COURSE PROGRESS: 0% COMPLETE

Salesforce Platform Developer I – Practice Test #4

Results

0 of 40 Questions answered correctly

Your time: 00:00:03

You have reached 0 of 40 point(s), (0%)

Restart Quiz View Questions

1 ✗	2 ✗	3 ✗	4 ✗	5 ✗	6 ✗	7 ✗	8 ✗	9 ✗	10 ✗	11 ✗	12 ✗	13 ✗	14 ✗	15 ✗	16 ✗
17 ✗	18 ✗	19 ✗	20 ✗	21 ✗	22 ✗	23 ✗	24 ✗	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40								

When preparing to migrate a complex data model from a Salesforce development environment to a full sandbox for further testing, which strategy ensures a comprehensive and efficient data migration, including both schema and data, with special attention to custom settings and metadata?

Use the Salesforce Data Import Wizard to migrate both the schema and data simultaneously, ensuring custom settings are manually configured post-import.

Live Agent

Utilize a third-party ETL tool to clone the entire development environment, including data, schema, and custom settings, directly into the sandbox.

Directly modify the production environment to reflect the development changes, then refresh the full sandbox to mirror these updates.

Deploy the schema and metadata changes using change sets, then use Data Loader for data migration, ensuring to map relationships and external IDs appropriately. Correct answer

Incorrect

Deploy the schema and metadata changes using change sets, then use Data Loader for data migration, ensuring to map relationships and external IDs appropriately. -> Correct. This approach allows for controlled migration of schema and metadata changes through change sets, while the Data Loader can be used to handle complex data migrations, including the management of relationships and external IDs.

Use the Salesforce Data Import Wizard to migrate both the schema and data simultaneously, ensuring custom settings are manually configured post-import. -> Incorrect. The Data Import Wizard is primarily designed for importing records and does not handle schema or metadata migrations, nor does it provide options for custom settings.

Directly modify the production environment to reflect the development changes, then refresh the full sandbox to mirror these updates. -> Incorrect. Direct modifications to production environments are highly discouraged due to the risk of impacting live operations. Changes should be tested in a sandbox before being deployed to production.

Utilize a third-party ETL tool to clone the entire development environment, including data, schema, and custom settings, directly into the sandbox. -> Incorrect. While third-party ETL tools can be powerful for data migrations, they might not handle Salesforce metadata and custom settings as seamlessly as Salesforce-native tools, potentially leading to incomplete migrations or inconsistencies.

The developer at CloudTech Solutions needs to create a custom field on the Account object to calculate the total value of all closed-won opportunities related to that account. Which field type should the developer use to fulfill this requirement?

Lookup Relationship

Formula Field

Roll-Up Summary Field — Correct answer

Master-Detail Relationship

Incorrect

Roll-Up Summary Field -> Correct. A roll-up summary field calculates and displays the sum, minimum, maximum, or count of related records' values, such as the total value of closed-won opportunities for an account.

Lookup Relationship -> Incorrect. A lookup relationship creates a link between two objects but does not aggregate data from related records.

Master-Detail Relationship -> Incorrect. Although a master-detail relationship can roll up summary data, the relationship itself is not a field type used to calculate values.

Formula Field -> Incorrect. A formula field calculates values based on other fields within the same record, not across related records.

A Salesforce developer needs to write an Apex method to process a list of Account records. The method should iterate through each Account, check if the Account's AnnualRevenue is greater than $1 million, and if so, update the Account's Status__c field to 'High Value'. Additionally, if the Account is in the 'Technology' industry, the method should further update the Account's IndustryStatus__c field to 'Tech Leader'. Which Apex control flow statement and logic best accomplishes this task?

Implement a while loop with nested switch statements for each condition.

Utilize a do-while loop and multiple if statements without else for each condition.

Apply a single if statement with logical AND operators to check all conditions simultaneously within a for loop.

Use a for loop to iterate through the Accounts and if-else statements for conditions. Correct answer

Incorrect

Use a for loop to iterate through the Accounts and if-else statements for conditions. -> Correct. A for loop efficiently iterates over each Account record, and if-else statements are ideal for checking conditions and applying logic based on the AnnualRevenue and Industry fields.

Implement a while loop with nested switch statements for each condition. -> Incorrect. While while loops can iterate through lists, they are less efficient than for loops for this purpose. Salesforce Apex does not support switch statements for string or object conditions, making this approach incorrect.

Utilize a do-while loop and multiple if statements without else for each condition. -> Incorrect. do-while loops execute at least once before checking the condition, which is not necessary for iterating through a list of records. Multiple if statements could work but are less efficient without using else for mutually exclusive conditions.

Apply a single if statement with logical AND operators to check all conditions simultaneously within a for loop. -> Incorrect. While this approach may seem concise, it can become more complex and harder to maintain as the number of conditions increases. Additionally, it might not be as readable or easy to understand compared to using separate if-else statements for each condition.

Given a complex scenario where you need to filter a list of Account records in Apex based on multiple criteria and then process each filtered record differently, which two of the following Apex control flow statements would be most effectively used to achieve this requirement?

- [ ] Employ the continue keyword within a loop to skip processing for Accounts that do not meet any of the specified criteria.
- [ ] Implement a for loop to iterate over the list of Accounts, applying nested if-else statements within the loop to perform different actions based on the records' attributes. Correct answer
- [ ] Utilize an if-else if-else ladder to apply different processing logic based on the criteria each Account meets. Correct answer
- [ ] Apply the switch statement to directly handle the processing of different Account types, assuming a predefined list of types is available.

Incorrect

Utilize an if-else if-else ladder to apply different processing logic based on the criteria each Account meets. -> Correct. An if-else if-else ladder is ideal for applying different logic based on varying conditions. In processing a list of Accounts, this allows for the execution of distinct blocks of code for each record based on specific criteria, making it highly suitable for filtering and processing records differently.

Implement a for loop to iterate over the list of Accounts, applying nested if-else statements within the loop to perform different actions based on the records' attributes. -> Correct. A for loop is perfectly suited for iterating over a collection like a list of Accounts. Using ne

if-else statements within this loop allows for detailed and nuanced control over how each record is processed, based on its attributes or other specific criteria.

Apply the switch statement to directly handle the processing of different Account types, assuming a predefined list of types is available. -> Incorrect. While switch statements are useful for branching logic based on distinct cases, they are less effective in scenarios requiring complex condition evaluation or when dealing with non-discrete values (like strings or enums), especially without clear, predefined cases as in filtering dynamic record attributes.

Employ the continue keyword within a loop to skip processing for Accounts that do not meet any of the specified criteria. -> Incorrect. The continue keyword does indeed skip the current loop iteration, but it is not a control flow statement used to directly apply different logic based on varying conditions. It's more a tool to optimize loop execution rather than a primary method for implementing conditional logic.

A Salesforce application needs to display a custom Visualforce page to end-users that shows a list of all active Contacts associated with the Account currently viewed by the user. Additionally, the page should allow users to select multiple Contacts and perform a mass update to mark them as "Inactive". This functionality requires a combination of displaying data and modifying Salesforce data based on user input. Which approach should be taken to implement this requirement using a Visualforce page and the appropriate controllers or extensions?

Rely solely on the Standard Controller for Account with built-in Visualforce functionality to display and update Contacts without any custom code.

Use a Standard Controller for Account and a Custom Extension to fetch and display active Contacts and handle the mass update operation.

Correct answer

Utilize a Visualforce Component with its own controller to manage the display and update of Contacts, embedded within a Visualforce page using the Standard Account Controller.

Implement a Custom Controller dedicated to handling all logic, including fetching Account data, displaying active Contacts, and the mass update functionality.

Incorrect

Use a Standard Controller for Account and a Custom Extension to fetch and display active Contacts and handle the mass update operation. -> Correct. The Standard Controller for Account provides built-in functionality for accessing Account data, and the Custom Extension allows for the additional logic required to fetch active Contacts, display them, and handle the mass update, aligning with the requirements.

