DUMPSBOSS.

Salesforce Certified Platform Developer II

Salesforce PDII

Version Demo

Total Demo Questions: 70

Total Premium Questions: 700

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, Advanced Developer Fundamentals	151
Topic 2, Process Automation	54
Topic 3, User Interface	187
Topic 4, Testing	100
Topic 5, Performance	90
Topic 6, Integration	118
Total	700

QUESTION NO: 1

Which Salesforce feature allows a developer to see when a user last logged in to

Salesforce if real-time notification is not required?

A. Calendar Events

B. Asynchronous Data Capture Events

C. Event Monitoring Log

D. Developer Log

ANSWER: C

Explanation:

Event Monitoring Log is the appropriate feature because it provides historical, downloadable records of user activity, including login activity. Salesforce generates event log files for supported event types, and the login-related event data contains timestamps and user identifiers that a developer or administrator can use to determine when a particular user accessed the org. This is well suited to an audit or reporting use case where the information can be reviewed after the activity occurs rather than requiring an immediate action at the moment of login.

Event Monitoring makes these records available through the EventLogFile object and related APIs, enabling programmatic retrieval and analysis of event data. A developer can query the relevant log files for a date range, retrieve the file contents, and evaluate the login timestamps for the user of interest. This provides a durable activity-history mechanism and supports security monitoring, compliance review, and usage analysis. Salesforce documents the available event log files and their access model in the Event Log File REST API reference and describes the broader capability in its Event Monitoring documentation.

QUESTION NO: 2

Recently, users notice that fields that were recently added for one department suddenly disappear without warning.

Which two statements are true regarding these issues and resolution?

Choose 2 answers

A. A sandbox should be created to use as a unified testing environment instead of deploying Change Sets directly to production.

B. Page Layouts should never be deployed via Change Sets, as this causes Field-Level Security to be reset and fields to disappear.

C. The administrators are deploying their own Change Sets over each other, thus replacing entire Page Layouts in production.

D. The administrators are deploying their own Change Sets, thus deleting each other's fields from the objects in production.

ANSWER: A C

Explanation:

The administrators are deploying their own Change Sets over each other, thus replacing entire Page Layouts in production. Page layouts are metadata components, and a deployment containing a page layout applies the layout definition that was retrieved and included in the outbound change set. If separate administrators modify and deploy versions of the same layout without coordinating or first reconciling their changes, a later deployment can replace the production layout with a version that does not contain fields another team recently placed on it. This produces the appearance that fields have suddenly disappeared for a department, even though the underlying custom fields have not necessarily been deleted.

A sandbox should be created to use as a unified testing environment instead of deploying Change Sets directly to production. A shared integration or testing sandbox lets administrators combine changes, validate the resulting metadata,

test the final page-layout experience, and establish a controlled deployment sequence before production release. This process reduces conflicting deployments and helps ensure that each intended layout change is present in the release artifact. Salesforce describes sandbox environments and their purpose for development and testing in Salesforce Sandbox documentation; deployment behavior for metadata is documented in the Metadata API deployment guide.

QUESTION NO: 3

A developer created a class that implement he Queueable interface, as follows:

As part of the deployment process, the developer is asked to create a corresponding test class.

Which two actions should the developer take to successfully execute the test class?

Choose 2 answers

A. Ensure the running user of the test class has, at least, the View All permission on the Order object

B. Enclose System.enqueueJob (new orderQueueable Job (}) within Test. starttest and Test, stoptest ()

C. Implement seeAllData-true to ensure the Queueable)ob is able to run in bulk mode.

D. Implement Test.isRunningtest ( ) to prevent chaining jobs during test execution.

ANSWER: A B

Explanation:

Enclose System.enqueueJob (new orderQueueable Job (}) within Test. starttest and Test, stoptest () is required because Queueable work is asynchronous. In an Apex test, the queueable job should be submitted after `Test.startTest()`, and `Test.stopTest()` forces asynchronous work queued within that test scope to run before execution returns to the test method. This allows the test to query the resulting records and make reliable assertions after the job has completed. The actual Apex syntax is `Test.startTest()` and `Test.stopTest()`.

Implement Test.isRunningtest ( ) to prevent chaining jobs during test execution is also appropriate when the queueable implementation can enqueue a follow-on job. Tests have special asynchronous-processing behavior and should keep the tested execution bounded to the intended queueable job. A `Test.isRunningTest()` guard can bypass production-only chaining while the test verifies the first job's behavior, avoiding an unnecessary chained submission during the test run. Salesforce documents the asynchronous test pattern, including execution at `Test.stopTest()`, in the Apex asynchronous testing documentation. Queueable-job behavior and chaining are covered in the Queueable Apex documentation.

QUESTION NO: 4

The Account object has a field, Audit_Code_c, that is used to specify what type of auditing the Account needs and a Lookup to User, zudizar_c, that is the assigned auditor. When an Account is initially created, the user specifies the Audit_Code c. Each User in the org has a unique text field, Audit_Code _e, that is used to automatically assign the correct user to the Account's Auditor_c field.

What should be changed to most optimize the code's efficiency?

Choose 2 answers

A. Add an initial SOQL query to get all distinct audit codes.

B. Build a Map> of audit code to accounts.

C. Build a Map of Account Id to audit codes.

D. Add a WHERE clause to the SOQL query to filter on audit codes.

ANSWER: B D

Explanation:

Building a `Map<String, List<Account>>` of audit code to Accounts groups all incoming Accounts that need the same auditor. This is an efficient bulk-processing pattern because a single User record identified for an audit code can be applied to every Account in that code's list, rather than repeatedly searching Accounts or Users. It also preserves the one-to-many relationship inherent in the requirement: multiple Accounts can share an audit code while each audit code identifies one assigned auditor.

Adding a `WHERE` clause to filter on audit codes ensures the User query retrieves only Users whose unique audit-code field matches codes present on the Accounts being processed. The code can first collect the map's key set and use it in a selective query such as `WHERE Audit_Code__c IN :auditCodes`. The resulting Users can then be indexed by audit code and used to assign the lookup field in bulk. Together, these changes avoid unnecessary queried rows and support Salesforce's governor-limit-aware bulkification practices. Salesforce documents collection types, including maps, at Apex Map Class, and describes binding collection values in SOQL at Use Apex Variables in SOQL Queries.

QUESTION NO: 5

Which statement is true regarding both Flow and Lightning Process? (Choose two.)

A. Can use Apex methods with the @InvocableMethod annotation

B. Are both server-side considerations in the Order of Execution

C. Can use Apex that implements the Process.Plugin interface

D. Are able to be embedded directly into Visualforce pages

ANSWER: A B

Explanation:

Both Flow and Lightning Process (Process Builder) can invoke Apex methods exposed with the `@InvocableMethod` annotation. This annotation makes a static Apex method available as a declarative Apex action, allowing admins and developers to reuse governed, bulk-capable Apex logic from automation without writing a trigger. The method must meet Salesforce's invocable-method requirements, including accepting the supported input form and being declared appropriately for Flow Builder and Process Builder to discover it.

Both are also server-side automation mechanisms that participate in Salesforce transaction processing and its order of execution. Process Builder processes and flows started by Process Builder are explicitly included in the order-of-execution processing that occurs after workflow-rule evaluation. More broadly, record-triggered flows execute on the Salesforce server within the transaction, at timing points determined by whether they are configured for before-save or after-save behavior. Consequently, updates, validation effects, and downstream automation from these tools occur as part of transactional server-side processing.

Salesforce documents invocable Apex actions for declarative tools at InvocableMethod Annotation and details automation processing in the Order of Execution.

QUESTION NO: 6

Universal Containers wants to notify an external system in the event that an unhandled exception occurs when their nightly Apex batch job runs.

What is the appropriate publish/subscribe logic to meet this requirement?

