COURSE PROGRESS: 0% COMPLETE

Salesforce Platform Developer I – Practice Test #1

Results

0 of 40 Questions answered correctly

Your time: 00:01:17

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								

A Salesforce developer is tasked with designing a robust error handling strategy for a complex Apex class that integrates Salesforce data with an external service. The integration process involves multiple steps, including data validation, transformation, and transmission over the network. The developer needs to ensure that the system gracefully handles both expected and unexpected errors, allowing for appropriate logging and user notification where necessary. Given this scenario, how should the developer implement exception handling in Apex, including the use of custom exceptions?

Live Agent

Use a try-catch block around the entire integration process code, catching only generic Exception types, to simplify the error handling logic.

Avoid using custom exceptions entirely, relying on built-in Salesforce exceptions for all error handling to reduce complexity.

Correct answer

Implement multiple try-catch blocks throughout the code, catching specific Salesforce exceptions and defining custom exceptions for domain-specific errors, allowing for detailed error handling and user feedback.

Implement a global exception handler class that uses reflection to dynamically handle exceptions, minimizing the need for explicit try-catch blocks in the integration code.

Incorrect

Implement multiple try-catch blocks throughout the code, catching specific Salesforce exceptions and defining custom exceptions for domain-specific errors, allowing for detailed error handling and user feedback. -> Correct. This method provides granular error handling, enabling the developer to address specific issues uniquely and use custom exceptions for more precise error categorization and handling.

Use a try-catch block around the entire integration process code, catching only generic Exception types, to simplify the error handling logic. -> Incorrect. This approach does not take full advantage of the granularity provided by catching specific exception types or the use of custom exceptions to handle different error conditions appropriately.

Avoid using custom exceptions entirely, relying on built-in Salesforce exceptions for all error handling to reduce complexity. -> Incorrect. While Salesforce exceptions handle many common error conditions, custom exceptions allow for handling specific error scenarios not covered by standard exceptions, offering more tailored error management.

Implement a global exception handler class that uses reflection to dynamically handle exceptions, minimizing the need for explicit try-catch blocks in the integration code. -> Incorrect. Salesforce Apex does not support reflection in this manner, and such an appr

Live Agent

would not provide the necessary specificity and control over error handling for different parts of the integration process.

Given a scenario where you need to retrieve the name and email of all contacts related to the "Acme" account in Apex, which of the following SOQL queries is correctly written to accomplish this task?

SELECT Name, Email FROM Account WHERE Name = 'Acme'

SELECT Name, Email FROM Contact WHERE Account.Name = 'Acme' **Correct answer**

FIND {Acme} IN ALL FIELDS RETURNING Contact(Name, Email)

SELECT Name, Email FROM Contact WHERE Account.Name LIKE 'Acme'

Incorrect

SELECT Name, Email FROM Contact WHERE Account.Name = 'Acme' -> Correct. This query correctly uses SOQL syntax to retrieve the name and email of contacts associated with the "Acme" account by filtering on the related Account's Name field.

FIND {Acme} IN ALL FIELDS RETURNING Contact(Name, Email -> Incorrect. This is an example of SOSL, not SOQL, which is used for different types of searches that span multiple objects and fields. It's not specific enough for the given scenario.

SELECT Name, Email FROM Contact WHERE Account.Name LIKE 'Acme' -> Incorrect. While this SOQL query attempts to filter contacts related to the "Acme" account, the use of LIKE without wildcards is unnecessary; exact matches should use =.

SELECT Name, Email FROM Account WHERE Name = 'Acme' -> Incorrect. This query incorrectly attempts to retrieve contact information from the Account object. The

Live Agent

requirement is to retrieve contact information, not account information, thus needing to query the Contact object and filter based on the related Account’s Name.

When designing an Apex trigger for a Salesforce application, understanding the relationship between Apex transactions, save order of execution, and managing the potential for recursion and cascading operations is crucial. Which of the following strategies best addresses these considerations to ensure efficient and reliable trigger execution?

Utilize separate triggers for before and after contexts for each DML operation (insert, update, delete) to clearly define and manage the execution order.

Correct answer

Implement a static variable within a class that acts as a trigger handler to prevent the same trigger from executing more than once in the same transaction, effectively managing recursion.

Code all business logic directly inside each trigger to ensure transactions are handled efficiently and to simplify the save order of execution.

Rely solely on Salesforce's built-in transaction management to automatically prevent recursion and manage the save order of execution, without additional coding.

Incorrect

Implement a static variable within a class that acts as a trigger handler to prevent the same trigger from executing more than once in the same transaction, effectively managing recursion. -> Correct. This is a recognized method to prevent trigger recursion. By using static variables to track execution, developers can prevent the same logic from running multiple times within a single transaction, addressing recursion directly.

Code all business logic directly inside each trigger to ensure transactions are handled efficiently and to simplify the save order of execution. -> Incorrect. Placing all logic directly in triggers can lead to complex, hard-to-maintain code and does not directly address th

Live Agent

challenges of recursion or cascading operations.

Utilize separate triggers for before and after contexts for each DML operation (insert, update, delete) to clearly define and manage the execution order. -> Incorrect. While organizing code this way might seem to offer clarity, it does not inherently manage recursion or cascading and can lead to redundant code execution and complexity.

Rely solely on Salesforce's built-in transaction management to automatically prevent recursion and manage the save order of execution, without additional coding. -> Incorrect. Salesforce provides mechanisms to manage transactions, but developers are responsible for explicitly coding to prevent recursion and properly manage execution order within triggers.

In a scenario where a Salesforce developer needs to debug a failure in a scheduled batch Apex job that processes large data volumes overnight, which tool or feature should be utilized first to identify and analyze the cause of the failure?

Modify the Apex code to include additional System.debug statements and redeploy.

Use the Salesforce CLI to directly query the AsyncApexJob object for detailed logs.

Check the Apex Jobs page in Setup to review the job status and error messages.

Utilize the Developer Console's Debug Logs to trace the execution of the batch job and identify errors. Correct answer

Incorrect

Utilize the Developer Console's Debug Logs to trace the execution of the batch job and identify errors. -> Correct. The Developer Console's Debug Logs provide detailed execution logs, including errors and execution paths, making it an ideal starting point for debugging batch job failures.

Modify the Apex code to include additional System.debug statements and redeploy. -> Incorrect. While adding debug statements can be helpful, it's not the first step for

Live Agent

debugging existing failures as it requires code modification and redeployment, which may not be immediately feasible.

Check the Apex Jobs page in Setup to review the job status and error messages. -> Incorrect. The Apex Jobs page gives a high-level overview of job statuses and basic error messages but lacks the detailed execution context needed for in-depth debugging.

Use the Salesforce CLI to directly query the AsyncApexJob object for detailed logs. -> Incorrect. Querying the AsyncApexJob object can provide job status and basic error information but does not offer the detailed execution logs needed for thorough analysis.

A Salesforce development team is planning to build a custom application within a multi-tenant environment that requires high performance, scalability, and adherence to Salesforce's best practices for development. The application will manage customer support cases and should provide a user-friendly interface for support agents to view, update, and manage cases efficiently. The team needs to decide on an architectural approach that leverages Salesforce's multi-tenant architecture and design frameworks effectively.
The application must:
Be responsive and provide a seamless experience across various devices (desktop, tablet, mobile).
Follow the Model-View-Controller (MVC) design pattern to separate the logic, UI, and data model.
Utilize Salesforce's built-in features and capabilities to ensure maintainability and upgradeability.
Considering the requirements, which technology should the development team use to build the application?

Aura Components

Visualforce Pages with Standard Controllers

Apex Classes with Web Services

Live Agent

Correct answer

Lightning Web Components (LWC)

Salesforce Mobile SDK

Incorrect

Lightning Web Components (LWC) -> Correct. Lightning Web Components (LWC) offer a modern framework that is standards-based and leverages web components for building efficient and scalable applications. LWC provides a responsive user experience across devices, adheres to the MVC architecture through its componentized design, and is optimized for performance in Salesforce's multi-tenant environment.