Implement a Custom Controller dedicated to handling all logic, including fetching Account data, displaying active Contacts, and the mass update functionality. -> Incorrect. While a Custom Controller can handle these requirements, it unnecessarily replicates functionality readily available in the Standard Controller for Account, making this approach less efficient.

Utilize a Visualforce Component with its own controller to manage the display and update of Contacts, embedded within a Visualforce page using the Standard Account Controller. -> Incorrect. Visualforce Components can enhance modularity but using them alone without a Custom Extension or Custom Controller for the specific logic required for fetching and updating Contacts misses the need for a cohesive solution integrating with the Account context.

Rely solely on the Standard Controller for Account with built-in Visualforce functionality to display and update Contacts without any custom code. -> Incorrect. The Standard Controller for Account does not provide the necessary functionality to fetch specific subsets of related records or perform complex operations like mass updates without custom code.

Given a scenario where you need to display a list of custom object records on a Visualforce page and allow users to edit these records directly from the page, which two approaches correctly utilize Visualforce pages and the appropriate controllers or extensions?

Correct answer

- [ ] Use a standard controller with a custom extension to handle record updates.
- [ ] Utilize an Apex custom controller to fetch and manage the object records. Correct answer
- [ ] Apply a third-party API directly on the Visualforce page for data manipulation, bypassing Salesforce controllers.
- [ ] Use a Visualforce page without any controller or extension, relying on HTML forms to submit data changes.
- [ ] Embed JavaScript directly in the Visualforce page to update records without a controller.

Incorrect

Use a standard controller with a custom extension to handle record updates. -> Correct. This approach leverages the built-in functionality of the standard controller for basic operations while using a custom extension to implement the specific logic needed for record updates, offering a balance of efficiency and customization.

Utilize an Apex custom controller to fetch and manage the object records. -> Correct. An Apex custom controller provides full control over the interaction with the database, allowing for complex data manipulation and business logic implementation, making it suitable for scenarios that require more than what standard controllers offer.

Embed JavaScript directly in the Visualforce page to update records without a controller. -> Incorrect. While JavaScript can enhance the user interface and interact with Salesforce via API calls, using it to directly update records without a controller is not a best practice. It can lead to security vulnerabilities and bypasses the built-in capabilities of Visualforce controllers for data manipulation.

Use a Visualforce page without any controller or extension, relying on HTML forms to submit data changes. -> Incorrect. This approach does not take advantage of Visualforce's integration with Apex controllers for data handling, leading to potential security risks and a lack of robustness in managing Salesforce data.

Apply a third-party API directly on the Visualforce page for data manipulation, bypassing Salesforce controllers. -> Incorrect. Using a third-party API for direct data manipulation bypasses the security and data integrity mechanisms provided by Salesforce controllers. This approach could lead to data consistency issues and security vulnerabilities.

When implementing exception handling in Apex, including the use of custom exceptions, which of the following practices is recommended?

Catch all exceptions using a generic Exception class catch block and handle them identically to simplify error handling.

Create custom exception classes for specific error scenarios and use try-catch blocks to handle these exceptions appropriately. Correct answer

Avoid using try-catch blocks for DML operations since Salesforce automatically rolls back transactions in case of errors.

Use the finally block to throw new exceptions if cleanup actions fail, ensuring that the original exception is not lost.

Incorrect

Create custom exception classes for specific error scenarios and use try-catch blocks to handle these exceptions appropriately. -> Correct. Creating custom exception classes allows for more granular control over error handling by providing a way to distinguish between different types of errors. This enables more tailored and effective error handling strategies.

Catch all exceptions using a generic Exception class catch block and handle them identically to simplify error handling. -> Incorrect. Catching all exceptions using a generic Exception class and handling them identically is not a best practice. It can obscure the root cause of different errors and lead to inadequate handling of specific error conditions.

Avoid using try-catch blocks for DML operations since Salesforce automatically rolls back transactions in case of errors. -> Incorrect. While Salesforce does automatically roll back transactions in the event of DML operation failures, using try-catch blocks allows developers to gracefully handle exceptions and provide users with more informative error messages or perform alternative logic.

Use the finally block to throw new exceptions if cleanup actions fail, ensuring that the original exception is not lost. -> Incorrect. The finally block is intended for cleanup actions that must occur regardless of whether an exception was thrown or not. Throwing new exceptions from within a finally block is not recommended as it can overshadow original exceptions and complicate error handling logic.

For a developer tasked with quickly identifying and resolving a runtime exception occurring in a complex Apex batch job, which Salesforce developer tool provides the most efficient means for debugging this issue in a non-production environment?

Developer Console's Debug Logs to trace execution logs and identify exceptions. Correct answer

Salesforce CLI to deploy the batch job code.

Salesforce DX to manage source code and version control.

Change Sets to migrate the batch job between environments.

Incorrect

Developer Console's Debug Logs to trace execution logs and identify exceptions. -> Correct. Developer Console's Debug Logs provide detailed execution information, including error messages and stack traces, making it ideal for diagnosing runtime exceptions.

Salesforce CLI to deploy the batch job code. -> Incorrect. While Salesforce CLI is powe for deployment and automation tasks, it is not primarily designed for in-depth debuggi

runtime exceptions.

Salesforce DX to manage source code and version control. -> Incorrect. Salesforce DX excels in source-driven development and version control but does not directly facilitate runtime exception debugging.

Change Sets to migrate the batch job between environments. -> Incorrect. Change Sets can move metadata and code between environments but do not offer debugging features for identifying runtime issues.

To enhance data integrity for a custom field on a Salesforce object, ensuring that each entry is not only present but also adheres to a predefined format, which two field properties should be configured?

- [ ] Length
- [ ] Unique
- [ ] Validation Rule — Correct answer
- [ ] Required — Correct answer
- [ ] Data Type

Incorrect

Validation Rule -> Correct. Validation rules enforce data quality by ensuring that data entered into a field meets specific criteria or patterns before the record can be saved, such as a specific format for phone numbers or email addresses.

Required -> Correct. Making a field required ensures that users cannot leave the field blank, contributing to data completeness and integrity.

Length -> Incorrect. Length specifies the maximum number of characters a field can contain. While it can enforce a maximum size, it does not ensure uniqueness or that the field is filled out.

Unique -> Incorrect. Setting a field to be unique ensures that each entry is distinct across all records. Although important for avoiding duplicates, it doesn't ensure the field adheres to a specific format.

Data Type -> Incorrect. The data type defines the kind of data the field can hold (e.g., text, number, date), but on its own, it does not ensure the field is filled out or follows a specific format.

In a scenario requiring the integration of Apex code with Salesforce UI components such as Lightning Components, Visualforce Pages, Flows, and Next Best Actions, which approach best facilitates efficient and scalable interactions?

Use Apex to directly manipulate the DOM of Lightning Components and Visualforce Pages for dynamic UI updates.

Create generic Apex services that expose methods for CRUD operations, which can be called from Lightning Components, Flows, and Next Best Actions.

Correct answer

Avoid using Apex with Next Best Actions and rely exclusively on configuration-based criteria within the Next Best Actions setup.

Design Apex classes to be tightly coupled with specific Lightning Component or Flow implementations for optimal performance.

Incorrect

Create generic Apex services that expose methods for CRUD operations, which can be called from Lightning Components, Flows, and Next Best Actions. -> Correct. Developi generic Apex services that provide a clear API for data operations ensures reusability a

decouples the backend logic from the specific UI components, supporting scalability and maintainability.

Design Apex classes to be tightly coupled with specific Lightning Component or Flow implementations for optimal performance. -> Incorrect. Tight coupling between Apex and specific UI components reduces code reusability and scalability, making it harder to maintain and adapt to new requirements.

