PassQueen
a.k.a dumpsqueen.com

Sitecore Experience Solution 9 Developer Exam

Sitecore Sitecore-Experience-Solution-9-Developer

Version Demo

Total Demo Questions: 17

Total Premium Questions: 177

Buy Premium PDF

https://passqueen.com

support@passqueen.com

QUESTION NO: 1

List the benefits of participating in the Sitecore Community.

A. Complete sample modules can be found on the Sitecore Marketplace.

B. You can get free 24/7 help from hundreds of community members around the globe, including Sitecore employees, Sitecore Certified Developers, and MVPs.

C. While contributing actively to the Sitecore Community, you can become a Sitecore MVP.

D. All

ANSWER: D

Explanation:

All

is correct because it encompasses the range of benefits described. The Sitecore Community connects practitioners with a worldwide network of developers, implementation specialists, partners, MVPs, and Sitecore employees. Community participation makes it possible to ask technical questions, exchange implementation knowledge, and draw on experience shared by other Sitecore professionals. Sitecore provides community channels and resources intended to support this collaboration; see the Sitecore Community.

The community ecosystem also includes the Sitecore Marketplace, where developers can discover and share modules and other extensions that can accelerate solution development. In addition, active and sustained contributions to the community can lead to recognition through the Sitecore MVP program. Sitecore describes this program as recognizing individuals who actively share knowledge and make valuable contributions to the Sitecore community; see Sitecore MVP. Therefore, access to shared modules, global peer assistance, and the opportunity for MVP recognition are collectively valid benefits, making the inclusive selection the single best answer.

QUESTION NO: 2

You need to insert a custom processor immediately before an existing processor in a Sitecore pipeline. Which configuration patch operation should you use?

A. patch:before

B. patch:instead

C. patch:source

D. patch:delete

ANSWER: A

Explanation:

Use `patch:before` on the custom processor element. Sitecore pipeline processors execute in the order in which they appear in the effective configuration. A `patch:before` instruction inserts the custom processor before the processor identified by the patch expression, allowing the custom logic to run at the required point without editing the original configuration file. This approach preserves upgradeability and follows Sitecore configuration patching practices.

QUESTION NO: 3

Where should you set insert options as a best practice?

A. In the Standard Values, so they are applied as a options to newly created items.

B. In the Standard Values, so they are applied by default to newly created items.

C. In the Standard templates, so they are applied as a default to newly created items.

D. In the base templates, so they are applied as a default to newly created items.

ANSWER: B

Explanation:

Insert options should be configured in a template's Standard Values item so that the permitted child-item templates are inherited as the default behavior for every item created from that template. Standard Values provide a central, reusable definition of default field values and template-level settings. This prevents authors from having to configure allowed child templates repeatedly on individual content items and ensures a consistent content-tree structure wherever that template is used.

In Sitecore, insert options are stored through the `__Masters` and `__Insert Rules` fields, which are available in the Standard Template's Insert Options section. When these fields are set on Standard Values, new items based on the associated template receive those settings through Sitecore's standard-value inheritance mechanism. This approach also makes future maintenance simpler: updating the template's Standard Values updates the default insert behavior without requiring configuration changes across each existing content branch. See Sitecore's documentation on the Standard Values item and insert options.

QUESTION NO: 4

How can you define renderings as compatible?

A. modifying a field in their data definition

B. modifying a field in their field definition

C. modifying a field in their component definition

D. None

ANSWER: C

Explanation:

You define renderings as compatible by modifying a field in their component definition. Specifically, the rendering definition item contains the *Compatible Renderings* field, where an administrator or developer can specify the rendering definition items that may be substituted for, or treated as compatible with, that rendering. Sitecore uses this compatibility relationship in authoring scenarios such as replacing a component in Experience Editor while preserving an appropriate editing experience and, where applicable, maintaining compatibility with the existing datasource and presentation context. The relationship is configured on the rendering's definition item in the Core database rather than in the datasource template or an individual field definition. This makes compatibility a property of the presentation component itself and allows the same rendering relationship to be reused consistently wherever the component is available. The configuration should be made deliberately: compatible renderings should represent alternatives that can reasonably fulfill the same presentation role and work with the expected content structure. Sitecore documents compatible-rendering configuration as part of rendering definition and Experience Editor presentation configuration. See the Sitecore documentation on compatible renderings and the rendering definition item reference.

QUESTION NO: 5

How are component parameters stored in the Sitecore database?

