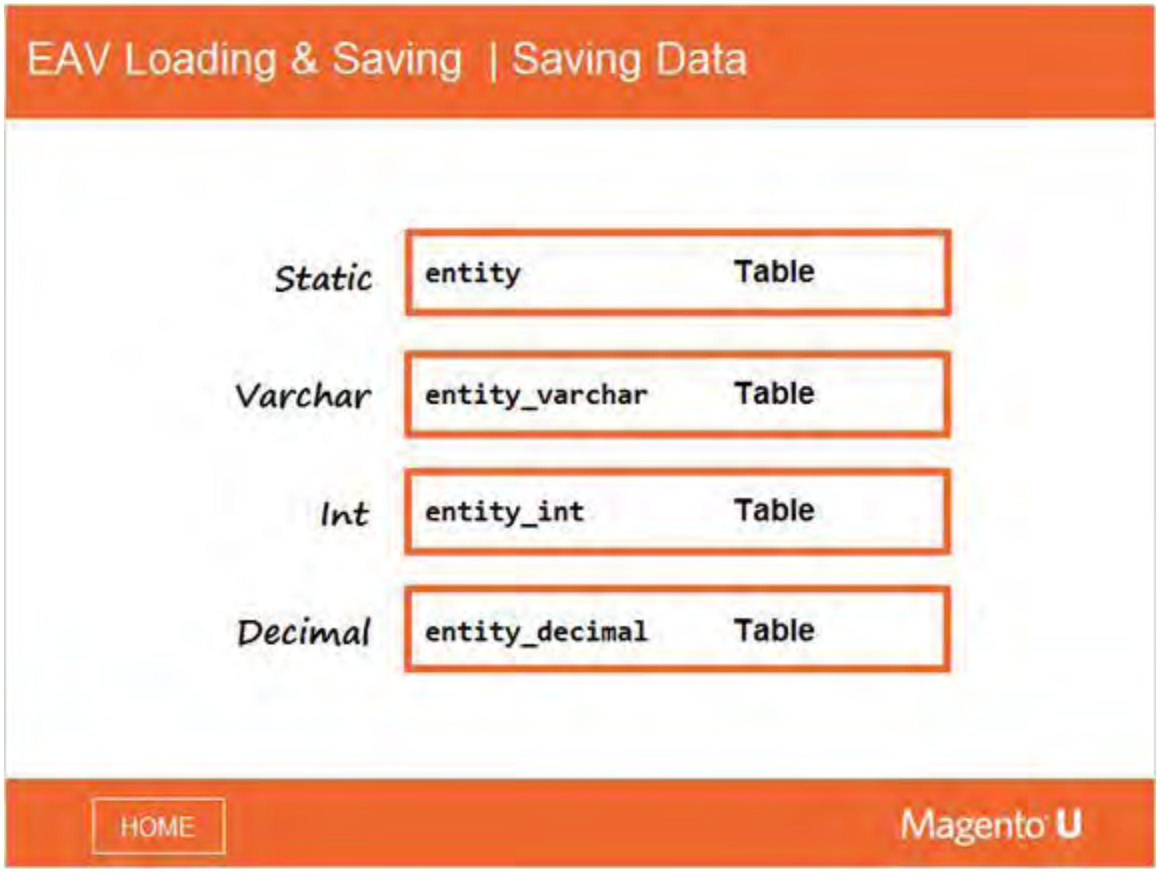
6.16 EAV Loading & Saving | EAV Save Structure



Notes:

Compared to the flat table save process, EAV adds one additional layer with `beforeSave()` and `afterSave()` method calls: the attribute backend model.

6.17 EAV Loading & Saving | Saving Data



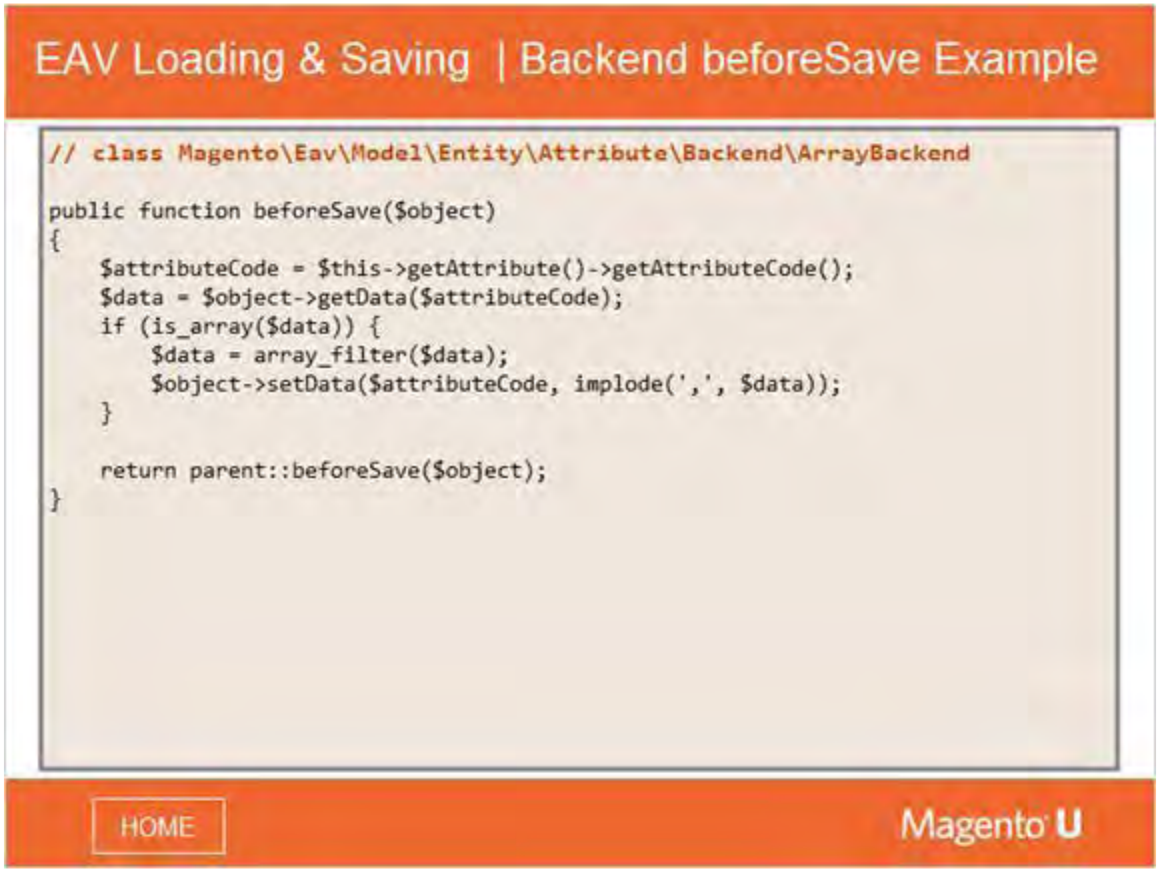
EAV Loading & Saving | Saving Data

Static	entity	Table
Varchar	entity_varchar	Table
Int	entity_int	Table
Decimal	entity_decimal	Table

Notes:

All attribute values are saved in the table according to their `backend_type` property.

6.18 EAV Loading & Saving | Backend beforeSave Example


EAV Loading & Saving | Backend beforeSave Example

```
// class Magento\Eav\Model\Entity\Attribute\Backend\ArrayBackend

public function beforeSave($object)
{
    $attributeCode = $this->getAttribute()->getAttributeCode();
    $data = $object->getData($attributeCode);
    if (is_array($data)) {
        $data = array_filter($data);
        $object->setData($attributeCode, implode(',', $data));
    }

    return parent::beforeSave($object);
}
```

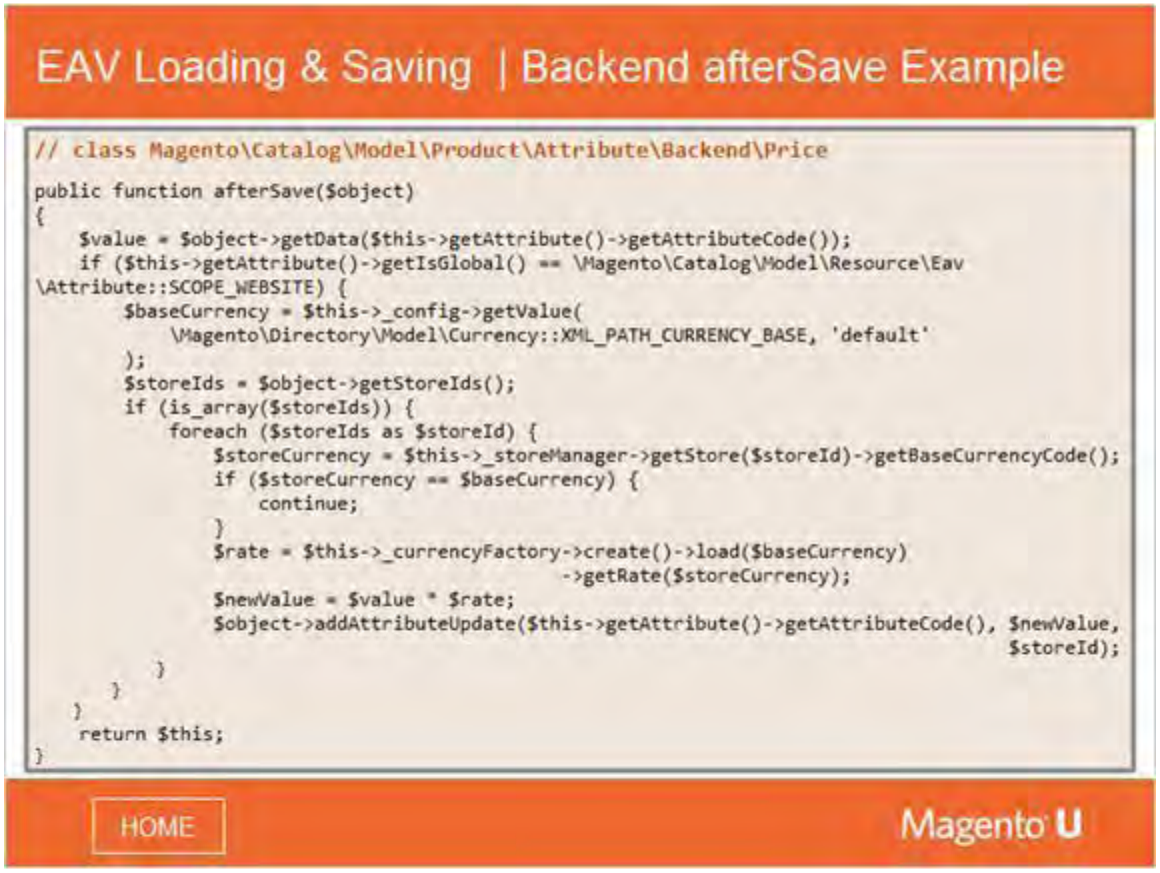
HOME
Magento U

Notes:

Attribute backend models can be used for the following, and more:

- Preparing values for storage in the database (as shown in the code example).
- Setting default values.
- Validation.
- Rehydration after loading.

6.19 EAV Loading & Saving | Backend afterSave Example



EAV Loading & Saving | Backend afterSave Example

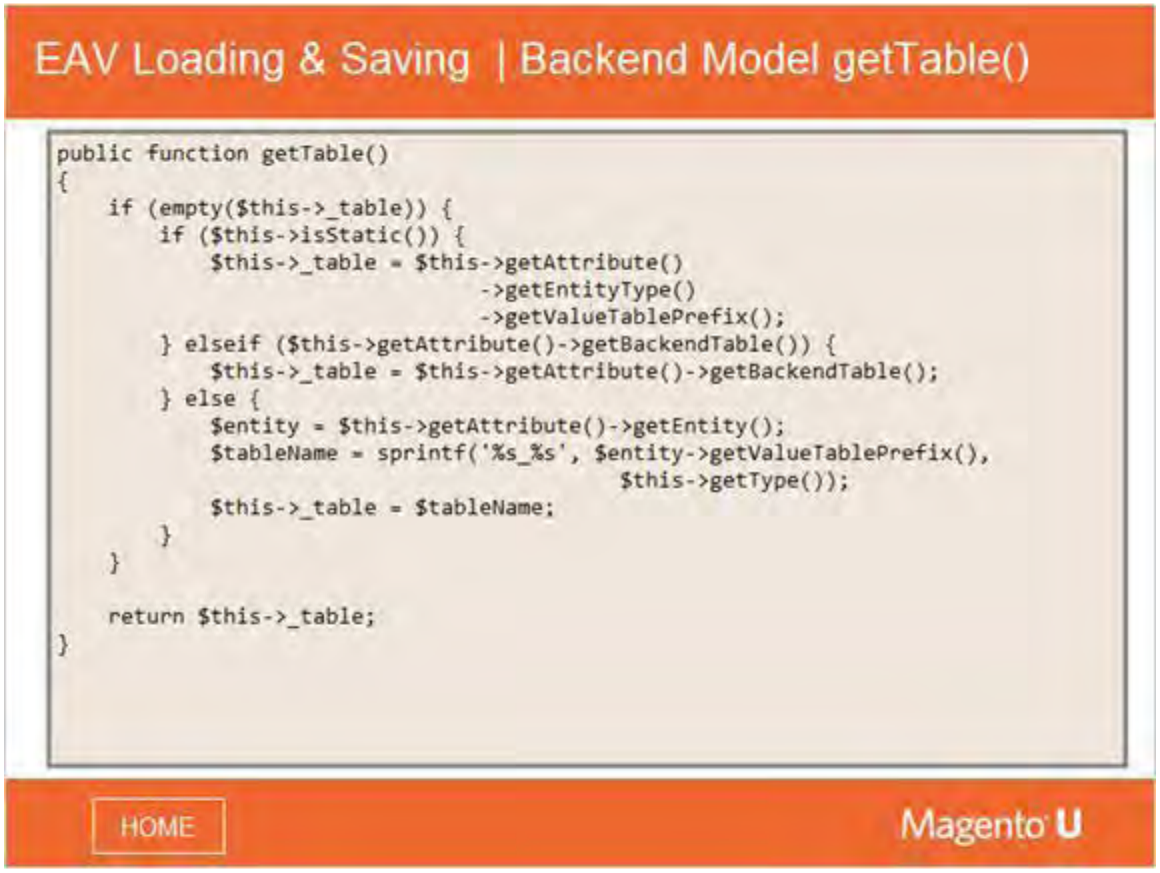
```
// class Magento\Catalog\Model\Product\Attribute\Backend\Price

public function afterSave($object)
{
    $value = $object->getData($this->getAttribute()->getAttributeCode());
    if ($this->getAttribute()->getIsGlobal() == \Magento\Catalog\Model\Resource\Eav
\Attribute::SCOPE_WEBSITE) {
        $baseCurrency = $this->_config->getValue(
            \Magento\Directory\Model\Currency::XML_PATH_CURRENCY_BASE, 'default'
        );
        $storeIds = $object->getStoreIds();
        if (is_array($storeIds)) {
            foreach ($storeIds as $storeId) {
                $storeCurrency = $this->_storeManager->getStore($storeId)->getBaseCurrencyCode();
                if ($storeCurrency == $baseCurrency) {
                    continue;
                }
                $rate = $this->_currencyFactory->create()->load($baseCurrency)
                                               ->getRate($storeCurrency);
                $newValue = $value * $rate;
                $object->addAttributeUpdate($this->getAttribute()->getAttributeCode(), $newValue,
                                                                                       $storeId);
            }
        }
    }
    return $this;
}
```

Notes:

The backend model of the price attribute saves the attribute value for different store views if they have different currencies in the backend model's `afterSave()` method.