A. Have the external system subscribe to a custom Platform Event that gets fired with addError().

B. Have the external system subscribe to a custom Platform Event published with EventBus.publish().

C. Have the external system subscribe to a Platform Event published with EventBus.publish().

D. Have the external system subscribe to the standard BatchApexErrorEvent Platform Event, and implement Database.RaisesPlatformEvents in the batch class.

ANSWER: D

Explanation:

Have the external system subscribe to the standard `BatchApexErrorEvent` Platform Event, and implement `Database.RaisesPlatformEvents` in the batch class. This is the purpose-built Batch Apex error-event mechanism for unhandled exceptions. When a batch class implements the `Database.RaisesPlatformEvents` marker interface, Salesforce publishes a `BatchApexErrorEvent` when a batch execution encounters an unhandled exception. The event includes useful diagnostic context, such as the asynchronous job ID, the failed record IDs, the exception type, and the exception message.

An external consumer can subscribe to this standard platform event through Salesforce event streaming APIs, such as Pub/Sub API, and use the received error event to initiate its notification or incident-handling process. This approach avoids requiring the batch code to catch every exception solely to publish a custom event, while preserving the details that identify the failed Batch Apex job and scope. Salesforce documents this pattern specifically for handling Batch Apex errors through platform events. See Batch Apex Error Events and Salesforce Pub/Sub API.

QUESTION NO: 7

An external application must create an Account and related Contact records in one REST request. The Contact requests must reference the Account created earlier in the same request, and all records must be rolled back if any request fails.

Which two Composite Graph API features should the developer use? Choose 2 answers

A. Use reference IDs to pass the Account ID to dependent Contact requests.

B. Set allOrNone to true for the graph.

C. Use the Bulk API to submit the related records.

D. Set the sObject Tree API as the graph execution mode.

E. Create a separate REST request for each Contact after the Account request completes.

ANSWER: A B

Explanation:

A Composite Graph request supports dependent subrequests by using a reference ID from one subrequest in another subrequest's URL or body. The application can create the Account in one graph node and use its returned ID when creating the related Contact records. This removes the need for the external application to wait for a separate Account-create request before creating Contacts.

The developer should also set `allOrNone` to `true` for the graph. With this setting, Salesforce treats the graph as a single transaction. If a request fails, Salesforce rolls back changes made by the other requests in that graph. This satisfies the requirement that the Account and Contacts are either all committed or all rolled back. Salesforce documents dependency handling and transactional behavior in Composite Graph.

QUESTION NO: 8

What are three benefits of using declarative customizations over code? (Choose three.)

A. Declarative customizations cannot generate run time errors.

B. Declarative customizations will automatically update with each Salesforce release.

C. Declarative customizations do not require user testing.

D. Declarative customizations are not subject to governor limits.

E. Declarative customizations generally require less maintenance.

ANSWER: B D E

Explanation:

Declarative customizations will automatically update with each Salesforce release, are not subject to governor limits, and generally require less maintenance. Salesforce manages the underlying platform implementation for point-and-click features, so these configurations are designed to remain compatible as the platform evolves through seasonal releases. This reduces the ongoing effort that would otherwise be needed to revise and redeploy custom programmatic implementations for platform changes.

Declarative tools also avoid the Apex transaction governor-limit model. Salesforce documents governor limits as runtime limits that apply to Apex to preserve shared multitenant resources. While declarative automation must still be designed thoughtfully for scale and can have its own platform limits, it does not require developers to implement logic around Apex SOQL, DML, CPU-time, or heap governor limits. Finally, because declarative configuration is maintained through supported setup interfaces rather than a custom codebase, it generally has a smaller long-term maintenance burden. Administrators can inspect, adjust, and deploy supported configuration without maintaining classes, tests, and code dependencies. See Salesforce's Apex Governor Limits documentation and its declarative versus programmatic development overview.

QUESTION NO: 9

A developer needs a Lightning web component to display in one column on phones and two columns on tablets/desktops. Which should the developer add to the code?

A. Add size="12" medium-device-size="6" to the elements

B. Add size="6" small-device-size="12" to the elements

C. Add small-device-size="12" to the elements

D. Add medium-device-size="6" to the elements

ANSWER: A

Explanation:

Add `size="12" medium-device-size="6"` to the `lightning-layout-item` elements. The `lightning-layout` base component uses the Salesforce Lightning Design System grid, whose layout widths are expressed as portions of a 12-column grid. Setting `size="12"` makes each layout item occupy all 12 columns by default. Consequently, each item takes an entire row on phone-sized screens, creating the required single-column presentation.

The `medium-device-size="6"` responsive setting changes each item's width to 6 of the 12 available columns beginning at the medium breakpoint. Two items therefore fit side by side, producing two columns on tablets and desktop-sized displays. This pattern supplies a mobile-first default while applying the wider-screen layout only where sufficient horizontal space is available. The attributes belong on each individual `lightning-layout-item`, while the parent `lightning-layout` provides the flexbox-based grid container. Salesforce documents the supported responsive sizing attributes and their 1–12 column values in the lightning-layout-item reference. The underlying responsive grid behavior is also described in the SLDS Grid utility documentation.

QUESTION NO: 10

What is a recommended practice with regard to the Apex CPU limit? (Choose two.)

A. Optimize SOQL query performance

B. Use Map collections to cache sObjects

C. Avoid nested Apex iterations

D. Reduce view state in Visualforce pages

ANSWER: B C

Explanation:

Using Map collections to cache sObjects and avoiding nested Apex iterations are recommended practices for staying within the Apex CPU limit. Apex CPU time measures the processing work performed by Apex in a transaction, so efficient in-memory access patterns and scalable loop design are particularly important. A Map lets code retrieve an already-loaded sObject by its Id or another key without repeatedly searching a collection or issuing unnecessary work inside a loop. This supports bulkified logic, where related records are collected and processed efficiently across an entire trigger or transaction batch.

Avoiding nested iterations is equally important because the amount of work can grow very quickly as record counts increase. Code that loops through one record collection for every item in another collection can consume CPU time disproportionately. Instead, developers should build keyed Maps, Sets, or grouped collections before the main processing loop, allowing direct lookups and generally linear processing. Salesforce identifies CPU time as a per-transaction governor limit and recommends designing Apex to handle bulk record processing efficiently. See Apex Governor Limits and Bulk Apex Trigger Best Practices.

QUESTION NO: 11

Refer to the following code snippet:

Java

public class LeadController {

public static List < Lead > getFetchLeadList(String searchTerm, Decimal aRevenue) {

String safeTerm = ' % ' +searchTerm.escapeSingleQuotes()+ ' % ' ;

return [

SELECT Name, Company, AnnualRevenue

FROM Lead

WHERE AnnualRevenue > = :aRevenue

AND Company LIKE :safeTerm

LIMIT 20

];

}

}

A developer created a JavaScript function as part of a Lightning web component (LWC) that surfaces information about Leads by wire calling getFetchLeadList when certain criteria are met. Which three changes should the developer implement in the Apex class above to ensure the LWC can display data efficiently while preserving security? 1

A. Use the WITH SECURITY_ENFORCED clause within the SOQL query.

B. Implement the with sharing keyword in the class declaration.

C. Annotate the Apex method with @AuraEnabled(Cacheable=true).

D. Implement the without sharing keyword in the class declaration.

E. Annotate the Apex method with @AuraEnabled.

ANSWER: A B C

Explanation:

“Annotate the Apex method with @AuraEnabled(Cacheable=true)” is required because an Apex method used through an LWC `@wire` adapter must be exposed with `@AuraEnabled` and marked `cacheable=true`. The cacheable designation identifies the method as read-only and enables the Lightning client to cache its returned data, reducing unnecessary server round trips for equivalent requests. “Implement the with sharing keyword in the class declaration” ensures that the SOQL query respects the running user’s record-level sharing access to Lead records. This is essential when a controller returns queried records to the client because Apex otherwise runs in system mode for sharing rules unless sharing behavior is explicitly declared. “Use the WITH SECURITY_ENFORCED clause within the SOQL query” provides object- and field-level permission enforcement for the fields and object referenced by the query, including `AnnualRevenue` and `Company`. Together, these changes enforce record access and query-level data permissions while supporting an efficient wired LWC data retrieval pattern. Salesforce documents the wire requirement in Wire Apex Methods to Lightning Web Components and describes query security enforcement in WITH SECURITY_ENFORCED.