A. Encoded form

B. As clear text in URL query string format

C. Decoded Form

D. All

ANSWER: B

Explanation:

Component parameters are stored as clear text in URL query string format. In Sitecore, a component is represented by a rendering definition in an item's Layout field. That definition includes a rendering-parameters attribute, commonly represented as `par`, whose content uses query-string-style name/value pairs. For example, a rendering can persist parameter values in a form conceptually similar to `ParameterName=Value&AnotherParameter=Value`. This representation lets Sitecore associate rendering-specific configuration with the component instance placed on a page while retaining the values in the serialized layout definition. The values are stored as part of the Layout field's XML, so XML escaping is applied where required—for example, an ampersand between parameter pairs is represented safely in XML—but the parameter payload itself follows URL query string conventions rather than being stored as a separate decoded database structure. This is why APIs that read rendering parameters expose a parameter collection based on those key/value pairs. Sitecore's layout documentation describes how presentation details are held in the Layout field and rendering definitions, while its rendering-parameter documentation covers parameters associated with individual renderings. See Sitecore: The Layout field and Sitecore: Rendering parameters.

QUESTION NO: 6

Provides a set of guidelines for your Sitecore projects.

A. Helix

B. Habitat

C. Sitecore Guidelines

D. None

ANSWER: A

Explanation:

Helix is Sitecore's recommended set of architecture principles and conventions for designing, building, and maintaining Sitecore solutions. It provides practical guidance for organizing a solution into independently understandable modules and arranging those modules within the Foundation, Feature, and Project layers. This structure helps development teams manage dependencies, separate reusable technical capabilities from business functionality, and make project-specific composition explicit. Helix also defines conventions that apply across solution structure, serialization, configuration, templates, rendering, and code, giving teams a common vocabulary and a consistent basis for implementation decisions. These guidelines are intended to improve maintainability, support parallel development, and reduce the coupling that can make large Sitecore implementations difficult to extend or upgrade. The official Helix documentation describes it as a collection of principles and conventions for Sitecore development, including its modular architecture and dependency rules. See the Sitecore Helix documentation and its overview of architecture principles.

QUESTION NO: 7

Add servers to each role, adding more servers that fulfill the same function

A. Vertical Scaling

B. Horizontal Scaling

C. Inclined Scaling

D. None

ANSWER: B

Explanation:

Horizontal Scaling is correct because it means increasing system capacity by adding more machines that perform the same role or workload. In a Sitecore deployment, this can mean running multiple Content Delivery instances, multiple Content Management instances where the topology supports them, or additional processing and reporting role instances as required. Requests and work can then be distributed across those equivalent servers, commonly through a load balancer or by role-specific workload distribution. This approach improves throughput and availability while allowing capacity to grow incrementally as traffic, publishing activity, or data-processing demand increases.

Sitecore's scaling guidance describes deploying roles across multiple instances to meet availability and performance requirements; roles can be scaled independently according to the workload they handle. That is precisely the model described by adding servers that fulfill the same function. Sitecore also emphasizes that topology and scaling decisions must account for role dependencies, databases, search, session-state configuration, and load-balancing behavior, so that each added instance participates consistently in the overall solution. See the Sitecore Experience Platform 9 developer documentation and Sitecore's platform administration and architecture documentation for deployment and scaling guidance.

QUESTION NO: 8

In an Image/File field, the field source defines the browsing location in the

A. Media Library

B. Data definition

C. Facets

D. Item Path or depends on Insert Options

ANSWER: A

Explanation:

Media Library is correct because the Source property on an Image or File field controls the media-item location presented when an editor selects or changes that field's value. In Sitecore, Image fields store a reference to a media item, and File fields similarly reference a file held as a media item. Configuring a source value lets a solution restrict the picker to an appropriate media-library branch, such as a site-specific images or downloads folder. This both guides authors to approved assets and helps prevent media references from being selected from unrelated areas of the repository.

The source is normally specified as a Sitecore item path or query that resolves to the desired starting or permitted media location. For example, a field source can point to a dedicated folder beneath `/sitecore/media library`; when the author opens the media selector, Sitecore uses that configuration to scope the available browsing context. This behavior is part of how Sitecore's field definitions tailor the Content Editor experience for a template field. Sitecore's documentation describes Image fields as selecting media items from the Media Library and documents the use of field source values to constrain selection: The Image field and The File field.

QUESTION NO: 9

What happens if field section names are the same?