Visualforce Pages with Standard Controllers -> Incorrect. While Visualforce pages can provide a custom UI and leverage standard controllers for MVC adherence, they may not offer the best performance and responsiveness across all devices compared to more modern frameworks.

Aura Components -> Incorrect. Aura Components allow for the development of responsive applications and adhere to the MVC pattern. However, they are not the latest technology offered by Salesforce for creating user interfaces, and newer frameworks might provide better performance and developer experience.

Apex Classes with Web Services -> Incorrect. Apex classes can be used to build the application's logic and expose web services, but this approach does not directly address the UI requirements or fully leverage Salesforce's design frameworks for a seamless user interface.

Salesforce Mobile SDK -> Incorrect. The Salesforce Mobile SDK is primarily used for developing mobile applications that connect to Salesforce data. While it supports mobile responsiveness, it is not specifically designed for creating comprehensive web applications within Salesforce and does not inherently follow the MVC architecture.

Live Agent

A Salesforce developer needs to integrate a Salesforce org with an external inventory management system to synchronize product information. The external system assigns a unique identifier to each product. Which approach should the developer take to ensure efficient and reliable integration between Salesforce and the external inventory management system?

Utilize an External ID field on the Product object to store the external system's unique identifiers.

Correct answer

Create a custom object to store the external system's unique identifiers and map them to corresponding Salesforce records.

Implement a formula field on the Product object to generate a unique identifier that matches the external system's format.

Use a standard field on the Product object as the unique identifier to match records between systems.

Incorrect

Utilize an External ID field on the Product object to store the external system's unique identifiers. -> Correct. External ID fields are specifically designed for this purpose, allowing developers to efficiently map and synchronize records between Salesforce and external systems using a unique identifier.

Use a standard field on the Product object as the unique identifier to match records between systems. -> Incorrect. Standard fields may not reliably match the unique identifiers from an external system, as they are not designed to store external unique IDs.

Create a custom object to store the external system's unique identifiers and map them to corresponding Salesforce records. -> Incorrect. While creating a custom object can store additional data, it's more efficient to use a field designed for external identifiers directly on related Salesforce objects.

Implement a formula field on the Product object to generate a unique identifier that matches the external system's format. -> Incorrect. Formula fields calculate values bas

Live Agent

other fields and cannot be used to store or synchronize external unique identifiers.

In the context of Apex development for Salesforce, especially regarding declarations and accessibility, which two of the following statements are true concerning the declaration of variables, constants, methods, and the use of modifiers and interfaces?

- ☐ Variables declared with the public modifier are accessible anywhere within the application, including external applications.
- ☐ The static keyword in Apex is used to declare variables or methods that belong to the class itself rather than any instance of the class, making them accessible without instantiating the class. Correct answer
- ☐ The interface keyword in Apex is used to define a contract that classes can implement, specifying the methods that must be implemented without providing their implementation. Correct answer
- ☐ The final keyword in Apex is used to prevent a class from being extended by any subclass.

Incorrect

The interface keyword in Apex is used to define a contract that classes can implement, specifying the methods that must be implemented without providing their implementation. -> Correct. Apex uses the interface keyword to declare an interface, which is a contract that any implementing class must follow. It specifies method signatures without providing their actual implementation, enforcing a certain structure while allowing for different implementations.

The static keyword in Apex is used to declare variables or methods that belong to the class itself rather than any instance of the class, making them accessible without instantiating the class. -> Correct. In Apex, the static keyword is indeed used to indicate that a variable or method is static, meaning it belongs to the class and not to any particular object instan

Live Agent

Static methods and variables can be accessed without creating an instance of the class

Variables declared with the public modifier are accessible anywhere within the application, including external applications. -> Incorrect. While variables declared as public are accessible from any class within the same application or namespace, they are not directly accessible by external applications unless exposed through global methods or API endpoints.

The final keyword in Apex is used to prevent a class from being extended by any subclass. -> Incorrect. In Apex, the final keyword is used to declare variables whose values cannot be changed once they are initialized, not to prevent a class from being extended. To prevent class extension, the virtual keyword is omitted since Apex classes are non-virtual by default.

In developing a complex Apex application that interacts with various Salesforce objects and external APIs, you decide to implement a comprehensive exception handling strategy to ensure reliability and maintainability. Part of your strategy includes the utilization of custom exceptions for specific error scenarios. Which two of the following practices are recommended when implementing exception handling in Apex, including the creation and use of custom exceptions?

- ☐ Implement a finally block to execute cleanup code, such as closing open resources, whether or not an exception is thrown. Correct answer
- ☐ Wrap every DML operation in a separate try-catch block to catch DmlException and rethrow it as a custom exception.
- ☐ Use a single try-catch block to catch all exceptions and handle them generically, regardless of the error source.
- ☐ Create a custom exception class that extends the Exception class and includes additional context information in its constructor. Correct answer

Incorrect

Create a custom exception class that extends the Exception class and includes additior

context information in its constructor. -> Correct. Extending the Exception class to cre

Live Agent

custom exception allows for more specific error handling and can include additional information about the error context, enhancing error reporting and debugging.

Implement a finally block to execute cleanup code, such as closing open resources, whether or not an exception is thrown. -> Correct. The finally block is crucial for ensuring that necessary cleanup actions are performed, such as releasing resources, even if an exception occurs, thereby preventing resource leaks.

Wrap every DML operation in a separate try-catch block to catch DmlException and rethrow it as a custom exception. -> Incorrect. While catching DmlException is a common practice, rethrowing every exception as a custom exception is unnecessary and could lead to code that is harder to maintain and debug.

Use a single try-catch block to catch all exceptions and handle them generically, regardless of the error source. -> Incorrect. This approach does not take full advantage of Apex's exception handling capabilities, such as catching specific exception types or using custom exceptions for more granular control.

Given a requirement to automate a business process that involves updating related records based on certain criteria and performing complex calculations that are not supported by declarative tools, how should a Salesforce Developer integrate declarative functionality with Apex to achieve an optimal solution?

Use Workflow Rules for the updates and write Apex triggers for the complex calculations, ensuring that they work independently of each other.

Create a Visualforce page to handle both the record updates and calculations, bypassing the need for separate declarative and programmatic solutions.

Develop the entire solution in Apex, including the updates and calculations, to maintain consistency in the implementation approach.

Correct a

Implement the logic for updating related records using Process Builder or Flow, and Apex for the complex calculations, invoking Apex from the declarative tools when

Live Agent

necessary.

Incorrect

Implement the logic for updating related records using Process Builder or Flow, and use Apex for the complex calculations, invoking Apex from the declarative tools when necessary. -> Correct. This approach utilizes the strengths of both declarative tools and Apex. Process Builder or Flow can efficiently handle updates to related records based on criteria, while Apex can be used for complex calculations that are beyond the capabilities of declarative tools. This method also allows for invoking Apex from within a Flow or Process Builder, providing a seamless integration.

Develop the entire solution in Apex, including the updates and calculations, to maintain consistency in the implementation approach. -> Incorrect. While developing the entire solution in Apex offers control and flexibility, it disregards the efficiency and simplicity that declarative tools can provide for certain aspects of the business logic, such as updating related records based on criteria.

Use Workflow Rules for the updates and write Apex triggers for the complex calculations, ensuring that they work independently of each other. -> Incorrect. Using Workflow Rules for updates and Apex triggers for calculations can work, but it may not leverage the full potential of Salesforce's declarative tools, like Process Builder or Flow, which offer more advanced capabilities and easier integration with Apex when needed.

