DUMPSARENA

# Magento 2 Certified Associate Developer Exam

## Magento Magento-2-Certified-Associate-Developer

**Version Demo**

**Total Demo Questions: 13**

**Total Premium Questions: 138**

**Buy Premium PDF**

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

# Topic Break Down

| Topic | No. of Questions |
|---|---|
| Topic 1, Magento Architecture | 54 |
| Topic 2, EAV/Database | 16 |
| Topic 3, Layout | 18 |
| Topic 4, Checkout | 3 |
| Topic 5, Catalog | 12 |
| Topic 6, Sales | 9 |
| Topic 7, Customers | 6 |
| Topic 8, Admin | 20 |
| Total | 138 |

QUESTION NO: 1

How can you access the select query of a collection?

**A.** You can only access the select query after the collection has been loaded by calling the public method query()

**B.** It is stored in a protected variable $query and can only be accessed from the inside of a collection class

**C.** You can get it by using public method getSelect() which returns an instance of Magento\Framework\DB\Select

**D.** The select query is not available in the collection class, it will be generated by the MySQL adapter right before executing a query

**ANSWER: C**

**Explanation:**

You can get it by using public method getSelect() which returns an instance of Magento\Framework\DB\Select. Magento resource collections build their database query as a Select object, and the public `getSelect()` method exposes that object so developers can inspect or extend the query before loading the collection. For example, a developer may call `$collection->getSelect()` to add joins, filters, ordering, grouping, or expressions when implementing collection-level data retrieval logic. The returned object is Magento's database Select abstraction, `Magento\Framework\DB\Select`, which extends the underlying Zend/Laminas database select functionality used to compose SQL statements. A collection does not need to be loaded first: the select object is initialized and may be modified during collection preparation, while loading is the later operation that executes the assembled query and populates collection items. This API is commonly used in resource-model collection customizations and is the supported direct means of obtaining the collection's query builder. See the Magento API reference for AbstractDb::getSelect() and the Magento Framework DB Select class.

**QUESTION NO: 2**

Which method of a Magento resource model will remove a record from the database?

**A.** remove

**B.** erase

**C.** clean

**D.** delete

**ANSWER: D**

**Explanation:**

The `delete` method removes an entity record through its Magento resource model. A resource model is responsible for persistence operations, including loading, saving, and deleting data from the database. In typical Magento service or model code, an entity instance is passed to the resource model's `delete()` method, such as `$this->resourceModel->delete($entity)`. The resource model resolves the entity's identifier and executes the database delete operation against the table mapped to that entity.

This approach keeps database access within Magento's persistence layer rather than requiring application code to construct direct SQL deletion statements. It also allows the resource model lifecycle and configured entity metadata to govern the operation consistently. Magento's framework API defines `delete(AbstractModel $object)` on resource models, and core resource-model implementations perform the deletion using the entity's primary key. See the Magento framework source for the resource-model delete implementation in Magento Framework AbstractDb and the resource-model guidance in Adobe Commerce Models documentation.

**QUESTION NO: 3**

A merchant requires the ability to configure contact information for their brick and mortar stores as a CSV file upload. The module already exists and contains an etc/adminhtml/system.xml file where the new field can be added.

How do you specify the class that will process the uploaded file?

**A.** \Magento\Config\Model\Config\Upload\File

**B.** \Magento\Config\Model\Config\Frontend\File

**C.** \Magento\Config\Model\Config\Backend\File

**D.** \Magento\Config\Model\Config\Source\File

**ANSWER: C**

**Explanation:**

`<backend_model>\Magento\Config\Model\Config\Backend\File</backend_model>` is correct because a system configuration field's `backend_model` declares the model responsible for processing the field value when the configuration form is saved. For a file-upload field, the backend model receives the uploaded file data, performs the upload handling, and saves the resulting value into configuration storage. Magento's core `Magento\Config\Model\Config\Backend\File` implementation is specifically intended for this purpose and extends the configuration value backend workflow with file-uploader behavior. The field should also be configured as a file input in `system.xml`; where appropriate, upload directory and allowed file-extension settings can be supplied so the upload is stored in the desired location and limited to CSV files. This cleanly separates admin form rendering from persistence-time processing: the configuration field defines the input, while the backend model owns validation and upload processing during save. Adobe's system configuration documentation describes `backend_model` as the model used to process a configuration value, and the Magento source shows the file backend model's upload implementation. See Adobe Commerce system configuration documentation and Magento Config Backend File source.

**QUESTION NO: 4**

You are developing a module MyCompany_StoreInfo to display information about brick and mortar stores on a frontend page. The displayed information varies based on the country of a given store.