QUESTION NO: 12

A developer is creating a Lightning web component that displays a list of records in a lightning-datatable. After saving a new record to the database, the list is not updating.

```
data;
@wire(recordList, {recordId : '$recordId'})
records(result){
    if(result.data){
        this.data = result.data;
    } else if(result.error){
        this.showToast(result.error);
    }
}
```

What should the developer change in the code above for this to

happen?

A. Call refreshApex() on this.data.

B. Create a new variable to store the result and annotate it with @track.

C. Create a variable to store the wired result and call refreshApex() on it.

D. Add the @track decorator to the data variable.

ANSWER: C

Explanation:

Create a variable to store the wired result and call `refreshApex()` on it. For a property wired to an Apex method, the value assigned to the component’s data property is the returned data, not the complete wire-service result that `refreshApex()` needs. The component should instead use a wired function, retain the full result object in a property such as `wiredRecordsResult`, and assign `result.data` to the datatable’s data property. After the record is successfully saved, invoke `refreshApex(this.wiredRecordsResult)`. This marks the Apex wire’s cached response as stale and retrieves current server data, causing the reactive data assignment to update the datatable. The Apex method used with `@wire` must

be cacheable for this refresh pattern. Salesforce documents that `refreshApex()` accepts the value provisioned by the wire service, which is why retaining the complete wire result is essential rather than passing only the array of records. See the Apex Result Caching documentation and the refreshApex reference.

QUESTION NO: 13

Universal Containers implements a private sharing model for Convention_Attendee__c. A lookup field Event_Reviewer__c was created. Management wants the event reviewer to automatically gain Read/Write access to every record they are assigned to. What is the best approach?

A. Create an after insert and after update trigger on the Convention Attendee custom object, and use Apex Sharing Reasons and Apex Managed Sharing.

B. Create a before insert trigger on the Convention Attendee custom object, and use Apex Sharing Reasons and Apex Managed Sharing.

C. Create criteria-based sharing rules on the Convention Attendee custom object to share the records with the Event Reviewers.

D. Create a criteria-based sharing rule on the Convention Attendee custom object to share the records with a group of Event Reviewers.

ANSWER: A

Explanation:

Create an after insert and after update trigger on the Convention Attendee custom object, and use Apex Sharing Reasons and Apex Managed Sharing. This approach supports granting access to the specific user whose Id is stored in `Event_Reviewer__c` for each individual Convention_Attendee__c record. In an after trigger, the record has a persistent Id, so the code can insert a corresponding `Convention_Attendee__Share` record. That share record uses the reviewer's user Id as `UserOrGroupId`, sets the object access level to `Edit` to provide Read/Write access, and uses a defined Apex Sharing Reason as its `RowCause`. Processing updates is important because the assigned reviewer can change after initial record creation; the automation can then remove obsolete managed shares and create the share for the newly assigned reviewer. Apex-managed sharing is designed for programmatic, record-specific sharing and allows the application to maintain access as assignment data changes. Salesforce documents creating custom-object share records through Apex and using an Apex sharing reason to identify and maintain shares created by the application. See Salesforce Apex Developer Guide: Bulk Sharing and Salesforce Help: Apex Sharing Reasons.

QUESTION NO: 14

An Apex Trigger creates a Contract record every time an Opportunity record is marked as Closed and Won. This trigger is working great, except (due to a recent acquisition) historical Opportunity records need to be loaded into the Salesforce instance.

When a test batch of records is loaded, the Apex Trigger creates Contract records. A developer is tasked with preventing Contract records from being created when mass loading the Opportunities, but the daily users still need to have the Contract records created.

What is the most extendable way to update the Apex Trigger to accomplish this?

A. Use a Hierarchy Custom Setting to skip executing the logic inside the trigger for the user who loads the data.

B. Add a Validation Rule to the Contract to prevent Contract creation by the user who loads the data.

C. Use a List Custom Setting to disable the trigger for the user who loads the data.

D. Add the Profile ID of the user who loads the data to the trigger so the trigger will not fire for this user.

ANSWER: A

Explanation:

Use a Hierarchy Custom Setting to skip executing the logic inside the trigger for the user who loads the data. A hierarchy custom setting supports organization-wide defaults plus profile-level and user-level values. The trigger can retrieve the effective setting for the current running user and conditionally bypass only the Contract-creation portion of its logic when that user's setting is enabled. This allows the dedicated data-load user to insert or update the acquired historical Opportunities without generating Contracts, while normal daily users continue through the same trigger logic and receive Contracts when Opportunities become Closed Won.

This approach is extendable because the bypass behavior is configuration-driven rather than hard-coded to a particular user ID or profile ID. Administrators can enable or remove the bypass without changing, testing, and deploying Apex, and the same pattern can be reused for additional controlled data-load users or operational scenarios. In Apex, hierarchy custom settings can be accessed using methods such as `getInstance()`, which resolves the appropriate user, profile, or organization value. Salesforce documents hierarchy custom settings and their user/profile-specific behavior at Custom Settings Methods and Custom Settings Overview.

QUESTION NO: 15

Refer to the Lightning component below:

```
<lightning:button label="Save" onclick="
{!c.handleSave}" />

({
    handleSave : function (component, event, helper) {
        helper.saveAndRedirect(component);
    }
})
```

The Lightning Component allows users to click a button to save their changes and then redirects them to a different page.

Currently when the user hits the Save button, the records are getting saved, but they are not redirected.

Which three techniques can a developer use to debug the JavaScript?

Choose 3 answers

A. Use console.log () messages in the JavaScript.

B. Use the browser's dev tools to debug the JavaScript.

C. Enable Debug Mode for Lightning components for the user.

D. Use Developer Console to view checkpoints.

E. Use Developer Console to view the debug log.

ANSWER: A B C

Explanation:

Use console.log () messages in the JavaScript., **Use the browser's dev tools to debug the JavaScript.**, and **Enable Debug Mode for Lightning components for the user.** are appropriate client-side debugging techniques for this situation. Console logging lets the developer verify that the save callback executes, inspect returned values and component state, and determine whether the redirect logic is reached. Browser developer tools provide a more interactive workflow: developers

can set breakpoints in the controller or helper JavaScript, step through callback execution, inspect variables, and view browser-console exceptions that might stop navigation after a successful save.

Lightning component debug mode is especially useful because Salesforce serves a less-minified version of framework JavaScript for the designated user. This makes client-side source code and error stack traces more readable in browser developer tools, which is critical when tracing a navigation failure that occurs after an asynchronous save action completes. Debug mode is enabled per user in Setup and should generally be used only while troubleshooting because it can affect page-load performance. Salesforce documents Lightning component debugging and the role of browser developer tools at Lightning Web Components Debugging and describes user-level debug mode at Debug Mode for Lightning Components.

QUESTION NO: 16

Part of a custom Lightning Component displays the total number of Opportunities in the org, which is in the millions. The Lightning Component uses an Apex Controller to get the data it needs. What is the optimal way for a developer to get the total number of Opportunities for the Lightning Component?