Use Apex to directly manipulate the DOM of Lightning Components and Visualforce Pages for dynamic UI updates. -> Incorrect. Apex runs on the server side and cannot directly manipulate the DOM of UI components. Client-side JavaScript is used for DOM manipulation in Lightning Components and Visualforce Pages.

Avoid using Apex with Next Best Actions and rely exclusively on configuration-based criteria within the Next Best Actions setup. -> Incorrect. Apex can enhance Next Best Actions by providing complex decision logic that goes beyond what is possible with configuration alone, enabling more sophisticated and dynamic recommendations.

When deploying Apex code from a sandbox environment to production, what is a prerequisite for the deployment to succeed in terms of code coverage?

At least 50% of the Apex code in the production environment must be covered by unit tests.

Each individual Apex class and trigger must have at least 75% code coverage.

At least 75% of the Apex code in the production environment must be covered by unit tests. Correct answer

100% of the Apex code must be covered by unit tests to ensure full functionality.

Incorrect

At least 75% of the Apex code in the production environment must be covered by unit tests. -> Correct. Salesforce requires a minimum of 75% code coverage for all Apex code in the production environment as a prerequisite for deployment.

At least 50% of the Apex code in the production environment must be covered by unit tests. -> Incorrect. The minimum code coverage required for Apex code deployment to production is higher than 50%.

Each individual Apex class and trigger must have at least 75% code coverage. -> Incorrect. While high code coverage is encouraged for individual classes and triggers, the platform enforces a different overall coverage requirement.

100% of the Apex code must be covered by unit tests to ensure full functionality. -> Incorrect. Although achieving 100% code coverage is ideal, Salesforce only requires a minimum of 75% code coverage for deployment to production.

A Salesforce developer is planning to use the Lightning Component framework for a new project. Which statement accurately describes the Lightning Component framework, its benefits, and the types of content that can be contained in a Lightning web component?

The Lightning Component framework allows for the creation of reusable components that can only contain HTML and custom JavaScript.

The Lightning Component framework enables developers to build highly performant and scalable web applications using standard web technologies, including HTML, JavaScript, and CSS, and allows for encapsulation of HTML, JavaScript, CSS, and Salesforce-specific features in a Lightning web component.

Correct answer

Lightning web components can include Apex code directly within the component for complex business logic implementation.

The framework supports only client-side rendering, limiting the ability to interact with Salesforce data and metadata directly.

Incorrect

The Lightning Component framework enables developers to build highly performant and scalable web applications using standard web technologies, including HTML, JavaScript, and CSS, and allows for encapsulation of HTML, JavaScript, CSS, and Salesforce-specific features in a Lightning web component. -> Correct. This statement accurately describes the Lightning Component framework. It highlights the framework's support for standard web technologies and Salesforce-specific features, the encapsulation capabilities within Lightning web components, and the benefits of using the framework for building scalable and performant web applications.

The Lightning Component framework allows for the creation of reusable components that can only contain HTML and custom JavaScript. -> Incorrect. While HTML and custom JavaScript are essential parts of Lightning web components, this choice incorrectly suggests that these are the only types of content that can be contained. Lightning web components can also include CSS and utilize Salesforce-specific features.

Lightning web components can include Apex code directly within the component for complex business logic implementation. -> Incorrect. Apex code cannot be directly included within a Lightning web component. Instead, Apex is called from Lightning web components using @wire or imperative methods to handle complex business logic.

The framework supports only client-side rendering, limiting the ability to interact with Salesforce data and metadata directly. -> Incorrect. The Lightning Component framework supports both client-side and server-side interactions, enabling components to interact with Salesforce data and metadata efficiently through Apex controllers or Salesforce APIs.

In the Salesforce Platform, which statement accurately describes the purpose and use of Apex triggers?

Apex triggers are primarily used for managing user sessions and authentication.

Apex triggers are specifically designed to update the Salesforce UI in real time.

Apex triggers are utilized to directly query and manipulate large data sets in external systems.

Apex triggers are used to perform DML operations before and after changes to Salesforce records. Correct answer

Incorrect

Apex triggers are used to perform DML operations before and after changes to Salesforce records. -> Correct. Apex triggers enable developers to execute code before or after changes to Salesforce records, such as insertions, updates, or deletions. This allows for complex business logic to be executed as part of the data manipulation process.

Apex triggers are specifically designed to update the Salesforce UI in real time. -> Incorrect. Apex triggers do not directly update the Salesforce UI; they operate on the server side to manage data operations. Real-time UI updates would typically involve Lightning components or Visualforce pages with JavaScript.

Apex triggers are utilized to directly query and manipulate large data sets in external systems. -> Incorrect. While Apex can interact with external systems via callouts, triggers themselves are designed to react to data changes within the Salesforce platform, not to directly handle large data sets in external systems.

Apex triggers are primarily used for managing user sessions and authentication. -> Incorrect. Managing user sessions and authentication is not the purpose of Apex triggers. Salesforce handles sessions and authentication through its built-in security model.

When determining the appropriate approach between declarative and programmatic customizations in Salesforce, which two statements accurately reflect common use cases or best practices in relation to governor limits, formula fields, and roll-up summaries? Select two correct answers.

Correct al

- [ ] For real-time processing of large data volumes that exceed declarative automation capabilities, programmatic solutions like batch Apex are recommended.

- [ ] Formula fields should be used over custom Apex code for calculated fields to minimize complexity and avoid consuming Apex governor limits. Correct answer

- [ ] When implementing custom user interfaces or integrating with external systems, declarative tools like Lightning App Builder and External Objects should be used exclusively.

- [ ] Custom roll-up summary fields on unrelated objects are best created using declarative workflow rules rather than programmatic approaches.

Incorrect

For real-time processing of large data volumes that exceed declarative automation capabilities, programmatic solutions like batch Apex are recommended. -> Correct. Batch Apex allows for the processing of large sets of data by breaking the operation into smaller batches, which helps manage and stay within governor limits, offering a solution when declarative options fall short.

Formula fields should be used over custom Apex code for calculated fields to minimize complexity and avoid consuming Apex governor limits. -> Correct. Formula fields are calculated at runtime and do not count towards Apex governor limits, making them a preferable choice for simple calculations to ensure efficiency and reduce the need for custom code.

When implementing custom user interfaces or integrating with external systems, declarative tools like Lightning App Builder and External Objects should be used exclusively. -> Incorrect. While declarative tools offer powerful capabilities for UI customization and external data integration, programmatic solutions such as Apex and Lightning Web Components may be necessary for complex integrations or custom behavior.

Custom roll-up summary fields on unrelated objects are best created using declarative workflow rules rather than programmatic approaches. -> Incorrect. Workflow rules ca...

create roll-up summaries for unrelated objects. Custom roll-up summaries for unrelated objects typically require programmatic solutions, such as Apex triggers or third-party tools.

Given a scenario where you need to display and use custom user interface components in Salesforce, including Lightning Components, Flow, and Visualforce, which approach is the most effective?

Use Visualforce exclusively for all custom UI development, as it is the oldest and most stable technology for building user interfaces in Salesforce.

Embed Visualforce pages directly within Lightning Components using iFrames to ensure compatibility and seamless user experience across different UI technologies.

Develop all custom UI components exclusively in Apex to maintain consistency in server-side logic, avoiding the use of client-side technologies like JavaScript and HTML.

Utilize the Lightning Web Components (LWC) framework for building custom UI components, and integrate them with Flow and Visualforce pages using Lightning Out to leverage their full potential within the Salesforce ecosystem.

Correct answer

Incorrect

Utilize the Lightning Web Components (LWC) framework for building custom UI components, and integrate them with Flow and Visualforce pages using Lightning Out to leverage their full potential within the Salesforce ecosystem. -> Correct. Utilizing the Lightning Web Components (LWC) framework for creating custom UI components and integrating them with other Salesforce technologies like Flow and Visualforce through Lightning Out represents a modern, efficient, and scalable approach. LWC is designed to coexist and complement existing UI development technologies in Salesforce, allowing developers to build highly performant, reusable UI components that can be leveraged across different parts of the Salesforce platform, including embedding within Visualfor pages or launching from Flow.