Create a Visualforce page to handle both the record updates and calculations, bypassing the need for separate declarative and programmatic solutions. -> Incorrect. Creating a Visualforce page for handling logic that could be managed through a combination of declarative tools and Apex is not the most efficient or scalable approach. It also requires additional development and maintenance effort compared to leveraging built-in Salesforce functionality.

Live Agent

In an Apex development scenario that requires implementing a complex business logic that interacts with multiple related Salesforce objects, what is the best practice for organizing your Apex classes and triggers to ensure code maintainability, efficiency, and adherence to Salesforce governor limits?

Create separate triggers for each operation (insert, update, delete, undelete) on each object to keep the logic simple and focused.

Correct answer

Utilize a Trigger Framework that delegates operations to handler classes, ensuring a single trigger per object which calls specific methods based on the trigger context.

Avoid using custom classes and focus on implementing logic within formula fields and workflow rules to reduce the need for testing.

Implement all business logic directly in triggers to ensure that the logic is executed anytime the related objects are modified.

Incorrect

Utilize a Trigger Framework that delegates operations to handler classes, ensuring a single trigger per object which calls specific methods based on the trigger context. -> Correct. This approach helps in organizing code, makes maintenance easier, ensures better control over execution order, and aids in managing governor limits efficiently.

Implement all business logic directly in triggers to ensure that the logic is executed anytime the related objects are modified. -> Incorrect. Directly coding complex logic in triggers can lead to maintenance issues and difficulties in managing governor limits.

Create separate triggers for each operation (insert, update, delete, undelete) on each object to keep the logic simple and focused. -> Incorrect. Creating separate triggers for each operation can lead to redundant code, increase the complexity of managing related operations, and make it harder to control the execution order.

Avoid using custom classes and focus on implementing logic within formula fields and workflow rules to reduce the need for testing. -> Incorrect. While formula fields and

Live Agent

workflow rules can simplify some logic, they cannot cover complex business requirements that require conditional logic, loops, or interactions with multiple objects, which are better handled in Apex.

In the context of building a Salesforce Lightning Web Component (LWC), which two of the following options correctly describe the use cases and best practices for utilizing LWC events?

- [ ] Implementing global application events to communicate state changes across all components in the application, regardless of their hierarchical relationship.
- [ ] Capturing a user's action within a component, such as clicking a button or entering text, to trigger a data fetch or a UI update. Correct answer
- [ ] Using system events to directly manipulate the DOM elements of another component for real-time UI updates.
- [ ] Broadcasting a change in one component's state to a sibling component within the same application.
- [ ] Notifying a parent component about a change in a child component's state, such as a user selection or input. Correct answer

Incorrect

Notifying a parent component about a change in a child component's state, such as a user selection or input. -> Correct. Custom events in LWC are designed for child-to-parent communication, allowing child components to inform parent components about state changes or user actions.

Capturing a user's action within a component, such as clicking a button or entering text, to trigger a data fetch or a UI update. -> Correct. LWC supports the use of standard DOM events, like clicks and keypresses, within components to respond to user actions with handlers that can trigger UI updates or data operations.

Live Agent

Broadcasting a change in one component's state to a sibling component within the same application. -> Incorrect. Direct sibling component communication is not a typical use case for LWC events. LWC recommends using a shared parent component or a pub-sub model for indirect communication between sibling components.

Implementing global application events to communicate state changes across all components in the application, regardless of their hierarchical relationship. -> Incorrect. LWC does not have a native concept of global application events like Aura components. For application-wide communication, LWC suggests using a service component or a pub-sub model.

Using system events to directly manipulate the DOM elements of another component for real-time UI updates. -> Incorrect. Direct DOM manipulation of another component's elements is discouraged in LWC. Components should communicate via events and data binding to maintain encapsulation and reactivity.

In developing a Salesforce application, how should a developer design the application to prevent user interface and data access security vulnerabilities, especially when handling sensitive data?

Embed sensitive information within Visualforce pages to optimize load times and reduce server-side calls.

Disable sharing rules in Apex by default to streamline data access and improve performance for all users.

Correct answer

Apply CRUD and FLS checks in Apex controllers before performing data operations to ensure data security and compliance with user permissions.

Utilize unescaped dynamic SOQL queries based on user input to provide flexible data retrieval options.

Live Agent

Incorrect

Apply CRUD and FLS checks in Apex controllers before performing data operations to ensure data security and compliance with user permissions. -> Correct. Enforcing CRUD (Create, Read, Update, Delete) and FLS (Field-Level Security) checks in Apex controllers ensures that the application respects the security settings and permissions configured in Salesforce, preventing unauthorized data access and modifications.

Embed sensitive information within Visualforce pages to optimize load times and reduce server-side calls. -> Incorrect. Embedding sensitive information directly within Visualforce pages can expose it to security risks such as cross-site scripting (XSS) attacks. Sensitive data should be securely handled on the server side and only displayed when necessary, with proper encoding and validation.

Utilize unescaped dynamic SOQL queries based on user input to provide flexible data retrieval options. -> Incorrect. Using dynamic SOQL queries without proper input validation and escaping can lead to SOQL injection vulnerabilities, where attackers could manipulate queries to access unauthorized data. Always validate and sanitize user input used in dynamic queries.

Disable sharing rules in Apex by default to streamline data access and improve performance for all users. -> Incorrect. Disabling sharing rules (running Apex in without sharing mode) can inadvertently give users access to data they should not see, violating data security and privacy principles. It's crucial to apply appropriate sharing settings in line with the principle of least privilege.

A Salesforce user tries to access a custom application built on the Salesforce platform after work hours, which is outside the login hours set on the user's profile. However, the user is within a location covered by the organization's trusted IP range. What will happen?

The user will be denied access due to login hours restrictions. **Correct answer**

The user will be granted access without any additional steps.

The user will be prompted to complete two-factor authentication.

Live Agent

The user will be granted access but with limited functionality.

Incorrect

The user will be denied access due to login hours restrictions. -> Correct. Login hours restrictions override IP range allowances. If a user attempts to log in outside of the permitted hours set on their profile, they will be denied access, regardless of the IP address.

The user will be granted access but with limited functionality. -> Incorrect. Salesforce does not typically reduce functionality based on login attempts; access is either granted or denied based on security settings.

The user will be prompted to complete two-factor authentication. -> Incorrect. Two-factor authentication may be required for security reasons, but in this scenario, the restriction is based on login hours, not the lack of two-factor authentication.

The user will be granted access without any additional steps. -> Incorrect. The trusted IP range does not bypass login hours restrictions.

In the context of Salesforce development, what is a recommended approach to ensure high code coverage and quality when writing tests for a custom Apex class that includes multiple methods and exception handling?

Use System.runAs to test the class under different user contexts. **Correct answer**

Only test positive scenarios to ensure the main functionality works as expected.

Rely exclusively on Salesforce's built-in testing framework to generate test data.

Write a single test method that covers all possible inputs and outcomes.

Live Agent

Incorrect

Use System.runAs to test the class under different user contexts. -> Correct. Using System.runAs allows testing how the class behaves under different user contexts, which is important for verifying permission-based logic and user-specific behavior.

Write a single test method that covers all possible inputs and outcomes. -> Incorrect. A single test method cannot effectively cover all scenarios, especially in complex classes with multiple methods and exception handling.

Only test positive scenarios to ensure the main functionality works as expected. -> Incorrect. Testing only positive scenarios ignores the importance of handling negative scenarios and exceptions, which are critical for robust applications.

Rely exclusively on Salesforce's built-in testing framework to generate test data. -> Incorrect. While Salesforce's built-in testing framework is useful, relying exclusively on it may not cover all custom logic and scenarios specific to the custom class.

In a complex development scenario requiring the migration of large volumes of data with extensive custom object relationships from one Salesforce sandbox to another, what is the most efficient strategy a Salesforce Platform Developer should employ to ensure data integrity and the maintenance of object relationships?