A. SUM() SOQL aggregate query on the Opportunity object

B. SOQL for loop that counts the number of Opportunities records

C. COUNT() SOQL aggregate query on the Opportunity object

D. Apex Batch job that counts the number of Opportunity records

ANSWER: C

Explanation:

COUNT() SOQL aggregate query on the Opportunity object is the optimal approach because the component needs only one value: the number of Opportunity records. An aggregate count lets Salesforce calculate that value in the database rather than returning Opportunity rows to Apex and incrementing a counter in application code. For example, an Apex controller can use `Integer opportunityCount = [SELECT COUNT() FROM Opportunity];`. This keeps the response compact and avoids unnecessary heap consumption, CPU work, and record processing when the organization contains millions of records.

SOQL's `COUNT()` function is specifically intended to return the number of records matching a query. If the displayed total must reflect only records visible to the running user, the controller should normally run in the appropriate sharing context; Salesforce then applies record access when evaluating the query. Where applicable, selective filter criteria can also be included in the count query without changing the basic pattern. Salesforce documents the behavior and syntax of SOQL aggregate functions, including `COUNT()`, at SOQL Aggregate Functions. The Apex language reference also describes using SOQL queries directly in Apex at SOQL and SOSL Reference.

QUESTION NO: 17

An org records customer order information in a custom object, Order__c, that has fields for the shipping address. A developer is tasked with adding code to calculate shipping charges on an order, based on a flat percentage rate associated with the region of the shipping address. What should the developer use to store the rates by region, so that when the changes are deployed to production no additional steps are needed for the calculation to work?3132

A. Custom metadata type

B. Custom list setting

C. Custom object

D. Custom hierarchy setting

ANSWER: A

Explanation:

Custom metadata type is the appropriate store for regional shipping-rate configuration because custom metadata records are metadata, rather than ordinary business data. A developer can define a custom metadata type with fields such as region and percentage rate, create one record for each region, and include both the type definition and its records in the deployment. Consequently, after the calculation code is deployed to production, the regional rate values are already present and available to the code without manual record entry, a separate data-load process, or post-deployment configuration.

Custom metadata is designed for application configuration that should move consistently between environments. Apex can query custom metadata records, allowing the shipping-charge logic to select the applicable rate using the order's shipping region and calculate the resulting charge. Keeping the values declarative also means an administrator can update rates through configuration and deploy those changes through the normal release process, without modifying the calculation implementation. Salesforce documents that custom metadata records are deployable and packageable, making them suitable for configuration that must accompany an application deployment. See Salesforce Apex Developer Guide: Access Custom Metadata Types and Salesforce Help: Custom Metadata Types.

QUESTION NO: 18

A developer is tasked with ensuring that email addresses entered into the system for Contacts and for a custom object called Survey_Response__c do not belong to a list of blocked domains. The list of blocked domains is stored in a custom object for ease of maintenance by users. The Survey_Response__c object is populated via a custom Visualforce page. What is the optimal way to implement this?

A. Implement the logic in validation rules on the Contact and the Survey_Response__c objects.

B. Implement the logic in a helper class that is called by an Apex trigger on Contact and from the custom Visualforce page controller.

C. Implement the logic in an Apex trigger on Contact and also implement the logic within the custom Visualforce page controller.

ANSWER: B

Explanation:

Implement the logic in a helper class that is called by an Apex trigger on Contact and from the custom Visualforce page controller. This design centralizes the domain-validation rule in one reusable Apex service, so both entry paths apply the same definition of a blocked domain. The helper can normalize email values, extract domains, query the user-maintained custom-object records once for the relevant domains, and return or apply validation errors before records are saved. The Contact trigger ensures the rule is enforced for Contact changes regardless of whether they originate in the standard interface, API, imports, or other automation. The Visualforce controller can invoke that same helper during its save action for Survey_Response__c, display an appropriate page message, and avoid performing the insert when a blocked domain is found. The helper should be designed for collections rather than individual records, because trigger invocations can contain multiple Contacts and Salesforce requires Apex code to stay within shared governor limits. Centralizing the behavior also provides one focused unit for tests and allows users to maintain blocked-domain records without changing the validation implementation. Salesforce recommends keeping trigger logic in handler classes rather than embedding substantial business logic in triggers; see Apex Trigger Best Practices. Visualforce controllers can execute Apex actions and perform database operations through controller methods, as described in Visualforce Controllers and Extensions.

QUESTION NO: 19

Consider the following code snippet:

Choose 2 answers

A. Store the URL of the endpoint in a custom Label named endpointURL.

B. Create a Named Credential, endpoint_NC, to store the endpoint and credentials.

C. Use req.setEndpoint(Label.endpointURL);

D. Use req.setEndpoint('callout:endpoint_NC'); within the callout request.

ANSWER: B D

Explanation:

Creating a Named Credential named `endpoint_NC` is the appropriate way to define an external service's base URL together with its authentication configuration outside Apex code. Named Credentials provide a managed, declarative endpoint and can use an External Credential to control the authentication method and principal access. This keeps sensitive connection details out of source code and allows administrators to update the target endpoint or authentication setup without requiring an Apex deployment.

The callout request must then reference that Named Credential with the `callout:` scheme, for example `req.setEndpoint('callout:endpoint_NC/resourcePath')`. Salesforce resolves the Named Credential, applies its configured authentication, and authorizes the outbound request through the configured connection. A resource path can be appended when the Named Credential stores only the service base URL. This approach is the standard Apex integration pattern for a configurable and securely authenticated HTTP callout. See Salesforce's Named Credentials callout guidance and the HttpRequest setEndpoint reference.

QUESTION NO: 20

A developer must create a way for external partners to submit millions of leads into Salesforce per day.

How should the developer meet this requirement?

A. Publicly expose a Visualforce page via Force.com Sites

B. Create a web service on Heroku that uses Heroku Connect

C. Host a Web-to-Lead form on the company website

D. Publicly expose an Apex Web Service via Force.com Sites

ANSWER: B

Explanation:

Create a web service on Heroku that uses Heroku Connect is the appropriate architecture for a very high-volume partner submission workload. A Heroku-hosted service can provide a scalable, internet-facing endpoint for partners and can be scaled independently to absorb bursts of traffic. Rather than having every partner request synchronously create a Lead directly in Salesforce, the service can persist submissions in Heroku Postgres. Heroku Connect maps the relevant PostgreSQL table to the Salesforce Lead object and asynchronously synchronizes inserted rows to Salesforce. This decoupling gives the integration durable staging, retry capability, and operational resilience when Salesforce processing is slower than inbound partner traffic.

Heroku Connect is specifically designed to synchronize Salesforce objects and Heroku Postgres data, including write-back changes made in the database to Salesforce. The application can therefore validate, authenticate, and normalize each submission before adding it to the mapped table, while Salesforce receives the resulting Lead records through the connector's managed synchronization process. This is a suitable integration pattern when the public-facing ingestion tier must handle volumes far beyond typical form-based lead capture. See the Heroku Connect documentation for synchronization behavior and architecture, and Salesforce's asynchronous processing guidance for the rationale behind decoupling high-volume data ingestion from synchronous request handling.

QUESTION NO: 21

The progress of an apex job queued is using the System.enqueueJob method and needs to be monitored.

Which options are valid? (Choose two.)

A. Use the Apex Jobs page in setup

B. Query the Queueable Apex record

C. Query the AsyncApexJob record

D. Use the Scheduled Jobs page in setup

ANSWER: A C

Explanation:

Jobs submitted through `System.enqueueJob` are Queueable Apex jobs, which Salesforce runs asynchronously. The **Apex Jobs** page in Setup is a valid monitoring tool because it displays asynchronous Apex executions, including Queueable jobs, with useful operational details such as status, job type, submission date, completion date, and any error information. This makes it appropriate for administrators and developers who need to inspect queued, processing, completed, failed, or aborted work.