Note: the code example shows a slightly abbreviated version of the real method.

6.20 EAV Loading & Saving | Backend Model getTable()



EAV Loading & Saving | Backend Model getTable()

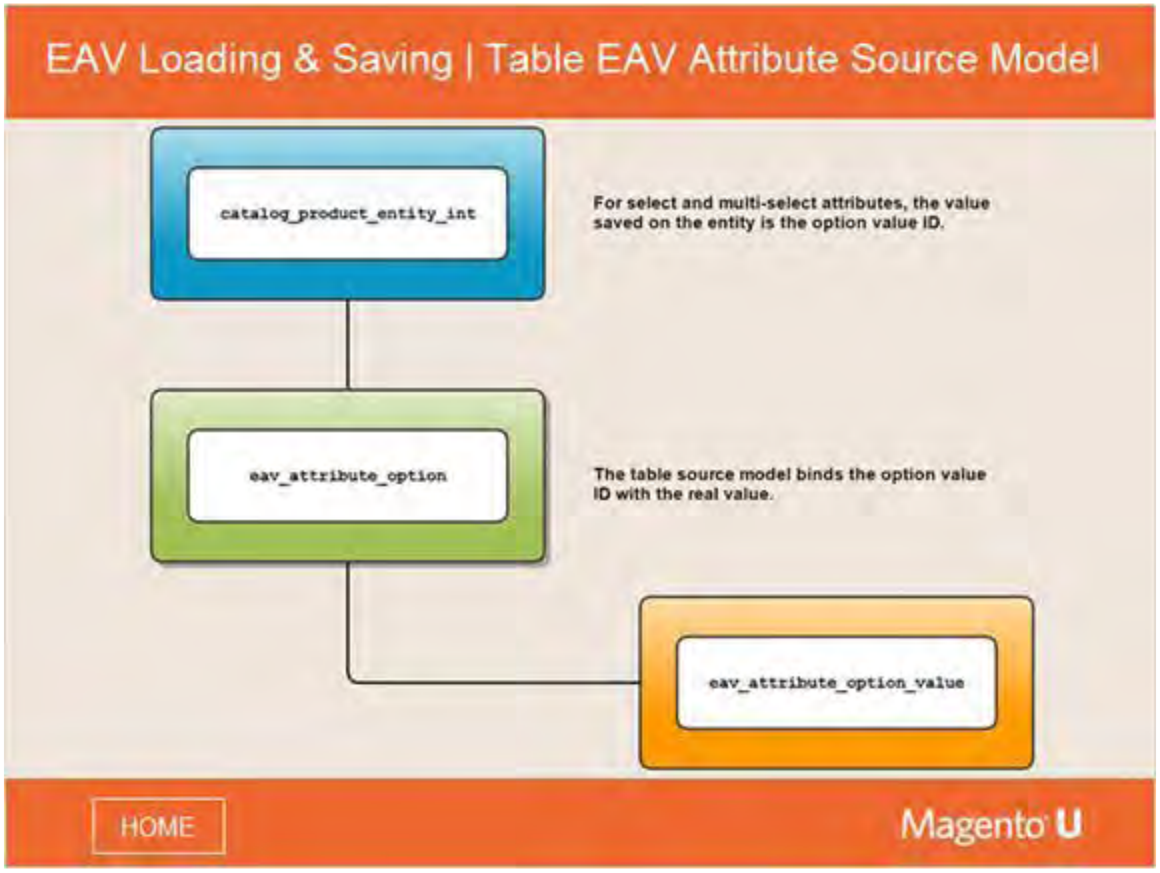
```
public function getTable()
{
    if (empty($this->_table)) {
        if ($this->isStatic()) {
            $this->_table = $this->getAttribute()
                                 ->getEntityType()
                                 ->getValueTablePrefix();
        } elseif ($this->getAttribute()->getBackendTable()) {
            $this->_table = $this->getAttribute()->getBackendTable();
        } else {
            $entity = $this->getAttribute()->getEntity();
            $tableName = sprintf('%s_%s', $entity->getValueTablePrefix(),
                                          $this->getType());
            $this->_table = $tableName;
        }
    }

    return $this->_table;
}
```

Notes:

When you need to load or save an EAV attribute value from any place, you can get the appropriate table by calling `getTable()` on the attribute backend model.

Example: `$product->getResource()->getAttribute($attributeCode)->getBackend()->getTable()`

6.21 EAV Loading & Saving | Table EAV Attribute Source Model


EAV Loading & Saving | Table EAV Attribute Source Model
catalog_product_entity_int
For select and multi-select attributes, the value saved on the entity is the option value ID.
eav_attribute_option
The table source model binds the option value ID with the real value.
eav_attribute_option_value
HOME
Magento U

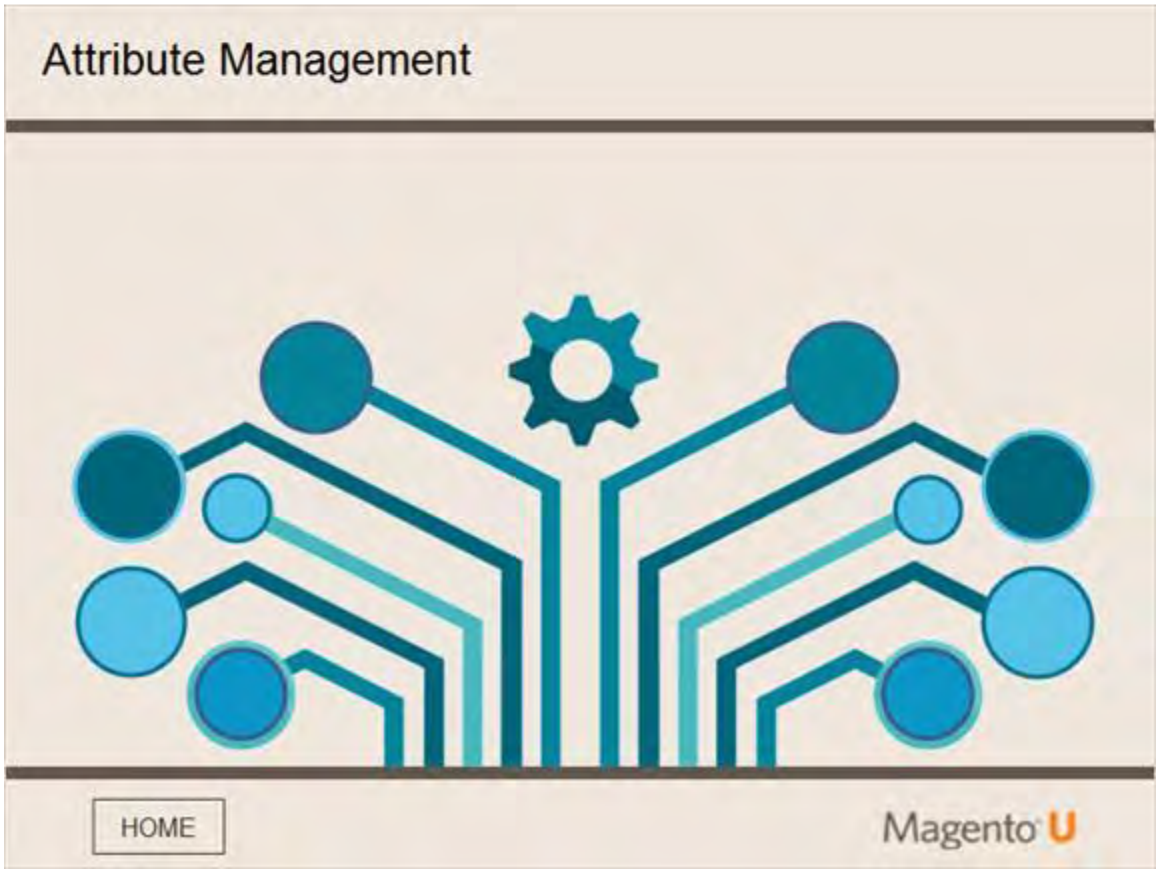
Notes:

The table EAV attribute source model:

- The native Magento select and multiselect attributes use the attribute source model `\Magento\Eav\Model\Entity\Attribute\Source\Table.`
- The value stored with the entity in the attribute value table is the `option ID`.
- The human-readable option label is stored in the `eav_attribute_option_value table`, mapped by the `eav_attribute_option` table.
- To get the option label from the `option ID`, use the associated attribute source model:
  `$eavEntity->getResource()->getAttribute($attributeCode)->getSource()->getOptionText($optionId);`
- To get the `option ID` for a given option label:
  `$eavEntity->getResource()->getAttribute($attributeCode)->getSource()->getOptionValue($optionText);`
- Retrieving the option label for products is easy:
  `$product->getAttributeText($attributeCode);`

7. Attribute Management

7.1 Attribute Management



Notes:

This module on EAV attribute management discusses the practical application of the EAV concepts in Magento 2.

7.2 Module Topics | Attribute Management

Module Topics | Attribute Management

In this module, we will discuss...

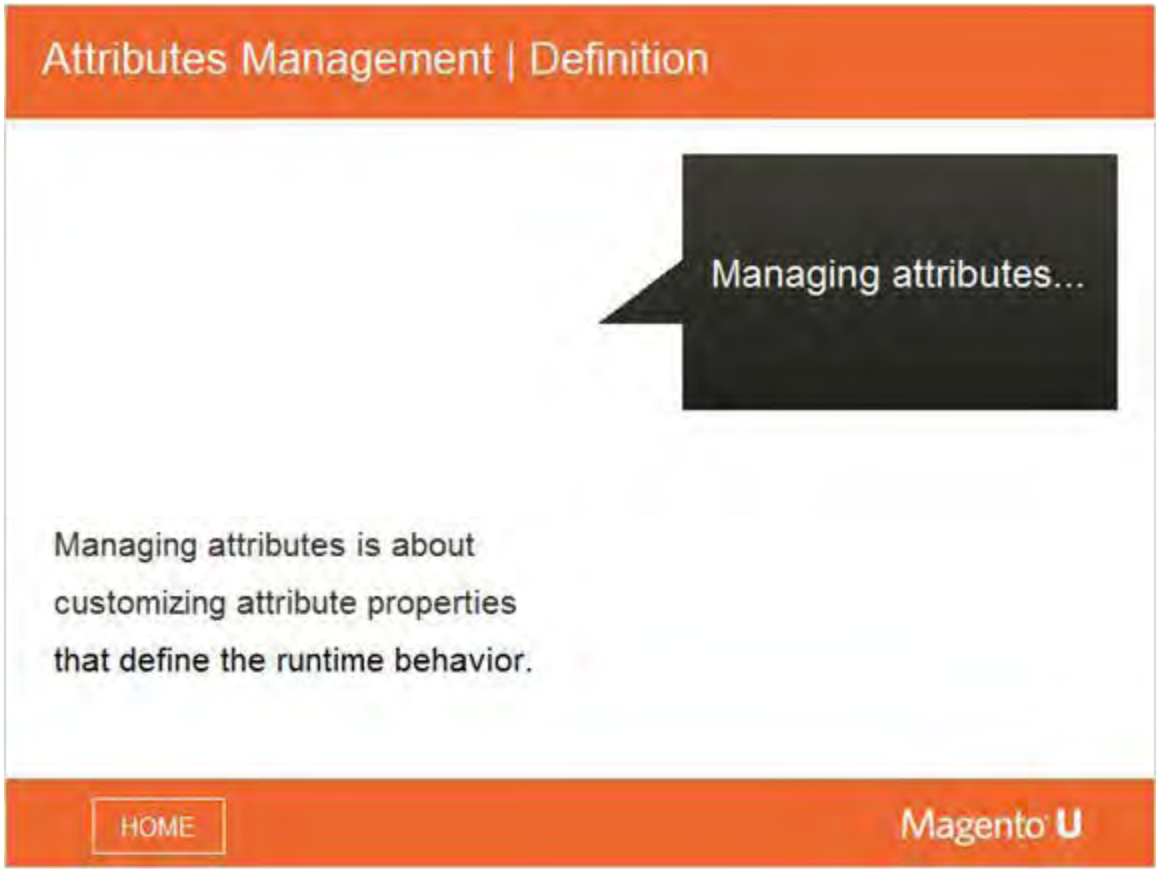
- Creating Attributes
- Customizing Attributes

HOME
Magento U

Notes:

In this module, we will discuss creating and customizing attributes.

7.3 Attributes Management | Definition



Attributes Management | Definition

Managing attributes...