Utilize the Salesforce Data Import Wizard for all data migration tasks.

Export data using reports and import via the Salesforce Data Loader, manually mapping relationships.

Rely on the Salesforce DX CLI to handle data exports and imports using simple scripts.

Correct answer

Employ an ETL (Extract, Transform, Load) tool to automate the data migration process, including complex relationships.

Live Agent

Incorrect

Employ an ETL (Extract, Transform, Load) tool to automate the data migration process, including complex relationships. -> Correct. ETL tools are designed to handle complex data migrations, allowing for automation of data transformation and ensuring integrity and relationship mapping across environments.

Utilize the Salesforce Data Import Wizard for all data migration tasks. -> Incorrect. The Data Import Wizard is limited to simple data imports and cannot handle complex object relationships or large volumes of data effectively.

Export data using reports and import via the Salesforce Data Loader, manually mapping relationships. -> Incorrect. While the Data Loader can handle large data volumes and complex relationships, relying on manual mapping for extensive custom object relationships is error-prone and inefficient.

Rely on the Salesforce DX CLI to handle data exports and imports using simple scripts. -> Incorrect. Salesforce DX is primarily focused on source-driven development and metadata management, not on the nuances of complex data migration.

A Salesforce developer is tasked with automating a custom lead scoring system. The system assigns scores to Leads based on a variety of factors, including web activity, engagement levels, and data quality. The scoring logic is too complex for declarative tools alone but requires immediate execution upon any Lead record update. Which combination of declarative functionality and Apex best meets these requirements?

Configure a Workflow Rule to trigger on Lead update and execute a field update for preliminary scoring, followed by an Apex trigger for detailed calculations.

Use a scheduled Apex class to calculate and update lead scores daily, based on changes in web activity and engagement.

Create a Process Builder process to call an Apex class for scoring whenever a Lead record is created or updated.

Live Agent

Correct answer

Utilize a before-save Flow to perform initial data quality checks, then call an Apex class to calculate and update the lead score.

Incorrect

Utilize a before-save Flow to perform initial data quality checks, then call an Apex class to calculate and update the lead score. -> Correct. This approach effectively combines the strengths of Flow for straightforward, declarative data checks with the power of Apex for complex calculations. The before-save Flow minimizes the need for DML operations by handling data quality checks upfront, while the Apex class focuses on the complex scoring logic, ensuring efficiency and adherence to best practices.

Configure a Workflow Rule to trigger on Lead update and execute a field update for preliminary scoring, followed by an Apex trigger for detailed calculations. -> Incorrect. Workflow Rules are limited in their ability to perform complex logic and cannot directly invoke Apex code. This method does not efficiently leverage the platform's capabilities for the described requirements.

Use a scheduled Apex class to calculate and update lead scores daily, based on changes in web activity and engagement. -> Incorrect. A scheduled Apex class does not provide the immediate execution required by the scenario. This method would result in delayed score updates, potentially affecting timely decision-making.

Create a Process Builder process to call an Apex class for scoring whenever a Lead record is created or updated. -> Incorrect. While this approach uses both declarative and programmatic features, Process Builder may not be as efficient as Flow for operations that can be executed before a record is saved, potentially leading to unnecessary processing and record saving steps.

Which two of the following statements accurately describe the Lightning Component framework, its benefits, and the types of content that can be contained within a Lightning web component?

Live Agent

- [ ] The Lightning Component framework is exclusively designed for building desktop interfaces.

- [ ] Lightning web components can only contain HTML and custom tags defined in the Lightning framework.

- [ ] It enables developers to build dynamic web applications for mobile and desktop devices using JavaScript on the client side and Apex on the server side. Correct answer

- [ ] Benefits include performance improvement, compatibility with modern web standards, and enhanced component encapsulation. Correct answer

Incorrect

It enables developers to build dynamic web applications for mobile and desktop devices using JavaScript on the client side and Apex on the server side. -> Correct. This framework supports client-side and server-side scripting, enhancing the development of dynamic web applications for both mobile and desktop devices.

Benefits include performance improvement, compatibility with modern web standards, and enhanced component encapsulation. -> Correct. The Lightning Component framework is optimized for performance, designed to be compatible with modern web standards, and supports enhanced component encapsulation for better code maintenance and reuse.

The Lightning Component framework is exclusively designed for building desktop interfaces. -> Incorrect. The Lightning Component framework is designed for building scalable, responsive applications for any device.

Lightning web components can only contain HTML and custom tags defined in the Lightning framework. -> Incorrect. Lightning web components can contain HTML, JavaScript, and CSS, in addition to using custom tags defined in the Lightning framework.

Live Agent

In the context of developing Lightning Web Components (LWC) for Salesforce, which of the following best illustrates the recommended use of custom events for component communication?

Broadcasting a change in user permissions from a child component to all components within an application.

Correct answer

Initiating a complex query in a parent component based on user input gathered by a child component.

Triggering a refresh of sibling component data directly after a user action in another sibling component.

Directly modifying the state of a parent component's variables from a child component upon a user action.

Incorrect

Initiating a complex query in a parent component based on user input gathered by a child component. -> Correct. This scenario correctly utilizes custom events to communicate user input from a child component to a parent component, allowing the parent to handle data fetching or complex logic.

Broadcasting a change in user permissions from a child component to all components within an application. -> Incorrect. Custom events in LWC are designed for a component to communicate up its containment hierarchy, not for broadcasting to unrelated components.

Triggering a refresh of sibling component data directly after a user action in another sibling component. -> Incorrect. Direct sibling communication is not a recommended practice. LWC encourages using a common parent component to manage the interaction between sibling components through events.

Directly modifying the state of a parent component's variables from a child component upon a user action. -> Incorrect. Direct manipulation of a parent component's state fro

Live Agent

child violates component encapsulation principles. Custom events should be used to notify the parent component of changes.

The marketing team at Global Enterprises wants a streamlined process to collaborate on campaign designs and securely share documents with external partners. Which two actions should an administrator take to facilitate this requirement?

- ☐ Utilize Salesforce Files Connect for external document access.
- ☐ Enable Salesforce CRM Content for document management. **Correct answer**
- ☐ Configure Web-to-Lead to capture partner interactions.
- ☐ Create a private Chatter group and invite external partners as members. **Correct answer**
- ☐ Implement Public Groups for external partner collaboration.

Incorrect

Enable Salesforce CRM Content for document management. -> Correct. Enabling Salesforce CRM Content allows the marketing team to manage, share, and collaborate on campaign documents more effectively within Salesforce, providing a centralized repository for all marketing materials.

Create a private Chatter group and invite external partners as members. -> Correct. Creating a private Chatter group and inviting external partners as members provides a secure and collaborative space for sharing documents and collaborating on campaign designs, leveraging Chatter's collaboration features while controlling access.

Configure Web-to-Lead to capture partner interactions. -> Incorrect. Web-to-Lead is designed to capture leads from a website into Salesforce, not for document sharing or collaboration purposes with partners.

Live Agent

Utilize Salesforce Files Connect for external document access. -> Incorrect. Salesforce Files Connect is a feature that allows users to access files from external repositories from within Salesforce. While useful, it doesn't directly facilitate collaboration on campaign designs or secure sharing with external partners without additional configuration.

Implement Public Groups for external partner collaboration. -> Incorrect. Public Groups are used within Salesforce to group users for sharing and security purposes, but they do not directly facilitate secure document sharing or collaboration with external partners.

In a scenario where an Apex batch process is designed to handle data cleanup on a large set of records, which approach effectively manages governor limits while ensuring efficient processing?

Store intermediate processing results in static variables to avoid re-querying data.

Correct answer

Implement the batch process with a small batch size to reduce the chance of exceeding governor limits per batch execution.

Utilize SOQL queries inside loop structures to process records individually for precise control.