What two elements automatically render their children? (Choose two.)

A. `<block class="\Magento\Framework\View\Element\AbstractBlock" name="shop.info.details"/>`
B. `<block class="\Magento\Framework\View\Element\Template" name="shop.info.details"/>`
C. `<container name="shop.info.details"/>`
D. `<block class="\Magento\Framework\View\Element\Text\ListText" name="shop.info.details"/>`

**A.** Option A

**B.** Option B

**C.** Option C

**D.** Option D

**ANSWER: B C**

**Explanation:**

The two selected elements are correct because Magento's layout system treats containers as structural output elements. A container does not provide business content of its own; instead, it is responsible for collecting its child layout elements and rendering them in the resolved layout order. This makes containers appropriate for grouping the country-specific store-information blocks while allowing Magento to output the group without requiring a template to explicitly call each child.

A container reference works with an existing container in the merged layout. When child elements are declared inside that reference, Magento adds them to the referenced container's child structure. During page rendering, the referenced container

consequently outputs those children through its normal container-rendering behavior. This lets a module place store-information components in an existing frontend region without replacing that region's template or manually rendering every child.

Magento's layout XML documentation describes containers as structural elements that render their child elements and documents container references as the mechanism for modifying an existing container. See Adobe Commerce layout XML instructions and Adobe Commerce layout overview.

## QUESTION NO: 5

Which two tasks are supported by Magento CLI? (Choose two.)

**A.** Customer password reset

**B.** Clearing cache

**C.** Codebase deployment from developer machine to staging server

**D.** Administrator account creation

## ANSWER: B D

**Explanation:**

Magento CLI directly supports both clearing cache and creating an Administrator account. Cache operations are provided through the `bin/magento cache:clean` and `bin/magento cache:flush` commands. These commands let a developer remove generated cached data after configuration, layout, code, or content changes, ensuring that Magento rebuilds and serves current data. The command can act on all cache types or on specified cache types, making it a standard maintenance operation available from the command line.

Magento CLI also includes the `bin/magento admin:user:create` command. This command creates a new Magento Admin user by accepting required account details such as the username, first name, last name, email address, and password as command arguments. It is particularly useful during installation, environment provisioning, recovery of administrative access, and automated deployment workflows where an administrator account must be created without using the Admin interface. Adobe's CLI documentation identifies cache management as a CLI command group and documents administrator-user creation as an explicit CLI operation. See Manage the cache and Create an administrator user.

## QUESTION NO: 6

During a code review of a module MyCompany_PaymentCurrencies you see a configuration field declared in the file etc/adminhtml/system.xml:

```
<field id="currency" type="select" showInDefault="1" showInWebsite="1" showInStore="0">
    ...
</field>
```

What is the consequence of the attribute showInStore being set to 0?

**A.** The field value will not be accessible on the store front by calling ScopeConfigInterface::getValue() with a $scopeType argument of 'store'.

**B.** The input field will not be visible if a store view scope is selected in the system configuration

**C.** The input field will only be visible if a website's default store scope is selected in the system configuration

**D.** The input field will be disabled if a store view scope is selected in the system configuration

## ANSWER: B

**Explanation:**

The input field will not be visible if a store view scope is selected in the system configuration. In an adminhtml `system.xml` declaration, `showInStore` controls whether a configuration element is available while the Configuration page is operating in the Store View scope. A value of `0` means that the field is not shown for that scope, so an administrator selecting an individual store view in the scope switcher will not see this input. This setting concerns the Admin configuration UI and the scope at which administrators can manage the value; it does not itself define how configuration values are read in PHP. Scope visibility attributes are used with the hierarchy of default, website, and store-view configuration scopes to ensure a setting is exposed only where it is intended to be maintained. Adobe's system-configuration tutorial documents the `showInStore` attribute as the control for Store View visibility in `system.xml`. See Adobe Commerce: Create a system configuration and Adobe Commerce: Change configuration scope.

## QUESTION NO: 7

You are adding a new entry to the backend menu that appears after

Marketing > SEO & Search > Site Map

You see the existing site map menu item is declared by the node:

```
<add id="Magento_Sitemap::catalog_sitemap"
    title="Site Map"
    translate="title"
    sortOrder="60"
    parent="Magento_Backend::marketing_seo"
    action="adminhtml/sitemap/"
    resource="Magento_Sitemap::sitemap"/>
```

What two actions do you take to configure the new menu entry location? (Choose two.)