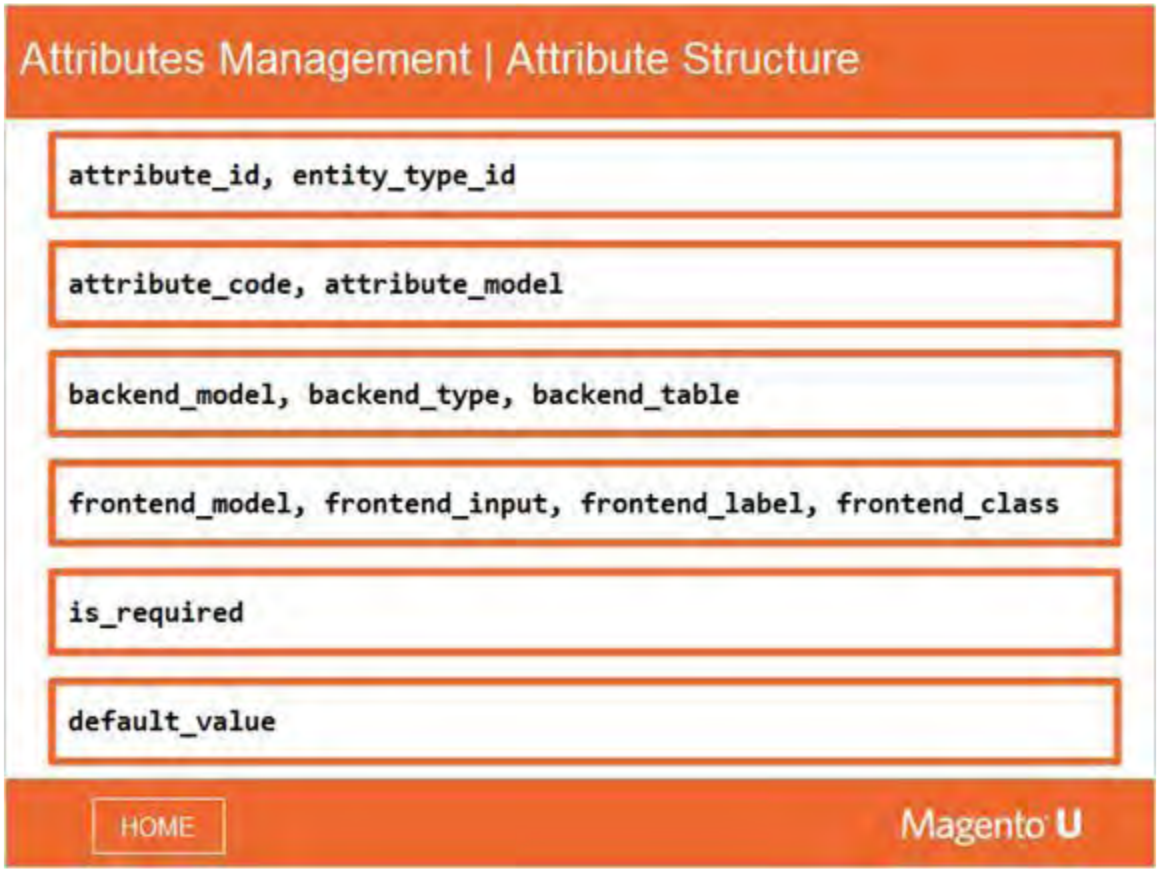
Managing attributes is about customizing attribute properties that define the runtime behavior.

HOME
Magento U

Notes:

Managing attributes is about customizing attribute properties that define the runtime behavior.

7.4 Attributes Management | Attribute Structure


Attributes Management | Attribute Structure

attribute_id, entity_type_id

attribute_code, attribute_model

backend_model, backend_type, backend_table

frontend_model, frontend_input, frontend_label, frontend_class

is_required

default_value

HOME
Magento U

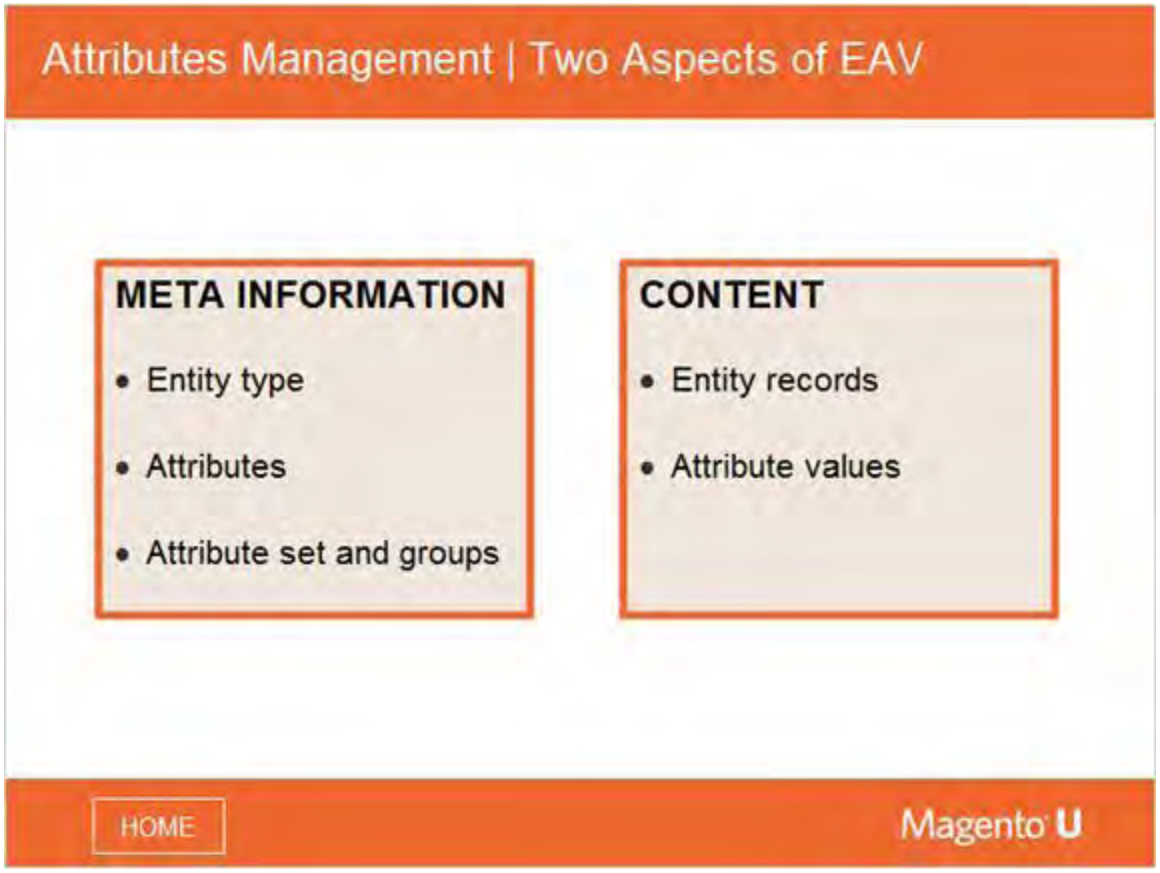
Notes:

Each attribute may have some additional `entity_type`-specific properties, which would be stored in additional tables such as `catalog_eav_attribute`.

- **`attribute_id`:** Unique ID from the `eav_attribute` table.
- **`entity_type_id`:** ID of the associated entity type.
- **`attribute_code`:** The attribute code must be unique within the associated entity type.
- **`attribute_model`:** Optional alternative model to use. Defaults to `Magento\Eav\Model\Entity\Attribute` if not specified.
- **`backend_model`:** Optional alternative backend model. Defaults to `Magento\Eav\Model\Entity\Attribute\Backend\DefaultBackend` if not specified.
- **`backend_type`:** Data type for the attribute; one of static, varchar, datetime, int, text or decimal. Custom types are possible, too.
- **`backend_table`:** Optional value table; defaults to the entity table name with the `backend_type` appended to it.
- **`frontend_model`:** Optional alternative frontend model; defaults to `Magento\Eav\Model\Entity\Attribute\Frontend\DefaultFrontend` if not specified.
- **`frontend_input`:** Input type for the attribute if the adminhtml form is rendered automatically.

- **`frontend_label:`** Default label if the adminhtml form is rendered automatically.
- **`frontend_class:`** Optional CSS class name, which will be added to the adminhtml input element if the adminhtml form is rendered automatically. This is useful for validation.
- **`is_required:`** Enables JavaScript validation if the adminhtml form is rendered automatically; is evaluated during `DataFlow` and `ImportExport` import processes.
- **`default_value:`** Optional default value for the attribute; displayed by the frontend model if the attribute has no value on an entity.

7.5 Attributes Management | Two Aspects of EAV



Attributes Management | Two Aspects of EAV

META INFORMATION

- Entity type
- Attributes
- Attribute set and groups

CONTENT

- Entity records
- Attribute values

HOME
Magento U

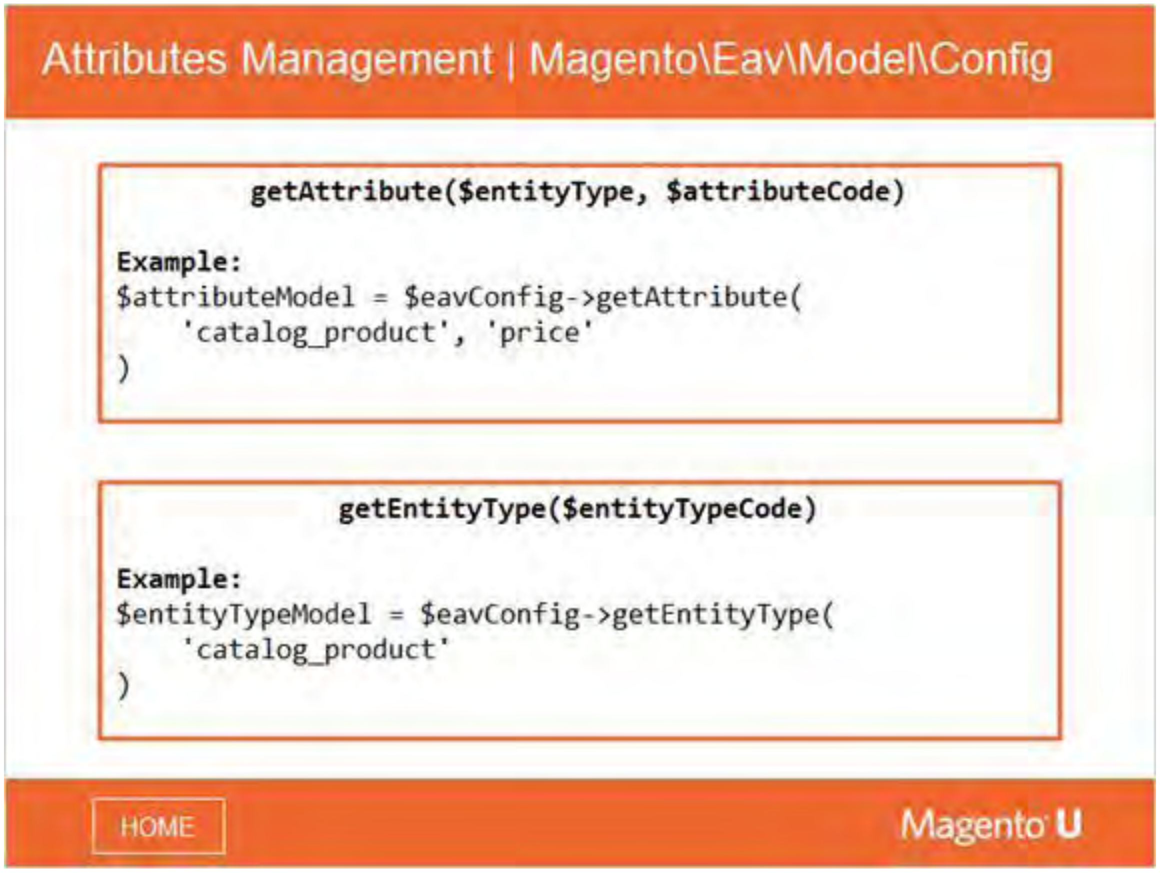
Notes:

Again recall that there are two aspects to the EAV process -- managing relationships and managing content.

Information about the “meta aspect” of the EAV system can be accessed using the EAV config model `Magento\Eav\Model\Config`.

It aggregates access to the meta information about entity types and attributes.

7.6 Attributes Management | Magento\Eav\Model\Config



Attributes Management | Magento\Eav\Model\Config

```
getAttribute($entityType, $attributeCode)

Example:
$attributeModel = $eavConfig->getAttribute(
    'catalog_product', 'price'
)
```

```
getEntityType($entityTypeCode)

Example:
$entityTypeModel = $eavConfig->getEntityType(
    'catalog_product'
)
```

HOME
Magento U

Notes:

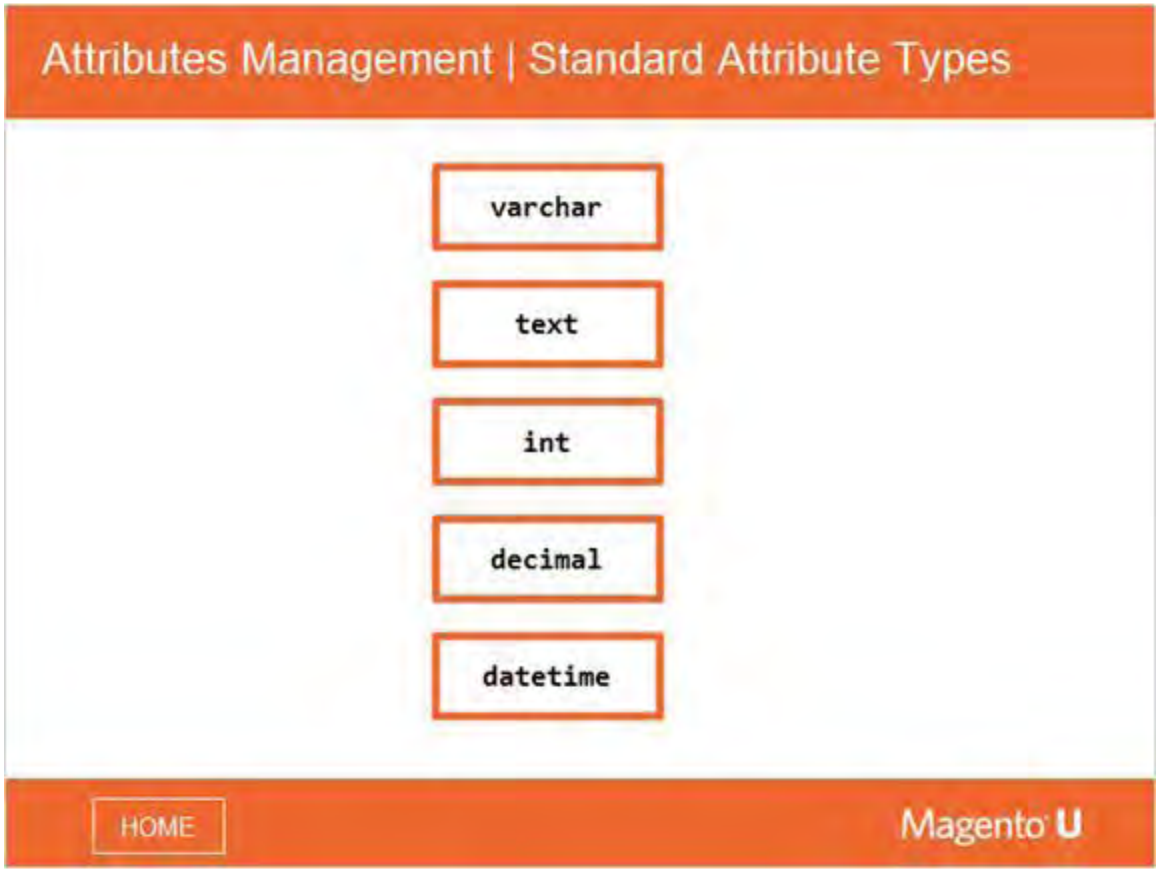
There are a couple of generic EAV classes that you can use to get information. One of these is `Magento\Eav\Model\Config`, mentioned on the previous slide. This is a very useful class for getting meta information.

Accessing the EAV entity type and attribute models is easy via the `Magento\Eav\Model\Config` class instance.