Execute all DML operations in a single transaction to minimize the total number of transactions.

Incorrect

Implement the batch process with a small batch size to reduce the chance of exceeding governor limits per batch execution. -> Correct. Adjusting the batch size allows for better control over the resources used during each batch execution. A smaller batch size helps in managing governor limits more effectively, especially in complex data processing scenarios.

Execute all DML operations in a single transaction to minimize the total number of transactions. -> Incorrect. Executing all DML operations in a single transaction can eas

Live Agent

exceed governor limits for DML statements and DML rows, especially when dealing with large datasets.

Utilize SOQL queries inside loop structures to process records individually for precise control. -> Incorrect. Utilizing SOQL queries inside loops is a common anti-pattern that can lead to hitting the governor limit on the number of SOQL queries allowed in a single transaction.

Store intermediate processing results in static variables to avoid re-querying data. -> Incorrect. Storing large amounts of data in static variables can lead to exceeding heap size limits. Additionally, static variables persist only for the duration of an Apex transaction, which may not be beneficial for batch processes that span multiple transactions.

In a complex Salesforce Lightning Web Component (LWC) application, a developer needs to create a mechanism where a child component must notify its parent component about a specific action, such as a user selecting an item from a list. Additionally, the parent component may need to communicate changes back down to one or more child components. What is the best practice for implementing events in LWC to handle this scenario?

Implement Platform Events to facilitate component messaging.

Use direct function calls from child to parent components to signal events.

Employ custom events for child-to-parent communication and public properties for parent-to-child communication. Correct answer

Utilize Aura events for communication between components.

Incorrect

Employ custom events for child-to-parent communication and public properties for par

to-child communication. -> Correct. Custom events are the standard for child-to-parer

Live Agent

communication in LWC, allowing children to dispatch events that parents can listen to. Public properties are used by parent components to pass data down to child components.

Utilize Aura events for communication between components. -> Incorrect. Aura events are specific to the Aura component framework and not applicable for communication between Lightning Web Components, which use a different event model.

Implement Platform Events to facilitate component messaging. -> Incorrect. Platform Events are used for event-driven architecture across Salesforce orgs, not for communication between LWC components on the client side.

Use direct function calls from child to parent components to signal events. -> Incorrect. Direct function calls from child to parent are not a supported communication pattern in LWC. Custom events are the recommended approach for child-to-parent signaling.

Given a scenario where you need to write an Apex trigger to update all related Contact records when an Account's address is changed, ensuring that the Contacts' mailing addresses are also updated to match the new Account address, which best practices should be followed in writing the Apex classes and triggers?

- ☐ Develop a helper class with a method that accepts a list of updated Accounts, identifies related Contacts using SOQL, and updates them in bulk. This method is called from an after update trigger on Account. Correct answer
- ☐ Use an after update trigger on Account to call a helper class method that queries and updates related Contacts, utilizing bulk patterns to handle large volumes of data efficiently. Correct answer
- ☐ Implement an after update trigger on Account that uses a for loop to iterate over each Contact individually and updates them, ensuring queries and DML are inside the loop for data freshness.
- ☐ Create a before update trigger on Account that instantiates a new Contact for eacl related Contact and updates the address, to ensure data integrity.

Live Agent

- [ ] Write the logic directly in the trigger, using before update to query and update related Contacts within the same trigger code.

Incorrect

Use an after update trigger on Account to call a helper class method that queries and updates related Contacts, utilizing bulk patterns to handle large volumes of data efficiently. -> Correct. Utilizing a helper class to encapsulate the logic and ensuring bulk operations are handled efficiently aligns with best practices for writing scalable and maintainable Apex.

Develop a helper class with a method that accepts a list of updated Accounts, identifies related Contacts using SOQL, and updates them in bulk. This method is called from an after update trigger on Account. -> Correct. This approach correctly separates the business logic from the trigger, focuses on bulk data handling, and adheres to best practices for writing efficient and maintainable Apex code.

Write the logic directly in the trigger, using before update to query and update related Contacts within the same trigger code. -> Incorrect. Directly writing complex logic inside a trigger can lead to difficult-to-maintain code and may violate best practices regarding bulkification and separation of concerns.

Create a before update trigger on Account that instantiates a new Contact for each related Contact and updates the address, to ensure data integrity. -> Incorrect. Instantiating new Contacts instead of updating existing ones does not address the requirement correctly and can lead to data duplication and integrity issues.

Implement an after update trigger on Account that uses a for loop to iterate over each Contact individually and updates them, ensuring queries and DML are inside the loop for data freshness. -> Incorrect. Placing queries and DML operations inside loops can lead to hitting governor limits and is not considered a best practice for handling bulk data in Salesforce.

Live Agent

When developing custom business logic in Salesforce, which scenario is best suited for using Apex triggers?

Customizing the appearance of Salesforce pages.

Adding new users to Salesforce.

Configuring security settings for objects and fields.

Creating complex page layouts.

Automatically updating related records when a record is inserted or updated. **Correct answer**

Incorrect

Automatically updating related records when a record is inserted or updated. -> Correct. Apex triggers are ideal for scenarios where actions need to be automatically triggered by data changes, such as updating related records when a record is inserted or updated.

Customizing the appearance of Salesforce pages. -> Incorrect. Apex triggers are not used for UI customization; they are used for executing business logic before or after data manipulation.

Adding new users to Salesforce. -> Incorrect. Adding new users is typically handled through the Salesforce UI or automated admin processes, not Apex triggers.

Creating complex page layouts. -> Incorrect. Page layouts are designed through the Salesforce Setup menu or by using Lightning App Builder, not through Apex triggers.

Configuring security settings for objects and fields. -> Incorrect. Security settings are configured in the Salesforce Setup, not via Apex triggers, which are meant for business logic.

Live Agent

In the context of Salesforce's multi-tenant architecture and design frameworks, which two statements accurately reflect best practices or concepts related to MVC architecture and the Lightning Component Framework? Select two correct answers.

☐ In MVC architecture, the Model is responsible for updating the View directly to ensure data consistency across the application.

☐ Apex classes used as Controllers in Salesforce should enforce CRUD and FLS (Field-Level Security) to comply with the platform's security model. Correct answer

☐ Visualforce pages are the preferred method for building responsive user interfaces in the Lightning Experience.

☐ The Lightning Component Framework utilizes event-driven programming, allowing components to communicate and update each other efficiently. Correct answer

☐ Controllers in MVC architecture should contain all the business logic, while Models are used merely for storing data structures.

Incorrect

The Lightning Component Framework utilizes event-driven programming, allowing components to communicate and update each other efficiently. -> Correct. The Lightning Component Framework supports an event-driven architecture, enabling components to be loosely coupled and to communicate through events, which improves the modularity and reusability of code.

Apex classes used as Controllers in Salesforce should enforce CRUD and FLS (Field-Level Security) to comply with the platform's security model. -> Correct. Enforcing CRUD and FLS in Apex controllers is a best practice to ensure that the application respects the organization's security settings, protecting sensitive data according to user permissions.

In MVC architecture, the Model is responsible for updating the View directly to ensure data consistency across the application. -> Incorrect. In MVC architecture, the Controller

Live Agent

updates the View based on changes in the Model; the Model does not update the View directly, maintaining a separation of concerns.

Controllers in MVC architecture should contain all the business logic, while Models are used merely for storing data structures. -> Incorrect. While Controllers handle the application's logic, the Model in MVC architecture also encapsulates business logic and data access, not just data structures.

Visualforce pages are the preferred method for building responsive user interfaces in the Lightning Experience. -> Incorrect. Lightning Components, not Visualforce pages, are the preferred method for building responsive and dynamic user interfaces in the Lightning Experience, taking full advantage of its modern capabilities.

In Salesforce, when designing a solution for automating business processes, which two of the following options accurately identify the capabilities of the declarative process automation features?

