Servicenow

(CAD)

ServiceNow Certified Application Developer

Total: **169 Questions**

Link:

Question: 1

Which of the following statements is true for the Form Designer? a) To add a field to the form layout, drag the field from the Fields tab to the desired destination on the form. b) To create a new field on a form's table, drag the appropriate data type from the Field Types tab to the form and then configure the new field. c) To remove a field from the form layout, hover over the field to enable the Action buttons, and select the Delete (X) button. d) To add a section to the form layout, drag it from the Field Types tab to the desired destination on the form.

A.a, b, c, and d
B.b, c, and d
C.a, b, and d
D.a, b, and c

Answer: D

Explanation:

The correct answer is D (a, b, and c). Let's break down why each statement is true concerning the ServiceNow Form Designer and why 'd' is incorrect:

a) To add a field to the form layout, drag the field from the Fields tab to the desired destination on the form. This is a fundamental function of the Form Designer. The Fields tab displays all available fields for the form's table. Dragging a field from this tab and dropping it onto the form layout adds that field to the form's user interface.

b) To create a new field on a form's table, drag the appropriate data type from the Field Types tab to the form and then configure the new field. This describes the process of creating new fields directly from the Form Designer. You select a data type (e.g., string, integer, date) from the Field Types tab, drag it onto the form, and then configure the new field's properties (name, label, etc.) within the interface. This avoids needing to navigate to the table configuration separately.

c) To remove a field from the form layout, hover over the field to enable the Action buttons, and select the Delete (X) button. This is the standard way to remove a field from the form's display. Hovering over a field reveals action buttons, including the "Delete" (X) button, which removes the field from the form layout. Note that this does not delete the field from the table definition; it only removes it from the form's presentation.

d) To add a section to the form layout, drag it from the Field Types tab to the desired destination on the form. This statement is incorrect. While the Form Designer allows you to add sections, you typically don't drag sections directly from the "Field Types" tab. Instead, you right-click on the form layout or use the context menu within the designer to insert a new section. Dragging from field type tab is exclusive to adding new fields, not sections.

Therefore, statements a, b, and c are accurate descriptions of how the Form Designer functions, making option D the correct choice.

For more information, consult the official ServiceNow documentation:

ServiceNow Docs - Using the Form Designer

Question: 2

Which of the following are configured in an Email Notification? a) Who will receive the notification. b) What content will be in the notification. c) When to send the notification. d) How to send the notification.

A.a, b and c
B.a, b, and d

C.b, c and d

D.a, c and d

Answer: A

Explanation:

The correct answer is A (a, b, and c) because ServiceNow Email Notifications are fundamentally defined by three key aspects: who receives the notification (a), what the notification contains (b), and when the notification is triggered and sent (c).

Who will receive the notification (a): This is configured using the "Who will receive" section of the notification record. You define recipients by selecting users, groups, email addresses, or through scripts that dynamically determine the recipients based on the event or record conditions.

What content will be in the notification (b): The "What it will contain" section configures the subject line and the body of the email. You can use static text, variables from the triggering record, or Jelly syntax to create dynamic and personalized content.

When to send the notification (c): The "When to send" section defines the conditions that must be met to trigger the notification. This can be based on events (e.g., record creation, update, or deletion), scheduled jobs, or other custom scripting logic.

The option "How to send the notification (d)" is less directly configured within the email notification record itself. The "how" part, such as the email server used or the format of the email, is managed at the system level through email properties and configurations, rather than within each notification. Email properties control global aspects such as default from address, mail server, and retry behavior. While you can set reply-to addresses in the notification, this affects where replies go, not how the notification is physically transmitted.

Therefore, the most accurate description of what's configured within a ServiceNow Email Notification record is who receives it, what it contains, and when it's sent.

Further research:

ServiceNow Documentation: https://docs.servicenow.com/ (Search for "Email Notifications")
ServiceNow Developer Site: https://developer.servicenow.com/

Question: 3

To see what scripts, reports, and other application artifacts will be in a published application:

A.Enter the name of the Application in the Global search field

B.Open the list of Update Sets for the instance

C.Examine the Application Files Related List in the application to be published

D.Open the artifact records individually to verify the value in the Application field

Answer: C

Explanation:

The correct answer is C: Examine the Application Files Related List in the application to be published.

Justification:

The Application Files related list within a ServiceNow application provides a centralized view of all the components that constitute the application. These components, also known as application artifacts, include

scripts (business rules, client scripts, script includes), UI elements (forms, lists, UI actions), reports, workflows, tables, and more. This list directly reflects what will be packaged and published when the application is distributed as an update set.

Option A is incorrect because using the global search to find an application by name will only navigate to the application record itself, not display the specific contents to be published. The global search primarily locates records based on keywords, not the detailed components of an application.

Option B is incorrect because while update sets are used for transporting application artifacts, inspecting the entire list of update sets won't isolate the artifacts specific to a single application. Instead, update sets may contain changes from various applications and configurations. Focusing on the specific Application Files related list ensures you only see components related to the application intended for publication.

Option D is incorrect because manually opening each artifact record to verify the application field is extremely time-consuming and inefficient. The Application Files related list offers an aggregated and consolidated view, making it far more practical for reviewing the application's contents before publishing. It's the most efficient approach for understanding the scope of what will be included in a published ServiceNow application. The Application Files Related List aggregates all related items making viewing all application components easy.

Therefore, examining the Application Files related list is the most direct and effective way to determine the scripts, reports, and other application artifacts that will be included when the application is published.

Supporting Resources:

1. ServiceNow Documentation on Application Development: https://developer.servicenow.com/devportal/$glide.url_portal_db.do?sys_id=77380514dbb877043982324e0b96190b
2. ServiceNow Documentation on Update Sets: https://docs.servicenow.com/bundle/quebec-application-development/page/build/system-update-sets/concept/c_SystemUpdateSets.html

Question: 4

Which one of the following is NOT a debugging strategy for client-side scripts?

A.g_form.addInfoMessage()
B.Field Watcher
C.jslog()
D.gs.log()

Answer: D

Explanation:

The correct answer is D, gs.log(). Here's why:

Client-side scripts in ServiceNow, like Client Scripts and UI Policies, execute within the user's web browser. Debugging these scripts requires methods accessible within that browser environment and designed for client-side scripting.

Options A, B, and C (g_form.addInfoMessage(), Field Watcher, and jslog()) are all valid debugging strategies for client-side scripts:

g_form.addInfoMessage() displays messages directly on the form, allowing developers to see variable values

or execution paths.
Field Watcher (available through the right-click context menu on form fields) monitors changes to a specific field, providing insights into how client-side scripts affect field values. This is a built-in browser feature that is available out-of-the-box in ServiceNow.
jslog() writes messages to the browser's console, which can be viewed using developer tools.

Option D, gs.log(), is not a client-side debugging strategy. gs.log() is a server-side function. Server-side scripts, like Business Rules and Script Includes, execute on the ServiceNow server. gs.log() writes messages to the system log on the server, which is inaccessible directly from the client's browser. Therefore, it's ineffective for debugging client-side code since it's logging information on the server, not within the browser's execution environment. The client-side scripts don't have access to server-side logs.

In short, you use client-side tools to debug client-side scripts, and gs.log() is strictly a server-side tool. The key concept to remember is the clear separation between client-side (browser) and server-side execution environments in ServiceNow.

For more information on client-side scripting, refer to the ServiceNow documentation:
https://developer.servicenow.com/dev.do#!/learn/learning-plans/quebec/new_to_servicenow/app_store_learnv2_client_side_scripting_quebec/app_store_learnv2_client_scri

And for server-side scripting (including gs.log()), see:
https://developer.servicenow.com/dev.do#!/learn/learning-plans/quebec/new_to_servicenow/app_store_learnv2_scripting_quebec/app_store_learnv2_server_side_scripting_

Question: 5

Which Application Access configuration field(s) are NOT available if the Can read configuration field is NOT selected?

A.All access to this table via web services
B.Can create, Can update, and Can delete
C.Can read does not affect the availability of other Application Access fields
D.Allow configuration

Answer: B

Explanation:

The correct answer is B. Can create, Can update, and Can delete.

If the "Can read" checkbox is not selected in the Application Access configuration within ServiceNow, it signifies that users from outside the application scope are denied the ability to read records from that table. Logically, if they cannot read, they also shouldn't be able to create, update, or delete records. ServiceNow's design reflects this dependency: the "Can create," "Can update," and "Can delete" fields become unavailable or non-functional because they are contingent on read access.

This mechanism enforces a fundamental security principle of least privilege. Limiting read access is a common security measure, and ServiceNow appropriately prevents unintended or unauthorized data modification if read access is restricted at the application access level. If users can't see the data, they shouldn't be able to modify it. The "All access to this table via web services" and "Allow configuration" options are managed independently and don't directly rely on 'Can read' functionality.

Authoritative links for further research:

ServiceNow Docs: Application Access Settings: (Search ServiceNow documentation portal for "Application Access Settings"). This explains the functionality of each of the application access settings, including Can read, Can create, Can update, and Can delete.

ServiceNow KB Article: Security Best Practices: (Search ServiceNow Knowledge Base for "Security Best Practices"). This provides best practices on how to secure your ServiceNow instance, including how to control access to data using application access controls.

Question: 6

Which of the following is NOT a trigger type in Flow Designer?

A.Outbound Email
B.Application
C.Record
D.Schedule

Answer: A

Explanation:

The correct answer is A. Outbound Email.

Flow Designer in ServiceNow utilizes triggers to initiate flows based on specific events or conditions. Triggers are the starting points for automating processes within the platform. Several trigger types exist to cater to diverse automation needs.

Record triggers initiate flows when a record is created, updated, or deleted in a specified table. Schedule triggers start flows based on a predefined schedule, such as daily or weekly. Application triggers launch flows when a specific application event occurs. These trigger types directly align with ServiceNow's automation capabilities, allowing administrators and developers to respond to database changes, recurring schedules, and application-specific events.

Outbound Email is not a standard trigger type in Flow Designer. While ServiceNow can send outbound emails as part of a flow action, the arrival of an outbound email cannot directly initiate a flow. Typically, email integrations are handled through inbound actions or email parsers that process incoming emails and trigger specific actions based on their content. Therefore, flows are triggered by events or schedules within the ServiceNow instance, not by external outbound email activity.