Querying the `AsyncApexJob` record is also valid. Salesforce creates an `AsyncApexJob` record for Queueable Apex execution, and the job ID returned by `System.enqueueJob` identifies that record. SOQL queries against fields such as `Status`, `JobType`, `NumberOfErrors`, `CreatedDate`, and `CompletedDate` enable programmatic monitoring, reporting, or custom operational tooling. For example, a developer can query the record by its ID to determine whether the queueable job has completed and whether errors occurred. Salesforce documents Queueable Apex monitoring through the Apex Jobs page and the `AsyncApexJob` object in its asynchronous Apex guidance: Queueable Apex and AsyncApexJob Object Reference.

QUESTION NO: 22

A developer wants to use an Aura component with a custom action.

What should be considered in order to do this?

A. A default value must be provided for each component attribute marked as required

B. The class “slds-modal__container” must be added to the top-level element of the component

C. The component must implement the flexipage:availableForRecordHome interface

D. The component’s JavaScript controller must handle a method on initialization

E. The component must implement the force:lightningQuickAction interface.

ANSWER: E

Explanation:

An Aura component intended for use as a custom Lightning quick action must implement the `force:lightningQuickAction` interface. This interface identifies the component as eligible to be selected when configuring an object-specific quick action in Salesforce Setup. After the component is exposed through this interface, an administrator can create a Lightning component action for the relevant object and choose the component as the action’s content.

The component can then provide a focused action experience directly from a record page, such as collecting user input, invoking Apex, or updating related data. If the action needs the current record’s identifier, the component can additionally implement `force:hasRecordId` and use the `recordId` attribute supplied by the framework. The quick-action interface itself is the essential requirement for making the Aura component available as a custom action. Salesforce documents Aura quick actions and their supported interfaces in the Lightning Component Developer Guide and explains how to configure them in Salesforce Help.

QUESTION NO: 23

Consider the following code snippet:

Java

```
HttpRequest req = new HttpRequest();

req.setEndpoint('https://TestEndpoint.example.com/some_path ');

req.setMethod('GET');

Blob headerValue = Blob.valueOf('myUserName' + ':' + 'strongPassword');

String authorizationHeader = 'BASIC ' + EncodingUtil.base64Encode(headerValue);

req.setHeader('Authorization', authorizationHeader);

Http http = new Http();

HTTPResponse res = http.send(req);
```

Which two steps should the developer take to add flexibility to change the endpoint and credentials without needing to modify code?1

A. Use req.setEndpoint(Label.endPointURL);

B. Use req.setEndpoint('callout:endPoint_NC'); within the callout request.2

C. Store the URL of the3 endpoint in a custom Label named endPointURL.4

D. Create a Named Credential, endPoint_NC, to store the endpoint and credentials.5

ANSWER: B D

Explanation:

Create a Named Credential, endPoint_NC, to store the endpoint and credentials. A Named Credential externalizes the integration's base URL and authentication configuration from Apex. The endpoint and credential values can therefore be maintained in Setup rather than being embedded in deployed source code. This lets an administrator update a host, authentication configuration, or secret without changing the Apex class or performing another deployment. It also avoids constructing and storing a Basic Authorization header directly in code, keeping sensitive values under Salesforce's managed credential configuration.

Use req.setEndpoint('callout:endPoint_NC'); within the callout request. The `callout:` endpoint syntax tells Apex to resolve the named configuration at runtime. Salesforce uses the Named Credential's configured URL and injects the applicable authentication details for the callout. A relative path can also be appended when the Named Credential represents a common base URL, allowing the same named configuration to support several resource paths. Together, these steps separate integration configuration from application logic while retaining a concise, portable Apex callout implementation.

Salesforce documents Named Credentials and their use in callouts at Named Credentials in Apex Callouts and Apex Named Credentials.

QUESTION NO: 24

Which scenario requires a developer to use an Apex callout instead of Outbound Messaging?

A. The target system uses a REST API.

B. The callout needs to be asynchronous.

C. The target system uses a SOAP API.

D. The callout needs to be invoked from a Workflow Rule.

ANSWER: A

Explanation:

The target system uses a REST API requires an Apex callout because Salesforce Outbound Messaging is a declarative integration feature that sends notifications in a predefined SOAP message format. An outbound message is delivered to an external endpoint as a SOAP request, with Salesforce-generated WSDL describing the message structure and an external listener acknowledging successful receipt. It does not provide a configurable REST client or support sending REST resources using methods such as GET, POST, PATCH, or DELETE with REST-specific headers, authentication patterns, and payload conventions.

Apex callouts provide the required flexibility for REST integrations. Using the Apex `HttpRequest`, `Http`, and `HttpResponse` classes, a developer can select the HTTP method, endpoint, request headers, authentication approach, and JSON or other request body required by the target REST service. Named Credentials can also centralize endpoint and authentication configuration. Salesforce documents REST callouts as HTTP-based integrations implemented through Apex, while Outbound Messaging is documented as SOAP-based workflow notification delivery. See Salesforce Apex Callouts documentation and Salesforce Outbound Messaging documentation.

QUESTION NO: 25

An org has a requirement that addresses on Contacts and Accounts should be normalized to a company standard by Apex code any time that they are saved.

What is the optimal way to implement this?

A. Apex trigger on Contact that calls the Account trigger to normalize the address

B. Apex triggers on Contact and Account that normalize the address

C. Apex trigger on Account that calls the Contact trigger to normalize the address

D. Apex triggers on Contact and Account that call a helper class to normalize the address

ANSWER: D

Explanation:

Apex triggers on Contact and Account that call a helper class to normalize the address is the optimal design. Each object needs its own trigger because Contacts and Accounts are independently saved and each trigger runs only for its associated object. The shared address-normalization logic belongs in a helper or service class so that the normalization rules are implemented once, rather than duplicated across object-specific triggers. This makes the implementation easier to test, maintain, and update when the company's address standard changes.

The triggers should invoke the shared class in a before-insert and before-update context, allowing the code to modify the address fields directly on the records being saved without requiring additional DML. The helper should accept record collections, not individual records, so it remains bulkified for imports, API operations, and other transactions that save many Contacts or Accounts at once. This follows Salesforce's trigger best practices: keep triggers lightweight, use a handler or helper layer for business logic, and design all Apex for bulk processing. See Salesforce Apex Trigger Best Practices and Bulk Apex Triggers.

QUESTION NO: 26

Which two scenarios require an Apex method to be called imperatively from a Lightning web component?

Choose 2 answers

A. Calling a method that makes a web service callout

B. Calling a method that is not annotated with cacheable=true

C. Calling a method with the click of a button

D. Calling a method that is external to the main controller for the Lightning web component

ANSWER: B C

Explanation:

Calling a method that is not annotated with cacheable=true requires an imperative Apex call. The Lightning Data Service wire infrastructure requires an Apex method used with `@wire` to be annotated `@AuraEnabled(cacheable=true)`, which indicates that the operation is read-only and that the client can use cached results. Imperative invocation is therefore the appropriate mechanism for server-side actions that do not meet that requirement, including operations that can change Salesforce data.

Calling a method with the click of a button also requires an imperative call because this is an event-driven interaction. An imperative invocation is made explicitly from the component's JavaScript event handler, allowing the component to control exactly when the server request occurs and to process the returned Promise with `async/await` or `then()`. This pattern is suited to user-initiated actions and lets the component manage loading states, results, and errors in the context of that action. Salesforce documents imperative Apex calls for both non-cacheable methods and methods invoked in response to a specific user event. See Call Apex Methods Imperatively and Wire Apex Methods to Lightning Web Components.

QUESTION NO: 27

Refer to the component code and requirements below:

HTML

{!v.account.Name}

{!v.account.AccountNumber}

{!v.account.Industry}

Requirements:

For mobile devices, the information should display in **three rows**.

For desktops and tablets, the information should display in a **single row**.

Requirement 2 is not displaying as desired. Which option has the correct component code to meet the requirements for desktops an7d tablets?