- [ ] Visual Workflow (Flow) can only update records related by master-detail relationship.
- [ ] Process Builder can update any related record, not just those directly related to the object that triggered the process. Correct answer
- [ ] Process Builder supports real-time and batch processing modes.
- [ ] Workflow Rules can initiate a flow as an action.
- [ ] Flow Builder can make decisions, loop over collections, and perform DML operations without code. Correct answer

Incorrect

Process Builder can update any related record, not just those directly related to the ob

that triggered the process. -> Correct. Process Builder is powerful in that it allows upd

Live Agent

to any related records, not strictly those in a direct relationship with the triggering object. This capability enables more complex and interconnected process automations.

Flow Builder can make decisions, loop over collections, and perform DML operations without code. -> Correct. Flow Builder is versatile and capable of executing logic such as making decisions, looping over record collections, and performing database operations (DML) like insert, update, delete, or undelete, all without requiring any Apex code.

Workflow Rules can initiate a flow as an action. -> Incorrect. Workflow Rules can perform actions such as field updates, sending emails, creating tasks, and sending outbound messages but cannot directly initiate a flow.

Process Builder supports real-time and batch processing modes. -> Incorrect. Process Builder operates in real-time mode, executing actions immediately when its criteria are met. It does not support batch processing, which is typically handled by scheduled Apex or batch Apex.

Visual Workflow (Flow) can only update records related by master-detail relationship. -> Incorrect. Flow is more flexible and can update any accessible records, not limited to those related by a master-detail relationship. This allows for broader automation capabilities across different objects and relationships.

Given a scenario where you need to aggregate the total sales from all opportunities related to a specific account, which approach best aligns with Salesforce best practices for customizations, considering governor limits, and the use cases for declarative versus programmatic solutions?

Develop a scheduled Apex batch job to calculate daily totals of sales from Opportunities and update the Account record.

Implement a complex Visualforce page and Apex controller to display the total sales on the Account page layout dynamically.

Use a declarative roll-up summary field on the Account to automatically calculate t total sales from related Opportunities.

Correct a

Live Agent

Create a custom Apex trigger to calculate and update the total sales on the Account record every time an Opportunity is created, updated, or deleted.

Incorrect

Use a declarative roll-up summary field on the Account to automatically calculate the total sales from related Opportunities. -> Correct. Roll-up summary fields are a declarative feature designed to aggregate data from child records (such as Opportunities) to a parent record (such as Account) without the need for custom code, thus avoiding the complexity and governor limits associated with programmatic solutions.

Create a custom Apex trigger to calculate and update the total sales on the Account record every time an Opportunity is created, updated, or deleted. -> Incorrect. While custom Apex triggers offer flexibility, they come with the burden of managing governor limits and complexity. Declarative options are preferred for straightforward aggregations like this.

Implement a complex Visualforce page and Apex controller to display the total sales on the Account page layout dynamically. -> Incorrect. A Visualforce page and Apex controller offer a high degree of customization but are overly complex for simply displaying aggregated data that can be handled declaratively.

Develop a scheduled Apex batch job to calculate daily totals of sales from Opportunities and update the Account record. -> Incorrect. Scheduled Apex batch jobs can handle complex data processing tasks but are not necessary for real-time aggregation of sales data, which can be more efficiently managed with declarative tools.

Given a scenario where you need to prevent user interface and data access security vulnerabilities in a Salesforce application, which of the following approaches should be implemented?

Correct answer

Utilize server-side controllers with enforced sharing rules to manage data access and ensure that users can only access data they are permitted to see.

Live Agent

Store sensitive data in client-side JavaScript variables to expedite data retrieval and improve user experience.

Embed raw SQL queries directly in JavaScript code for dynamic data fetching, ensuring flexibility in data access and manipulation.

Disable Salesforce's built-in security features like CRUD and FLS checks to simplify development and reduce the complexity of code.

Incorrect

Utilize server-side controllers with enforced sharing rules to manage data access and ensure that users can only access data they are permitted to see. -> Correct. Utilizing server-side controllers with enforced sharing rules is a best practice for managing data access in Salesforce applications. This approach ensures that users can only access the data they are authorized to see, thereby preventing unauthorized data exposure and access. It leverages Salesforce's robust security model, which includes organization-wide defaults, role hierarchies, sharing rules, and profiles and permission sets to safeguard data effectively.

Store sensitive data in client-side JavaScript variables to expedite data retrieval and improve user experience. -> Incorrect. Storing sensitive data in client-side JavaScript variables poses a significant security risk as it can be easily accessed by malicious users through browser tools.

Embed raw SQL queries directly in JavaScript code for dynamic data fetching, ensuring flexibility in data access and manipulation. -> Incorrect. Embedding raw SQL (or SOQL in the context of Salesforce) queries in JavaScript code is unsafe because it exposes the application to injection attacks and does not enforce Salesforce's security standards.

Disable Salesforce's built-in security features like CRUD and FLS checks to simplify development and reduce the complexity of code. -> Incorrect. Disabling Salesforce's built-in security features, such as CRUD (Create, Read, Update, Delete) and FLS (Field Level Security) checks, would remove critical layers of security, making the application vulne to unauthorized access and data manipulation.

Live Agent

A Salesforce developer needs to implement a solution to automatically update a custom field on Account records called “Customer Status” based on the total value of closed-won Opportunities associated with that Account. The “Customer Status” should be set to “Platinum” if the total value exceeds $1,000,000, “Gold” if the total is between $500,000 and $1,000,000, and “Silver” otherwise. Which process automation tool should the developer use to achieve this requirement most efficiently?

Lightning Component

Process Builder

Workflow Rule

Apex Trigger **Correct answer**

Flow

Incorrect

Apex Trigger -> Correct. An Apex Trigger on the Opportunity object can efficiently handle aggregate calculations of closed-won Opportunities and update the “Customer Status” field on the related Account based on the calculated total. This approach provides the necessary flexibility and precision for the requirement.

Workflow Rule -> Incorrect. Workflow Rules can perform field updates based on specific criteria but lack the ability to perform aggregate calculations across related records before setting the field value.

Process Builder -> Incorrect. While Process Builder is more powerful than Workflow Rules and can handle related records, it does not natively support aggregate calculations without involving additional Apex code for this specific scenario.

Live Agent

Flow -> Incorrect. Flows are versatile in handling complex logic and user interactions but might not be as efficient as Apex Triggers for real-time aggregate calculations and updates based on related record changes.

Lightning Component -> Incorrect. Lightning Components are used to create custom user interfaces. While they can display aggregated data, they do not automatically update a record's field based on changes in related records without being invoked by the user or another process.

Global Shipping Inc. wants to ensure that updates made to contact records in Salesforce are automatically reflected in their separate, custom CRM system. What approach should a developer recommend to facilitate this integration?

Correct answer

Implement outbound messages from Salesforce to trigger updates in the custom CRM system.

Configure Web-to-Lead forms to capture changes and update the external system.

Use Apex triggers to directly update records in the external CRM database.

Utilize Salesforce reports to manually export and update the custom CRM system.

Incorrect

Implement outbound messages from Salesforce to trigger updates in the custom CRM system. -> Correct. Outbound messages can be configured to send specific information to external systems when certain criteria are met, enabling automatic updates.

Utilize Salesforce reports to manually export and update the custom CRM system. -> Incorrect. This method relies on manual processes and does not provide automatic synchronization.

Live Agent

Use Apex triggers to directly update records in the external CRM database. -> Incorrect. While Apex triggers can respond to data changes, directly updating an external system's database is not a recommended practice due to security and architecture concerns.

Configure Web-to-Lead forms to capture changes and update the external system. -> Incorrect. Web-to-Lead is designed for capturing new leads from web forms into Salesforce, not for synchronizing data between systems.