Further research can be done on the official ServiceNow documentation:

ServiceNow Flow Designer Triggers: https://docs.servicenow.com/bundle/utah-platform-administration/page/administer/flow-designer/concept/flow-triggers.html

Question: 7

When creating new application files in a scoped application, cross scope access is turned on by default in which of the following?

A.REST messages
B.Table
C.Script Include
D.Workflow

Answer: B

Explanation:

The correct answer is B (Table). When a new table is created within a scoped application, the ServiceNow platform automatically configures specific cross-scope access properties to manage how other applications can interact with the table. By default, the system intends to restrict access. However, the cross-scope access configurations control what operations other scopes may perform on the table (read, write, create, delete).

Script Includes (C) do require cross-scope access configuration but it is not turned on by default. They need to be explicitly granted access to be called from other scopes. Similarly, REST Messages (A) need explicit configuration for access from other scopes and do not have default cross-scope access upon creation. Workflows (D) also don't enable cross-scope access by default, but are governed by roles and permissions.

The reason tables have initial cross-scope access configuration involves security and data integrity.
ServiceNow aims to limit unintentional access to sensitive data stored within the table. The default configuration for tables defines a baseline restriction, and administrators can then selectively open up access as required.

Therefore, tables uniquely feature these default settings for cross-scope access compared to other listed application file types upon their creation.

For further research, refer to the official ServiceNow documentation on Scoped Applications and Cross-Scope Access:

1. ServiceNow Docs: Scoped Applications
2. ServiceNow Docs: Cross-Scope Access

Question: 8

In an Email Notification, which one of the following is NOT true for the Weight field?

A.Only Notifications with the highest weight for the same record and recipients are sent

B.A Weight value of zero means that no email should be sent

C.The Weight value defaults to zero

D.A Weight value of zero means the Notification is always sent when the Notification's When to send criteria is met

Answer: B

Explanation:

The correct statement that is NOT true about the Weight field in ServiceNow Email Notifications is **B. A Weight value of zero means that no email should be sent.**

Here's why:

Weight Field Functionality: The Weight field in ServiceNow email notifications determines the order in which notifications are processed when multiple notifications are triggered by the same event.

Weight Hierarchy: ServiceNow evaluates notifications based on weight. Notifications with higher weight values are given priority and are more likely to be sent. If multiple notifications meet the criteria, only the one with the highest weight is typically sent.

Statement A (Correct): Only Notifications with the highest weight for the same record and recipients are sent: This is generally true. If multiple notifications match, the one with the highest weight wins.

Statement C (Correct): The Weight value defaults to zero: By default, the Weight field is set to zero,

indicating a low priority.

Statement D (Correct): A Weight value of zero means the Notification is always sent when the Notification's When to send criteria is met: If no other notifications have a higher weight and the 'When to Send' conditions are met, a notification with weight zero will be sent. It simply means it has the lowest priority compared to others.

Statement B (Incorrect): A Weight value of zero means that no email should be sent: This is false. A weight of zero doesn't prevent an email from being sent; it simply means the notification has the lowest priority. If no other notifications are triggered or if no other notifications have a higher weight and the 'When to send' conditions are met, then a notification with a weight of 0 will be sent. It just has the lowest priority.

In essence, the weight field helps manage email overload by preventing redundant or less important notifications from being sent when a more relevant or critical notification is triggered.For more in-depth information on ServiceNow email notifications and weight, consult the official ServiceNow documentation:

ServiceNow Docs - Email Notifications

Question: 9

Which of the following objects does a Display Business Rule NOT have access to?

A.previous
B.GlideSystem
C.g_scratchpad
D.current

Answer: A

Explanation:

The correct answer is A (previous). A Display Business Rule executes **before** the form is rendered to the user. Its primary purpose is to add information to the g_scratchpad object, which the client-side script can then access. The current GlideRecord object is available, representing the record being displayed. GlideSystem (accessible via gs) provides access to server-side scripting APIs. However, since Display Business Rules run before any form interaction or submission, there is no "previous" version of the record available. The previous object typically holds the values of the record before a change, which only exists when records are updated or inserted (as in a before or after Business Rule). Since a display rule is for displaying not changing, there's no previous record to reference. Therefore, Display Business Rules inherently lack access to the previous object, making option A the correct choice. Display Business Rules run on the server but are primarily used to prepare data for client-side scripts by using the g_scratchpad object. They operate in a read-only context regarding record changes.

Relevant Resources:

ServiceNow Docs on Business Rules:
https://developer.servicenow.com/devportal/$knowledge.do#!/api/latest/server/no-namespace/c_GlideSystemAPI
ServiceNow Docs on GlideRecord:
https://developer.servicenow.com/devportal/$knowledge.do#!/api/latest/server/no-namespace/c_GlideRecordAPI
ServiceNow Docs on Client-Side Scripting:
https://developer.servicenow.com/devportal/$knowledge.do#!/api/latest/client/

Question: 10

Which of the following features are available to Global applications? (Choose two.)

A.Automated Test Framework
B.Source Control
C.Delegated Development
D.Flow Designer

Answer: AD

Explanation:

The correct answer highlights features accessible to Global applications within ServiceNow. Global applications, existing outside any specific application scope, benefit from the broadest range of platform capabilities.

A. Automated Test Framework (ATF): ATF is indeed available to Global applications. It is a core platform feature that allows developers to create and run automated tests to ensure the quality and stability of their applications, regardless of scope. ATF tests can be used on Global applications to perform various testing functions such as UI testing, server-side testing, and API testing.

D. Flow Designer: Flow Designer is another critical feature accessible to Global applications. It allows developers to automate business processes through a visual, drag-and-drop interface. Global applications can leverage Flow Designer to create workflows, approvals, and other automated processes that span across the entire ServiceNow instance.

Why the other options are less suitable:

B. Source Control: While Source Control Integration is a core capability in ServiceNow, its primary purpose is version control and team collaboration. Source Control itself doesn't directly define functionality that is exclusively used by Global applications. It is a feature generally helpful across all application scopes.

C. Delegated Development: Delegated Development is primarily used to grant specific users or groups the permission to develop within a specific application scope. This feature aims at controlled application development process within a particular application, where the Global scope doesn't necessarily require delegated development for its wider accessibility.

In essence: Global applications, by their nature, require access to core platform functionalities for comprehensive development, testing, and automation, making ATF and Flow Designer particularly suitable.

Supporting Links:

ServiceNow Docs - Automated Test Framework:https://docs.servicenow.com/bundle/utah-application-development/page/administer/auto-test-framework/concept/atf-concept.html
ServiceNow Docs - Flow Designer:https://docs.servicenow.com/bundle/utah-servicenow-platform/page/administer/flow-designer/concept/flow-designer.html

Question: 11

Which one of the following is NOT a UI Action type?

A.List choice
B.Form button
C.List banner button

D.Form choice

Answer: D

Explanation:

The correct answer is D, "Form choice." Here's why:

UI Actions in ServiceNow are scripts that add buttons, links, and context menu items to forms and lists, making the platform more interactive and user-friendly. They allow users to perform actions directly from these interfaces. ServiceNow provides several UI Action types designed to trigger specific actions or display information in a controlled manner.

Let's break down the other options:

A. List choice: A list choice UI Action adds a choice list (a dropdown menu) to a list view. This allows users to select an action from the list and apply it to one or more selected records. It's a common way to provide batch processing capabilities.

B. Form button: A form button UI Action adds a button to a form. Clicking this button typically triggers a script that can perform various actions, such as updating records, creating related records, or triggering workflows.

C. List banner button: A list banner button UI Action appears as a button in the banner at the top of a list view. This is useful for actions that apply to the list as a whole, like exporting data or refreshing the list view.

"Form choice," however, is not a standard or recognized UI Action type in ServiceNow. While you can certainly implement dropdowns or choice lists on forms through other means (like using choice fields), there isn't a dedicated UI Action type specifically called "Form choice." UI Actions focus on triggering backend logic rather than rendering form elements directly. The closest concept would be a UI Policy which can influence the appearance or behavior of existing form elements like choice fields, but that's a distinct feature and not a UI Action type.

In summary: List choice, Form button, and List banner button are valid UI Action types within ServiceNow, each serving a specific purpose in enhancing user interaction. Form choice isn't an established UI Action type, making it the correct answer.

Authoritative Links:

ServiceNow Docs: UI Actions:https://developer.servicenow.com/dev.do#!/learn/learning-plans/quebec/new_to_servicenow/app_store_learnv2_ui_actions_quebec/app_store_learnv2_ui_actions_concept
ServiceNow Docs: Defining UI Actions:https://docs.servicenow.com/bundle/sandiego-platform-administration/page/scripting/ui-actions/task/t_DefineAUIAction.html

Question: 12

Which of the following is NOT supported by Flow Designer?

A.Call a subflow from a flow
B.Test a flow with rollback
C.Use Delegated Developer
D.Run a flow from a MetricBase Trigger

Answer: B

Explanation:

The correct answer is B. Here's why:

Flow Designer is a ServiceNow feature that allows developers to automate processes using a visual, low-code environment. It offers functionalities such as calling subflows, enabling modularity and reusability (A). Delegated Developers can indeed work on flows, allowing admins to grant specific development rights (C).

MetricBase triggers, introduced later in ServiceNow releases, can absolutely initiate flows based on predefined conditions and data thresholds (D).

However, Flow Designer's testing capabilities do not directly support rollback functionality. While you can test a flow and examine the data and results, there isn't an automatic rollback mechanism built into the testing interface to revert changes made by the flow during testing. In essence, if the flow makes unwanted changes, you would have to manually revert them.

Therefore, the statement about testing a flow with rollback is the functionality that is not natively supported in Flow Designer.

For further reading about Flow Designer features and capabilities, refer to the official ServiceNow documentation.https://docs.servicenow.com/bundle/sandiego-servicenow-platform/page/administer/flow-designer/concept/flow-designer.html

Question: 13

Which of the following are true for reports in ServiceNow? (Choose three.)

A.Any user can see any report shared with them.
B.Can be a graphical representation of data.
C.All users can generate reports on any table.
D.Can be run on demand by authorized users.
E.Can be scheduled to be run and distributed by email.