Embed Visualforce pages directly within Lightning Components using iFrames to ensure compatibility and seamless user experience across different UI technologies. -> Incorrect. Embedding Visualforce pages within Lightning Components using iFrames is generally not recommended due to potential security and performance issues, and it does not provide a seamless integration between these technologies.

Develop all custom UI components exclusively in Apex to maintain consistency in server-side logic, avoiding the use of client-side technologies like JavaScript and HTML. -> Incorrect. Apex is a server-side technology and cannot be used directly for building UI components. Client-side technologies like JavaScript, HTML, and CSS are essential for UI development.

Use Visualforce exclusively for all custom UI development, as it is the oldest and most stable technology for building user interfaces in Salesforce. -> Incorrect. Relying exclusively on Visualforce limits the ability to leverage the modern capabilities of the Lightning Component framework and does not support the advanced client-side performance optimizations available with Lightning Web Components.

In a scenario where an Apex trigger is designed to aggregate and update summary fields on a parent object whenever related child records are inserted, updated, or deleted, identify the implications of governor limits on this Apex transaction and select the correct approaches to handle it. Select two.

- [ ] Use a @future method to process each child record update asynchronously, performing SOQL queries and DML operations within the @future method.
- [ ] Implement the aggregation logic directly within the trigger, using a SOQL query for each parent record to calculate the summary fields whenever a child record is inserted, updated, or deleted.
- [ ] Opt for a batch Apex class that is scheduled to run during off-peak hours, summarizing the day's changes to child records and updating parent records accordingly.

Correct answer

Live Agent

Correct ar

- [ ] Aggregate the child records in a collection within the trigger, and perform a single SOQL query outside of loops to retrieve necessary parent records, then update them in bulk.

Incorrect

Aggregate the child records in a collection within the trigger, and perform a single SOQL query outside of loops to retrieve necessary parent records, then update them in bulk. -> Correct. By aggregating data and minimizing the number of SOQL queries and DML operations, this approach adheres to best practices for managing governor limits.

Opt for a batch Apex class that is scheduled to run during off-peak hours, summarizing the day's changes to child records and updating parent records accordingly. -> Correct. This approach minimizes the impact on system performance during peak hours and ensures that bulk updates are handled efficiently, respecting governor limits.

Implement the aggregation logic directly within the trigger, using a SOQL query for each parent record to calculate the summary fields whenever a child record is inserted, updated, or deleted. -> Incorrect. This approach risks exceeding governor limits for SOQL queries if the trigger processes multiple child records related to different parent records in a single transaction.

Use a @future method to process each child record update asynchronously, performing SOQL queries and DML operations within the @future method. -> Incorrect. While @future methods help with asynchronous processing, using them for each individual record update does not effectively manage bulk processing and may hit the governor limits on asynchronous calls.

When preparing to deploy code and associated configurations in Salesforce, which environments and processes should be utilized to ensure successful deployment? Select two.

- [ ] Utilize Version Control System as the Deployment Tool

- [ ] Use a Sandbox for Testing Changes
- [ ] Deploy Code During Peak Business Hours for Immediate Feedback
- [ ] Directly Modify Production Org for Quick Fixes
- [ ] Deploy Using Change Sets

Correct answer

Incorrect

Use a Sandbox for Testing Changes -> Correct. Sandboxes provide a separate environment from the production org, allowing developers to build, test, and stage changes without affecting the live environment. This is critical for testing code and configurations before deployment.

Deploy Using Change Sets -> Correct. Change sets allow for the selective transfer of configuration changes and Apex code between Salesforce orgs, such as from a sandbox to production, ensuring a controlled deployment process.

Directly Modify Production Org for Quick Fixes -> Incorrect. Direct modifications in the production org can lead to unexpected issues and are not recommended. Changes should be tested in a non-production environment first.

Utilize Version Control System as the Deployment Tool -> Incorrect. While a Version Control System (VCS) is essential for tracking changes and collaboration, it does not directly deploy code to Salesforce orgs. Deployment tools or Salesforce's Metadata API are required for actual deployment.

Deploy Code During Peak Business Hours for Immediate Feedback -> Incorrect. Deploying code during peak business hours can disrupt business operations. It is advisable to schedule deployments during off-peak hours to minimize impact and allow for thorough post-deployment testing.

A Salesforce development team needs to implement a functionality that automatically updates a custom field on the Account object to classify accounts based on the number of high-value opportunities (value greater than $50,000) associated with each account. The classifications are “Silver” for 1-3 high-value opportunities, “Gold” for 4-6, and “Platinum” for more than 6 high-value opportunities. Considering governor limits, best practices for declarative versus programmatic customizations, formula fields, and roll-up summaries, which approach should the developer take?

Create a roll-up summary field on the Account to count high-value opportunities and a formula field to set the account classification based on the count.

Develop an Apex trigger on the Opportunity object that updates the Account classification each time an Opportunity is created, updated, or deleted. Correct answer

Implement a scheduled Apex class that recalculates Account classifications daily, based on the total value of high-value opportunities.

Use Process Builder to update the Account classification each time an Opportunity is created or updated, based on the criteria for high-value opportunities.

Incorrect

Develop an Apex trigger on the Opportunity object that updates the Account classification each time an Opportunity is created, updated, or deleted. -> Correct. This approach allows for precise control over the logic to classify accounts based on the dynamic conditions of related opportunities, including handling bulk data operations efficiently and respecting governor limits.

Create a roll-up summary field on the Account to count high-value opportunities and a formula field to set the account classification based on the count. -> Incorrect. Roll-up summary fields cannot directly evaluate conditions (such as opportunity value greater than $50,000) on child records to count them, and formula fields cannot dynamically categorize accounts based on related records beyond their direct parent-child relationships.

Use Process Builder to update the Account classification each time an Opportunity is created or updated, based on the criteria for high-value opportunities. -> Incorrect. While Process Builder can handle complex logic and update related records, it may not efficiently manage bulk data changes and can lead to hitting governor limits in cases with high volumes of opportunity records.

Implement a scheduled Apex class that recalculates Account classifications daily, based on the total value of high-value opportunities. -> Incorrect. While this approach can ensure account classifications are updated, it introduces unnecessary delay in classification updates and may not reflect real-time changes in opportunity data.

Given a scenario where you need to develop a custom logic that cannot be achieved through declarative features in Salesforce, you decide to write an Apex trigger. When following best practices for writing Apex triggers, which of the following approaches should you adopt?

Avoid using context-specific trigger methods (like before insert, after update) to make the trigger more flexible.

Create multiple triggers for the same object to handle different events separately.

Use a Trigger Handler pattern, where the trigger delegates the processing logic to a separate class.

Correct answer

Write all the logic directly inside the trigger body for simplicity.

Incorrect

Use a Trigger Handler pattern, where the trigger delegates the processing logic to a separate class. -> Correct. The Trigger Handler pattern is a best practice because it separates concerns, organizes the logic into manageable units, and makes the code easier to test and maintain.

Live Agent

Write all the logic directly inside the trigger body for simplicity. -> Incorrect. Writing all logic directly in the trigger body is not a best practice. It makes the code hard to maintain, test, and can lead to issues with governor limits.

Create multiple triggers for the same object to handle different events separately. -> Incorrect. Creating multiple triggers for the same object is not recommended because it can lead to unpredictable behavior due to the uncertain order of execution.

Avoid using context-specific trigger methods (like before insert, after update) to make the trigger more flexible. -> Incorrect. Avoiding context-specific methods makes the trigger less efficient and can lead to unnecessary processing. It's best to use context-specific trigger methods to ensure the trigger only runs when necessary.