A Salesforce Platform Developer needs to import and export data between multiple development environments for testing purposes. The data includes complex relationships between custom objects. Given this scenario, which method should the developer use to ensure data integrity and relationship mappings are maintained?

Rely on the Salesforce Ant Migration Tool for both exporting and importing data.

Export data using the Data Loader and import using the Import Wizard.

Use the Data Loader for both import and export, ensuring to export parent records before child records. Correct answer

Export data to CSV files and manually adjust relationships before importing.

Incorrect

Use the Data Loader for both import and export, ensuring to export parent records before child records. -> Correct. The Data Loader can handle complex object relationships by exporting and importing records in the correct order, maintaining data integrity.

Export data using the Data Loader and import using the Import Wizard. -> Incorrect. The Import Wizard is limited in handling complex relationships and large volumes of data, potentially compromising data integrity.

Live Agent

Export data to CSV files and manually adjust relationships before importing. -> Incorrect. Manually adjusting relationships in CSV files is error-prone and inefficient for maintaining complex data integrity across development environments.

Rely on the Salesforce Ant Migration Tool for both exporting and importing data. -> Incorrect. The Ant Migration Tool is primarily designed for deploying metadata rather than for importing and exporting records, which is essential for testing data integrity in development environments.

In Apex, when defining a class method that should not be overridden in any subclass, which keyword should be used to ensure this behavior?

static

virtual

global

final **Correct answer**

Incorrect

final -> Correct. The final keyword is used in Apex to indicate that a method cannot be overridden by any subclass. This is useful when you want to lock down the implementation of a particular method to ensure its behavior remains consistent across all instances of inheritance.

virtual -> Incorrect. The virtual keyword allows a class or method to be extended or overridden by subclasses, which is the opposite of what is asked in the question.

static -> Incorrect. The static keyword indicates that a method or variable belongs to the class itself, rather than an instance of the class. It does not control whether a method c

Live Agent

overridden.

global -> Incorrect. The global access modifier specifies that the method or variable is accessible across all Apex code, regardless of namespace. It does not influence the ability to override a method.

Cloud Solutions Inc. is planning to integrate their external inventory management system with Salesforce. The integration requires the creation of a custom object to store inventory data in Salesforce, which should be updated daily. Which two approaches should a Salesforce Developer use to automate the creation and update of inventory records in Salesforce from the external system?

- ☐ Schedule an Apex class using the Salesforce Scheduler to initiate daily data retrieval from the external inventory management system.
- ☐ Configure an Outbound Message to notify the external system each time inventory data needs to be updated.
- ☐ Use the Salesforce Bulk API for daily batch updates of inventory records from the external system. Correct answer
- ☐ Implement an Apex trigger on the custom object to pull data from the external system upon record creation.
- ☐ Utilize the Salesforce REST API to upsert inventory records based on a unique identifier. Correct answer

Incorrect

Utilize the Salesforce REST API to upsert inventory records based on a unique identifier. -> Correct. The Salesforce REST API supports upsert operations, which allows for the creation of new records or updating existing ones based on a unique identifier, making it suitable for integrating and synchronizing data with external systems.

Use the Salesforce Bulk API for daily batch updates of inventory records from the exte system. -> Correct. The Salesforce Bulk API is optimized for loading or updating large s

Live Agent

of data. It is an ideal choice for the daily batch update of inventory records from an external system, ensuring efficient data synchronization.

Configure an Outbound Message to notify the external system each time inventory data needs to be updated. -> Incorrect. Outbound Messages are used to send notifications from Salesforce to external systems when certain events occur in Salesforce, not for updating Salesforce records with external data.

Implement an Apex trigger on the custom object to pull data from the external system upon record creation. -> Incorrect. Apex triggers react to events within Salesforce, such as record creation, updates, or deletions. They are not designed to pull data from external systems directly.

Schedule an Apex class using the Salesforce Scheduler to initiate daily data retrieval from the external inventory management system. -> Incorrect. While scheduling an Apex class can automate processes within Salesforce, the class itself cannot directly initiate data retrieval from external systems without specifying a method of integration, such as calling out to an external API. This option does not directly address the integration mechanism needed for updating Salesforce records with data from an external system.

In a complex Salesforce project, your team needs to implement a solution to automatically update the Account status field to “Preferred” when a related Opportunity is marked as “Closed Won” and the total value of closed-won Opportunities for that Account exceeds $100,000. The solution must ensure bulk data processing efficiency and adhere to best practices for writing Apex classes and triggers. To achieve this, which approach should be used to update the Account status?

Implement a batch Apex class that periodically scans all Accounts and updates the status based on the total value of "Closed Won" Opportunities.

Create a trigger on the Opportunity object that updates the Account status directly when an Opportunity is marked as "Closed Won".

Live Agent

Correct a

Use an Apex trigger on Opportunity with a helper class to handle the logic and aggregate calculations, ensuring bulk-safe operations through the use of maps and lists.

Use a Process Builder to update the Account status when an Opportunity is marked as "Closed Won".

Incorrect

Use an Apex trigger on Opportunity with a helper class to handle the logic and aggregate calculations, ensuring bulk-safe operations through the use of maps and lists. -> Correct. This approach follows best practices by isolating complex logic in a helper class and using collection types (maps and lists) to ensure bulk-safe operations. It allows for real-time processing with efficient handling of related record updates and aggregate calculations.

Use a Process Builder to update the Account status when an Opportunity is marked as “Closed Won”. -> Incorrect. While Process Builder is useful for automating simple tasks without code, it may not efficiently handle bulk data operations and complex logic required for aggregating Opportunity values before updating an Account.

Create a trigger on the Opportunity object that updates the Account status directly when an Opportunity is marked as “Closed Won”. -> Incorrect. Directly updating related records in a trigger can lead to bulk processing issues and governor limit exceptions. It’s not the best practice for handling related record updates, especially when dealing with aggregate calculations.

Implement a batch Apex class that periodically scans all Accounts and updates the status based on the total value of “Closed Won” Opportunities. -> Incorrect. Batch Apex is designed for large data volumes and can efficiently process records in batches, but it may not provide real-time updates which might be a requirement in this scenario.

Live Agent

A Salesforce developer is preparing to deploy a new set of features, including Apex classes, Lightning components, and various customizations, from a sandbox environment to production. To ensure a smooth deployment and adherence to Salesforce best practices, which strategy should the developer employ for testing, debugging, and deployment?

Manually test all new features in production after deployment during off-peak hours to reduce the impact of potential issues on end-users.

Correct answer

Implement Continuous Integration (CI) and Continuous Deployment (CD) practices, including automated testing in a staging environment before deployment to production.

Use anonymous Apex blocks to perform final testing in production after deployment to quickly identify and fix any issues.

Deploy the new features directly to production without running tests to minimize deployment time.

Incorrect

Implement Continuous Integration (CI) and Continuous Deployment (CD) practices, including automated testing in a staging environment before deployment to production. -> Correct. Implementing CI/CD practices ensures that code is automatically tested and deployed in a controlled manner. Automated testing in a staging environment helps catch issues early, and deploying through CI/CD pipelines can streamline the process and reduce errors.

Deploy the new features directly to production without running tests to minimize deployment time. -> Incorrect. Deploying directly to production without running tests is against Salesforce best practices. It risks introducing bugs and negatively impacting the existing setup. Salesforce requires that all Apex code have a minimum of 75% code coverage and that all tests pass before deployment.

Use anonymous Apex blocks to perform final testing in production after deployment tc quickly identify and fix any issues. -> Incorrect. While anonymous Apex can be used foı

Live Agent

quick testing or data manipulation, it's not a suitable method for final testing of new features. Testing should be thoroughly completed in a sandbox or developer environment before deployment to ensure stability and functionality.