Answer: BDE

Explanation:

The correct answer is BDE. Here's why:

B. Can be a graphical representation of data: ServiceNow reports are versatile and support various visualizations like bar charts, pie charts, line graphs, and more. This allows users to easily understand trends and patterns within their data. ServiceNow Docs: Reports

D. Can be run on demand by authorized users: Access to reports, and the ability to run them, is controlled by roles and permissions within ServiceNow. Users with appropriate permissions can execute reports whenever they need the information. This is a key aspect of access control and data security in cloud platforms.

E. Can be scheduled to be run and distributed by email: ServiceNow allows administrators or report creators to schedule reports to run at specific intervals (e.g., daily, weekly, monthly). The generated reports can then be automatically emailed to designated recipients. This automation streamlines data delivery and ensures timely access to crucial information, reflecting the automation capabilities of cloud platforms. ServiceNow Docs: Scheduled Reports

Let's examine why the other options are incorrect:

A. Any user can see any report shared with them: While users can see reports shared specifically with them, it's not true that they can see any report. ServiceNow's role-based access control determines who can see

which reports. Reports can be shared with specific users, groups, or roles.

C. All users can generate reports on any table: Similarly, not all users can generate reports on any table. Reporting access is dependent on roles and permissions associated with tables and reporting capabilities within the ServiceNow instance. The report_admin role, report_global role, etc., give users permission to create reports and can be restricted by ACLs.

Question: 14

Modules must have a Link type. Which one of the following is a list of Link types?

A.List of Records, Separator, Catalog Type, Roles
B.Assessment, List of Records, Separator, Timeline Page
C.List of Records, Content Page, Order, URL (from arguments:)
D.Assessment, List of Records, Content Page, Roles

Answer: B

Explanation:

The correct answer is B: Assessment, List of Records, Separator, Timeline Page.

Modules in ServiceNow are navigation elements that appear within an application. They serve as shortcuts to specific functionalities and data within the platform. Each module needs a "Link type," which defines the action the module performs when clicked. Understanding these link types is crucial for application development.

Option B provides a legitimate list of Link types available in ServiceNow module configuration. "Assessment" allows users to directly access or initiate assessments. "List of Records" is a standard type that displays a table's records based on specified filters. "Separator" acts as a visual divider in the application navigator to organize modules. Finally, "Timeline Page" can present a timeline view of related activities, tasks or records.

Options A, C and D contain elements that are either not valid link types, or include inaccurate components. "Catalog Type" and "Order" are related to Service Catalog items or requests, but not directly module link types. "URL (from arguments:)" while a possibility, it should really just be "URL (External)" and "Roles" is not a Link type.

Therefore, Option B offers a combination of valid and common ServiceNow module link types, making it the correct response.

For further information, you can research the below links:

ServiceNow Docs - Navigation modules: https://docs.servicenow.com/bundle/sandiego-platform-administration/page/build/system-navigation/concept/c_NavigationModules.html
ServiceNow Community: (Search for "Module Link Types")

Question: 15

Which one of the following is true for a table with the `Allow configuration` Application Access option selected?

A.Only the in scope application's scripts can create Business Rules for the table
B.Any user with the application's user role can modify the application's scripts
C.Out of scope applications can create Business Rules for the table

D.Out of scope applications can add new tables to the scoped application

Answer: C

Explanation:

The correct answer is C: Out of scope applications can create Business Rules for the table.

Here's a detailed justification:

The Allow configuration Application Access option on a ServiceNow table determines the extent to which applications outside the scope of the application owning the table can interact with and modify the table's configuration. When this option is selected, it essentially grants broader access for customization by other applications. It does not mean that only the in-scope application can create business rules.

Specifically, allowing configuration enables out-of-scope applications to create Business Rules that operate on that table. These rules can run independently of the original application's scope and perform actions based on events occurring on the table.

Option A is incorrect because the Allow configuration option explicitly lets out-of-scope applications create Business Rules. Restricting Business Rule creation solely to the in-scope application would be the default behavior without this option enabled or with restricted settings.

Option B is incorrect because the Allow configuration setting does not relate to user roles; instead, it's tied to application scopes and access control between applications. Application security is enforced at the application level, where elevated roles are required to modify the application's scripts. Standard users with the application's user role are generally restricted from script modification, unless specifically granted access.

Option D is incorrect because Allow configuration deals with configuring existing tables, not adding new tables to a scoped application. Adding new tables would typically be governed by other permissions and design considerations within the application's scope, often requiring administrative or developer roles within the scope.

In essence, the Allow configuration option provides a mechanism for controlled access to the configuration of a table from outside the scope of the application that defined it, empowering other applications to interact with and extend the table's functionality.

For further research and authoritative references, consider the following:

ServiceNow Docs: Search for "Application Access Settings" or "Scoped Application Access" on the ServiceNow documentation site (https://docs.servicenow.com/) to find detailed explanations of how these settings work.
ServiceNow Developer Site: Explore the ServiceNow Developer Site (https://developer.servicenow.com/) for tutorials, articles, and best practices related to application development and security.

Question: 16

When working in the Form Designer, configuring the label of a field in a child table changes the label on which table(s)?

A.base table
B.child table
C.parent table
D.all tables

Answer: B

Explanation:

The correct answer is **B. child table**.

When you modify the label of a field directly within the Form Designer while working on a child table, the change is localized to that specific child table. The Form Designer allows granular control over UI elements within the context of the current table. The modification only changes how the field label is displayed on the child table's forms and lists. The base table or any parent tables remain unaffected.

The ServiceNow platform utilizes a hierarchical table structure. Changes made to a child table typically do not propagate upwards to parent tables unless explicitly configured to do so through mechanisms such as dictionary overrides or business rules. Without such explicit configurations, modifications are contained within the scope of the table where they were made. Consequently, altering the label of a field in the Form Designer of a child table specifically changes the label presentation for that child table's records and views, preserving the integrity and configuration of parent and base tables. This controlled inheritance and modification capability is fundamental to ServiceNow's application development framework, allowing for tailored user experiences on different tables while maintaining a unified data structure at the base level.

For further reading, consult the ServiceNow documentation on table relationships and form design:

Table Administration:https://developer.servicenow.com/devportal/$glide.home.do (Navigate to the Tables section after logging in)
Form Designer:https://developer.servicenow.com/devportal/$glide.home.do (Navigate to the Forms section after logging in)

Question: 17

Which one of the following is true?

A.A UI Policy's Actions execute before the UI Policy's Scripts
B.The execution order for a UI Policy's Scripts and Actions is determined at runtime
C.A UI Policy's Scripts execute before the UI Policy's Actions
D.A UI Policy's Actions and Scripts execute at the same time

Answer: A

Explanation:

A. A UI Policy's Actions execute before the UI Policy's Scripts.

In ServiceNow, when a UI Policy runs:

The UI Policy condition is evaluated.

If the condition is true, the UI Policy Actions (like setting fields visible, read-only, or mandatory) are executed first.

After that, any associated UI Policy Script runs.

Question: 18

Here is the Business Rule script template:

```
(function executeRule (current, previous */null when async*/) {

    }) (current, previous);
```

This type of JavaScript function is known as:

A.Constructor
B.Scoped
C.Anonymous
D.Self-invoking

Answer: D

Explanation:

D. Self-invoking .

A function that runs immediately after being defined, e.g., (function() ...)();

Question: 19

Which method call returns true only if the currently logged in user has the catalog_admin role and in no other case?

A.g_user.hasRole('catalog_admin')
B.g_user.hasRoleExactly('catalog_admin')
C.g_user.hasRoleOnly('catalog_admin')
D.g_user.hasRoleFromList('catalog_admin')

Answer: B

Explanation:

The correct answer is **B. g_user.hasRoleExactly('catalog_admin')**.

g_user is a GlideUser object, a server-side API in ServiceNow that allows you to retrieve information about the currently logged-in user. Each method of g_user has a specific purpose when checking user roles.

g_user.hasRole('catalog_admin') returns true if the user has the 'catalog_admin' role or any role that contains 'catalog_admin' (including inherited roles from groups they belong to). It does not check exclusively for 'catalog_admin'.
g_user.hasRoleExactly('catalog_admin') returns true only if the user has the 'catalog_admin' role and no other roles. This perfectly fits the question's requirement. It performs an exact match.
g_user.hasRoleOnly('catalog_admin') is not a valid or recognized method in the GlideUser API. It is likely a distractor.
g_user.hasRoleFromList('catalog_admin') is also not a standard GlideUser method. A variant exists that takes a list of roles as input and returns true if the user has at least one of the roles in the list, but this is unrelated to the question's requirement.

Therefore, g_user.hasRoleExactly('catalog_admin') ensures that the logged-in user possesses only the 'catalog_admin' role, fulfilling the "in no other case" condition. This method accurately reflects the scenario presented, making it the correct and intended answer. This is useful in scenarios where you want the user to have only specific permissions and nothing beyond that.

Authoritative Links:

ServiceNow Docs - GlideUser API:
https://developer.servicenow.com/devportal/$DOXP_SCRIPTING_INCLUDES.htm

Question: 20

There is a basic strategy when creating a Utils Script Include. Identify the step that does not belong.

A.Identify the table
B.Script the function(s)
C.Create a class
D.Create a prototype object from the new class

Answer: A

Explanation:

The question asks which step is not part of a basic strategy when creating a Utils Script Include. The correct answer is A, "Identify the table."

Here's why: Utils Script Includes are designed to contain reusable functions that can be used across multiple scripts and applications. They are intended to be table-agnostic. Their purpose is to provide general utility functions, like string manipulation, date calculations, or complex logic, that don't directly depend on the data or structure of a specific ServiceNow table.

Steps B, C, and D are all fundamental to creating a well-structured Utils Script Include. "Script the function(s)" (B) is the core purpose: defining the reusable logic. "Create a class" (C) is the standard ServiceNow best practice for organizing script includes, especially Utils. "Create a prototype object from the new class" (D) is also crucial; it makes the functions within the script include accessible for use in other scripts. This instantiation provides a mechanism to call and reuse the functions defined within the Script Include.

Identifying a specific table (A) contradicts the utility nature of these script includes. If a script include is tied to a specific table, it should instead be a business rule, client script, or another appropriate type of script more closely tied to that table's context. Utils are intended for global, reusable functionality.

In summary, table identification isn't a step in creating a utility script include because the function is designed for general use and not specific to a table. Script Includes act as modular blocks of reusable code, promoting efficiency and maintainability within the ServiceNow platform.