**A.** Specify item XML in the file etc/adminhtml/menu/marketing/seo/menu.xml

**B.** Specify parent="Magento_Sitemap::catalog_sitemap"

**C.** Specify parent="Magento_Backend::marketing_seo"

**D.** Specify sortOrder="100"

## ANSWER: C D

**Explanation:**

To place a new entry in the same *SEO & Search* section as *Site Map*, the menu item must declare `parent="Magento_Backend::marketing_seo"`. A Magento admin menu's `parent` attribute identifies the container node under which the item is rendered. Using the same parent as the existing Site Map item makes the new item a sibling in the intended Marketing submenu rather than a child of the Site Map entry itself.

The relative position among sibling menu entries is controlled through the `sortOrder` attribute. Specifying `sortOrder="100"` assigns an ordering value that places the entry after lower-numbered items, including the shown Site Map item when its declared order is lower. Magento assembles menu configuration from module `etc/adminhtml/menu.xml` files and sorts entries sharing a parent according to this value. Thus, the parent value establishes the menu path, while the sort order establishes that the new entry follows Site Map within that path. See Adobe's configuration guidance and the Magento framework's backend menu configuration for the menu XML structure and parent-node pattern.

## QUESTION NO: 8

Your module adds an Admin controller that must be accessible only to users granted permission for the custom resource `MyCompany_MyModule::manage_records`.

Which two actions are required to enforce this permission? (Choose two.)

**A.** Declare `MyCompany_MyModule::manage_records` in `etc/acl.xml`

**B.** Set the controller `ADMIN_RESOURCE` constant to `MyCompany_MyModule::manage_records`

**C.** Add the resource ID as a constructor argument in `etc/adminhtml/di.xml`

**D.** Add the resource ID to the controller route in `etc/adminhtml/routes.xml`

**ANSWER: A B**

**Explanation:**

The ACL resource must be declared in `etc/acl.xml`. This adds the resource to Magento's authorization tree so it can be selected for an Admin role. A resource normally belongs beneath an existing parent resource and has an ID such as `MyCompany_MyModule::manage_records`.

The Admin controller must identify the same resource through its `ADMIN_RESOURCE` constant. Backend controller authorization checks this constant before the action is executed. If the current Admin user does not have the resource through an assigned role, Magento denies access to the controller action.

A menu item can reference the resource to control its visibility, but a menu declaration alone does not secure a controller URL. See Adobe Commerce ACL documentation and Adobe Commerce backend controller documentation.

**QUESTION NO: 9**

The constructor function for \Magento\Catalog\Model\Category contains this excerpt:




With the automatic dependency injection that Magento provides, how is the StoreManagerInterface resolved?

**A.** If no $storeManager is provided, Magento's code generator creates a shell concrete class based on \Magento\Store\Model\StoreManagerInterface

**B.** Magento finds all classes that implement \Magento\Store\Model\StoreManagerInterface (ordered alphabetically) and injects the first class.

**C.** Magento resolves \Magento\Store\Model\StoreManagerInterface using the preference configured in the merged di.xml configuration, then constructs and injects that class.

**D.** Magento throws an exception because you cannot instantiate an interface

**ANSWER: C**

**Explanation:**

Magento looks to the merged `di.xml` configuration for a preference declared for `\Magento\Store\Model\StoreManagerInterface`, then constructs and injects the configured implementation. Dependency injection cannot directly instantiate an interface, so Magento's object manager uses the interface-to-class preference configured by the `Magento_Store` module. That module maps `StoreManagerInterface` to `Magento\Store\Model\StoreManager`. As a result, when Magento needs to supply this constructor dependency, it resolves the requested interface through the DI configuration and creates the associated concrete store manager, including its own required dependencies.

Magento compiles and merges DI configuration from enabled modules and application configuration by area; it does not select an implementation based on alphabetical class discovery. Preferences are the standard Magento mechanism for specifying which implementation should be used when a constructor requests an interface. The optional constructor argument in this legacy model also allows the category model to obtain the same interface through the object manager when no instance was supplied, and that lookup still follows the configured preference. See Adobe's Dependency Injection documentation and the Magento Store module's DI configuration.

## QUESTION NO: 10

Your module must add a custom field named `warehouse_code` to the GraphQL response for a product.

Which two actions are required to implement the custom GraphQL field? (Choose two.)

**A.** Declare the field on the appropriate type in `etc/schema.graphqls`

**B.** Implement and configure a resolver for the custom field

**C.** Declare a REST route for the field in `etc/webapi.xml`

**D.** Add the field to an Admin UI component XML file

## ANSWER: A B

**Explanation:**

The module must declare the field in `etc/schema.graphqls`. Magento merges GraphQL schema files from enabled modules, and the schema declaration adds the field to the relevant GraphQL type, such as `ProductInterface`. The declaration identifies the field name, its GraphQL type, and the resolver class where a custom resolver is needed.