- **`getAttribute($entityType, $attributeCode)`:** Allows you to get an attribute instance.
  Example: `$eavConfig->getAttribute('catalog_product', 'price')`

- **`getEntityType($entityTypeCode)`:** Returns an entity type instance.
  Example: `$eavConfig->getEntityType('catalog_product')`

7.7 Attributes Management | Standard Attribute Types


Attributes Management | Standard Attribute Types

varchar

text

int

decimal

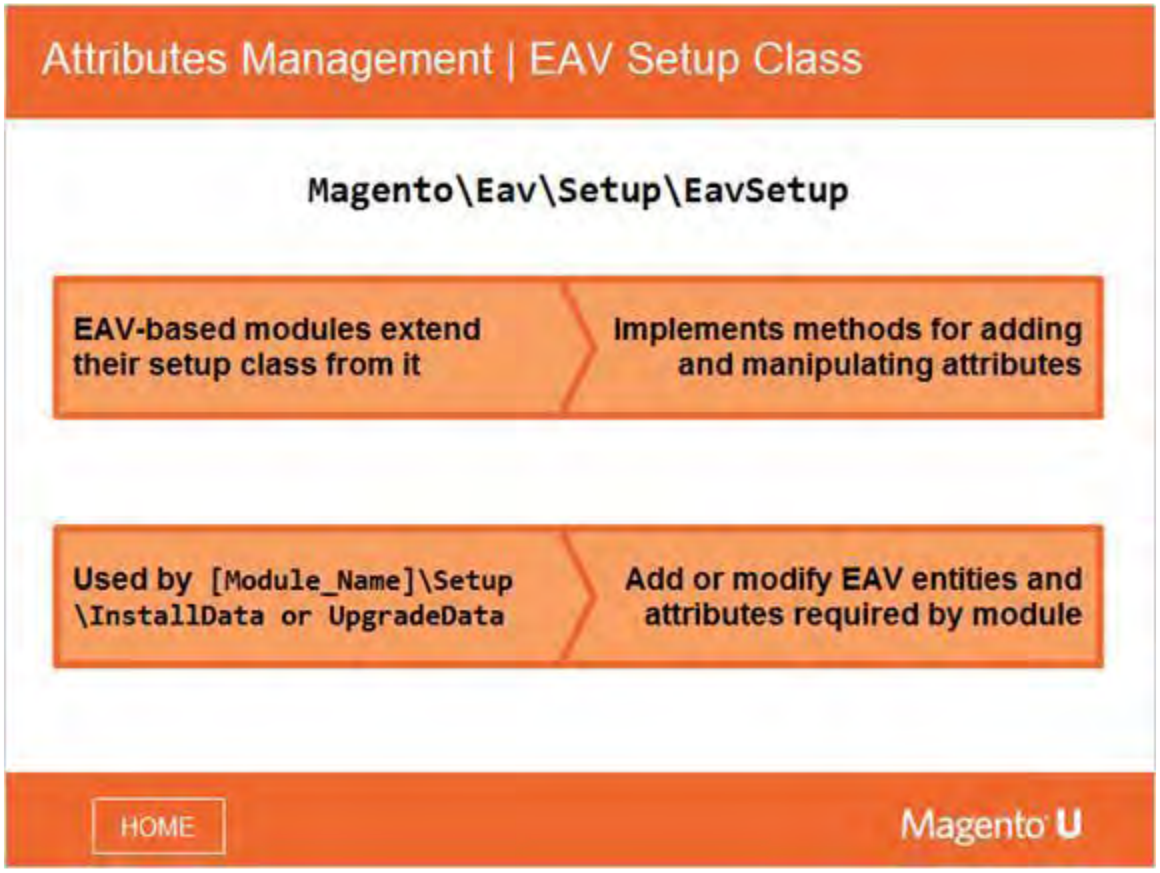
datetime

Notes:

The Magento EAV system offers the listed `backend_type` variations by default. Custom types can be used as needed, too.

There is also the static attribute `backend_type`, which indicates that the attributes values are stored in the entity table.

7.8 Attributes Management | EAV Setup Class



Attributes Management | EAV Setup Class

`Magento\Eav\Setup\EavSetup`

EAV-based modules extend their setup class from it → Implements methods for adding and manipulating attributes

Used by `[Module_Name]\Setup\InstallData` or `UpgradeData` → Add or modify EAV entities and attributes required by module

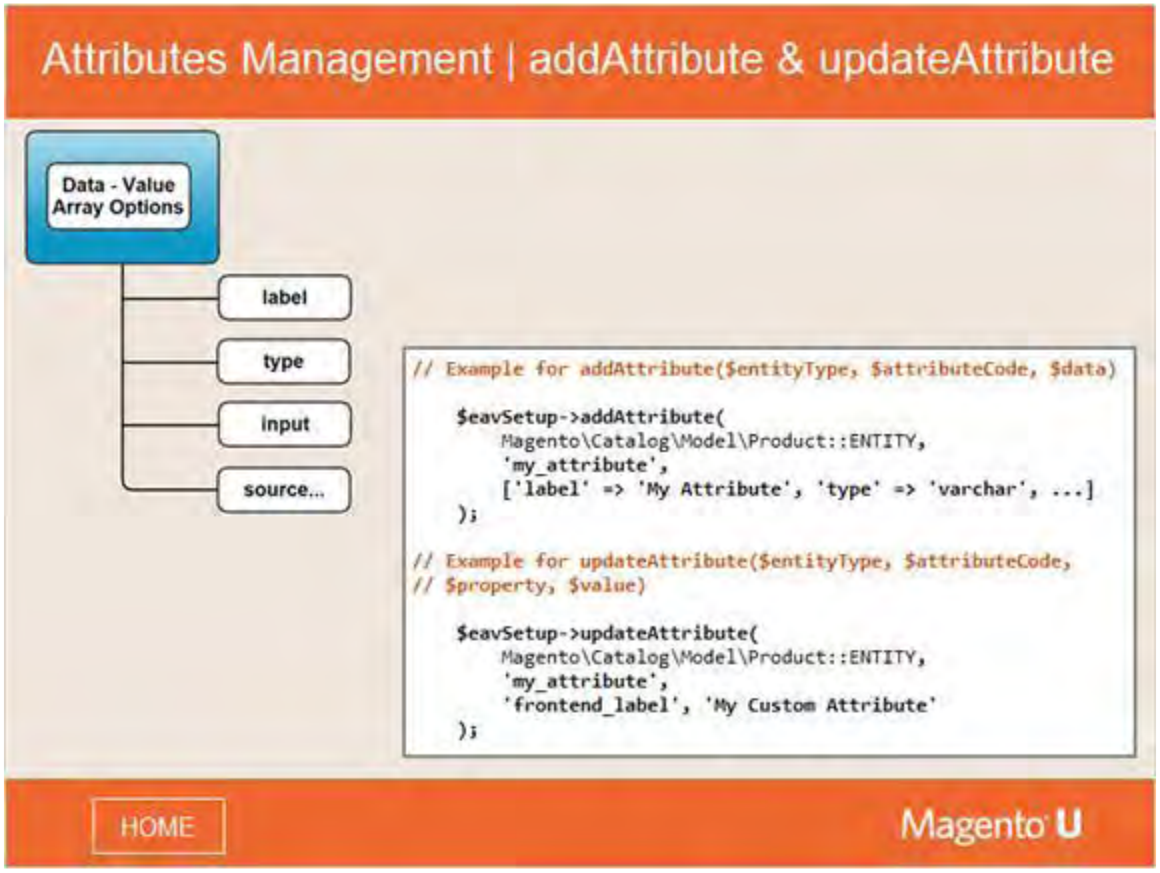
HOME
Magento U

Notes:

The EAV setup class `Magento\Eav\Setup\EavSetup` must be used to work with attributes and entity types in module setup scripts. For this reason, it is a very important class.

The EAV setup class implements methods for manipulating entity types and attributes. It is used by the `InstallData` or `UpgradeData` setup classes of modules to add or modify EAV entities or attributes.

7.9 Attributes Management | addAttribute & updateAttribute



Attributes Management | addAttribute & updateAttribute

```
// Example for addAttribute($entityType, $attributeCode, $data)

    $eavSetup->addAttribute(
        Magento\Catalog\Model\Product::ENTITY,
        'my_attribute',
        ['label' => 'My Attribute', 'type' => 'varchar', ...]
    );

// Example for updateAttribute($entityType, $attributeCode,
// $property, $value)

    $eavSetup->updateAttribute(
        Magento\Catalog\Model\Product::ENTITY,
        'my_attribute',
        'frontend_label', 'My Custom Attribute'
    );
```

Notes:

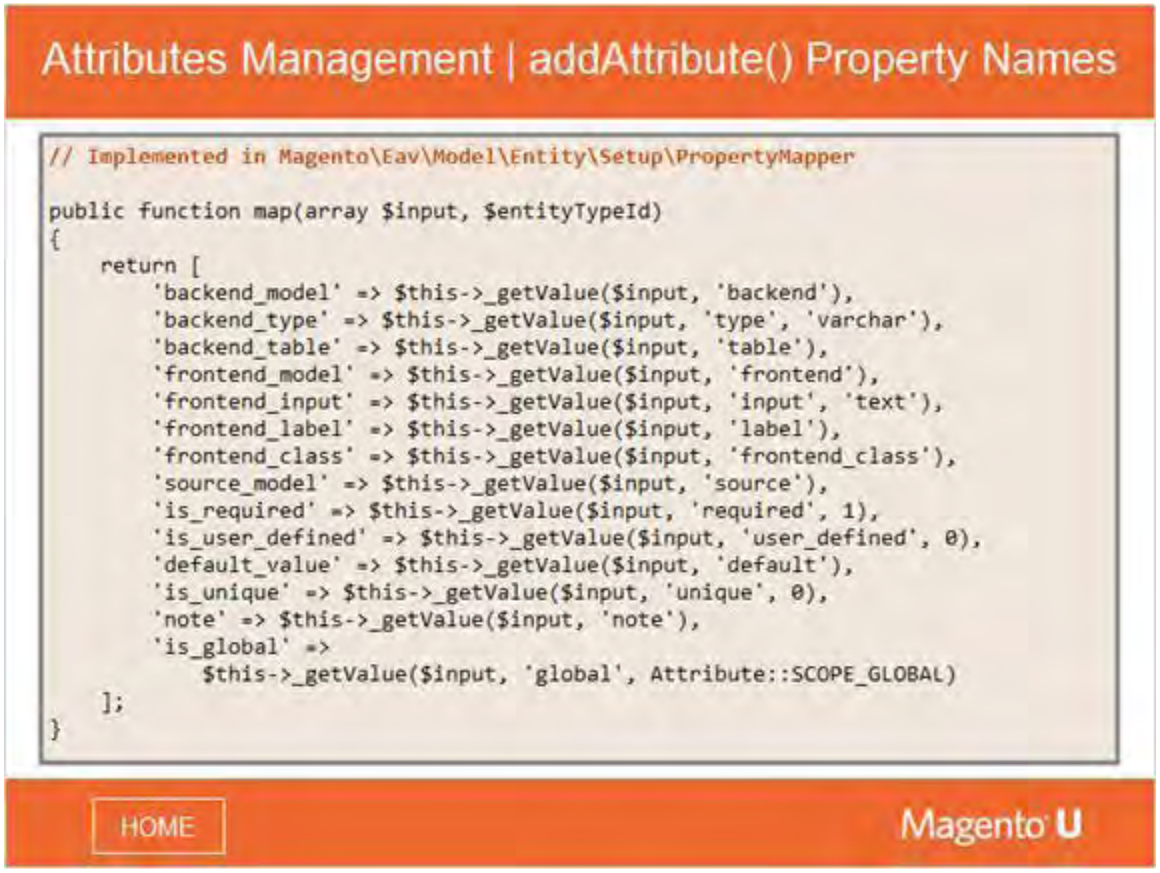
Two examples of the methods that the Setup class implements are `addAttribute()` and `updateAttribute()`.

Note: The `updateAttribute()` method `$property` argument exactly matches the column name in the database, while for `addAttribute()` the property names of the `$data` array are filtered through `Magento\Eav\Model\Entity\Setup\PropertyMapperInterface::map()` by the setup class.

For example, `addAttribute()` uses `label` instead of `frontend_label`.

This mapping is defined in the mapper implementation `Magento\Eav\Model\Entity\Setup\PropertyMapper`.

7.10 Attributes Management | addAttribute() Property Names



Attributes Management | addAttribute() Property Names

```
// Implemented in Magento\Eav\Model\Entity\Setup\PropertyMapper

public function map(array $input, $entityTypeId)
{
    return [
        'backend_model' => $this->_getValue($input, 'backend'),
        'backend_type' => $this->_getValue($input, 'type', 'varchar'),
        'backend_table' => $this->_getValue($input, 'table'),
        'frontend_model' => $this->_getValue($input, 'frontend'),
        'frontend_input' => $this->_getValue($input, 'input', 'text'),
        'frontend_label' => $this->_getValue($input, 'label'),
        'frontend_class' => $this->_getValue($input, 'frontend_class'),
        'source_model' => $this->_getValue($input, 'source'),
        'is_required' => $this->_getValue($input, 'required', 1),
        'is_user_defined' => $this->_getValue($input, 'user_defined', 0),
        'default_value' => $this->_getValue($input, 'default'),
        'is_unique' => $this->_getValue($input, 'unique', 0),
        'note' => $this->_getValue($input, 'note'),
        'is_global' =>
            $this->_getValue($input, 'global', Attribute::SCOPE_GLOBAL)
    ];
}
```

Notes:

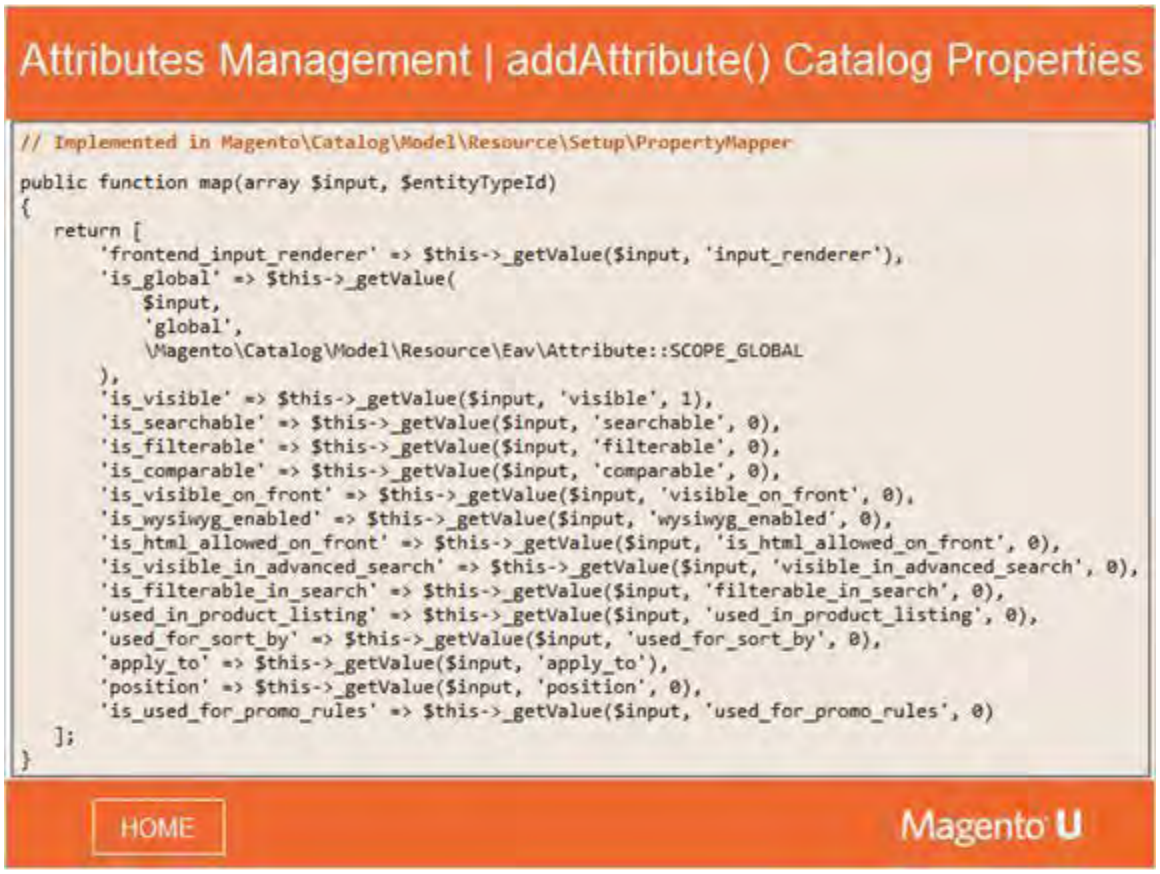
Two examples of the methods that the Setup class implements are `addAttribute()` and `updateAttribute()`.