Manually test all new features in production after deployment during off-peak hours to reduce the impact of potential issues on end-users. -> Incorrect. Testing in production, especially after deployment, is not recommended due to the potential risk to live data and user experience. Testing should be completed in sandbox environments prior to deployment.

In the context of developing a secure and robust Salesforce application, a Salesforce Platform Developer needs to implement best practices to prevent user interface and data access security vulnerabilities. Considering the various strategies available, which of the following approaches is the most effective in preventing user interface and data access security vulnerabilities in a Salesforce application?

Correct answer

Apply Field-Level Security (FLS) and Sharing Rules in Apex classes to enforce data access permissions, ensuring that data visibility in the UI matches the organization's security settings.

Disable Salesforce's built-in Cross-Site Scripting (XSS) protection in Visualforce pages to allow for more dynamic and interactive user interfaces.

Implement hard-coded credentials in Lightning Components to ensure consistent authentication and authorization checks across the application.

Utilize unescaped dynamic queries in Apex to fetch data based on user input, enhancing flexibility in data retrieval and user interface adaptability.

Incorrect

Apply Field-Level Security (FLS) and Sharing Rules in Apex classes to enforce data acce

permissions, ensuring that data visibility in the UI matches the organization's security

Live Agent

settings. -> Correct. Applying FLS and Sharing Rules in Apex code aligns data access in custom code with the platform's security model, effectively preventing unauthorized data access and ensuring that the UI respects security constraints.

Utilize unescaped dynamic queries in Apex to fetch data based on user input, enhancing flexibility in data retrieval and user interface adaptability. -> Incorrect. Using unescaped dynamic queries can lead to SOQL injection vulnerabilities, posing a significant security risk by allowing attackers to inject malicious SOQL code.

Implement hard-coded credentials in Lightning Components to ensure consistent authentication and authorization checks across the application. -> Incorrect. Hard-coding credentials is a severe security flaw that exposes sensitive information and can be easily exploited, compromising the entire application.

Disable Salesforce's built-in Cross-Site Scripting (XSS) protection in Visualforce pages to allow for more dynamic and interactive user interfaces. -> Incorrect. Disabling XSS protection removes a crucial security layer, making the application vulnerable to XSS attacks where malicious scripts can be injected into web pages viewed by other users.

When designing an Apex trigger for a Salesforce application, understanding the relationship between Apex transactions, save order of execution, and managing the potential for recursion and cascading operations is crucial. Which of the following strategies best addresses these considerations to ensure efficient and reliable trigger execution?

Rely solely on Salesforce's built-in transaction management to automatically prevent recursion and manage the save order of execution, without additional coding.

Code all business logic directly inside each trigger to ensure transactions are handled efficiently and to simplify the save order of execution.

Utilize separate triggers for before and after contexts for each DML operation (insert, update, delete) to clearly define and manage the execution order.

Live Agent

Correct a

Implement a static variable within a class that acts as a trigger handler to prevent the same trigger from executing more than once in the same transaction, effectively managing recursion.

Incorrect

Implement a static variable within a class that acts as a trigger handler to prevent the same trigger from executing more than once in the same transaction, effectively managing recursion. -> Correct. This is a recognized method to prevent trigger recursion. By using static variables to track execution, developers can prevent the same logic from running multiple times within a single transaction, addressing recursion directly.

Code all business logic directly inside each trigger to ensure transactions are handled efficiently and to simplify the save order of execution. -> Incorrect. Placing all logic directly in triggers can lead to complex, hard-to-maintain code and does not directly address the challenges of recursion or cascading operations.

Utilize separate triggers for before and after contexts for each DML operation (insert, update, delete) to clearly define and manage the execution order. -> Incorrect. While organizing code this way might seem to offer clarity, it does not inherently manage recursion or cascading and can lead to redundant code execution and complexity.

Rely solely on Salesforce's built-in transaction management to automatically prevent recursion and manage the save order of execution, without additional coding. -> Incorrect. Salesforce provides mechanisms to manage transactions, but developers are responsible for explicitly coding to prevent recursion and properly manage execution order within triggers.

In Apex, what is the correct way to declare a method that is intended to be overridden by subclasses, ensuring it is accessible within the same package and by subclasses outside the package?

Live Agent

public final void calculateDiscount() {}

protected virtual void calculateDiscount() {} **Correct answer**

private void calculateDiscount() {}

protected void calculateDiscount() {}

Incorrect

The protected access modifier ensures that the method is accessible within the same package and by subclasses, even if they are outside the package. The virtual keyword indicates that the method can be overridden by subclasses.

private void calculateDiscount() {} -> Incorrect. The private access modifier restricts the method's accessibility to the class it is declared in only, making it impossible for subclasses to override or access it.

public final void calculateDiscount() {} -> Incorrect. The final keyword prevents the method from being overridden by any subclass, which contradicts the requirement for the method to be intended for overriding.

protected void calculateDiscount() {} -> Incorrect. Although protected allows access within the same package and by subclasses, the absence of virtual (or abstract for abstract methods) means the method is not explicitly intended to be overridden, which is a key part of the question's requirement.

EcoEnergy Solutions utilizes a custom Salesforce form on their website for service inquiries. They want Salesforce to automatically schedule a follow-up task for the sales team when an inquiry form is submitted with a service type labeled as "Solar Panel Installation". What configuration should an administrator implement to meet this requirement?

Email Alert

Live Agent

Web-to-Lead

Validation Rule

Process Builder **Correct answer**

Incorrect

Process Builder -> Correct. Process Builder can be configured to automatically create a follow-up task when a record meets specified criteria, such as a specific service type in an inquiry.

Web-to-Lead -> Incorrect. Web-to-Lead is used to capture lead information from a website into Salesforce but does not directly automate the creation of follow-up tasks based on form submissions.

Email Alert -> Incorrect. While an email alert can notify team members of the submission, it does not schedule tasks directly within Salesforce for follow-up actions.

Validation Rule -> Incorrect. Validation rules enforce data entry standards and cannot be used to automate the creation of tasks or other actions based on form submissions.

When implementing a custom sharing model using Apex managed sharing, what must a developer ensure to comply with Salesforce best practices?

Apex managed sharing code must be written in a without sharing context. **Correct answer**

Manually share records with all users by default for maximum visibility.

Create a public group for each record that needs to be shared.

Live Agent

The sharing reason must be associated with every user in the org.

Incorrect

Apex managed sharing code must be written in a without sharing context. -> Correct. Apex managed sharing should be executed in a without sharing context to ensure that the sharing calculations are correctly applied regardless of the running user's permissions.

The sharing reason must be associated with every user in the org. -> Incorrect. Sharing reasons are associated with specific records and not users directly.

Manually share records with all users by default for maximum visibility. -> Incorrect. This approach contradicts the principle of least privilege and Salesforce's sharing model best practices, which advocate for starting with the most restrictive access and then selectively granting access.

Create a public group for each record that needs to be shared. -> Incorrect. While public groups are a mechanism for sharing, they are not specifically related to the best practices for Apex managed sharing, which involves directly granting access through sharing rules or Apex sharing reasons.

Which feature is primarily used for adding custom logic that executes before or after changes to Salesforce records in the Salesforce Platform?

Visualforce Pages

Process Builder

Workflow Rules

Triggers Correct answer

Live Agent

Incorrect

Triggers -> Correct. Triggers are Apex code that execute before or after specific Salesforce record events, such as inserts, updates, or deletions, allowing for custom logic implementation.

Workflow Rules -> Incorrect. Workflow Rules automate standard internal procedures and processes but do not contain custom logic that executes before records are changed.

Process Builder -> Incorrect. Although Process Builder allows for advanced automation compared to Workflow Rules, it primarily focuses on automating business processes without writing code.

Visualforce Pages -> Incorrect. Visualforce pages are used to create custom UIs for Salesforce applications, not for executing logic before or after record changes.

Live Agent