A. Sitecore will remove these fields into the same field section

B. Sitecore will merge these fields into the same field section

C. Sitecore will edit these fields into the same field section

D. Sitecore will merge these fields into the different field section

ANSWER: B

Explanation:

Sitecore will merge these fields into the same field section. In a Sitecore template, fields are organized beneath section items, which control how related fields are grouped and displayed in Content Editor. When template inheritance brings together fields from base templates or other applicable template definitions, Sitecore identifies sections by their names. Sections that have the same name are treated as one logical section for presentation, so their fields appear together under the shared section heading rather than as duplicate headings. This is useful when multiple templates contribute fields to a common business grouping, such as content, metadata, or page settings. The behavior helps keep the editing interface organized and lets template designers extend a shared section through inheritance without requiring authors to navigate repeated sections with identical labels. Template inheritance is a core Sitecore mechanism for composing reusable field definitions, and field sections remain part of the template structure used to render an item's editing experience. See the Sitecore documentation on template inheritance and the template architecture for the related template and field organization concepts.

QUESTION NO: 10

:

What is the purpose of the Standard Template?

A. To define default values for templates

B. Define fields that Sitecore uses internally but are not shown to the user

C. To define some fields you will need in all your pages: title, text, keywords, etc.

D. To define the overall structure of your page

ANSWER: B

Explanation:

The correct answer is **"Define fields that Sitecore uses internally but are not shown to the user"**. The Sitecore Standard Template is a built-in base template from which all Sitecore templates ultimately inherit. It supplies the standard fields required for core platform capabilities, including item identity, security, workflow, publishing restrictions, language/version details, locking, layout and presentation settings, and item statistics. These fields let Sitecore manage an item consistently throughout its lifecycle without requiring developers to recreate the same infrastructure fields in every content template.

Standard fields are normally hidden in the Content Editor to keep the authoring interface focused on fields intended for everyday content entry. Administrators and developers can reveal them through the *View* ribbon when they need to configure such features as workflow, publishing dates, security, or presentation details. Thus, the Standard Template provides the internal system-level metadata and behavior shared by Sitecore items, rather than defining the business-content fields for a particular page type. See Sitecore's documentation on the Standard Template and standard fields.

QUESTION NO: 11

What are some recommended practices when creating templates?

A. provide default field values

B. avoid duplicating a field name within the same template

C. limit Rich Text Editors

D. All the above

ANSWER: D

Explanation:

All the above is correct because each listed practice supports a maintainable, predictable Sitecore content model. Supplying default field values helps authors begin with useful, consistent content and reduces unnecessary manual entry when new items are created. Ensuring that a field name is not duplicated within a template avoids ambiguity when developers and content authors access fields through Sitecore APIs, rendering code, search, and serialization. Template inheritance can make field resolution harder to understand, so clear, unique field naming is particularly important for long-term solution maintenance. Limiting use of Rich Text Editor fields is also recommended because rich text permits broad, presentation-oriented markup that can be difficult to control consistently across channels and components. A structured template design, using focused fields for specific content where practical, gives developers more reliable rendering control and makes content more reusable. Together, these practices align with Sitecore's template-design guidance: templates define the structure and defaults for content items, and a well-designed schema improves authoring and solution maintainability. See Sitecore's template data model documentation and its template best practices guidance.

QUESTION NO: 12

What additional databases are required when running xDB?

A. Collection Database (MongoDB)

B. Reporting

C. Session State

D. All

ANSWER: D

Explanation:

All is correct because an xDB-enabled Sitecore Experience Platform deployment relies on each of the listed data stores to support distinct parts of the experience-data architecture. The Collection Database, implemented with MongoDB in Sitecore Experience Platform 9, stores xDB contact and interaction data collected from visits and other engagement activity. The Reporting database provides the reporting data store used by Sitecore reporting and analytics features. Session State is also required to persist session information appropriately, particularly in production and scaled environments where sessions cannot depend on a single application instance's memory. Together, these stores allow Sitecore to collect experience data, maintain session continuity, process it, and make it available for reporting. Sitecore's installation documentation describes configuring the required SQL Server and MongoDB data stores as part of an XP/xDB deployment, while its xDB documentation describes the collection and processing architecture that uses them. See the Sitecore XP 9 installation documentation and the Sitecore xDB documentation.

QUESTION NO: 13

An author adds a component to a page in the Experience Editor and the selects the Associated Content dialog box automatically appears. What triggers this behavior?