Note: The `updateAttribute()` method `$property` argument exactly matches the column name in the database, while for `addAttribute()` the property names of the `$data` array are filtered through `Magento\Eav\Model\Entity\Setup\PropertyMapperInterface::map()` by the setup class.

For example, `addAttribute()` uses `label` instead of `frontend_label`.

This mapping is defined in the mapper implementation `Magento\Eav\Model\Entity\Setup\PropertyMapper`.

7.11 Attributes Management | addAttribute() Catalog Properties


Attributes Management | addAttribute() Catalog Properties

```
// Implemented in Magento\Catalog\Model\Resource\Setup\PropertyMapper

public function map(array $input, $entityTypeId)
{
    return [
        'frontend_input_renderer' => $this->_getValue($input, 'input_renderer'),
        'is_global' => $this->_getValue(
            $input,
            'global',
            \Magento\Catalog\Model\Resource\Eav\Attribute::SCOPE_GLOBAL
        ),
        'is_visible' => $this->_getValue($input, 'visible', 1),
        'is_searchable' => $this->_getValue($input, 'searchable', 0),
        'is_filterable' => $this->_getValue($input, 'filterable', 0),
        'is_comparable' => $this->_getValue($input, 'comparable', 0),
        'is_visible_on_front' => $this->_getValue($input, 'visible_on_front', 0),
        'is_wysiwyg_enabled' => $this->_getValue($input, 'wysiwyg_enabled', 0),
        'is_html_allowed_on_front' => $this->_getValue($input, 'is_html_allowed_on_front', 0),
        'is_visible_in_advanced_search' => $this->_getValue($input, 'visible_in_advanced_search', 0),
        'is_filterable_in_search' => $this->_getValue($input, 'filterable_in_search', 0),
        'used_in_product_listing' => $this->_getValue($input, 'used_in_product_listing', 0),
        'used_for_sort_by' => $this->_getValue($input, 'used_for_sort_by', 0),
        'apply_to' => $this->_getValue($input, 'apply_to'),
        'position' => $this->_getValue($input, 'position', 0),
        'is_used_for_promo_rules' => $this->_getValue($input, 'used_for_promo_rules', 0)
    ];
}
```

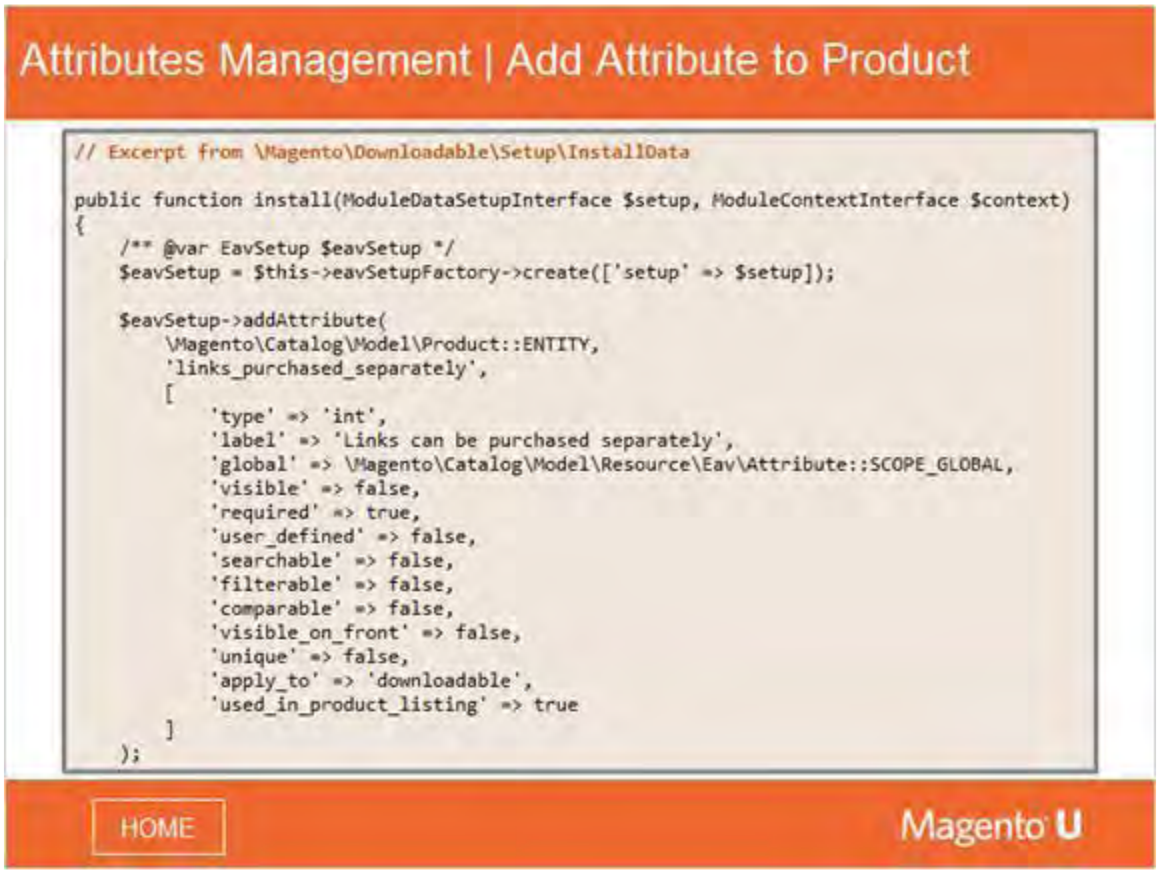
Notes:

To accommodate additional attribute table column mappings, a composite property mapper is used: Magento\Eav\Model\Entity\Setup\PropertyMapper\Composite, which is configured using `di.xml`.

For example, in the `Mage_Catalog di.xml` file, the catalog property mapper is added to the composite:

```
<type name="Magento\Eav\Model\Entity\Setup\PropertyMapper\Composite">
    <arguments>
        <argument name="propertyMappers" xsi:type="array">
            <item name="catalog" xsi:type="string">Magento\Catalog\Model\Resource\Setup\PropertyMapper</item>
        </argument>
    </arguments>
</type>
```

7.12 Attributes Management | Add Attribute to Product



Attributes Management | Add Attribute to Product

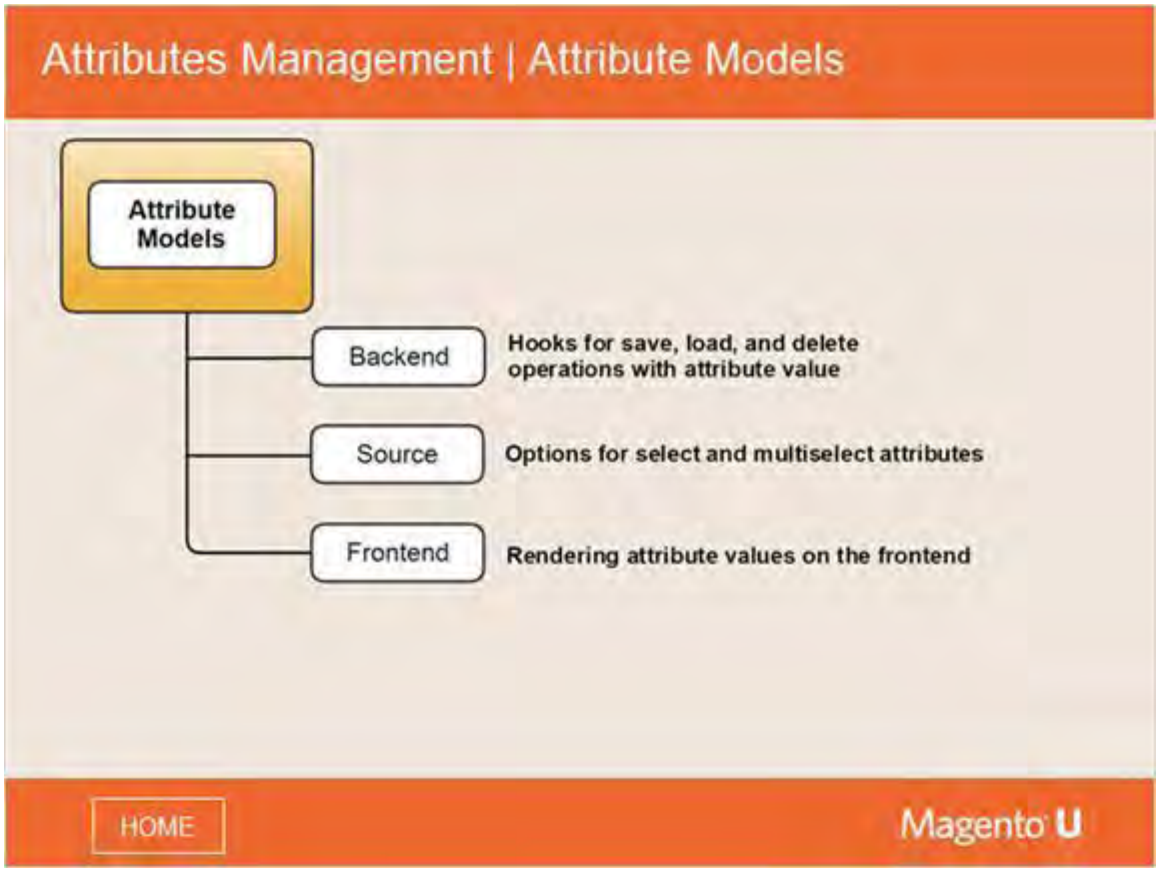
```
// Excerpt from \Magento\Downloadable\Setup\InstallData

public function install(ModuleDataSetupInterface $setup, ModuleContextInterface $context)
{
    /** @var EavSetup $eavSetup */
    $eavSetup = $this->eavSetupFactory->create(['setup' => $setup]);

    $eavSetup->addAttribute(
        \Magento\Catalog\Model\Product::ENTITY,
        'links_purchased_separately',
        [
            'type' => 'int',
            'label' => 'Links can be purchased separately',
            'global' => \Magento\Catalog\Model\Resource\Eav\Attribute::SCOPE_GLOBAL,
            'visible' => false,
            'required' => true,
            'user_defined' => false,
            'searchable' => false,
            'filterable' => false,
            'comparable' => false,
            'visible_on_front' => false,
            'unique' => false,
            'apply_to' => 'downloadable',
            'used_in_product_listing' => true
        ]
    );
```

Notes:

A code example of adding an attribute to a product.

Note: Properties do not have to be specified in the `$input` array if the default values are sufficient.

7.13 Attributes Management | Attribute Models

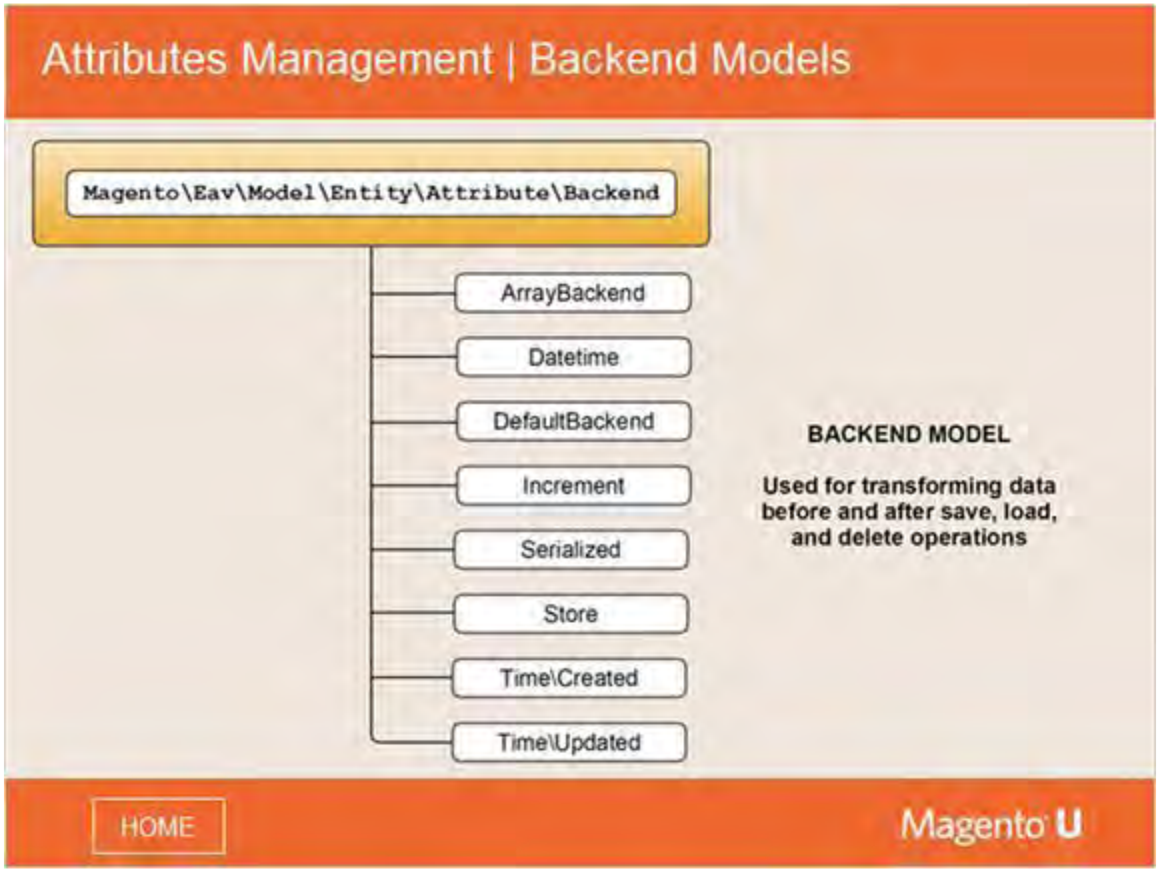

Attributes Management | Attribute Models
Attribute Models
Backend
Hooks for save, load, and delete operations with attribute value
Source
Options for select and multiselect attributes
Frontend
Rendering attribute values on the frontend
HOME
Magento U

Notes:

A reminder of the function of attribute backend, source, and frontend models:

- `Backend:` Provides hooks before and after save, load, and delete operations with an attribute value.
- `Source:` Provides option values and labels for select and multi-select attributes.
- `Frontend:` Prepares an attribute value for rendering on the storefront.