Relevant resources:

ServiceNow Docs on Script Includes: https://developer.servicenow.com/dev.do#!/learn/learning-plans/quebec/new_to_servicenow/app_store_learnv2_scripting_quebec/concept/app_store_script_include
Community article discussing best practices: https://community.servicenow.com/community?id=community_article&sys_id=62c8f1a1dbb290d066f1fff8ca9619b5

Question: 21

Which roles grant access to source control repository operations such as importing applications from source control, or linking an application to source control?
(Choose two.)

A.source_control
B.source_control_admin
C.admin
D.git_admin

Answer: AC

Explanation:

The correct answer is A and C: source_control and admin. Here's a detailed justification:

The ServiceNow platform uses roles to control access and permissions. Managing source control operations like importing applications from source control or linking an application to source control requires specific privileges.

The source_control role directly grants the user the ability to perform basic operations on the source control integration. This generally includes the capability to commit changes, update from the repository, and view the status of the source control repository linked to the application. It enables developers to work with source control without granting full administrative rights to the ServiceNow instance.

The admin role in ServiceNow is a superuser role that grants almost unrestricted access to the entire platform. This role inherently includes the ability to manage source control configurations and perform all operations related to integrating applications with source control repositories. This role bypasses all other role restrictions.

The source_control_admin role (option B) does not exist as a standard role within ServiceNow. While you can create custom roles, this question specifically references standard, out-of-the-box roles.

The git_admin role (option D) is also not a standard ServiceNow role. While one might expect a role with this name to manage Git integrations, ServiceNow does not provide this role by default.

Therefore, the source_control role provides specific access to source control operations, and the admin role grants global access, including source control. These two roles collectively cover the necessary permissions to import applications from source control and link applications to source control. The other two roles, source_control_admin and git_admin, are not standard ServiceNow roles.

Relevant documentation:

ServiceNow Docs on Roles: https://docs.servicenow.com/en-US/bundle/quebec-platform-administration/page/administer/roles/concept/c_Roles.html
ServiceNow Docs on Source Control Integration: https://docs.servicenow.com/en-US/bundle/quebec-application-development/page/build/system-applications/concept/source-control-integration.html

Question: 22

When configuring the content of an Email Notification, which syntax should be used to reference the properties of an event triggering the Notification?

A.$ event.<property name>
B.$ current.<property name>
C.$ <property name>.getDisplayValue()
D.$ gs.<property name>

Answer: A

Explanation:

The correct syntax for referencing event properties within a ServiceNow email notification triggered by an event is $ event.<property name> . This syntax allows you to directly access parameters passed when the event was fired.

Here's why the other options are incorrect:

$ current.<property name> : This syntax refers to properties of the current record on which the notification is being triggered. It's relevant for notifications based on record changes (e.g., incident updates), but not specifically for event-driven notifications.
$ <property name>.getDisplayValue() : This syntax also relates to record properties, and getDisplayValue() is used to retrieve the display value of a field, not event parameters. It is associated with accessing values from a GlideRecord object.
$ gs.<property name> : gs (GlideSystem) provides access to various system-level methods and properties, but it's not the correct way to access event parameters.

When an event is fired in ServiceNow, you can pass parameters along with it. These parameters can then be referenced in an email notification triggered by that event using the $ event.<property name> syntax. For example, if an event named 'u_alert' is fired with parameters 'user' and 'message', you would access them in the notification as $ event.parm1 (user) and $ event.parm2 (message). These parameters are automatically mapped to the event object during the notification processing. The parm1 and parm2 parameters of an event correspond directly with what becomes accessible via $ event.parm1 and $ event.parm2 within the notification script.

Therefore, the most accurate and direct way to reference the properties associated with the event triggering the email notification is by using $ event.<property name> . This ensures that the correct data from the triggering event is utilized within the email content. It is a critical component of making event-driven email notifications dynamic and informative. This syntax is supported natively within ServiceNow's email engine when handling event-driven notifications.

Authoritative Links:

ServiceNow Documentation on Events: https://developer.servicenow.com/dev.do#!/learn/learning-plans/quebec/new_to_servicenow/app_store_learnv2_devstudio_quebec_application_foundations/app_store_learn
ServiceNow Documentation on Notifications: https://docs.servicenow.com/bundle/vancouver-platform-administration/page/administer/notification/concept/c_Notifications.html

Question: 23

Which one of the following is true for a Script Include with a Protection Policy value of Protected?

A.Any user with the protected_edit role can see and edit the Script Include
B.The Protection policy option can only be enabled by a user with the admin role
C.The Protection Policy is applied only if the glide.app.apply_protection system property value is true
D.The Protection Policy is applied only if the application is downloaded from the ServiceNow App Store

Answer: D

Explanation:

The correct answer is **D. The Protection Policy is applied only if the application is downloaded from the ServiceNow App Store.**

Here's a detailed justification:

ServiceNow's Application Protection mechanism provides a way to safeguard the intellectual property of applications distributed through the ServiceNow App Store. A Script Include marked as "Protected" is primarily intended to restrict access and modification once an application is installed from the App Store.

Why A is incorrect: While users with specific roles might have broader access, the core purpose of protection is to limit modification. The protected_edit role doesn't automatically bypass the protection applied by the App Store.

Why B is incorrect: The ability to set the Protection Policy is typically tied to application scoping and developer roles within the application scope, not exclusively to the admin role. Application developers with appropriate rights within their scoped application can set the Protection Policy.

Why C is incorrect: The glide.app.apply_protection property controls the enforcement of application scopes in general. It's not directly related to the specific protection applied by the App Store Protection Policy. That is enforced once an App from the App Store is installed.

Why D is correct: The Protection Policy is specifically designed for applications distributed via the ServiceNow App Store. When an application is downloaded and installed, the Protection Policy restricts modifications, hiding implementation details from the customer. This helps ensure the integrity and functionality of the application as designed by the vendor. The scripts are locked from modifications unless the vendor takes action.

In essence, the Protection Policy is a gatekeeper that locks down specific aspects of an application to safeguard the vendor's IP and ensure that the application behaves as intended in a customer's instance.Here's an authoritative link that confirms this information:https://developer.servicenow.com/devportal/$glide.document.getrecord.do?sys_id=4a4a5d590b05520030b3638482673a08https://docs.servicenow.com/bundle/utah-application-development/page/build/applications/concept/application_protection.html

Question: 24

Which one of the following is true for GlideUser (g_user) methods?

A.Can be used in Client Scripts and UI Policies only
B.Can be used in Business Rules only
C.Can be used in Client Scripts, UI Policies, and UI Actions
D.Can be used in Business Rules, and Scripts Includes

Answer: C

Explanation:

The correct answer is C: Can be used in Client Scripts, UI Policies, and UI Actions.

GlideUser (g_user) is a client-side API in ServiceNow providing access to information about the currently logged-in user. Its methods allow scripts to interact with user-specific details like roles, user ID, name, and location. Because it exposes user-specific data for the client (the web browser), it is designed to be usable in client-side scripting environments.

Client Scripts, UI Policies, and UI Actions all execute on the client-side. Client Scripts respond to events happening in forms and lists, modifying the user interface behavior. UI Policies control the visibility and editability of fields based on conditions, also operating on the client-side. UI Actions, such as buttons and

context menu items, when configured to be client-side, execute JavaScript in the browser.

Therefore, g_user's client-side nature makes it perfectly suited for use within these three client-side scripting contexts. Business Rules and Script Includes, however, are server-side. Using the client-side g_user object on the server side would not work, as the user's browser context and its associated information are unavailable to the server. Business Rules run on the server in response to database events, and Script Includes define reusable server-side functions. G_user is primarily designed for modifying UI appearance or client behavior based on the currently logged in user.

For additional information, refer to the ServiceNow documentation on GlideUser (g_user): https://developer.servicenow.com/dev.do#!/reference/api/sandiego/client/GlideUser and Client-Side Scripting documentation.

Question: 25

When configuring a module, what does the Override application menu roles configuration option do?

A.Users with the module role but without access to the application menu access the module

B.Self-Service users can access the module even though they do not have roles

C.Admin is given access to the module even if Access Controls would ordinarily prevent access

D.Users with access to the application menu can see the module even if they don't have the module role

Answer: A

Explanation:

The correct answer is A: Users with the module role but without access to the application menu access the module. This option controls how roles associated with the module interact with the application menu's roles.

When the "Override application menu roles" option is selected, the roles specified directly on the module take precedence over the roles associated with the application menu itself. This means a user who does not have a role that grants access to the application menu can still access the module if they have the role(s) specifically defined on the module configuration. This enables more granular control over access.

Without this override, users must possess a role granting access to the application menu and possess the role specified on the module to access the module. In essence, it acts like an "AND" condition. With the override, the module's roles become the primary gatekeeper, potentially bypassing the application menu's role requirements.

This override is useful when you want to provide access to a specific module to a particular user group (defined by a role) without granting them wider access to the entire application through the application menu.

It promotes the principle of least privilege, ensuring users only have the minimum necessary permissions to perform their tasks.

Options B, C, and D are incorrect. B describes a situation that would likely require scripting or other configuration, not just a checkbox setting on the module. C is about bypassing ACLs, which is an entirely different security mechanism. D is the opposite of the correct behavior; without the override, this is closer to the true scenario.

For more detailed information about roles and modules in ServiceNow, consult the official ServiceNow documentation:

ServiceNow Documentation: Modules:https://docs.servicenow.com/en-US/bundle/sandiego-platform-user-interface/page/use/navigation/concept/c_ApplicationNavigator.html

ServiceNow Documentation: Roles:https://docs.servicenow.com/en-US/bundle/sandiego-platform-administration/page/administer/security/concept/c_Roles.html

Question: 26

Which platform feature can be used to determine the relationships between field in an Import Set table to field in an existing ServiceNow table?

A.Business Service Management Map
B.Data Sources
C.Transform Map
D.CI Relationship Builder

Answer: C

Explanation:

The correct answer is C, Transform Map. A Transform Map in ServiceNow is the mechanism by which data from an Import Set table is mapped and transformed into the target ServiceNow table's fields. It defines the relationships between fields in the source (Import Set) and the destination table (e.g., Incident, User).

Data Sources (B) are primarily used to define where the data is coming from (e.g., a file, a JDBC connection), but they don't define the field-level relationships. They are a prerequisite for using Transform Maps.