A. The component's Datasource Template field is filled in.

B. The component's Datasource Location field is filled in.

C. None

D. Both

ANSWER: B

Explanation:

The component's Datasource Location field is filled in. A rendering's Datasource Location field tells Sitecore where authors may look for, select, or create the content item that will serve as that rendering's data source. When an author adds the

rendering in Experience Editor, Sitecore detects this configured data-source location and opens the Select Associated Content dialog so the author can associate an appropriate item with the new component. This supports the Experience Editor workflow by ensuring that a component needing context-specific content is connected to an item from an approved content location at the point it is added. The selected item is then stored as the rendering's data source in the page's presentation details. The configured location can be a content-tree path or a supported query, allowing implementers to constrain the authoring choice to the relevant folder or set of items. Sitecore's rendering data-source configuration is described in the Sitecore documentation on defining data sources. Additional background on configuring renderings and their presentation settings is available in the Sitecore rendering documentation.

QUESTION NO: 14

contains only the published version of each item including the different language and version that have been published.

A. web

B. master

C. core

D. All

ANSWER: A

Explanation:

The **web** database is correct. In Sitecore, publishing transfers approved item versions from the master database to the web database, which is the database normally used by the public-facing content delivery website. It therefore contains the versions of content that have been published and are intended to be available to site visitors. Sitecore maintains language-specific content and item versions, and publishing evaluates which versions are publishable according to their publishing restrictions, workflow state, and related settings. As a result, the web database represents the published content set, including published language variants and versions where applicable, rather than the complete authoring history. This separation allows content authors to create, edit, and manage items safely in the authoring environment before those changes are made live. Sitecore's database model identifies master as the primary authoring database and web as the published-content database used for delivery. See the Sitecore database documentation and the Sitecore publishing documentation.

QUESTION NO: 15

What are Standard Values?

A. Child item of data templates

B. Standard Template

C. Parent item of data templates

D. Parent item of standard Template

ANSWER: A

Explanation:

Child item of data templates is correct because Sitecore stores a template's standard values in a special child item named `__Standard Values`, beneath the template definition. This item is created under the relevant template in the Template Manager and provides default field values for every item created from that template, unless a value is set directly on the individual item. Standard values can also hold layout details and presentation settings, allowing a template to supply a consistent initial presentation configuration to its items. In addition, Sitecore supports token replacement in standard values, such as `$name` and `$date`, so values can be generated dynamically when an item is created. These defaults are inherited through template inheritance, which makes the standard-values item a core part of defining reusable content behavior and

structure. Sitecore documents the `__Standard Values` item as the child item that contains the default values for fields in a data template. See the official Sitecore documentation on the Standard Values item and standard values.

QUESTION NO: 16

Name the different types of MVC renderings

A. View rendering

B. controller rendering

C. item rendering

D. All

ANSWER: D

Explanation:

All is correct because Sitecore MVC provides view renderings, controller renderings, and item renderings. A view rendering uses a Razor view to generate markup and is appropriate when rendering logic can remain primarily in the view and its model. A controller rendering invokes a controller action, allowing the solution to run server-side logic, prepare a model, and select a view. An item rendering renders another Sitecore item, enabling reusable, item-based presentation composition. These rendering definitions are created under `/sitecore/layout/Renderings` and can be assigned to a page's layout details or controlled through presentation rules. The rendering type determines how Sitecore's MVC rendering pipeline resolves the component at request time, while datasource and rendering parameters can supply context-specific content and configuration. Therefore, the complete list in the question is represented by **All**, rather than only one rendering form. Sitecore's developer documentation describes MVC renderings and how they participate in the MVC implementation: MVC and Sitecore. For the associated layout and presentation concepts, see Presentation details.

QUESTION NO: 17

How can you add custom code that runs when an item is saved in Sitecore?

A. Add an event handler to the appropriate Sitecore event.

B. Add a processor to the publish pipeline only.

C. Create a Placeholder Settings item.

D. Add a workflow command.

ANSWER: A

Explanation:

Add an event handler to the appropriate Sitecore event. Sitecore exposes events for important item operations, including saving, deleting, moving, and publishing items. A developer can create a handler class and register the handler in configuration using the event definition for the required operation, such as an item save event.

Event handlers are appropriate for cross-cutting behavior that must occur in response to a Sitecore operation, for example updating related items, validating dependent data, or invoking an integration. The handler should be efficient and should avoid unnecessary item edits that could trigger additional events. Custom registrations should be made in an include patch file rather than by directly changing the product configuration. See the Sitecore events documentation and configuration patching documentation.