7.14 Attributes Management | Backend Models


Attributes Management | Backend Models

BACKEND MODEL

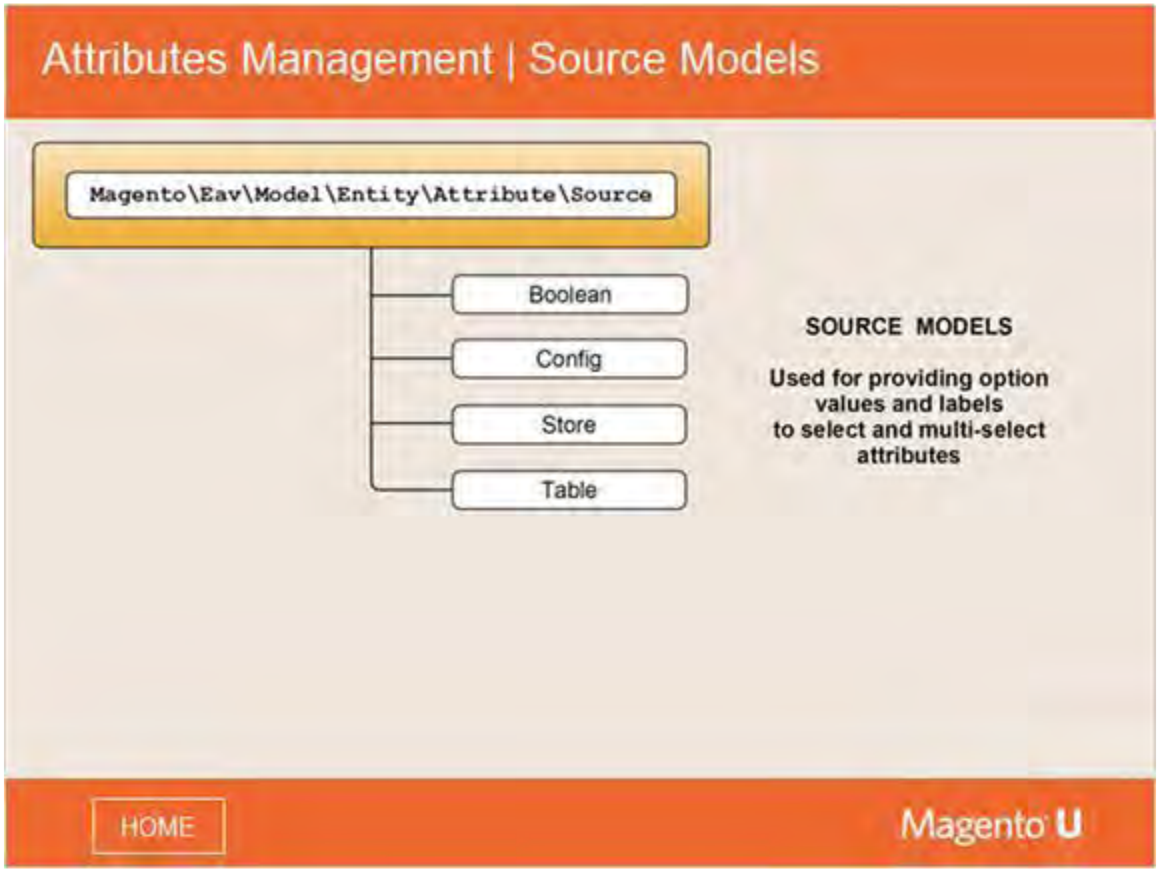
Used for transforming data before and after save, load, and delete operations

Notes:

Backend models can be an alternative to an observer; for example, when you have to do something that depends on an attribute value when an entity is saved.

Magento comes with a number of backend models that can be used for custom attributes as well. The models can be found in the `app/code/Magento/Eav/Model/Entity/Attribute/Backend` directory.

7.15 Attributes Management | Source Models

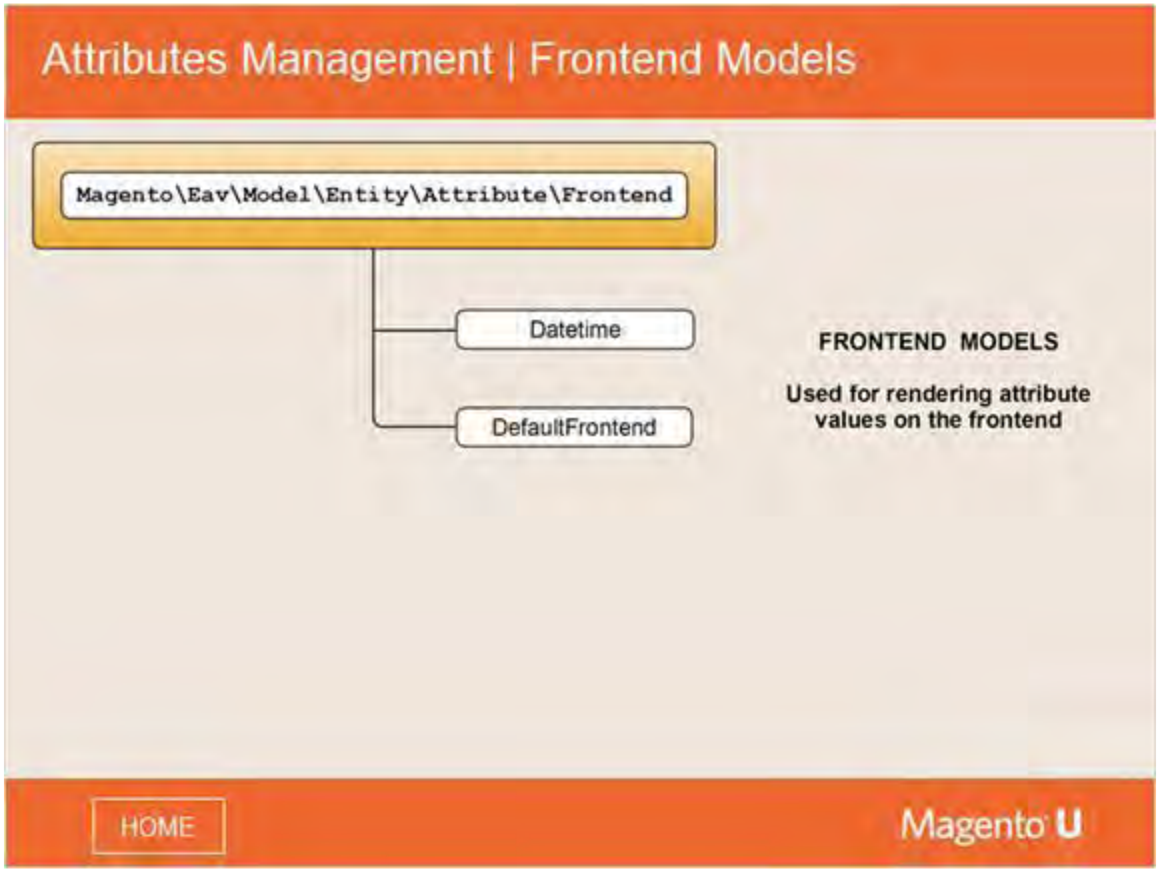

Attributes Management | Source Models
Magento\Eav\Model\Entity\Attribute\Source
Boolean
Config
Store
Table
SOURCE MODELS
Used for providing option values and labels to select and multi-select attributes
HOME
Magento U

Notes:

Magento comes with a number of attribute source models that can be used for custom attributes, too.

The models can be found in the `app/code/Magento/Eav/Model/Entity/Attribute/Source` directory.

7.16 Attributes Management | Frontend Models



Attributes Management | Frontend Models

Magento\Eav\Model\Entity\Attribute\Frontend
Datetime
DefaultFrontend
HOME
Magento U

FRONTEND MODELS

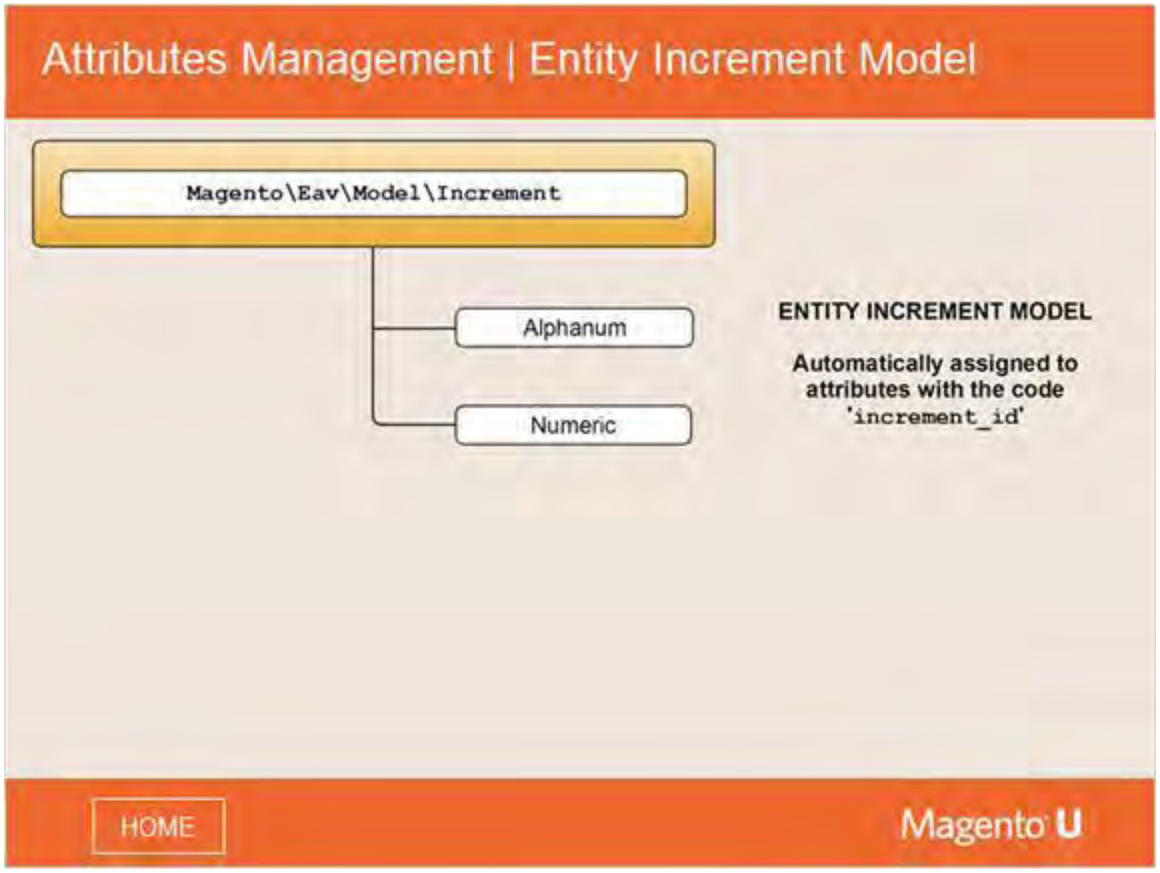
Used for rendering attribute values on the frontend

Notes:

The number of frontend models Magento provides out of the box is very limited.

The models can be found in the `app/code/Magento/Eav/Model/Entity/Attribute/Frontend` directory.

7.17 Attributes Management | Entity Increment Model


Attributes Management | Entity Increment Model
Magento\Eav\Model\Increment
Alphanum
Numeric
ENTITY INCREMENT MODEL
Automatically assigned to attributes with the code 'increment_id'
HOME
Magento U

Notes:

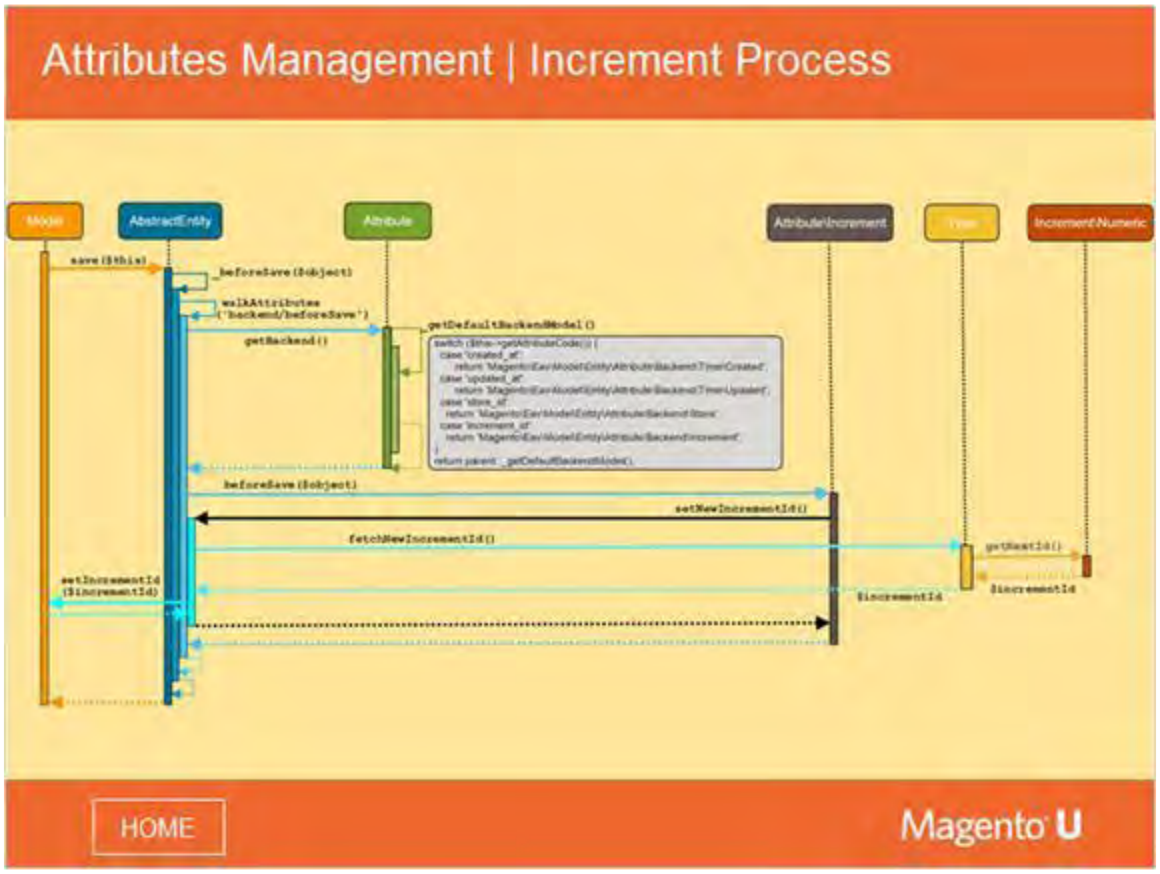
If an EAV entity type has an attribute with the code `increment_id`, and no backend model is set for that attribute, the `Magento\Eav\Model\Entity\Attribute\Backend\Increment` model is assigned automatically.

