**Working with Database**

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

**ORM -** Object-relational mapping

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

ORM : **Elements**

* **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.

ORM : **Element Relationships**

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

**Model :** Models encapsulate storage independent business logic

**Resource Model**: Resource Models encapsulate storage related logic. All storage layer related actions are the responsibility of the resource model.

**Resource Collection:** Resource Collections represent a list of models of a specific type

**Model**

* Create model using extends \Magento\Framework\Model\AbstractModel class
* Create \_construct() method to call AbstractModel class \_init() method pass resourceModel object

EX. protected function \_construct()

{

$this->\_init(‘Vendorname\Modulename\Model\ResourceModel\Modelname’);

}

**AbstractModel \_init() method**

protected function \_init($resourceModel){

$this->\_setResourceModel($resourceModel);

$this->\_idFieldName= $this->\_getResource()->getIdFieldName();

}

Domain model implementation:Behavior + data object Model

Deprecated some model from Model class (ex save(), load() and delete()).

**ResourceModel**

* Extends \Magento\Framework\Model\ResourceModel\Db\AbstractDb class to create ResourceModel
* Create \_construct() method to call \_init() method of AbstractDb class.
* Pass two parameter (1) entity\_name (2) primary\_key field

Ex. public function \_construct()

{

$this->\_init('bytes\_yotporeview', 'entity\_id');

}

**AbstarctDb \_init() method**

protected function \_init($mainTable, $idFieldName)

{

$this->\_setMainTable($mainTable, $idFieldName);

}

**Resource Collection**

* Extends to this class you create a Resource Collection Model \Magento\Framework\Model\ResourceModel\Db\Collection\AbstractCollection
* Create a \_construct() method to init a collection model,It’s call \_init() of AbtractCollection class
* There have two parameter (1) Model and (2) ResourceModel

Ex. public function \_construct()

{

$this->\_init('Bytes\Yotporeview\Model\Reviews', 'Bytes\Yotporeview\Model\ResourceModel\Reviews');

}

**AbstractCollection \_init() method**

protected function \_init($model, $resourceModel)

{

$this->setModel($model);

$this->setResourceModel($resourceModel);

return $this;

}

* Container for storing object collection
* Prevents unnecessary loading of data
* Storing all object during a session
* Provide an interface for filtering & storing entity of a specific kind

**Model Type Interfaces**

Declare setters and getters for API

**Database Adapters**

Provide read/write connection to the database

**Operations Operators**

|  |  |
| --- | --- |
| Operator | sign |
| eq | = |
| neq | != |
| like | like |
| nlike | not like |
| is | is |
| in | in |
| nin | not in |
| notnull | Is not null |
| null | is null |
| gt | > |
| lt | < |
| gteq | >= |
| lteq | <= |
| moreq | >= |
| finset | find\_in\_set |
| from,to | >=,<= |

**CRUD Workflow | Reading**

* Magento model to call load() with one parameter it’s return the maching with peramete to primary key value.
* Magento Model to call load() with two perameter it’s return check with value to second perameter database field with entity.

**\_beforeLoad()**

Event: model\_load\_before, <event\_prefix>\_load\_before

**\_afterLoad()**

Event: model\_load\_after, <event\_prefix>\_load\_after

**afterLoad()**

The afterLoad call complate the collection object

Event : core\_collection\_abstract\_load\_before, core\_collection\_abstract\_load\_after,

<event\_prefix>\_load\_before, <event\_prefix>\_load\_after

**getStoreData()**

Retriving a current store data.

**hasDataChanges()**

Return true or false, if model data is change then it’s retrun true otherwise it’s getting false.

**CRUD Workflow | Creating & Updating**

Composite primary key are not supported by the magento.

**isDeleted()**

It’s return the current \_isDeleted flag value

**hasDataChanges()**

Return true or false, if model data is change then it’s retrun true otherwise it’s getting false.

**validateBeforeSave()**

contains all the rules to validate the current model.

**beforeSave()**

Event: model\_save\_before, <event\_prefix>\_save\_before

**isSaveAllowed()**

Return \_dataSaveAllowed flag in beforeSave() method rules

**afterSave()**

Event : model\_save\_after, <event\_prefix>\_save\_after

**afterCommitCallBack()**

This method to use commit all transection.

Event: model\_save\_commit\_after, <event\_prefix>\_save\_commit\_after

**CRUD Workflow| Deleting**

* 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

**beforeDelete()**

Event : model\_delete\_before, <event\_prefix>\_delete\_before

**afterDelete()**

Event : model\_delete\_after, <event\_prefix>\_delete\_after

**afterDeleteCommit()**

Event : model\_delete\_commit\_after, <event\_prefix>\_delete\_commit\_after

**Setup Scripts & Setup Resources**

1. **Creating new entities**

* **Install Script**: Run only once per module.
* **Upgrade Script**: Run after an install and upon subsequent upgrade.
* The module version is used to determine which setup scripts to apply.
* Every module.xml is required to have a setup\_version declaration.
* Install a module

Run bin/magento module:enable --clear-static-content Module\_Name.

Run bin/magento setup:upgrade.

Check the module was successfully installed, run bin/magento module:status

* Module database configuration save to “setup\_module” entity.

**InstallSchema**

Using 3 interface to create a InstallSchema class to create new entity.