Given a scenario where you need to display a custom list of Account records on a Salesforce Visualforce page, which approach would you use to ensure the page displays the data using the appropriate controller or extension?

Implement a Visualforce Component without a controller, relying on built-in Visualforce functionality to access the Account object.

Utilize a Standard Controller without extensions, directly querying the Account object in the Visualforce page markup.

Develop an Apex Trigger on the Account object to automatically populate a Visualforce page when an Account is accessed.

Create a Custom Controller that queries the Account records and utilize this controller in the Visualforce page.

Correct answer

Incorrect

Create a Custom Controller that queries the Account records and utilize this controlle the Visualforce page. -> Correct. A Custom Controller is the best choice for displaying

Live Agent

custom list of records, as it allows for the execution of specific SOQL queries to fetch and display the data as required.

Utilize a Standard Controller without extensions, directly querying the Account object in the Visualforce page markup. -> Incorrect. A Standard Controller without extensions cannot execute custom queries within Visualforce markup.

Implement a Visualforce Component without a controller, relying on built-in Visualforce functionality to access the Account object. -> Incorrect. Visualforce Components require a controller or extension to fetch custom data sets; they cannot directly query Salesforce data.

Develop an Apex Trigger on the Account object to automatically populate a Visualforce page when an Account is accessed. -> Incorrect. Apex Triggers are used to execute code before or after specific data manipulation operations and cannot directly influence Visualforce page displays without a controller.

When designing a solution to automatically update all related Contact records when an Account's address is changed, which Salesforce process automation tool should be used to efficiently implement this requirement while adhering to best practices for scalability and maintainability?

Implement an Apex trigger on the Account object to handle the address update and propagate changes to related Contacts.

Create a Process Builder process to listen for changes on the Account's address fields and update all related Contacts accordingly.

Correct answer

Use the Salesforce Lightning Component Framework to develop a component that updates Contacts when an Account's address is changed via the UI.

Utilize Workflow Rules to update the Contact records whenever the Account's address fields are modified.

Incorrect

Create a Process Builder process to listen for changes on the Account's address fields and update all related Contacts accordingly. -> Correct. Process Builder is a powerful tool for automating complex business processes without writing code. It can handle the requirement to update related records based on changes to a record field efficiently and with less maintenance overhead than code-based solutions. It also allows for more complex logic than Workflow Rules and is generally recommended over Apex for declarative solutions.

Utilize Workflow Rules to update the Contact records whenever the Account's address fields are modified. -> Incorrect. While Workflow Rules can automate simple field updates, they lack the ability to perform complex logic or update related records based on criteria other than field updates on the same record.

Implement an Apex trigger on the Account object to handle the address update and propagate changes to related Contacts. -> Incorrect. Implementing an Apex trigger is a more complex solution that requires coding and testing. It's generally recommended to use declarative tools like Process Builder for such requirements unless the automation cannot be achieved through declarative means.

Use the Salesforce Lightning Component Framework to develop a component that updates Contacts when an Account's address is changed via the UI. -> Incorrect. Using the Lightning Component Framework to handle data updates is not the recommended approach for process automation. This framework is primarily for developing user interfaces and not for backend data processing or automation.

When developing an Apex class that processes data from various sources, it's necessary to implement robust exception handling to manage runtime exceptions. Additionally, the requirement specifies that for a particular type of data inconsistency, a custom exception must be thrown to signal the need for manual data review. How should this be correctly implemented in Apex?

Utilize a try-catch block with a finally section that throws the custom exception regardless of error occurrence.

Use a try-catch block, and within the catch, check the exception type before throwing a custom exception. Correct answer

Implement a custom exception class but do not use try-catch blocks in the method.

Use a try-catch block, catching only generic Exception types, and log the error.

Incorrect

Use a try-catch block, and within the catch, check the exception type before throwing a custom exception. -> Correct. This approach correctly implements robust exception handling by catching exceptions that occur during execution. By checking the exception type or error condition within the catch block before throwing a custom exception, it ensures that the custom exception is thrown only for specific data inconsistencies, as required.

Use a try-catch block, catching only generic Exception types, and log the error. -> Incorrect. Catching only generic Exception types might handle unexpected errors, but it doesn't specifically address the requirement to throw a custom exception for a particular type of data inconsistency.

Implement a custom exception class but do not use try-catch blocks in the method. -> Incorrect. While creating a custom exception class is necessary for throwing specific types of exceptions, not using try-catch blocks means you cannot effectively handle exceptions that occur during execution.

Utilize a try-catch block with a finally section that throws the custom exception regardless of error occurrence. -> Incorrect. Throwing a custom exception in the finally block regardless of whether an error occurred does not meet the requirement to throw it specifically for a type of data inconsistency. The finally block should be used for cleanup actions, not for throwing exceptions based on conditional checks.

A Salesforce developer is tasked with ensuring comprehensive test coverage for a suite of new features, including triggers, controllers, custom classes, automated flows, and process builders. The features interact with multiple standard and custom objects, and the tests must validate both positive and negative scenarios under various data conditions. Considering best practices for Salesforce development, which approach should the developer use to write and execute tests effectively for these components?

Utilize SeeAllData=true in all test classes to ensure tests are running with a full set of data.

Employ Test.loadData() to load test data from static resources for complex data models. Correct answer

Write a single, comprehensive test method to cover all features and scenarios to minimize test execution time.

Hard-code user IDs and record IDs in test methods to ensure consistency across test executions.

Incorrect

Employ Test.loadData() to load test data from static resources for complex data models. -> Correct. Test.loadData() allows for loading complex test data from static resources, which is efficient for setting up detailed test scenarios involving multiple objects and relationships without hard-coding test data creation in each test class.

Utilize SeeAllData=true in all test classes to ensure tests are running with a full set of data. -> Incorrect. Using SeeAllData=true makes tests dependent on the org's data, which can lead to unreliable test results and failures in different environments. It's best practice to create test data within test classes.

Write a single, comprehensive test method to cover all features and scenarios to minimize test execution time. -> Incorrect. Writing a single test method for all features undermines the test's ability to isolate issues and verify specific functionalities. It's better to write modular tests focusing on specific behaviors or scenarios.

Hard-code user IDs and record IDs in test methods to ensure consistency across test executions. -> Incorrect. Hard-coding IDs in test methods is not recommended as it creates dependencies on specific org data, making tests fail in other environments. Test data should be dynamically created and queried within test methods.

TechSolutions Corp has set the organization-wide default for Cases to public read-only to facilitate customer service collaboration. However, customer service representatives have indicated that the case list views are overcrowded with cases not relevant to their specific areas of expertise, complicating their workflow. How can the System Administrator streamline the process for these users to see only the cases relevant to them?

Adjust the sharing rules to restrict case visibility.

Create personalized list views for each area of expertise. Correct answer

Customize the case page layout to include only relevant fields.

Modify the role hierarchy to limit case visibility.

Incorrect

Create personalized list views for each area of expertise. -> Correct. Personalized list views allow users to filter and see only the cases that are relevant to their specific duties or areas of expertise, effectively streamlining their workflow.

Adjust the sharing rules to restrict case visibility. -> Incorrect. Sharing rules are used to extend access, not restrict it, and would not help in filtering cases for individual users based on their areas of expertise.

Customize the case page layout to include only relevant fields. -> Incorrect. Modifying page layouts changes the information displayed on case records but does not affect the list c cases a user sees.

Modify the role hierarchy to limit case visibility. -> Incorrect. The role hierarchy primarily affects record access in the context of the organization's sharing model and is not a tool for filtering records in list views or reports based on content or user preference.

A Salesforce development team is preparing to deploy a significant update to their application, including new Apex classes, Visualforce pages, and complex configuration changes. The deployment needs to be executed with minimal disruption to the production environment and must comply with Salesforce best practices for deployment. What strategy should the team adopt to ensure a successful deployment?

Rely solely on manual configuration changes in production to ensure accuracy and control.