Business Service Management (BSM) Map (A) visually displays the relationships between business services and the underlying infrastructure. It helps understand service dependencies and doesn't play a role in defining field mappings during data imports.

The CI Relationship Builder (D) is used to establish and visualize relationships between Configuration Items (CIs), which are components of IT infrastructure. While it's about relationships, it's not related to mapping data during import processes.

A Transform Map allows specifying field mappings, specifying field transformations via scripting, and handling coalescing (identifying existing records to update versus creating new ones). It allows users to define how the data should be transformed from the staging Import Set table into the desired ServiceNow table. This is crucial for ensuring data accuracy and integrity during the import process. Without the Transform Map, the data from the Import Set table would not know where to write to in the ServiceNow platform.

For more information, refer to the official ServiceNow documentation:

Transform Maps
Data Sources

Question: 27

When configuring a REST Message, the Endpoint is:

A.The commands to the REST script to stop execution
B.The URI of the data to be accessed, queried, or modified
C.Information about the format of the returned data
D.The response from the provider indicating there is no data to send back

Answer: B

Explanation:

The correct answer, B, emphasizes the core function of the Endpoint field within a ServiceNow REST Message. Let's break down why:

REST (Representational State Transfer) is an architectural style for building networked applications. REST relies on stateless, client-server communication. A crucial aspect of RESTful interactions is specifying where to find the resources you want to interact with. That "where" is the URI (Uniform Resource Identifier).

Within ServiceNow, a REST Message allows you to easily integrate with external systems via REST APIs. The Endpoint field in a REST Message configuration is specifically designed to hold the URI. This URI points to the resource you are targeting on the external system.

Think of it like a postal address. The Endpoint URI tells ServiceNow exactly where to send its request (to access, query, or modify data). Without a valid and accurate Endpoint URI, the REST Message won't know where to send the request and the integration will fail.

Option A, "The commands to the REST script to stop execution," describes a scripting control mechanism, not a URI. Option C, "Information about the format of the returned data," relates to the content type (e.g., JSON, XML) which is handled separately. Option D, "The response from the provider indicating there is no data to send back," is a response from the external system, not something you configure before sending the request.

In essence, the Endpoint defines the "address" of the REST API, ensuring that your ServiceNow instance can connect to and interact with the appropriate resources on the target system. Without this address, the communication can't be established.

Authoritative Links for Further Research:

ServiceNow Docs on REST API Integration:
https://developer.servicenow.com/devportal/learn/courses/quebec/rest_api_integration/rest_api_integration_introd **REST Architectural Style:**https://restfulapi.net/

Question: 28

When evaluating Access Controls, ServiceNow searches and evaluates:

A.Only for matches on the current table
B.Only for matches on the current field
C.From the most specific match to the most generic match
D.From the most generic match to the most specific match

Answer: C

Explanation:

The correct answer is C: From the most specific match to the most generic match. ServiceNow's Access Control List (ACL) evaluation process prioritizes specificity. This means the system first looks for ACL rules that are very precisely tailored to the specific table and field being accessed. If no direct match exists, it then broadens the search to include rules applying to the parent table and the (any field) or specific field. If still no match is found, the system continues moving up the table hierarchy to more generic ACLs. This "most specific to most generic" approach ensures granular control over data access. For instance, an ACL rule on *incident.priority* would take precedence over one on just *incident.*. This ensures that specific rules override more general ones, allowing for fine-grained permission management. This approach allows for efficient

management of permissions in complex data models. By moving from specific to generic rules, organizations can grant broad access by default while creating granular exceptions as needed, ensuring both usability and security. The system ultimately checks for an ACL rule with the table name and * which acts as a fallback if no specific field rules were found. The system uses the highest role requirement out of all rules if more than one rule is evaluated to grant access.

Further research:

ServiceNow Docs on Access Control Lists: https://docs.servicenow.com/bundle/utopia-platform-administration/page/administer/security/concept/access-control-rules.html
ServiceNow Guru on Access Control: https://www.servicenowguru.com/scripting/security-scripting/

Question: 29

In a Business Rule, which one of the following returns true if the currently logged in user has the admin role?

A.g_form.hasRoleExactly('admin')
B.gs.hasRole('admin')
C.g_form.hasRole('admin')
D.gs.hasRoleExactly('admin')

Answer: B

Explanation:

The correct answer is B, gs.hasRole('admin'). Let's break down why:

gs vs. g_form: In a Business Rule, gs refers to the GlideSystem API, the server-side scripting API. g_form, on the other hand, is a client-side API primarily used for manipulating the form interface. Since role checks often need to be done on the server side to securely determine access and behavior, gs is the appropriate object.

hasRole() vs. hasRoleExactly():hasRole() returns true if the user has the specified role or any role that contains the specified role. For example, if a user has the admin role, or a role like itil_admin, hasRole('admin') will return true. hasRoleExactly() only returns true if the user has exactly the specified role and no other roles are considered. In most cases, the broader check using hasRole() is what's intended for checking elevated privileges.

Why gs.hasRole('admin') is best: This method efficiently determines whether the current user possesses the 'admin' role, or any role that includes 'admin'. This is vital for enforcing security policies and controlling access to sensitive operations within the ServiceNow platform. It operates server-side, providing a reliable mechanism for verifying administrative privileges before executing privileged code.

Why other options are incorrect:

g_form.hasRole('admin') and g_form.hasRoleExactly('admin') are incorrect because g_form is a client-side object and role checks should occur on the server-side for security.
gs.hasRoleExactly('admin') is less likely to be the desired approach because it requires the user to only have the 'admin' role, which is too restrictive in most practical scenarios where users may have other roles in addition to admin.

Authoritative Links:

GlideSystem API (gs):https://developer.servicenow.com/devportal/reference/api/glide-system/index.html
GlideForm (g_form):https://developer.servicenow.com/devportal/reference/api/client/GlideForm/

Question: 30

From the list below, identify one reason an application might NOT be a good fit with ServiceNow. The application:

A.Needs workflow to manage processes
B.Requires as-is use of low-level programming libraries
C.Requires reporting capabilities
D.Uses forms extensively to interact with data

Answer: B

Explanation:

The correct answer is B. Requires as-is use of low-level programming libraries. Here's why:

ServiceNow is a high-productivity platform-as-a-service (PaaS) designed to rapidly develop and deploy applications using a low-code/no-code approach. It abstracts away much of the underlying infrastructure and complex coding typically associated with application development. ServiceNow prioritizes configuration over custom coding.

Option A, "Needs workflow to manage processes," aligns well with ServiceNow's core strengths. ServiceNow is renowned for its robust workflow engine, enabling the automation and management of complex business processes.

Option C, "Requires reporting capabilities," is also a good fit. ServiceNow provides extensive reporting and analytics capabilities, allowing users to track performance, identify trends, and make data-driven decisions.

Option D, "Uses forms extensively to interact with data," is also a suitable scenario for ServiceNow. ServiceNow's platform is designed around forms-based data entry and management.

However, Option B, "Requires as-is use of low-level programming libraries," indicates a need for functionalities that are more easily implemented outside the purview of the ServiceNow platform. If an application's functionality is intimately tied to specific, unchangeable low-level libraries (e.g., complex, heavily optimized mathematical routines or device driver interactions), integrating it with ServiceNow's abstracted environment becomes difficult and counterproductive. ServiceNow applications are written in JavaScript on the client-side and server-side, and do not allow direct, low-level library integration. The emphasis on low-code development means that developers do not have the freedom to directly import and utilize these external dependencies. Thus the application would be a poor fit.

Further Reading:

ServiceNow Documentation:https://docs.servicenow.com/ (Search for "low-code platform," "workflow," "reporting," "forms")
PaaS Overview:https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-paas/

Question: 31

Identify the incorrect statement about Delegated Development in ServiceNow.

A.Administrators can grant non-admin users the ability to develop global applications.
B.Administrators can specify which application file types the developer can access.
C.Administrators can grant the developer access to script fields.

D.Administrators can grant the developer access to security records.

Answer: A

Explanation:

The incorrect statement is A: "Administrators can grant non-admin users the ability to develop global applications." Here's why:

Delegated Development in ServiceNow is a feature that allows administrators to grant specific development rights to non-admin users. This enables a division of labor and empowers business users to contribute to application development without requiring full administrative access.

Administrators have fine-grained control over what delegated developers can do. Statement B is correct because administrators can specify which application file types (e.g., business rules, client scripts, UI policies) a developer can access. This is crucial for controlling the scope of development and ensuring that developers only modify files relevant to their assigned tasks.

Statement C is also correct. Administrators can grant delegated developers access to script fields. This allows them to write and modify server-side or client-side scripts within the defined application scope and file types. Script access is essential for implementing custom logic and functionality.

Statement D is also correct. Administrators can grant developers access to security records such as ACLs (Access Control Lists) that govern data access and system security. This option should be used cautiously but might be necessary for developers working on specific security-related aspects of an application.

However, delegated developers are not typically granted the ability to create or modify global applications directly. Global applications operate outside of the application scope boundaries and require full administrative privileges for proper management. Allowing non-admin users direct control over the global scope could pose security risks and destabilize the entire ServiceNow instance. Instead, the purpose of delegated development is to empower scoped application development. Delegated developers work within specific applications that are scoped and well-defined. Therefore, statement A is the incorrect one because it contradicts the fundamental principle of delegated development, which aims to limit access and control within predefined application boundaries, not provide access to the global scope.

Further Reading:

ServiceNow Docs - Delegated Development:https://docs.servicenow.com/bundle/vancouver-platform-administration/page/administer/delegated-developers/concept/c_DelegatedDevelopment.html

Question: 32

One of the uses of the ServiceNow REST API Explorer is:

A.Practice using REST to interact with public data providers
B.Find resources on the web for learning about REST
C.Convert SOAP Message functions to REST methods
D.Create sample code for sending REST requests to ServiceNow

Answer: D

Explanation:

The correct answer is D, "Create sample code for sending REST requests to ServiceNow." The ServiceNow REST API Explorer is a powerful tool within the ServiceNow platform specifically designed to facilitate

interaction with ServiceNow's REST APIs. Its primary function isn't about interacting with external public data providers (A), finding general REST resources (B), or converting SOAP to REST (C). Instead, it allows developers to explore available ServiceNow APIs, understand their parameters and expected responses, and, crucially, generate ready-to-use code snippets in various programming languages (like JavaScript, Python, cURL, etc.). These snippets demonstrate how to construct and send REST requests to ServiceNow instances, making it significantly easier for developers to integrate external applications and systems with ServiceNow. By generating this sample code, the API Explorer greatly accelerates the development process and reduces the learning curve associated with implementing ServiceNow REST integrations. This functionality is critical for developing applications and workflows that leverage the power of ServiceNow.