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();

$setup->endSetup();

}

}

**UpgradeSchema**

Using 3 interface to create UpgradeSchema class to create & update entity structure

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();

$setup->endSetup();

}

}

**InstallData**

Using InsallData to change to database data. And add new data or delele data from database.

use Magento\Framework\Setup\InstallDataInterface;

use Magento\Framework\Setup\ModuleContentInterface;

use Magento\Framework\Setup\ModuleDataSetupInterface;

class InstallData implements InstallDataInterface

{

public function install(ModuleDataSetupInterface $setup, ModuleContextInterface $context)

{

$setup->startSetup();

$setup->endSetup();

}

}

**UpgradeData**

* Using UpgradeData class to add or upgrade database data.

use Magento\Framework\Setup\UpgradeDataInterface;

use Magento\Framework\Setup\ModuleContentInterface;

use Magento\Framework\Setup\ModuleDataSetupInterface;

class UpgradeData implements UpgradeDataInterface

{

public function install(ModuleDataSetupInterface $setup, ModuleContextInterface $context)

{

$setup->startSetup();

$setup->endSetup();

}

}

* version\_compare($context->getVersion(), '1.0.0', '<') using to compare module version.
* Run bin/mageto setup:db:status return module version status.
* Run bin/mageto setup:upgrade to install module & run install & upgrade scripts.
* There are two ways to perform required changes to a database:

1. **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.
2. **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.

**Entity Attribute Value [EAV]**

1. **EAV storage concept**

**Data Storage Issue**

* **Multiple Scope values:** do not provide to store value in deference store and website.
* **Adding/removing new Fields:** if you need one more column to entity it’s must be alter.
* **Cleanging the type of fields:** if you change the field data type you must have alter table.

**Benefits Using EAV**

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

**EAV Aspects**

1. Meta Information

Entity type

Attributes per entity type

Attribute set and group

1. Content

Entity records

Attribute values

**Classic EAV structure**

entity\_type

entiry\_type\_id

entity\_type\_code

Ex . (1,product),(2,customer)

attribute

attribute\_id

entity\_type\_id

attribute\_code

Ex. (1,1,”name”),(2,2,”fname”)..etc

entity

entity\_id

entity\_type\_id

Ex. (1,1),(2,1),(3,1),(4,2)....etc

attribute\_value

value\_id

entity\_id

attribute\_id

value

Ex. (1,1,1,”Sample Product”),(2,4,2,”Rajesh”)

**EAV Structure**

eav\_entity\_type

entity\_type\_id

entity\_type\_code

Ex . (1,product),(2,customer)

eav\_attribute

attribute\_id

entity\_type\_id

atttribute\_code

backend\_type

Ex (1,1,’name’,’varchar’),(2,2,’fname’,’vharchar’),(3,1,’price’,decimal’)...etc

product\_entity

entity\_id

entity\_type\_id

sku

Ex (1,1,’test1’)

product\_entity\_varchar

value\_id

entity\_type\_id

entity\_id

attribute\_id

value

Ex (1,1,1,1,”Sample Product1”), (2,1,1,1,”Sample Product2”)...etc

product\_entity\_decimal

value\_id

entity\_type\_id

entity\_id

attribute\_id

value

Ex (1,1,1,3,10.45),(2,1,2,3,20.45)....etc

**EAV Hierarchy**

* Each entity type has one or more attribute & attribute set
* Each attribute set has one or more attribute group.
* Every attribute is associated to exactly one entity type or more attribute set via attribute set

**EAV Meta Tables**

eav\_entity\_type

entity\_type\_id

entity\_type\_code

default\_attribute\_set\_id

entity\_model

entity\_table

eav\_attribute

attribute\_id

attribute\_code

entity\_type\_id

backend\_type

backend\_model

frontend\_model

source\_model

additional\_attribute\_table

attribute\_id

eav\_entity\_attribute

entity\_attribute\_id

entity\_type\_id

attribute\_set\_id

attribute\_group\_id

attribute\_id

eav\_entity\_group

attribute\_group\_id

attribute\_set\_id

eav\_attribute\_set

attribute\_set\_id

entity\_type\_id

* 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
* The eav\_attribute\_set table contains the attribute set declarations. The eav\_attribute\_group contains the attribute groups

1. **Load & Save EAV**

**EAV Model & Resource**

* EAV Model extends Magento\Framework\Model\AbstractModel class, just like flat table
* EAV Resource Model extends Magento\Eav\Model\Entity\AbstractEntity rather than Magento\Framework\Model\Resource\Db\AbstractDb

**EAV Resource Methods**

* **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.

**EAV Collection Method**

* **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.

All attribute values are saved in the table according to their backend\_type property.

**Attribute Structure**

* **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 for the adminhtml form; is also evaluated during product import.
* **default\_value**: Optional default value for the attribute; displayed by the frontend model if the attribute has no value on an entity.

**Get EAV Attribute & EAV Entity Type**

Magento\Eav\Model\Config

* **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’)

**Attribute Type**

* Varchar
* Text
* Int
* Decimal
* Datetime

**EAV Setup class**

**Magento\Eav\Setup\EavSetup**

* This class used to create attribute and attribute group in setup script(InstallData, UpgradeData)

**addAttribute & updateAttribute**

* **addAttribute(ENTITY\_TYPE, ENTITY\_CODE, ATTRIBUTE\_DATA)**: this method to use create new entity type attribute in magento
* **updateAttribute(ENTITY\_TYPE, ENTITY\_CODE, ATTRIBUTE\_DATA)**: this method to use update entity\_type attribute in magento

**Attribute Properties**

* backend\_model
* backend\_type
* backend\_table
* frontend\_model
* frontend\_input
* frontend\_lable
* frontend\_class
* source\_model
* is\_required
* is\_user\_defined
* default\_value
* is\_unique
* note
* is\_global

**Attribute Models**

* **Backend**: Provides hooks before and after save, load, and delete operations with an attribute value.

ArrayBackend

Datetime

DefaultBackend

Increment

Serialized

Store

Time/Created

Time/Updated

* **Source**: Provides option values and labels for select and multi-select attributes.

Boolean

Config

Store

Table

* **Frontend**: Prepares an attribute value for rendering on the storefront.

DateTime

DefaultFrontend