Utilize unmanaged packages for all deployments to maintain flexibility in development and deployment processes.

Deploy directly from a developer's sandbox to production to reduce deployment time.

Use change sets to promote changes from a staging environment to production, after comprehensive testing in the staging environment. Correct answer

Incorrect

Use change sets to promote changes from a staging environment to production, after comprehensive testing in the staging environment. -> Correct. Utilizing change sets to move changes from a staging (or full sandbox) environment to production, following extensive testing, aligns with Salesforce best practices. This approach ensures that any issues are identified and resolved in a controlled environment before affecting end-users.

Deploy directly from a developer's sandbox to production to reduce deployment time. -> Incorrect. Deploying directly from a developer sandbox to production bypasses critical steps such as testing in a staging environment, which is essential for identifying and resolving issues before they impact the production environment.

Rely solely on manual configuration changes in production to ensure accuracy and control. -> Incorrect. Manual changes in production are prone to human error and lack the traceability and rollback capabilities provided by deploying through change sets or Salesforce DX. This approach also fails to leverage the benefits of version control and automated testing.

Utilize unmanaged packages for all deployments to maintain flexibility in development and deployment processes. -> Incorrect. While unmanaged packages can be useful for distributing open-source projects or applications, they are not the best solution for controlled deployments within the same organization, as they do not support the same level of traceability and rollback capabilities as change sets or Salesforce DX.

When developing a Salesforce application, which two practices should a developer implement to prevent user interface and data access security vulnerabilities?

- [ ] Use direct SQL queries in Apex to fetch data for dynamic web content generation.
- [ ] Rely on Visualforce page's built-in mechanisms to automatically escape HTML content. Correct answer
- [ ] Apply Field-Level Security (FLS) checks in Apex code to respect user permissions on data access. Correct answer
- [ ] Disable Salesforce's built-in CSRF protection on Visualforce pages to enhance performance.
- [ ] Embed JavaScript directly into Visualforce pages to perform data validation.

Incorrect

Apply Field-Level Security (FLS) checks in Apex code to respect user permissions on data access. -> Correct. Implementing FLS checks in Apex code ensures that the application

respects the field-level permissions set in Salesforce, preventing unauthorized access to sensitive data.

Rely on Visualforce page's built-in mechanisms to automatically escape HTML content. -> Correct. Visualforce pages have built-in mechanisms to escape HTML content, preventing XSS (Cross-Site Scripting) attacks by ensuring that any user input is rendered safely.

Embed JavaScript directly into Visualforce pages to perform data validation. -> Incorrect. Embedding JavaScript for data validation on the client side can be bypassed by attackers, making the application vulnerable to malicious inputs. Server-side validation is more secure.

Use direct SQL queries in Apex to fetch data for dynamic web content generation. -> Incorrect. Salesforce does not support direct SQL queries in Apex. SOQL is used instead, and dynamic queries should be handled carefully to avoid SOQL injection vulnerabilities.

Disable Salesforce's built-in CSRF protection on Visualforce pages to enhance performance. -> Incorrect. Disabling CSRF (Cross-Site Request Forgery) protection removes a critical security layer that prevents unauthorized commands from being transmitted from a user that the application trusts. This should not be disabled.

In the context of Salesforce, what is the primary requirement for deploying custom Apex code from a sandbox to a production environment?

All custom Apex classes and triggers must be manually reviewed by Salesforce before deployment.

A deployment package must include at least one Visualforce page.

Apex code must have at least 75% code coverage and all tests must pass. Correct answer

Apex code must be accompanied by relevant Lightning components for successful deployment.

Incorrect

Apex code must have at least 75% code coverage and all tests must pass. -> Correct. For a successful deployment to production, Apex code must achieve at least 75% code coverage, and all associated unit tests must pass without errors.

All custom Apex classes and triggers must be manually reviewed by Salesforce before deployment. -> Incorrect. Salesforce does not require manual review of custom Apex code by their team before deployment to production.

A deployment package must include at least one Visualforce page. -> Incorrect. Including a Visualforce page is not a requirement for deploying Apex code.

Apex code must be accompanied by relevant Lightning components for successful deployment. -> Incorrect. Apex code deployment is independent of Lightning components unless specifically related to the functionality being deployed.

A Salesforce developer is designing a Lightning Web Component (LWC) application that requires components to communicate efficiently. Which scenario best illustrates the use case and best practices for handling events in Lightning Web Component (LWC)?

Creating a parent component that listens for custom events dispatched by child components to orchestrate complex interactions and data sharing. Correct answer

Emitting custom events with detailed payload for simple inter-component messages, even when only a signal is needed.

Using standard HTML events to manage all component communications, avoiding custom events to reduce complexity.

Dispatching events from child components directly to unrelated components, bypassing the containment hierarchy.

Live Agent

Incorrect

Creating a parent component that listens for custom events dispatched by child components to orchestrate complex interactions and data sharing. -> Correct. This approach follows best practices for component communication in LWC, leveraging custom events to enable child-to-parent communication. It allows the parent component to act as a mediator or orchestrator for interactions between child components, ensuring loose coupling and modularity.

Using standard HTML events to manage all component communications, avoiding custom events to reduce complexity. -> Incorrect. While standard HTML events are useful for common user interactions, they are not sufficient for all types of component communication, especially for custom data or complex interactions specific to the application's logic. Custom events are necessary for these scenarios.

Emitting custom events with detailed payload for simple inter-component messages, even when only a signal is needed. -> Incorrect. Custom events should be used judiciously, with payloads only as complex as needed. For simple signals where no data is required, an event with no payload or a very simple one is more efficient and adheres to best practices.

Dispatching events from child components directly to unrelated components, bypassing the containment hierarchy. -> Incorrect. Events in LWC are designed to bubble up the containment hierarchy and are not intended to communicate directly between unrelated components. This approach would violate the encapsulation principles of LWC and could lead to maintenance and scalability issues.

How does the Salesforce platform support the principle of data encapsulation and object-oriented programming?

Utilizing Apex Classes to define data structures and methods for data manipulation. Correct answer

Through the Schema Builder, allowing the definition of objects and fields without co

Live Agent

By using Visualforce pages to encapsulate data and logic within standard controllers.

Through the use of List Views to encapsulate data fields and display them based on user permissions.

Incorrect

Utilizing Apex Classes to define data structures and methods for data manipulation. -> Correct. Apex Classes allow developers to write object-oriented code, encapsulating data (fields/properties) and behavior (methods) in a single unit, adhering to the principles of object-oriented programming.

By using Visualforce pages to encapsulate data and logic within standard controllers. -> Incorrect. Visualforce pages are used for defining user interfaces in Salesforce, not for encapsulating data and business logic, which are typically handled by controllers and Apex classes, respectively.

Through the Schema Builder, allowing the definition of objects and fields without code. -> Incorrect. The Schema Builder is a tool for visually designing database schemas, not for implementing object-oriented programming concepts like data encapsulation.

Through the use of List Views to encapsulate data fields and display them based on user permissions. -> Incorrect. List Views are for displaying records from a specific object in Salesforce based on criteria; they do not encapsulate data or logic in the sense required by object-oriented programming principles.

A Salesforce Developer needs to implement an Apex trigger for a high-volume custom object, CustomObj__c, to update related records based on changes to CustomObj__c. This object can have thousands of records updated in a single transaction. Considering the Salesforce governor limits, which approach should the developer take to ensure the trigger can handle bulk operations without hitting governor limits?

Use a static variable within the trigger to prevent it from executing more than once in a transaction.

Query related records within a for loop for each CustomObj__c record that is being processed.

Directly perform DML operations on related records within a loop each time a CustomObj__c record is processed.

Collect the IDs of all CustomObj__c records in a single list, then perform a single SOQL query outside of loops to retrieve related records, and process them in bulk. Correct answer

Incorrect

Collect the IDs of all CustomObj__c records in a single list, then perform a single SOQL query outside of loops to retrieve related records, and process them in bulk. -> Correct. This method follows best practices for bulkifying Apex code, reducing the number of SOQL queries, and efficiently handling large volumes of data without exceeding governor limits.