Further research:

ServiceNow REST API Explorer Documentation
ServiceNow REST API Guide

Question: 33

Which one of the following is true regarding Application Scope?

A.All applications are automatically part of the Global scope
B.Applications downloaded from 3 party ServiceNow application developers cannot have naming conflicts rd
C.Any developer can edit any application
D.Developers can choose the prefix for a scope's namespace

Answer: B

Explanation:

The correct answer is B: Applications downloaded from 3rd party ServiceNow application developers cannot have naming conflicts. Here's why:

Application scoping in ServiceNow is a crucial security and governance feature designed to prevent naming collisions and maintain the integrity of the platform. When an application is created within a specific scope, ServiceNow enforces a unique namespace based on a prefix determined by ServiceNow. This prefix helps distinguish applications and their components from those in other scopes, including the Global scope and other custom scopes.

The primary benefit of scoping is isolation. By isolating applications, ServiceNow prevents custom scripts and configurations from interfering with other applications or the core ServiceNow platform. This prevents unintended consequences arising from different applications using the same name for a table, script, or other component. Third-party applications obtained through the ServiceNow Store are inherently scoped, thereby avoiding naming conflicts with existing applications within your instance. The scoping mechanism ensures that even if two different developers use the same name for an object, such as a field, the ServiceNow platform can still distinguish them based on their respective scopes.

Option A is incorrect because only legacy applications or applications specifically intended to have broad access reside in the Global scope. Modern ServiceNow development strongly encourages scoped applications for better maintainability and security. Option C is also incorrect. Scoped applications limit the ability to directly edit applications across scopes. Only those with proper roles or within the application's scope have edit access. Option D is incorrect because the prefix for a scope's namespace is automatically assigned by ServiceNow.

In summary, application scoping eliminates naming conflicts, especially for applications downloaded from third-party developers. It helps maintain stability and predictability within the ServiceNow platform.

Reference:

ServiceNow Docs: Application Scoping
ServiceNow Docs: Scoped Applications

Question: 34

Which one of the following is the baseline behavior of a table in a privately-scoped application?

A.The table and its data are not accessible using web services
B.Any Business Rule can read, write, delete, and update from the table
C.Only artifacts in the table's application can read from the table
D.All application scopes can read from the table

Answer: D

Explanation:

The correct answer is **D. All application scopes can read from the table**.

Here's a detailed justification:

In ServiceNow, application scoping provides a mechanism to encapsulate and protect application data and logic. However, baseline behavior leans towards openness, especially for reading data. Privately-scoped applications do restrict write access by default, enhancing data integrity. But read access is generally allowed from all application scopes. This design choice facilitates cross-application functionality and integration if desired, while still providing isolation at the write level.

Option A is incorrect because web services can be explicitly configured to access scoped tables, contradicting the statement that they are inaccessible. Option B is incorrect because application scoping by default, prevents Business Rules from other scopes from directly reading, writing, or deleting data in a privately-scoped table. Option C is not entirely accurate; while it's true that artifacts in the table's application can read the table, the baseline behavior allows other scopes to read as well.

Therefore, the most accurate description of the baseline behavior of a table in a privately-scoped application is that all application scopes can read from the table. This promotes data sharing capabilities when needed while still maintaining a reasonable level of isolation through write restrictions. The read access can later be further restricted by implementing appropriate Access Controls Lists (ACLs).

For further research, consult the official ServiceNow documentation on application scoping and table access:

ServiceNow Application Scoping
Controlling Table Access with Application Scopes

Question: 35

Which one of the following is NOT a purpose of application scoping?

A.Provide a relationship between application artifacts
B.Provide a way of tracking the user who developed an application
C.Provide a namespace (prefix and scope name) to prevent cross application name collisions
D.Provide controls for how scripts from another scope can alter tables in a scoped application

Answer: B

Explanation:

The correct answer is B, "Provide a way of tracking the user who developed an application." Application scoping in ServiceNow primarily addresses application isolation, namespace management, and controlled access to data and functionality between different applications.

Option A is correct because scoping establishes relationships between different components (artifacts) within an application. This allows developers to understand dependencies and manage the application as a cohesive unit.

Option C is correct because application scoping creates a unique namespace (identified by a prefix and scope name) for an application. This ensures that naming conflicts are avoided when multiple applications are installed on the same ServiceNow instance, preventing one application from inadvertently overwriting or interfering with another.

Option D is also correct because application scoping allows administrators to define how scripts from other applications can interact with tables and data within a scoped application. This provides granular control over data security and integrity, preventing unauthorized access or modification. Cross-scope access is a key element of application scoping that defines rules and permissions for inter-application communication.

Option B, on the other hand, is incorrect. While ServiceNow does track user activity and changes within the system through update sets and history logs, application scoping is not the primary mechanism for tracking the developer of an application. Developer information is typically captured within the update set created by the developer, as well as through system logs and auditing features. Scoping is more about functional isolation and preventing namespace collisions than direct user tracking for development purposes. The system tracks who modified records, but this is independent of the scoping mechanism itself.

Further research can be done here:

ServiceNow Documentation on Scoped Applications:
https://developer.servicenow.com/devportal/devdb/topic?id=scoped_applications
ServiceNow Documentation on Cross-Scope Access:
https://developer.servicenow.com/devportal/devdb/topic?id=r_CrossScopePrivilege

Question: 36

It is best practice to define the business requirements and the process(es) an application will manage as part of the application development plan. What are some of the considerations to document as part of the business process?

A.Business problem, data input/output, users/stakeholders, and process steps
B.Business problem, data input/output, project schedule, and process steps
C.Business problem, data input/output, users/stakeholders, and database capacity
D.Business problem, users/stakeholders, available licenses, and database capacity

Answer: A

Explanation:

The best answer is **A. Business problem, data input/output, users/stakeholders, and process steps**.

A robust business process documentation, crucial for successful application development, needs to explicitly outline the core problem the application aims to solve. Understanding the pain points clarifies the

application's purpose and scope. Equally important is detailing the data flow: what data is fed into the system (input) and what the system produces as a result (output). This data specification informs data modeling and integration requirements. Identifying all users and stakeholders ensures that the application caters to their needs and roles, considering permissions and user experience. Finally, meticulously documenting each step of the process, from initiation to completion, is essential for guiding application design, development, and testing. This process mapping facilitates automation and optimization.

Option B includes "project schedule," which, while important for overall project management, isn't directly a core component of the business process itself. The schedule influences how the application is built but doesn't define what the application does. Option C mentions "database capacity," which is a technical consideration handled separately, not part of the process definition. Option D replaces "data input/output" and "process steps" with "available licenses" and "database capacity," both falling under technical implementation details rather than components that thoroughly define business processes.

Therefore, Option A encapsulates the most critical considerations needed when documenting business processes during application development planning, focusing on understanding and documenting the essential elements of the process itself. The other options mix important project considerations with the specific elements needed to define a business process.

Supporting documentation:

ServiceNow Documentation: Application Development: Although generic, ServiceNow application development training outlines the importance of understanding the customer's process before initiating development.

Business Process Management Body of Knowledge (BPM CBOK): provides a comprehensive overview of business process management principles. Although this documentation is not ServiceNow specific, the same principles of documenting business processes apply when developing applications on the platform.

Question: 37

Which of the following statements does NOT apply when extending an existing table?

A.The parent table's Access Controls are evaluated when determining access to the new table's records and fields

B.The new table inherits the functionality built into the parent table

C.The new table inherits all of the fields from the parent table

D.You must script and configure all required behaviors

Answer: D

Explanation:

The correct answer is **D. You must script and configure all required behaviors.**

Here's why:

When you extend a table in ServiceNow, you're creating a child table that inherits properties and behaviors from its parent table. This inheritance significantly reduces the amount of scripting and configuration needed for the new table.

A. The parent table's Access Controls are evaluated when determining access to the new table's records and fields: This is correct. ServiceNow's Access Control Lists (ACLs) are hierarchical. When accessing data in an extended table, the system checks both the child table's ACLs and the parent table's ACLs. If the parent table's ACL denies access, access will be denied, regardless of the child table's ACLs.

B. The new table inherits the functionality built into the parent table: This is also correct. The extended table automatically inherits the business rules, client scripts, UI policies, and other functionalities defined on the parent table. This is a core benefit of table extension, enabling code reuse and consistency.

C. The new table inherits all of the fields from the parent table: This is a fundamental aspect of table extension. The child table automatically includes all the columns (fields) defined in the parent table. You can then add new fields specific to the child table, but you don't need to redefine the parent's fields.

D. You must script and configure all required behaviors: This statement is incorrect. While you might need to add custom scripting and configurations specific to the new table's unique requirements, you don't have to script and configure all behaviors. The inheritance from the parent table provides a significant head start, automatically providing many behaviors already. The purpose of extending a table is to take advantage of the existing behaviors and fields of the parent table and just add or modify things as necessary.

In essence, extending a table promotes code reuse and simplifies development by leveraging the existing structure and functionality of the parent table. You should configure the behaviors, but you are not required to re-script all behaviors.

References:

ServiceNow Docs: Table Administration
ServiceNow Community: Extending Tables

Question: 38

Which of the following CANNOT be debugged using the Field Watcher?

A.Business Rules
B.Script Includes
C.Client Scripts
D.Access Controls

Answer: B

Explanation:

The correct answer is B. Script Includes.

Field Watcher is a debugging tool in ServiceNow specifically designed for monitoring the behavior of fields on a form or record. It allows developers to track changes to field values and understand how client-side scripts (like Client Scripts) and server-side scripts (like Business Rules) interact with those fields in real-time. Access Controls (ACLs) define user access permissions on data and are triggered on record read, write, and create operations; their effect on field values can be indirectly observed via the Field Watcher. Client Scripts run in the browser, directly manipulating the form's fields, making them easily monitored. Business Rules execute on the server but often involve setting or modifying field values on a record, changes observable through Field Watcher when the rules are triggered by form interactions.