The module must implement a resolver class for the field and configure that resolver in the schema declaration. The resolver implements Magento's GraphQL resolver interface and returns the field value from the parent product data or another required data source. This keeps field resolution within Magento's GraphQL execution flow.

A REST route in `webapi.xml` does not expose a field through GraphQL, and a UI component configuration affects only Admin or storefront user interfaces. See Adobe Commerce GraphQL schema extension documentation.

## QUESTION NO: 11

You need to add a new text attribute to all products in the Magento store. When this attribute is displayed on the product page, its values must be different depending on the selected language.

Keeping simplicity in mind, how do you add this attribute?

**A.** Use the Magento CLI to create a new custom attribute, then generate dictionaries for all supported languages

**B.** Use a Data Patch to create a new EAV attribute

**C.** Add a new column to the catalog_product_entity table using declarative schema

**D.** Use the admin panel to create a new extension attribute

## ANSWER: B

**Explanation:**

Use a Data Patch to create a new EAV attribute. Product data in Adobe Commerce and Magento Open Source uses the Entity-Attribute-Value (EAV) model, so a product-specific text field should be created as a catalog product EAV attribute rather than as a database column or an extension attribute. A data patch provides a version-controlled, deployable way for a module to add the attribute during setup upgrades. The patch can use

`\Magento\Catalog\Setup\CategorySetupFactory` or the EAV setup facilities to add an attribute to the `catalog_product` entity.

To support distinct values for each selected language, configure the attribute scope as `Store View`. Store views commonly represent localized storefronts, and store-view scope allows an administrator to save a separate attribute value for each view. Magento resolves and displays the value for the active store view on the product page, allowing translated or localized text without custom database structures. The attribute should also be configured to be visible on the storefront as needed, and assigned to the relevant attribute set(s), ensuring it is available for all intended products. Adobe's documentation describes product attributes and their scope behavior in the Product attributes documentation, while the supported declarative mechanism for data changes is documented in Data patches.

## QUESTION NO: 12

What are two functions of a resource model? (Choose two.)

**A.** It executes create, retrieve, update and delete actions for an entity

**B.** It loads lists of entity models

**C.** It is made available in the Magento API for the purpose of data manipulation

**D.** It maps an entity to one or more database rows

## ANSWER: A D

**Explanation:**

A resource model is the persistence-layer class that performs database operations for a Magento entity. It contains the logic used to save an entity's data, retrieve it by its identifier, and delete it, thereby implementing the entity's create, retrieve, update, and delete interactions with storage. Magento models delegate these persistence operations to their associated resource models rather than embedding database-specific code in the model itself.

A resource model also defines how the entity is mapped to database storage. For the common single-table entity pattern, its constructor associates the resource model with the main database table and the entity ID field. This mapping lets Magento translate an entity model's data to the appropriate database row when loading or saving it. Resource models can also support more complex persistence arrangements, but their central role remains connecting an entity to its underlying database representation. Adobe's architecture documentation describes resource models as the layer responsible for database access and CRUD behavior, while the PHP component documentation explains the resource model's entity-table and identifier-field mapping. See Adobe Commerce persistence layer documentation and Adobe Commerce models and resource models documentation.

## QUESTION NO: 13

You want to remove a column introduced by a third-party extension via declarative schema.

How do you do that?

**A.** Create the etc/db_schema.xml file and specify disabled="true" on the column

**B.** Modify the original etc/db_schema.xml file and remove the column from there

**C.** Create a SchemaPatch file and remove the column programmatically

**D.** Copy the etc/db_schema.xml file into your module and remove the column from your copy

## ANSWER: A

**Explanation:**

Create the `etc/db_schema.xml` file in a custom module and declare the target column with `disabled="true"`. Magento merges declarative-schema definitions from enabled modules, so a module can use a matching table and column declaration to disable a column contributed by another module. During the declarative schema comparison, Magento identifies that disabled element and generates the required schema operation to remove it from the database. This is the declarative approach: the intended database state is expressed in XML, rather than being implemented with imperative PHP setup logic.

The declaration must identify the same table and column accurately, and the custom module must be enabled when Magento runs schema updates. After deploying the change, run `bin/magento setup:upgrade` so Magento processes the merged schema and applies the database change. Before removing a column, ensure that no active application code, indexes, constraints, or integrations still require its data, and back up production data as appropriate. Adobe Commerce documents that declarative schema supports disabling database elements through the `disabled` attribute and manages the necessary database changes from the merged XML schema. See the Declarative schema configuration documentation and the Declarative schema overview.