A. HTML{!v.account.Name}{!v.account.AccountNumber}{!v.account.Industry}

B. 1213

C. 1415HTML{!v.account.Name}{!v.account.AccountNumber}{!v.account.Industry}

D. HTML{!v.account.Name}{!v.account.AccountNumber}{!v.account.Industry}

E. Enable Debug Mode for Lightning components for the user.9

ANSWER: D

Explanation:

`<lightning:layout multipleRows="true">...</lightning:layout>` with each nested `<lightning:layoutItem size="12" mediumDeviceSize="4" largeDeviceSize="4">` correctly implements the required responsive behavior. Lightning layout uses a 12-column grid. The base `size` value applies at the smallest breakpoint, so `size="12"` gives each account field the full available width on phones. Consequently, Name, Account Number, and Industry are displayed as three separate rows on mobile devices.

At the medium and large breakpoints, each item is assigned four columns. The three fields therefore consume 12 columns in total: 4 + 4 + 4. They fit side by side in one row on tablet and desktop screens. Setting both `mediumDeviceSize` and `largeDeviceSize` is important because the requirement explicitly applies to both device categories. The layout items must also be children of the layout component, and the component attribute names use the documented casing, including

`multipleRows`, `mediumDeviceSize`, and `largeDeviceSize`. Salesforce documents these responsive sizing attributes and their 12-column behavior in the lightning:layoutItem reference and the lightning:layout reference.

QUESTION NO: 28

An Apex trigger creates a Contract record every time an Opportunity record is marked as Closed and Won. A developer is tasked with preventing Contract records from being created when mass loading historical Opportunities, but the daily users still need the logic to fire. What is the most extendable way to update the Apex trigger to accomplish this?

A. Add the Profile ID of the user who loads the data to the trigger.

B. Add a validation rule to the Contract to prevent creation by the data load user.

C. Use a hierarchy custom setting to skip executing the logic inside the trigger for the user who loads the data.

D. Use a list custom setting to disable the trigger for the user who loads the data.

ANSWER: C

Explanation:

Use a hierarchy custom setting to skip executing the logic inside the trigger for the user who loads the data. A hierarchy custom setting is appropriate because it supports values at the organization, profile, and individual user levels. The developer can create a checkbox setting, such as `Disable_Opportunity_Trigger__c`, and have the trigger evaluate the effective setting for the current user before running the Contract-creation logic. For example, the trigger handler can obtain the setting with `MySetting__c.getInstance()` and return without creating Contracts when the checkbox is enabled. During the historical Opportunity load, an administrator enables the setting for the dedicated integration or data-load user. The same trigger continues to execute normally for daily users because their effective setting remains disabled. This design is extendable because the behavior is configuration-driven rather than tied to environment-specific identifiers in Apex, and the setting can be administered without deploying code. Hierarchy custom settings resolve values based on the applicable user, profile, or organization context, making them well suited to controlled, user-specific automation bypasses. See Salesforce's Apex Custom Settings methods reference and Custom Settings overview.

QUESTION NO: 29

What is a consideration when using bind variables with dynamic SOQL? (Choose two.)

A. Dynamic SOQL cannot reference fields on bind variables

B. Dynamic SOQL cannot use bind variables

C. Bind variables must be public or global

D. Bind variables must be in local scope

ANSWER: A D

Explanation:

With traditional dynamic SOQL executed through `Database.query()`, a bind expression refers to an Apex variable that is available in the current execution context. Therefore, **Bind variables must be in local scope**: declare or otherwise make the value available to the code constructing and executing the query, then reference it with the colon prefix, such as `WHERE Name = :accountName`. This lets Salesforce safely substitute the runtime value without requiring string concatenation.

Dynamic SOQL cannot reference fields on bind variables because the bind syntax accepts a variable reference, not a dotted field dereference. Assign the desired field value to a separate variable first—for example, set `String accountName = account.Name;`—and bind `:accountName` in the query. This approach makes the query valid and keeps query values parameterized. Salesforce documents dynamic SOQL bind-variable behavior and scope requirements in its Apex Developer Guide: Dynamic SOQL. For the general syntax and permitted use of bind expressions in SOQL, see Use Apex Variables as Bind Variables.

QUESTION NO: 30

A Salesforce org has more than 50,000 contacts. A new business process requires a calculation that aggregates data from all of these contact records. This calculation needs to run once a day after business hours.

which two steps should a developer take to accomplish this?

Choose 2 answers

A. Implement the Queueable interface.

B. Use the @future annotation.

C. Implement the Schedulable interface.

D. Implement the Database.Stateful interface.

E. Implement the Database.Batchable interface.

ANSWER: C E

Explanation:

Implementing the `Schedulable` interface and implementing the `Database.Batchable` interface provide the required scheduled, large-volume processing design. A class that implements `Schedulable` can be registered with `System.schedule` using a CRON expression for the required after-hours daily time. Its `execute` method can then launch the batch job with `Database.executeBatch`. This separates the timing requirement from the record-processing work.

Batch Apex is specifically designed for data volumes that exceed the synchronous Apex query-row limit. The batch class's `start` method can return a `QueryLocator` for the Contact query, allowing Salesforce to divide the full result set into independently governed execution chunks. The aggregation can be performed in those chunks, with final persisted results handled as appropriate in `finish`. A `QueryLocator` used by Batch Apex can retrieve up to 50 million records, which comfortably supports processing all contacts in this org. If the aggregate must be accumulated across execution chunks, the batch class can additionally implement `Database.Stateful`, but that is not the fundamental interface required to process the records. See Salesforce's Batch Apex documentation and Scheduled Apex documentation.

QUESTION NO: 31

Which of the following exceptions cannot be caught and will force an error? (Choose three.)

A. LimitException

B. AssertException

C. SObjectExceptions

D. DMLException

E. License exceptions

F. ListException

ANSWER: A B E

Explanation:

`LimitException`, `AssertException`, and license exceptions are exceptions that Apex does not allow application code to handle with a `try-catch` block. A `LimitException` represents exhaustion of a governor limit, such as exceeding the allowed number of SOQL queries or CPU time in a transaction. Salesforce treats this as a transaction-level failure, so execution stops rather than permitting recovery in Apex code. An `AssertException` is raised when an assertion fails; it is intentionally uncatchable so that a failed assertion reliably terminates the relevant execution, particularly in tests. License exceptions are likewise treated as unrecoverable platform conditions and cannot be intercepted by Apex exception handling.

Salesforce's Apex documentation explicitly identifies limit and license exceptions as exceptions that can't be caught, and documents assertion failures through the built-in `AssertException` type. See the Salesforce Apex exception-handling documentation at Exception Statements and the built-in exception reference at Exception Class and Built-In Exceptions.

QUESTION NO: 32

Which statement is true about using ConnectApi namespace (also called Chatter in Apex)? (Choose two.)

A. Chatter in Apex methods honor the 'with sharing' and 'without sharing' keywords

B. Chatter in Apex operations are synchronous, and they occur immediately

C. Chatter in Apex methods do not run in system mode; they run in the context of the current user

D. Many test methods related to Chatter in Apex require the IsTest (SeeAllData=true) annotation

ANSWER: C D

Explanation:

Chatter in Apex methods do not run in system mode; they run in the context of the current user. ConnectApi calls enforce the current user's Chatter access and visibility context, independently of the sharing declaration on the surrounding Apex class. This behavior is important when building features such as custom feeds, posts, comments, groups, and topics: the user invoking the code must be allowed to perform and view the requested Chatter operation. Salesforce documents this user-context behavior as a core consideration for Chatter in Apex.

Many test methods related to Chatter in Apex require the IsTest (SeeAllData=true) annotation because ConnectApi methods often operate on existing Chatter entities and organization data, such as users, groups, and feed content. Unlike ordinary unit-test database records created within a test, some required Chatter context or resources cannot always be fully created or substituted in an isolated test. Salesforce's Chatter in Apex testing guidance therefore identifies cases where access to existing organization data is necessary. See Chatter in Apex (ConnectApi) documentation and Salesforce test data and SeeAllData documentation.