All the increment attribute backend model does is call `$this->getAttribute()->getEntity()->setNewIncrementId($object)` in the `beforeSave()` hook method.

Note: See `\Magento\Eav\Model\Entity\Attribute::_getDefaultBackendModel()` for more automatic backend model assignment rules.

By default, Magento 2 uses only the numeric increment model for customers, orders, shipments, invoices and credit memos. The increment models shipped with Magento 2 out of the box can be found in the directory `app/code/Magento/Eav/Model/Entity/Attribute/Increment/`.

7.18 Attributes Management | Increment Process



Attributes Management | Increment Process

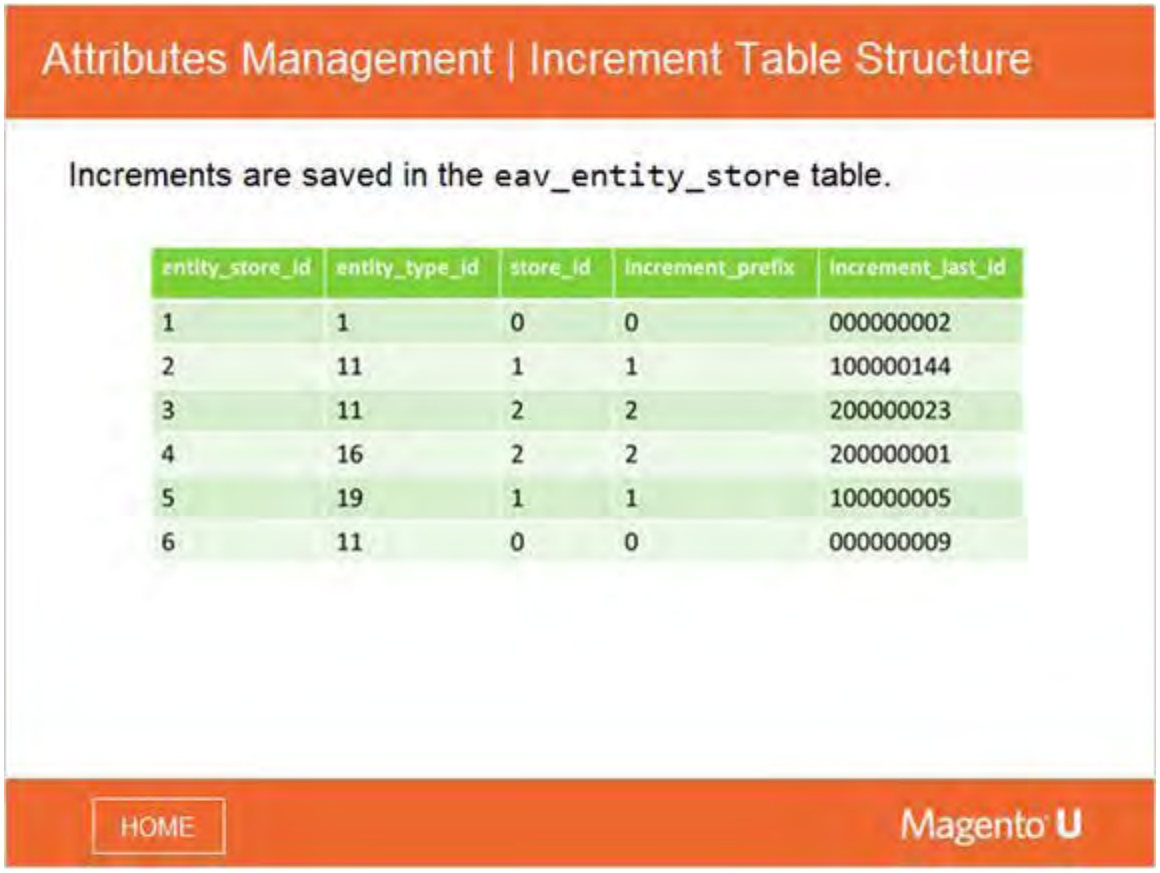
```
switch ($this->getAttributeCode()) {
  case 'created_at':
      return 'Magento\Eav\Model\Entity\Attribute\Backend\Time\Created';
  case 'updated_at':
      return 'Magento\Eav\Model\Entity\Attribute\Backend\Time\Updated';
  case 'store_id':
    return 'Magento\Eav\Model\Entity\Attribute\Backend\Store';
  case 'increment_id':
    return 'Magento\Eav\Model\Entity\Attribute\Backend\Increment';
}
return parent::_getDefaultBackendModel();
```

Notes:

After the entity type model's `fetchNewIncrementId()` method is called, it delegates to the increment model to actually generate the new increment ID. The increment model to use is set on the entity type.

Currently in Magento 2, all entities that use increments use the Numeric increment model.

7.19 Attributes Management | Increment Table Structure



Attributes Management | Increment Table Structure

Increments are saved in the `eav_entity_store` table.

entity_store_id	entity_type_id	store_id	increment_prefix	increment_last_id
1	1	0	0	000000002
2	11	1	1	100000144
3	11	2	2	200000023
4	16	2	2	200000001
5	19	1	1	100000005
6	11	0	0	000000009

HOME

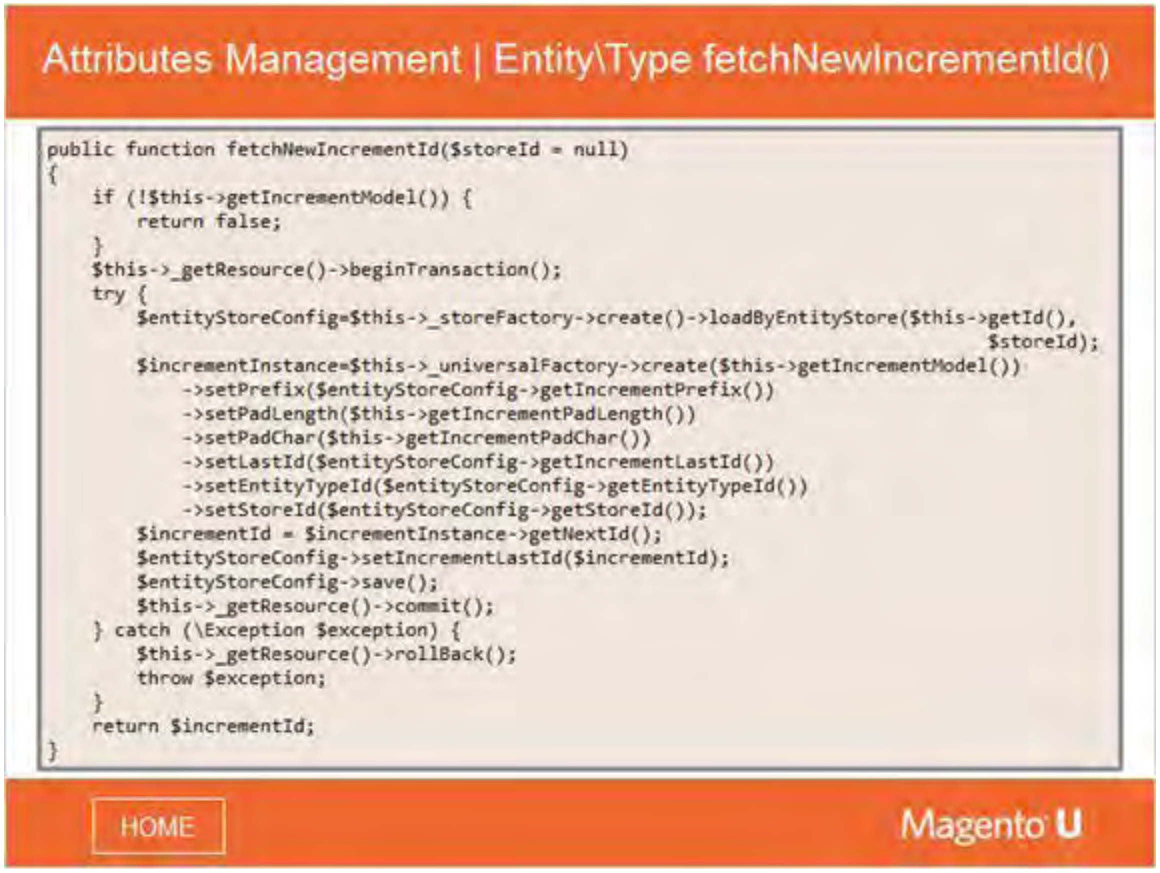
Magento U

Notes:

Increments are saved in the `eav_entity_store` table, one record for each entity type and store.

The values from the `eav_entity_store` table, together with the increment configuration stored in the `eav_entity_type` table, are set on the increment models so they may be used to generate a new increment ID.

7.20 Attributes Management: Entity\Type fetchNewIncrementId()



Attributes Management | Entity\Type fetchNewIncrementId()

```
public function fetchNewIncrementId($storeId = null)
{
    if (!$this->getIncrementModel()) {
        return false;
    }
    $this->_getResource()->beginTransaction();
    try {
        $entityStoreConfig=$this->_storeFactory->create()->loadByEntityStore($this->getId(),
                                                                              $storeId);
        $incrementInstance=$this->_universalFactory->create($this->getIncrementModel())
            ->setPrefix($entityStoreConfig->getIncrementPrefix())
            ->setPadLength($this->getIncrementPadLength())
            ->setPadChar($this->getIncrementPadChar())
            ->setLastId($entityStoreConfig->getIncrementLastId())
            ->setEntityTypeId($entityStoreConfig->getEntityTypeId())
            ->setStoreId($entityStoreConfig->getStoreId());
        $incrementId = $incrementInstance->getNextId();
        $entityStoreConfig->setIncrementLastId($incrementId);
        $entityStoreConfig->save();
        $this->_getResource()->commit();
    } catch (\Exception $exception) {
        $this->_getResource()->rollBack();
        throw $exception;
    }
    return $incrementId;
}
```

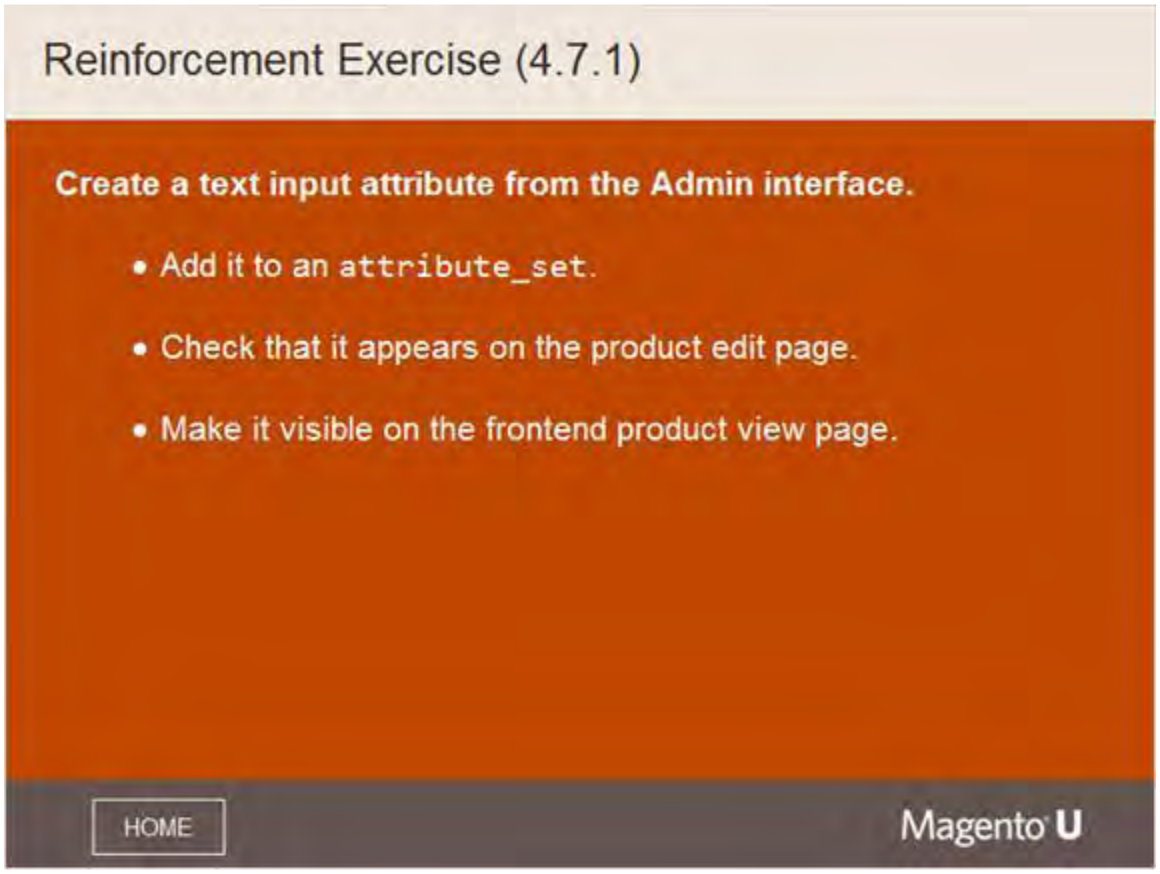
Notes:

The `fetchNewIncrementId()` method is implemented on the entity type model `Magento\\Eav\\Model\\Entity\\Type`.

It instantiates the increment model associated with the entity type, and then sets the increment-related values from the `eav_entity_type` table and the associated values from the `eav_entity_store` table on the model.

Then the new increment ID is generated by calling `getNextId()` on the increment instance.