Script Includes, on the other hand, are reusable server-side script modules that define functions and classes. They are typically called from other scripts (Business Rules, Workflow Activities, etc.) to perform specific tasks. They don't directly interact with or modify fields on a form in a way that Field Watcher can directly observe. Field Watcher monitors changes happening on the form itself. While Script Includes may indirectly cause field changes through the scripts that call them, the Script Include's execution itself isn't monitored, only the result of its operations on the fields if triggered via a business rule or other record action.

Therefore, while you might indirectly observe the effects of a Script Include through other scripts and their

impact on fields, you cannot directly debug the execution of the Script Include itself using Field Watcher. Field Watcher focuses on the context of the record being viewed in the form/list. To debug Script Includes, you would typically use server-side debugging tools like script logging (gs.log) or the ServiceNow script debugger.

For further research, refer to the official ServiceNow documentation on Field Watcher:https://developer.servicenow.com/dev.do#!/learn/courses/kingston/scripting_in_servicenow/scripting_in_s

Question: 39

Which objects can be used in Inbound Action scripts?

A.current and previous
B.current and email
C.current and event
D.current and producer

Answer: B

Explanation:

The correct answer is **B. current and email**.

In ServiceNow Inbound Actions, the current object represents the target record that the inbound email is intended to create or update. It behaves similarly to the current object in other server-side scripts, allowing you to set field values, update existing records, or create new ones based on information extracted from the email. Think of current as a direct handle to the record being processed.

The email object provides access to the inbound email's properties, such as the sender (email.from), subject (email.subject), body (email.body), and attachments (email.attachments). You use this object to extract data from the email and then populate the current record accordingly. For example, you might extract a user's name from the email body and use it to update the current.caller_id field on an incident record.

The previous object, available in many ServiceNow scripting contexts, is not available in Inbound Actions. This object holds the record's state before a change is made, which isn't applicable in the context of processing an incoming email to trigger an action. The event object pertains to events triggered within ServiceNow, and while Inbound Actions can trigger events, the event object itself isn't directly accessible within the script.
Similarly, the producer object is relevant for Service Catalog items and record producers, not inbound email processing.

Therefore, the current object for manipulating the target record and the email object for accessing email properties are essential and the only readily available objects in Inbound Action scripts. These objects enable the core functionality of parsing emails and acting on them within ServiceNow. They allow the application to understand and process the context of the email, acting as a connector between external communication and the application's internal processes.ServiceNow Docs: Inbound Email ActionsServiceNow Community: Inbound Email Scripting

Question: 40

Which one of the following is part of the client-side scripting API?

A.workflow.scratchpad
B.GlideUser object (g_user)
C.current and previous objects
D.GlideSystem object (gs)

Answer: B

Explanation:

The correct answer is B, the GlideUser object (g_user). Client-side scripting in ServiceNow, executed within the user's browser, has specific APIs available for interaction. These APIs allow developers to manipulate the user interface, handle events, and communicate with the server. The GlideUser object (g_user) is a critical part of this client-side API. It provides access to information about the currently logged-in user, such as their user ID, name, roles, and preferences. Using g_user, client-side scripts can personalize the user experience, enforce access controls, and tailor functionality based on user attributes. For example, a script might hide or show a field based on the user's role.

Options A, C, and D are not primarily client-side. workflow.scratchpad is typically used in workflows, which are server-side processes. The current and previous objects are mainly used in business rules, which also run on the server. While it's true they can sometimes be accessible via AJAX calls from the client, they aren't core to the client-side API itself. The GlideSystem object (gs), while having some client-side counterparts like gs.addInfoMessage, is largely a server-side API used for logging, accessing system properties, and other system-level operations. g_user is specifically designed for client-side use, enabling scripts to directly interact with the logged-in user's information without server round trips in many common scenarios. That functionality is pivotal for creating dynamic and responsive user interfaces.

Authoritative Links:

ServiceNow Docs - GlideUser (g_user) - Client:
https://developer.servicenow.com/devportal/devdb/api_docs/api/client/GlideUser/
ServiceNow Docs - Client Script API:
https://developer.servicenow.com/devportal/devdb/api_docs/api/client/c_GlideFormAPI.html

Question: 41

Application developers configure ServiceNow using industry standard JavaScript to`¦

A.Enable the right-click to edit the context menus on applications in the navigator
B.Extend and add functionality
C.Customize the organization's company logo and banner text
D.Configure the outgoing email display name

Answer: B

Explanation:

The correct answer is B: Extend and add functionality. ServiceNow's application development framework heavily relies on JavaScript to customize and enhance the platform's capabilities beyond its out-of-the-box features. Application developers use JavaScript in various areas, including client scripts (running in the user's browser), server-side scripts (running on the ServiceNow instance), business rules, UI policies, and workflow activities. These scripts manipulate data, automate processes, create dynamic user interfaces, and integrate with external systems. While options A, C, and D represent possible configurations within ServiceNow, they don't encapsulate the core, widespread application of JavaScript across the platform's development

landscape. Right-clicking context menu customization often involves UI actions tied to specific tables, not a broad JavaScript-driven mechanism. Customizing logos and banners typically relies on system properties and UI branding tools. While JavaScript could potentially contribute to manipulating email display names, it's not the standard primary method; system properties handle that directly. Fundamentally, application developers use JavaScript to build custom applications, integrate systems, automate complex workflows, and tailor user experiences, all extending the platform's functionality.

Relevant resources:

ServiceNow Developer Site:https://developer.servicenow.com/dev.do - This site offers comprehensive documentation, tutorials, and resources for ServiceNow application development, including JavaScript.

ServiceNow Docs - Scripting:https://docs.servicenow.com/bundle/utopia-platform-administration/page/script/concept/scripting.html - Provides detailed information on various scripting aspects in ServiceNow, including client-side and server-side scripting.

Question: 42

How many applications menus can an application have?

A.3, one for an application's user modules, one for an application's administrator modules, and one for the ServiceNow administrator's modules

B.As many as the application design requires

C.2, one for an application's user modules and one for an application's administrator modules

D.1, which is used for all application modules

Answer: B

Explanation:

The correct answer is B: "As many as the application design requires."

ServiceNow's application development paradigm provides substantial flexibility in structuring applications. Application menus serve as primary navigation points for users to access different modules within an application. ServiceNow doesn't impose a hard limit on the number of application menus an application can possess. Instead, the design of the application and its intended functionality dictates the number of required menus. If an application requires a complex structure with multiple distinct user roles or functionalities, multiple application menus become necessary to organize the various modules effectively. Option A, C, and D are all incorrect because they suggest limited numbers of application menus that are not consistent with ServiceNow's design flexibility. The key concept here is modularity and customizable user experience; each application should be tailored to its specific needs. Having multiple menus allows for better organization of modules for different users.https://developer.servicenow.com/devportal-docs/dev-guide/main-structure/application-files.html

Question: 43

The source control operation used to store local changes on an instance for later application is called a(n) <blank>.

A.Branch

B.Tag

C.Stash

D.Update set

Answer: C

Explanation:

The correct answer is C. Stash.

A stash is a temporary storage area in Git (and by extension, ServiceNow's source control integration, which is based on Git) where you can save changes you've made to your working directory without committing them. This allows you to switch branches, pull in updates, or perform other operations that would be problematic with uncommitted changes. Think of it as a "pause" button for your work-in-progress. You can then "pop" the stash later to restore those changes. Branches, tags, and update sets serve different purposes. A branch is a parallel version of the repository. A tag marks a specific point in the repository's history. Update sets are ServiceNow's older mechanism for capturing and moving configurations and customizations between instances, now largely replaced by source control for application development. The key distinction is the temporary, non-committal nature of a stash, making it perfect for storing local changes temporarily before committing them. The question explicitly asks for an operation to store local changes "for later application," pointing directly to the stash functionality.

Further reading:

Git Stash documentation: https://git-scm.com/docs/git-stash
ServiceNow Source Control Integration:
https://developer.servicenow.com/devportal/devdb/reference/api/glide.sg/sg-API.html (While this doesn't directly discuss stashing in ServiceNow, it details the underlying Git integration.)

Question: 44

What syntax is used in a Record Producer script to access values from Record Producer form fields?

A.producer.field_name
B.producer.variablename
C.current.variable_name
D.current.field_name

Answer: B

Explanation:

B. producer.variablename

In Record Producer scripts, the form fields (variables) are accessed via the producer object.

You use producer.variable_name to get the value entered in that variable.

The current object refers to the record being created or updated after the producer runs, so it’s not used to access the form input values directly in the script.

Question: 45

Which of the following methods prints a message on a blue background to the top of the current form by default?

A.g_form.addInfoMsg()
B.g_form.addInfoMessage()

C.g_form.showFieldMessage()
D.g_form.showFieldMsg()

Answer: B

Explanation:

The correct answer is B, g_form.addInfoMessage(). This method is specifically designed in ServiceNow to display an informational message at the top of the current form. By default, the message appears with a blue background, indicating its informational nature and distinguishing it from error or warning messages.

Option A, g_form.addInfoMsg(), is not a valid ServiceNow API method. ServiceNow API names are case-sensitive and must match exactly. Therefore, this option is incorrect due to a naming discrepancy.

Option C, g_form.showFieldMessage(), is used to display a message associated with a specific field on the form, not at the top of the form. It can be used to highlight errors or provide context relating to particular field values. It does not default to a blue background for general informational messages at the top of the form.

Option D, g_form.showFieldMsg(), similar to option A, is not a recognized or valid ServiceNow API method due to incorrect capitalization and naming convention. This immediately renders it incorrect.

g_form.addInfoMessage() is the standard and recommended method for displaying general informational messages at the top of a ServiceNow form, defaulting to a blue background which visually categorizes the message for users. The other options either target specific fields or are simply invalid method names, making them unsuitable for displaying general information at the top of the form with a blue background by default.

Further information on this method can be found in the ServiceNow documentation: https://developer.servicenow.com/dev.do#!/reference/api/sandiego/client/g_form (refer to the addInfoMessage section).