QUESTION NO: 33

Salesforce users consistently receive a "Maximum trigger depth exceeded" error when saving an Account.

How can a developer fix this error?

A. Split the trigger logic into two separate triggers.

B. Use a helper class to set a Boolean to TRUE the first time a trigger is fired, and then modify the trigger to only fire when the Boolean is FALSE.

C. Convert the trigger to use the @future annotation, and chain any subsequent trigger invocations to the Account object.

D. Modify the trigger to use the isMultiThread=true annotation.

ANSWER: B

Explanation:

Use a helper class to set a Boolean to TRUE the first time a trigger is fired, and then modify the trigger to only fire when the Boolean is FALSE. This implements a recursion guard: a static Boolean declared in an Apex helper class retains its value for the duration of the current transaction. The trigger can check that flag before performing logic that updates an Account or related records in a way that would cause the Account trigger to execute again. On the initial entry, the logic sets the flag and performs the required work. If the same trigger is invoked again during that save transaction, the flag prevents the recursive processing path from running again.

The guard should be designed carefully for bulk processing. Rather than using the Boolean as a substitute for correct bulkified logic, place the trigger handler's intended processing behind the guard and operate on record collections. In more

complex scenarios, a static `Set<Id>` can track records already processed, allowing legitimate processing for other records in the transaction while preventing repeated work for the same record. Salesforce documents that static variables are scoped to a transaction, which makes them appropriate for transaction-level recursion control. See Apex static variables and Apex triggers.

QUESTION NO: 34

A developer Is writing a Listener for implementing outbound messaging.

Which three considerations must the developer keep in mind in this case?

Choose 3 answers

A. Messages can be delivered out of order.

B. Messages are retried Independent of their order In the queue.

C. The session in an outbound message Is scoped for both API requests and UT requests.

D. The Organization 1D is included only in the first outbound message.

E. The Listener must be reachable from the public internet.

ANSWER: A B E

Explanation:

Messages can be delivered out of order. Outbound Messaging is an asynchronous integration mechanism: Salesforce places notifications on its delivery queue and does not guarantee that notifications reach the listener in the same sequence as the underlying record changes. A listener should therefore be designed to be idempotent and able to evaluate each notification safely even when a more recent update arrives before an earlier one.

Messages are retried Independent of their order In the queue. Salesforce tracks delivery acknowledgment for outbound notifications and retries a notification when the endpoint does not return the expected acknowledgment. Retry processing is independent, so a delayed or failing notification must not be assumed to block, or remain ordered relative to, other notifications. The listener should tolerate duplicate deliveries and use record identifiers and update data to handle them reliably.

The Listener must be reachable from the public internet. Salesforce delivers outbound messages by making a SOAP call to the configured endpoint URL. Consequently, the listener must be available to Salesforce's delivery infrastructure over the network, with an endpoint configuration that Salesforce can access and that can return the required acknowledgment response. See Salesforce's Outbound Messaging API documentation and outbound messaging setup guidance.

QUESTION NO: 35

Invokable methods accept sObjects as parameters.

A. True

B. False

ANSWER: A

Explanation:

True is correct. An Apex method annotated with `@InvocableMethod` can receive Salesforce records as input, using a list of a specific sObject type, such as `List<Account>` or `List<Contact>`. This enables declarative tools that invoke Apex, including Flow and Process Builder where applicable, to pass collections of records to Apex for bulk processing. The list-based parameter structure is important because invocable Apex is designed to operate efficiently when automation runs for multiple records in one transaction.

Invocable methods have a constrained public interface so that Salesforce automation tools can discover and call them. The method must be static and can have one input parameter; that input can be a list of sObjects, primitive values, Apex-defined types, or supported request-wrapper types. When the automation supplies record data, using an sObject list gives the method direct access to the record fields needed for its logic. Salesforce documents these supported invocable-method parameter patterns and the related bulk-processing expectations in its Apex Developer Guide: @InvocableMethod Annotation and Invocable Apex Methods.

QUESTION NO: 36

A developer has created a Visualforce page that uses a third-party JavaScript framework. The developer has decided to supply data to the JavaScript functions using JavaScript Remoting for Apex Controllers.

What is the correct syntax to declare a remote method in Apex? (Choose two.)

A. @RemoteAction global static String getTable()

B. @RemoteAction global String getTable()

C. @RemoteAction public static String getTable()

D. @RemoteObject global static String getTable()

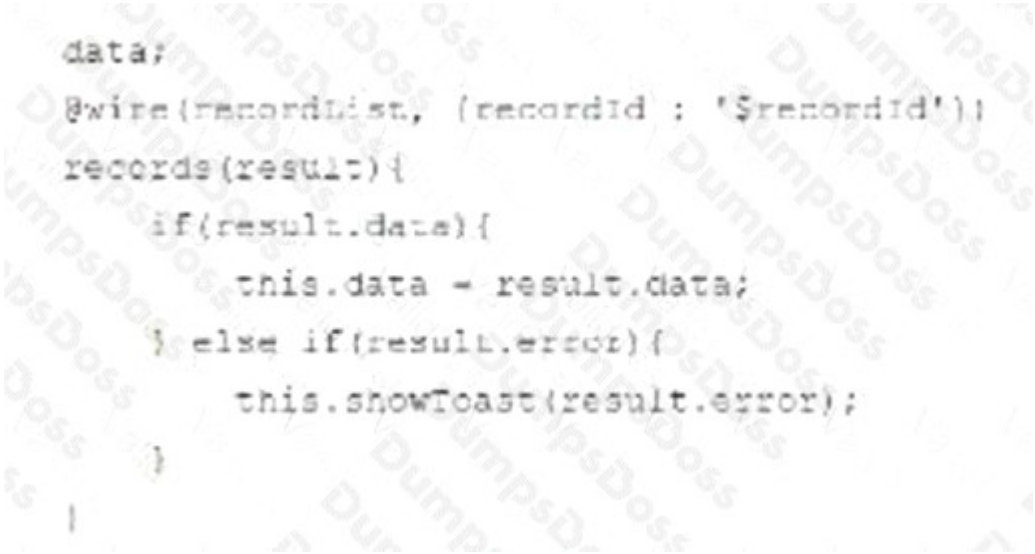
ANSWER: A C

Explanation:

For JavaScript Remoting, an Apex controller method must be exposed with the `@RemoteAction` annotation and must be declared `static`. The static requirement is important because the Visualforce remoting framework invokes the method directly through the controller class rather than through a controller instance. Therefore, both `@RemoteAction global static String getTable()` and `@RemoteAction public static String getTable()` are valid declarations. The method's return type can be a `String`, as shown, provided that the value is suitable for serialization and use by the client-side JavaScript callback. A `public` remote action is appropriate for a controller used within the same namespace, while `global` provides broader visibility when cross-namespace access is required. In either case, combining the appropriate access modifier with `static` and `@RemoteAction` makes the method available to JavaScript code through Visualforce remoting. Salesforce's Visualforce documentation describes remote actions as static controller methods annotated with `@RemoteAction`, and the Apex reference documents the annotation and its use in remoting calls. See JavaScript Remoting for Apex Controllers and RemoteAction Annotation Reference.

QUESTION NO: 37

A developer is creating a Lightning web component that displays a list of records in a lightning-datatable. After saving a new record to the database, the list is not updating.



```
data;
@wire(recordList, {recordId : '$recordId'})
records(result){
    if(result.data){
        this.data = result.data;
    } else if(result.error){
        this.showToast(result.error);
    }
}
```

What should the developer change in the code above for this to

happen?