7.21 Reinforcement Exercise (4.7.1)



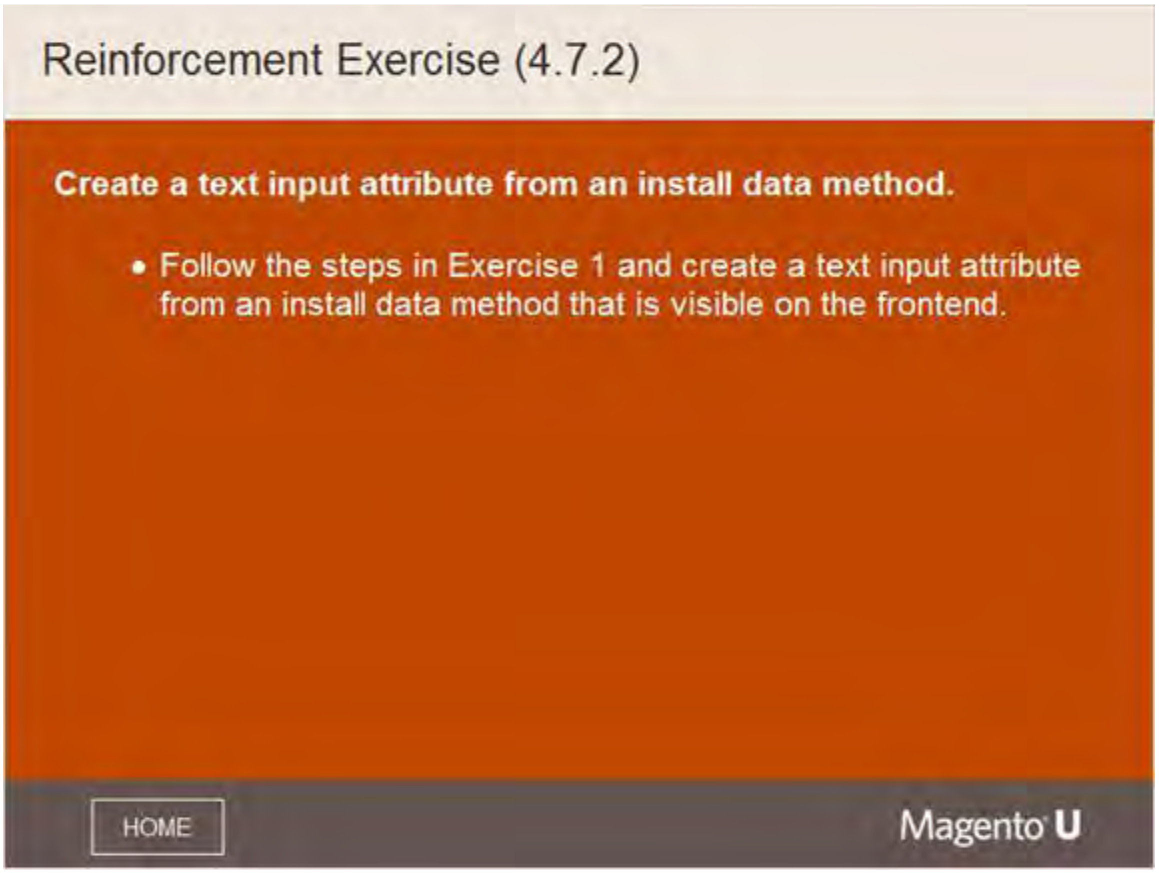
Reinforcement Exercise (4.7.1)

Create a text input attribute from the Admin interface.

- Add it to an `attribute_set`.
- Check that it appears on the product edit page.
- Make it visible on the frontend product view page.

HOME
Magento U

7.22 Reinforcement Exercise (4.7.2)



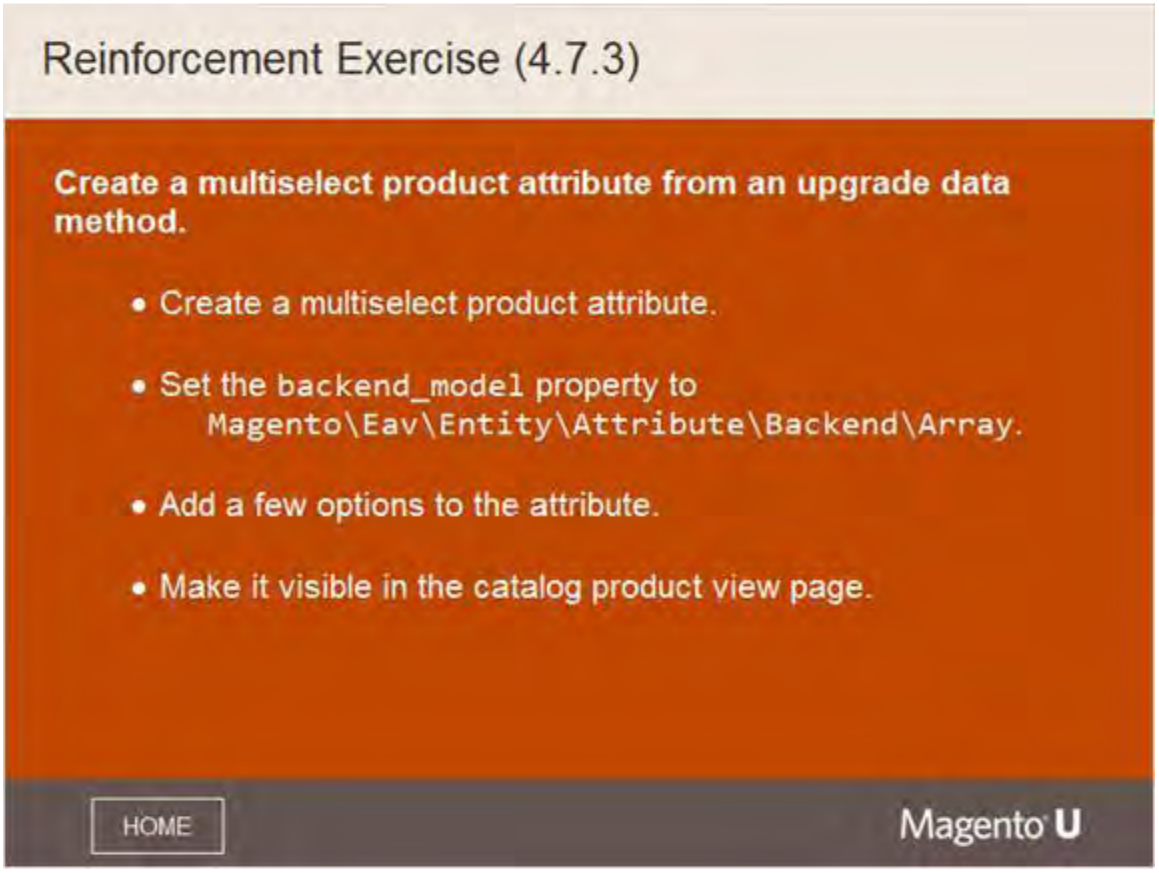
Reinforcement Exercise (4.7.2)

Create a text input attribute from an install data method.

- Follow the steps in Exercise 1 and create a text input attribute from an install data method that is visible on the frontend.

HOME
Magento U

7.23 Reinforcement Exercise (4.7.3)



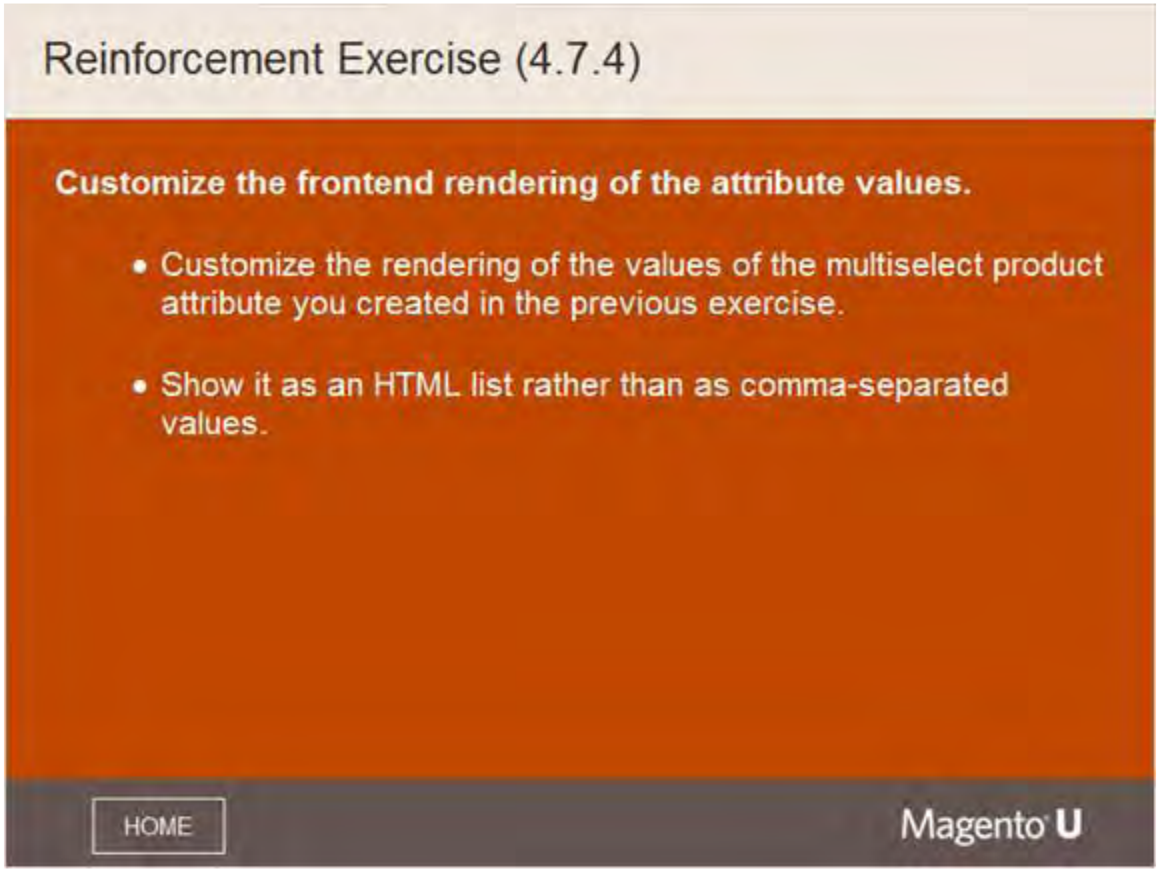
Reinforcement Exercise (4.7.3)

Create a multiselect product attribute from an upgrade data method.

- Create a multiselect product attribute.
- Set the `backend_model` property to `Magento\Eav\Entity\Attribute\Backend\Array`.
- Add a few options to the attribute.
- Make it visible in the catalog product view page.

HOME
Magento U

7.24 Reinforcement Exercise (4.7.4)



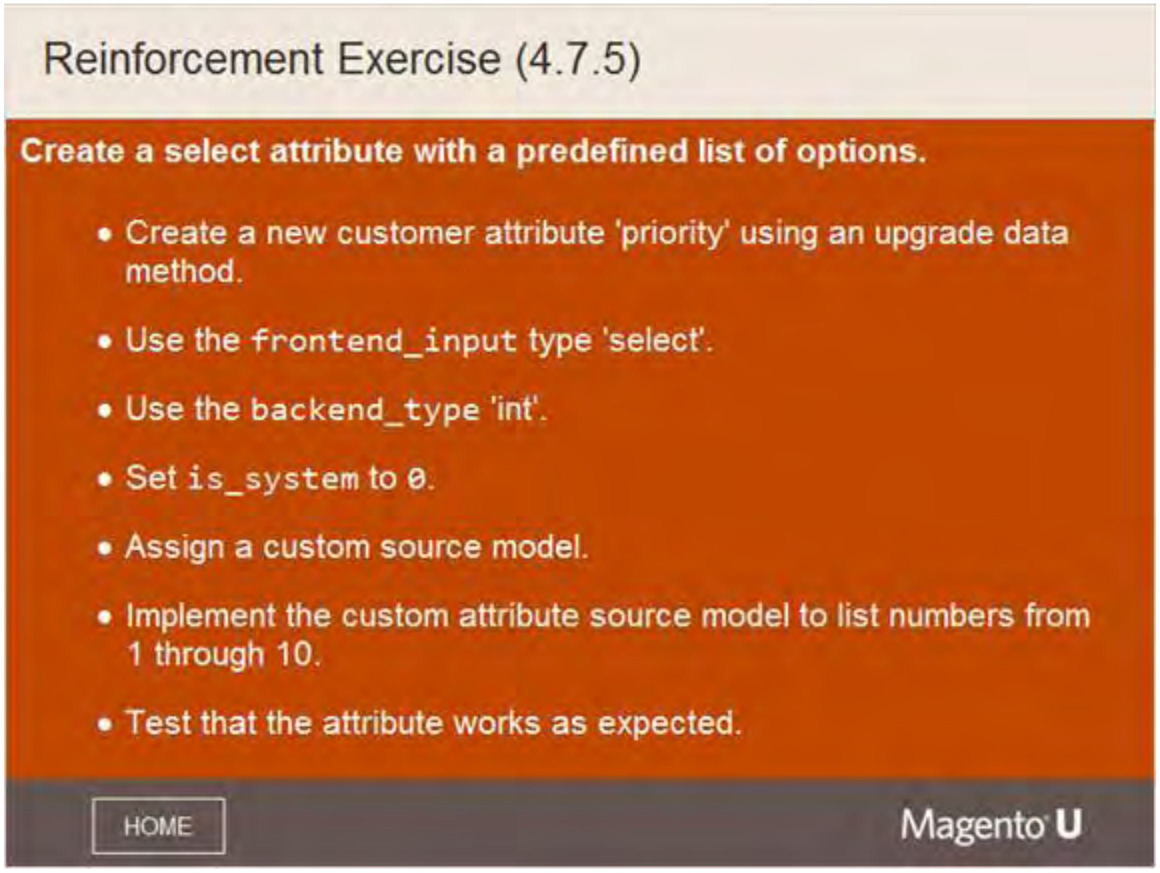
Reinforcement Exercise (4.7.4)

Customize the frontend rendering of the attribute values.

- Customize the rendering of the values of the multiselect product attribute you created in the previous exercise.
- Show it as an HTML list rather than as comma-separated values.

HOME
Magento U

7.25 Reinforcement Exercise (4.7.5)



Reinforcement Exercise (4.7.5)

Create a select attribute with a predefined list of options.

- Create a new customer attribute 'priority' using an upgrade data method.
- Use the `frontend_input` type 'select'.
- Use the `backend_type` 'int'.
- Set `is_system` to `0`.
- Assign a custom source model.
- Implement the custom attribute source model to list numbers from 1 through 10.
- Test that the attribute works as expected.

7.26 End of Unit Four

End of Unit Four

Congratulations!

You have completed Unit Four

of the Magento 2 Fundamentals course.