Question: 46

A scoped application containing Flow Designer content dedicated to a particular application is called a(n):

A.Spoke
B.Bundle
C.Action
D.Flow

Answer: A

Explanation:

Here's a detailed justification for why the correct answer is A (Spoke), and why the other options are less suitable within the context of ServiceNow Flow Designer and scoped applications:

Justification for A (Spoke):

A Spoke in ServiceNow Flow Designer is a scoped application that contains pre-built actions, flows, and subflows designed to automate tasks related to a specific application or integration. Spokes allow developers to package and reuse automation logic, simplifying the development process and promoting consistency across different instances. The key feature of a Spoke is its focus on providing specific functionality for a particular area, making it highly reusable. A spoke allows developers to extend the capabilities of the Now Platform by integrating with external systems or automating application-specific processes. This aligns with

the description in the question, making "Spoke" the best fit. ServiceNow officially uses the term "Spoke" to describe pre-built automation content packaged for a specific application.

Why the other options are incorrect:

B (Bundle): While a bundle can refer to a collection of related items, it's not the specific term used in ServiceNow's Flow Designer for a scoped application containing automation content. Bundles are more commonly associated with packaging software updates or related plugins together.

C (Action): An action is a single, discrete operation within a flow or subflow. It's a building block of automation but not a container for multiple flows and subflows within a scoped application. Actions are smaller units within a Spoke.

D (Flow): A flow is a sequence of actions and conditions that automate a task or process. While a Spoke contains flows, the Spoke itself is the higher-level container for the scoped application, and not the other way around. Flows are components within a Spoke.

Supporting Links:

ServiceNow Documentation: Spokes:https://docs.servicenow.com/bundle/utah-platform-flow-designer/page/administer/flow-designer/concept/spokes.html
ServiceNow Community Article on Spokes:https://community.servicenow.com/community?id=community_article&sys_id=361111a2db12c0506648fb243996190a

These links confirm that "Spoke" is the designated term for a scoped application containing Flow Designer content specific to an application or integration within the ServiceNow ecosystem. They also clarify the distinctions between Spokes, Flows, Actions, and other related concepts in Flow Designer.

Question: 47

What is a Module?

A.The functionality within an application menu such as opening a page in the content frame or a separate tab or window

B.A group of menus, or pages, providing related information and functionality to end-users

C.A way of helping users quickly access information and services by filtering the items in the Application Navigator

D.A web-based way of providing software to end-users

Answer: A

Explanation:

The provided answer, A, is the correct definition of a Module in ServiceNow. A module is a link within an application menu that navigates users to a specific area or functionality within the ServiceNow instance. It represents a distinct element within the application, allowing users to directly access specific forms, lists, reports, or other functionalities. Modules control what appears in the content frame (the main working area) of ServiceNow, or may even open a separate tab or window depending on their configuration.

Option B is incorrect because it describes an Application rather than a module. An application is a collection of modules, menus, and tables providing related information and functionality. Option C alludes to Filters or potentially Favorites, which are user-specific ways to customize navigation but not the inherent definition of a module. Option D speaks more generally about Software as a Service (SaaS), a cloud delivery model that ServiceNow embodies, but not the specific function of a module.

Modules are configurable objects within ServiceNow that link to records, external URLs, UI Pages, or other ServiceNow interfaces. They are a core component of the platform's navigation and application design. Understanding modules is crucial for application developers because they determine how users interact with and access the application's features. Modules allow developers to guide users to the essential parts of the application quickly. The ServiceNow documentation specifically defines modules as navigational links within applications.

For further reading, refer to the official ServiceNow documentation:

ServiceNow Documentation on Modules:https://docs.servicenow.com/bundle/sandiego-platform-administration/page/build-applications/application-administration/concept/c_ApplicationMenuAndModules.html

Question: 48

Which source control operation is available from BOTH Studio and the Git Repository?

A.Create Branch
B.Apply Remote Changes
C.Stash Local Changes
D.Edit Repository Configurations

Answer: A

Explanation:

The question asks which source control operation is available from both ServiceNow Studio and the Git Repository itself. The correct answer is A. Create Branch.

Here's why:

Create Branch: Creating a new branch is a fundamental operation in Git version control. Branches allow developers to work on new features or bug fixes in isolation without affecting the main codebase (e.g., the main or master branch). This operation is core to Git's branching model and can be initiated from both ServiceNow Studio (within the Studio's source control integration) and directly from Git (using command-line or a Git GUI client pointed at the Git repository). It's how independent development streams are initiated.

Apply Remote Changes (Pull): While Studio allows developers to pull remote changes, the actual operation applies them to the local branch. The Git repository itself (on the remote server like GitHub, GitLab, or Azure DevOps) doesn't "apply" changes; it serves as the source of those changes. The application happens on the local machine (either via Studio or Git command-line).

Stash Local Changes: Stashing is a local operation. It temporarily saves changes that are not ready to be committed, allowing developers to switch branches or perform other actions without committing incomplete work. It's primarily a local function. The Git repository only stores commits, branches, and other high-level metadata; stashed changes exist only in the developer's local working directory.

Edit Repository Configurations: Editing repository configurations (like access control, webhooks, or branch protection rules) is typically done directly within the Git repository hosting platform (e.g., GitHub, GitLab, Azure DevOps). While Studio can be configured to point to a specific repository, it doesn't manage the repository's global settings in the same way the repository hosting platform does.

Therefore, branch creation is a core Git operation available directly both through the command line (pointed to the repository) and from Studio's interface. The other operations have a different locality regarding where they are directly available.

Authoritative Links:

Git Branching:https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell
ServiceNow Studio Source Control:
https://developer.servicenow.com/dev.do#!/learn/courses/quebec/app_store_certification/app_store_certification_

Question: 49

Which one of the following is NOT required to link a ServiceNow application to a Git repository?

A.Password
B.URL
C.User name
D.Application name

Answer: D

Explanation:

The correct answer is D (Application name). Linking a ServiceNow application to a Git repository involves authenticating and authorizing ServiceNow to access the repository. This necessitates credentials like a username and password (or a token), and the URL of the Git repository.

Username and password (or token) are essential for authentication. These credentials verify that ServiceNow has the right to access and modify the Git repository. The URL points to the specific location of the repository on platforms like GitHub, GitLab, or Bitbucket. This tells ServiceNow where to find the application's code.

The application name, while important within ServiceNow, is not directly used for establishing the connection with the external Git repository. The Git repository is identified by its URL, and the access is controlled by the provided credentials. The application name in ServiceNow is metadata that is often embedded inside commit messages, but is not directly involved in authentication or location of the remote repository. The connection is made using the repository URL and the provided credentials.

Therefore, while the application name is important for identification within ServiceNow, it's not a required parameter for setting up the connection to the external Git repository. The connection is established based on the repository URL and the provided authentication credentials.

Here are some links for further research:

ServiceNow documentation on Source Control Integration: https://docs.servicenow.com/bundle/tokyo-application-development/page/build/applications/concept/source_control_integration.html
GitHub documentation on authentication: https://docs.github.com/en/authentication

Question: 50

Which Report Type(s) can be created by right-clicking on a column header in a table's list?

A.Bar Chart, Pie Chart, Histogram, and Line
B.Bar Chart
C.Bar Chart, Pie Chart, and Histogram
D.Bar Chart and Pie Chart

Answer: D

Explanation:

The correct answer is **D. Bar Chart and Pie Chart** because ServiceNow's user interface provides a quick way to generate these two report types directly from a column header in a table list. Right-clicking a column header and selecting "Quick Chart" (or similar wording, depending on the ServiceNow version) offers these two common visualization options for a fast overview of the data distribution in that column.

ServiceNow's reporting module aims to simplify data visualization for users of all skill levels. Choosing "Bar Chart" presents data in a column-based format, effectively demonstrating the frequency or count of each distinct value within the selected column. This quickly reveals the most common values.

Similarly, opting for a "Pie Chart" visualizes the proportion of each unique value relative to the total number of records. It highlights which values contribute most significantly to the whole dataset. Both chart types provide immediate insights without needing to navigate the full report designer interface.

Histograms, while valuable for showing data distribution, and Line charts, used for visualizing trends over time, are typically not offered as direct, one-click options from the column header. Creating them generally requires accessing the report designer and configuring the report more explicitly. The column header shortcut targets frequently-used, simpler visualizations that deliver immediate insights. The design reflects a trade-off between ease-of-use for basic reports and the flexibility for complex visualizations.

Therefore, the column context menu provides the most common initial report choices that leverage the specific column's data.

For further research, refer to the official ServiceNow documentation on creating quick reports:

ServiceNow Docs - Reporting (replace "sandiego" with your version if needed). Although this link may not directly point to the Quick Chart feature, it gives a comprehensive overview of the reporting capabilities within ServiceNow. Explore the different sections to find related info.

Question: 51

Which one of the following is NOT a method used for logging messages in a server-side script for a privately-scoped application?

A.gs.log()
B.gs.error()
C.gs.warn()
D.gs.debug()

Answer: A

Explanation:

The correct answer is A, gs.log(), because privately-scoped ServiceNow applications have restrictions on logging mechanisms compared to global-scoped applications. The gs.log() method, while available globally, is **not** intended for use within privately scoped applications for standard logging purposes. This is because its output is typically more broadly accessible and might not adhere to the strict security and isolation principles that privately scoped applications are designed to uphold.

Privately scoped applications are designed to encapsulate and protect their data and logic, preventing unauthorized access from other applications. Using gs.log() within a private scope could inadvertently expose sensitive information.

Instead, privately scoped applications primarily rely on the gs.error(), gs.warn(), and gs.debug() methods for

logging, which are designed with more control and specific use cases in mind. gs.error() is used to log error messages indicating significant issues. gs.warn() is used for warnings about potential problems or unexpected behavior. gs.debug() is meant for detailed debugging information that's typically filtered out in production environments.

The key distinction is that gs.error(), gs.warn(), and gs.debug() offer better mechanisms for filtering and controlling the visibility of logged messages within the private scope's intended boundaries. gs.debug() especially benefits from the system property glide.script.debug allowing for granular control. gs.log() is more generally available and its logs can be viewed by users who may not have appropriate access to private scope data.

Therefore, while gs.log() technically might not immediately throw an error within a private scope, its usage is discouraged and not a standard practice. The best practice is to use the dedicated gs.error(), gs.warn(), and gs.debug() methods for logging within privately scoped applications.

For additional information regarding scoped applications logging, consult the official ServiceNow documentation:

Scoped Applications
gs.log() documentation (note the general nature, suitable for global scope)
Logging levels in Servicenow (Useful for understanding the purpose of different log levels)