Query related records within a for loop for each CustomObj__c record that is being processed. -> Incorrect. This approach will quickly hit the governor limit for SOQL queries (100 SOQL queries per transaction) due to querying within a loop, which is not scalable for bulk operations.

Use a static variable within the trigger to prevent it from executing more than once in a transaction. -> Incorrect. While using a static variable can prevent reentrant trigger execution, it does not address the main concern of optimizing SOQL queries and DML operations for bulk processing and avoiding governor limits.

Directly perform DML operations on related records within a loop each time a CustomObj__c record is processed. -> Incorrect. Performing DML operations within a loop is a common mistake that can lead to exceeding the governor limit on DML operations (150 DML operations per transaction) and does not scale for bulk data processing.

A Salesforce developer needs to implement a custom user interface component that displays a dynamic list of records from a custom object, “Projects”. This list should allow users to interact with each record, such as editing or deleting, directly from the component. The solution must also be compatible across Salesforce’s mobile and desktop environments. Which technology should the developer use to meet these requirements?

Visualforce Pages

Apex Classes

Lightning Components (Aura) Correct answer

Salesforce Flow

Incorrect

Lightning Components (Aura) -> Correct. Lightning Components (Aura) provide a robust framework for creating custom, reusable, and dynamic user interface components that are compatible across Salesforce mobile and desktop environments.

Apex Classes -> Incorrect. Apex Classes are used for the business logic layer in Salesforce, not for creating user interface components.

Visualforce Pages -> Incorrect. While Visualforce Pages can be used to create custom user interfaces, they are not as responsive or mobile-friendly as newer technologies like Lightning Web Components.

Salesforce Flow -> Incorrect. Salesforce Flow is a powerful tool for automating business processes but is not primarily used for creating custom user interface components.

When writing and executing tests for triggers, controllers, classes, flows, and processes in Salesforce, which two practices should be followed? Select two correct answers.

- [ ] Use hard-coded IDs retrieved from your production environment to reference records needed for tests. Correct answer
- [ ] Systematically assert the behavior of the code by checking the values of variables and the database state after operations.
- [ ] Avoid writing test methods for triggers and flows since these are automatically tested by Salesforce during deployments.
- [ ] Test methods should ideally cover both positive and negative test cases to ensure comprehensive coverage. Correct answer
- [ ] Use the SeeAllData=true attribute on test classes to access real, existing data for more realistic test outcomes.

Incorrect

Systematically assert the behavior of the code by checking the values of variables and the database state after operations. -> Correct. Assertions are crucial in test methods to verify that the code behaves as expected. Checking variables and database states after operations ensures that the code performs correctly under various conditions.

Test methods should ideally cover both positive and negative test cases to ensure comprehensive coverage. -> Correct. Covering both positive and negative test cases ensures that your code handles expected paths correctly and gracefully degrades or handles errors as expected, which is essential for robust software.

Use the SeeAllData=true attribute on test classes to access real, existing data for more realistic test outcomes. -> Incorrect. Using SeeAllData=true should be avoided when possible because it can lead to tests that are dependent on the organization's data, whi can change and cause tests to fail unpredictably. Salesforce recommends creating test within test methods.

Avoid writing test methods for triggers and flows since these are automatically tested by Salesforce during deployments. -> Incorrect. Salesforce does not automatically test triggers and flows during deployments. Developers must write test methods to ensure these components work as expected and meet the required code coverage for deployments.

Use hard-coded IDs retrieved from your production environment to reference records needed for tests. -> Incorrect. Hard-coding IDs from production is not a best practice because these IDs will not exist in the test execution context. Salesforce recommends creating necessary test data within the test methods or using test setup methods.

Which situation would prevent a system administrator from deleting a custom object in Salesforce?

The object has existing records.

The object is referenced in a custom report type.

The object is included in a dashboard.

The object is used in an Apex class. Correct answer

Incorrect

The object is used in an Apex class. -> Correct. If a custom object is referenced within an Apex class, it cannot be deleted without first removing or modifying the Apex code that references it.

The object has existing records. -> Incorrect. While the presence of records may require additional steps before deletion, it does not inherently prevent the deletion of a custom object.

The object is referenced in a custom report type. -> Incorrect. Although referenced in custom report types, objects can still be deleted; however, the custom report type may

affected or deleted as a result.

The object is included in a dashboard. -> Incorrect. Objects used in dashboards do not prevent deletion, but deleting an object could impact dashboard components that rely on that object's data.

A developer needs to write an Apex method to find all Opportunities that are in the "Closed Won" stage and have a closing date in the current fiscal quarter. Then, the method must update a custom field HasBeenReviewed__c on each retrieved Opportunity to true. Which sequence of steps should the developer take to accomplish this using SOSL, SOQL, and DML statements?

Use SOQL to query Opportunities; Use a For loop to set HasBeenReviewed__c to true; Use DML to update the records. Correct answer

Use DML to search for and update Opportunities in a single step.

Use SOSL to find Opportunities; Use DML to update HasBeenReviewed__c directly.

Use SOQL to query Opportunities; Use a Switch statement to update HasBeenReviewed__c; Use DML to update.

Incorrect

Use SOQL to query Opportunities; Use a For loop to set HasBeenReviewed__c to true; Use DML to update the records. -> Correct. SOQL is used to precisely query records based on specific criteria. A for loop can iterate through the results to modify each record, followed by a DML operation to commit the changes.

Use SOSL to find Opportunities; Use DML to update HasBeenReviewed__c directly. -> Incorrect. SOSL is designed for searching large text fields and global search across multiple objects, not for querying records based on specific criteria like stage and closing date.

Use DML to search for and update Opportunities in a single step. -> Incorrect. DML operations are used for inserting, updating, deleting, and undeleting records, not for searching or querying records.

Use SOQL to query Opportunities; Use a Switch statement to update HasBeenReviewed__c; Use DML to update. -> Incorrect. While SOQL is correctly used for querying, a Switch statement is not necessary for this operation. A for loop is more appropriate for iterating and updating each record before using DML to commit the updates.

In Salesforce, which declarative process automation feature is specifically designed to create complex, multi-step automated processes that can handle decision logic, loop through data, and update records?

Approval Processes

Flow — Correct answer

Workflow Rules

Apex Triggers

Process Builder

Incorrect

Flow -> Correct. Flow is the most versatile declarative automation tool in Salesforce, capable of handling complex logic, looping through data, and updating records. It allows for the creation of multi-step processes with decision branches.

Process Builder -> Incorrect. While Process Builder can automate complex processes b on record changes and can call flows, it does not inherently support looping through da

handling complex logic to the same extent as Flow.

Workflow Rules -> Incorrect. Workflow Rules are more limited in scope, designed for simple automatic actions like sending email alerts, creating tasks, or updating fields without the capacity for multi-step logic or looping.

Approval Processes -> Incorrect. Approval Processes manage the approval of records in Salesforce. They do not facilitate the creation of multi-step processes or handle complex logic and data manipulation outside of approvals.

Apex Triggers -> Incorrect. Apex Triggers are programmatic solutions for handling complex logic that executes before or after database operations. Unlike Flow, they require coding knowledge and are not considered a declarative automation feature.

When developing an Apex method to categorize a list of custom object records named CustomObject__c based on a numeric field Amount__c, which control flow statement combination is most efficient for sorting these records into three categories: 'Low', 'Medium', and 'High', with thresholds at $10,000 and $50,000?

Implement a single switch statement on the Amount__c field of CustomObject__c records to categorize them directly into 'Low', 'Medium', and 'High'.

Utilize a for loop to iterate over CustomObject__c records and nested if statements to assign categories based on the Amount__c field values. Correct answer

Use a while loop with a counter to go through each CustomObject__c record, applying a switch statement on Amount__c for categorization.

Create a do-while loop to process each CustomObject__c record, using if-else-if statements to check Amount__c and categorize the records.

Incorrect

Utilize a for loop to iterate over CustomObject__c records and nested if statements to assign categories based on the Amount__c field values. -> Correct. This approach is straightforward and efficient for categorizing records based on numeric thresholds. The for loop allows for seamless iteration over the list of records, while nested if statements enable precise categorization based on the Amount__c field, making it easy to implement and understand the logic for threshold-based categorization.

Implement a single switch statement on the Amount__c field of CustomObject__c records to categorize them directly into 'Low', 'Medium', and 'High'. -> Incorrect. Apex's switch statement does not directly support range-based cases (like numeric ranges for 'Low', 'Medium', and 'High'), making it unsuitable for this scenario without additional logic to determine the case expressions.

Use a while loop with a counter to go through each CustomObject__c record, applying a switch statement on Amount__c for categorization. -> Incorrect. While a while loop could theoretically be used, it is less practical for iterating over a list of records in Apex due to the need for manual index management. Moreover, as mentioned, a switch statement does not support range-based categorization directly.

Create a do-while loop to process each CustomObject__c record, using if-else-if statements to check Amount__c and categorize the records. -> Incorrect. A do-while loop ensures that the loop's body is executed at least once before checking the condition, which might not be necessary or efficient for this scenario. While if-else-if statements could categorize records based on Amount__c, the initial use of a do-while loop is less conventional for this case compared to a straightforward for loop.

In the Salesforce ecosystem, how does the Lightning Component Framework leverage the multi-tenant architecture to enhance application development?

Correct answer

It employs MVC architecture by strictly enforcing a separation of concerns, where Lightning components represent the View and Controller, and Apex code acts as the Model.

It requires developers to manage their own server infrastructure for hosting Lightning components.

It allows developers to directly modify the Salesforce core platform code for custom functionality.

It uses a single shared data model for all tenants, allowing custom Lightning components to directly access and manipulate data across multiple tenants.

Incorrect

It employs MVC architecture by strictly enforcing a separation of concerns, where Lightning components represent the View and Controller, and Apex code acts as the Model. -> Correct. The Lightning Component Framework aligns with the MVC pattern by separating the user interface (View) and logic (Controller) within components, while Apex code interacts with the database (Model), ensuring a clear separation of concerns.

It allows developers to directly modify the Salesforce core platform code for custom functionality. -> Incorrect. The Lightning Component Framework provides a component-based development model. It does not allow direct modification of Salesforce core platform code but enables developers to create reusable components and applications on top of it.

It requires developers to manage their own server infrastructure for hosting Lightning components. -> Incorrect. Salesforce's multi-tenant architecture hosts and manages the server infrastructure, ensuring that developers focus on building applications without worrying about underlying servers.

It uses a single shared data model for all tenants, allowing custom Lightning components to directly access and manipulate data across multiple tenants. -> Incorrect. While Salesforce operates on a multi-tenant architecture, each tenant's data is isolated. Lightning components operate within a tenant's context and cannot directly access data from other tenants.

Given a scenario where a Salesforce developer needs to design a data model to manage a custom project management application within Salesforce, which two actions correctly describe best practices in determining, creating, and accessing the appropriate data model including objects, fields, relationships, and external IDs? Select two correct answers.

- [ ] Employ formula fields to establish relationships between objects, enabling dynamic referencing of data across related objects without using relationship fields.
- [ ] Utilize external ID fields on custom objects to facilitate efficient and reliable record matching when integrating with external systems. Correct answer
- [ ] Use standard objects for all project management features to ensure rapid development and avoid the complexity of custom object permissions and relationships.
- [ ] Define a master-detail relationship between the Task and Project objects to tightly couple task records to their respective projects, ensuring data integrity and cascading deletions. Correct answer
- [ ] Create a custom object for projects and use auto-number fields as the primary method for tracking project uniqueness and external integrations.

Incorrect

Utilize external ID fields on custom objects to facilitate efficient and reliable record matching when integrating with external systems. -> Correct. External ID fields are designed for integration purposes, allowing Salesforce to efficiently identify records during upsert operations, making them ideal for linking Salesforce records with external system records.

Define a master-detail relationship between the Task and Project objects to tightly couple task records to their respective projects, ensuring data integrity and cascading deletions. -> Correct. Master-detail relationships are perfect for situations where there is a clear ownership and dependent relationship between records. This setup ensures that wher

Project record is deleted, all associated Task records are automatically deleted, maintaining data cleanliness.

Create a custom object for projects and use auto-number fields as the primary method for tracking project uniqueness and external integrations. -> Incorrect. While auto-number fields are useful for internal uniqueness, they are not suitable for external integrations, where external IDs are specifically designed to facilitate record matching and integration with external systems.

Employ formula fields to establish relationships between objects, enabling dynamic referencing of data across related objects without using relationship fields. -> Incorrect. Formula fields are used for calculations and deriving values based on other fields; they cannot establish relationships between objects. Relationships are defined by lookup or master-detail fields.

Use standard objects for all project management features to ensure rapid development and avoid the complexity of custom object permissions and relationships. -> Incorrect. While leveraging standard objects can speed up development, custom project management requirements often necessitate custom objects to capture unique data, enforce business rules, and establish specific relationships that standard objects cannot accommodate.

Universal Containers wishes to automate the process where any Opportunity that reaches the ‘Negotiation/Review’ stage triggers a discount approval request to the sales manager and logs this activity for audit purposes. Which Salesforce feature should be used to automate this process?

Validation Rules

Process Builder — Correct answer

Apex Code

Workflow Rules

Incorrect

Process Builder -> Correct. Process Builder allows for the automation of complex workflows, including sending approval requests and logging activities when specific criteria are met, such as an Opportunity reaching a certain stage.

Apex Code -> Incorrect. While Apex Code can be used for complex custom automations, it's not necessary for this requirement as Process Builder offers a no-code solution.

Workflow Rules -> Incorrect. Workflow rules can automate tasks, email alerts, and field updates but do not directly handle approval processes or complex conditional logic involving multiple actions like logging for audit purposes.

Validation Rules -> Incorrect. Validation rules enforce data quality by ensuring that the data entered by users meets certain criteria before it can be saved, but they do not automate processes or actions.

In a scenario where an Apex class is required to update contact records based on data from related account records, which approach best demonstrates an understanding of Salesforce governor limits and ensures that the Apex transaction remains within these limits?

Use a SOQL query within a for loop to retrieve and update each contact record related to an account, ensuring each contact is updated immediately after retrieval.

Collect all updates in a List and perform a single DML operation outside the loop after collecting all necessary changes to contact records. Correct answer

Perform DML operations individually on each contact record inside a for loop to ensure real-time processing and error handling for each record.

Use static variables to aggregate data from account records before starting the loop to update contact records, reducing the need for SOQL queries.

Incorrect

Collect all updates in a List and perform a single DML operation outside the loop after collecting all necessary changes to contact records. -> Correct. This approach minimizes the number of DML operations by bulkifying them, which is essential for staying within governor limits and enhancing performance.

Use a SOQL query within a for loop to retrieve and update each contact record related to an account, ensuring each contact is updated immediately after retrieval. -> Incorrect. Placing SOQL queries inside for loops can rapidly exhaust governor limits by performing too many queries in a single transaction.

Perform DML operations individually on each contact record inside a for loop to ensure real-time processing and error handling for each record. -> Incorrect. Executing DML operations within a for loop is inefficient and risks exceeding the governor limit on DML operations.

Use static variables to aggregate data from account records before starting the loop to update contact records, reducing the need for SOQL queries. -> Incorrect. While reducing SOQL queries is beneficial, the use of static variables does not directly mitigate the risk of exceeding specific governor limits related to SOQL queries and DML operations.