A. Call rafrasnApex() ON this.dazta.

B. Create a new variable to store the result and annotate it with @track.

C. Create a variable to store the result and call refreshApex().

D. Add the @track decorator to the data variable.

ANSWER: C

Explanation:

Create a variable to store the result and call refreshApex(). When an Apex method is wired to a Lightning web component, the wire service provides a result object containing the returned data or error information. The component must retain this complete result object—not just the data array—so that it can later pass the wired result to `refreshApex()`. After the record-save operation succeeds, calling `refreshApex(wiredResult)` marks the wired Apex data as stale and requests the latest records from the server. Once the wire service receives the updated response, it assigns the new data to the component property used by `lightning-datatable`, and LWC re-renders the table reactively. This pattern is appropriate for cacheable Apex methods used with `@wire`, because the original data can otherwise remain available from the client-side cache. Store the result in a property in the wired-function handler, then invoke `refreshApex()` with that property only after the database save has completed successfully. Salesforce documents this refresh pattern for refreshing data returned by wired Apex and other wire adapters. See Client-Side Caching of Apex Method Results and Refresh View API.

QUESTION NO: 38

A developer is writing code that requires making callouts to an external web service. Which scenario necessitates that the callout be made in an @future method?

A. The callouts will be made in an Apex Test class.

B. The callouts will be made in an Apex Trigger.

C. The callout could take longer than 60 seconds to complete.

D. over 10 callouts will be made in a single transaction.

ANSWER: B

Explanation:

The callouts will be made in an Apex Trigger. is correct because trigger execution is part of the database transaction that caused the record change. Salesforce does not allow a trigger to perform a synchronous HTTP callout directly after it has performed uncommitted database work, because the platform must avoid holding an open transaction while waiting for an external system. The callout therefore needs to be moved to asynchronous processing. An `@future(callout=true)` method is a supported mechanism for this pattern: the trigger passes primitive values or collections to the future method, the original transaction can complete, and the asynchronous method makes the HTTP request in a separate transaction.

The future method should be declared `static` and must use the `callout=true` annotation parameter to permit HTTP callouts. In a production design, the developer should also ensure that the payload contains the data needed by the external service and that failures are handled appropriately, because the callout no longer executes as part of the initiating transaction. Salesforce documents future methods as an asynchronous option for callouts initiated from trigger-based processing. See Invoke Future Methods and Apex Triggers.

QUESTION NO: 39

After a Platform Event is defined in a Salesforce org, events can be published via which two mechanisms? (Choose two.)

A. Internal Apps can use Outbound Messages

B. External Apps can use the REST API

C. Internal Apps can use Process Builder

D. External Apps require a custom Apex web service

ANSWER: B C

Explanation:

Platform events can be published by internal Salesforce automation using Process Builder. A process can evaluate record changes or other configured criteria and use a Create a Record action to create a platform-event record; creating that event record publishes the event to the event bus. This makes platform events useful for decoupling declarative business automation from downstream event consumers.

External applications can publish a platform event through Salesforce APIs, including the REST API sObject resource for the platform event's API name. The external client authenticates to Salesforce and submits a create request for the event record; Salesforce then publishes it for subscribers. Salesforce also supports API-based event publishing through other supported interfaces depending on the integration architecture. The key principle is that both declarative internal automation and authenticated API clients can act as publishers, without requiring a custom Apex web service. See Salesforce's Platform Events documentation and the Publish Platform Events with APIs guide.

QUESTION NO: 40

The "Webservice" keyword _____.

A. Method must be static, and class must be global

B. Can be used on all classes

C. Used for any member variables included

D. All of the above

ANSWER: A

Explanation:

Method must be static, and class must be global is correct because the `webservice` keyword exposes Apex functionality for consumption through a SOAP web service. Salesforce requires an Apex method intended for SOAP exposure to be declared `global static` and annotated with `webservice`. The containing class must also be declared

`global`, which makes the class visible outside the org's namespace and allows Salesforce to include the method in the generated WSDL.

For example, a SOAP operation can be declared as `global static webservice String getStatus()` inside a global Apex class. After deployment, Salesforce can generate an enterprise or custom WSDL that external clients use to invoke that operation. The global access level and static method requirement are therefore fundamental parts of creating an Apex SOAP web service, rather than optional conventions. Salesforce documents the keyword's use and the required method declaration in its Apex webService annotation documentation. Additional implementation guidance is available in the Apex SOAP Web Services guide.

QUESTION NO: 41

Universal Containers is using a custom Salesforce application to manage customer support cases. The support team needs to collaborate with external partners to resolve certain cases. However, they want to control the visibility and access to the cases shared with the external partners. Which Salesforce feature can help achieve this requirement?

A. Apex managed sharing

B. Sharing sets

C. Role hierarchy

D. Criteria-based sharing rules

ANSWER: B

Explanation:

Sharing sets are the appropriate declarative feature for providing controlled record access to external Experience Cloud users. A sharing set grants access by mapping values on the external user's account or contact to fields on the target record. For example, Universal Containers can configure access so that an external partner user can view or update Case records whose Account matches the user's Account. This makes the shared cases visible only to users associated with the relevant customer or partner relationship, rather than exposing all support cases.

Sharing sets are particularly useful for external-user license types that do not use roles, including high-volume external users. They support scalable, configuration-based sharing without requiring custom code to create and maintain individual sharing records. Administrators can define the profiles that receive the access, the objects being shared, the access level, and the account/contact-to-record field mappings. This directly meets the requirement to collaborate with outside partners while retaining precise control over which Case records those partners can access. Salesforce documents sharing sets as a mechanism for granting external users access to records associated with their accounts or contacts: Salesforce Help: Sharing Sets. See also Salesforce Developer Guide: Sharing Sets.

QUESTION NO: 42

A developer must create a custom pagination solution for accessing approximately 2000 records and displaying 50 records on each page. Data from Salesforce will be accessed via an API and not via Apex.

How can the developer meet these requirements? (Choose two.)

A. Use a StandardSetController

B. Use CURSOR 50 in SOQL queries

C. Use OFFSET in SOQL queries

D. Use LIMIT 50 in SOQL queries

ANSWER: C D

Explanation:

Use OFFSET in SOQL queries and Use LIMIT 50 in SOQL queries together provide API-based pagination for this volume. `LIMIT 50` defines the fixed page size, so each request returns no more than 50 records. The client can request successive pages by increasing the `OFFSET` value in increments of 50: for example, use `OFFSET 0` for the first page, `OFFSET 50` for the next page, and continue through `OFFSET 1950` for the final page of approximately 2,000 records. Include an `ORDER BY` clause on a stable, deterministic field or set of fields so that records remain in a predictable order while users move between pages. Salesforce supports `OFFSET` values up to 2,000 rows, which fits this requirement when the data set is approximately 2,000 records. These clauses are part of SOQL and can be supplied in queries issued through Salesforce APIs; no Apex controller is required. See Salesforce's SOQL OFFSET documentation and the SOQL LIMIT documentation.

QUESTION NO: 43

Given the following code:

Assuming there were 10 Contacts and five Accounts created today, what is the expected result?

A. System.QueryException: Too many DML Statement errors on Contact

B. System.QueryException: List has more than one row for Assignment on Account

C. Systemn.LimitException: Too many SOQL Queries on Account

D. System.LimitException: Too many SOQL Queries on Contact

E. Cannot be determined because the referenced code is not included.

ANSWER: E

Explanation:

Cannot be determined because the code referenced by the question is not present. The number of Contacts and Accounts created today is not enough to establish an execution result by itself. The missing code would determine whether the records are queried in a loop, whether a query is assigned to a single sObject rather than a list, whether DML is performed, and whether the transaction is synchronous or asynchronous. Each of those details is essential to evaluating whether the transaction completes or raises a particular exception. For example, Salesforce enforces per-transaction governor limits, but the applicable limit and the number of operations consumed can only be calculated from the omitted statements and their execution context. Salesforce also documents that SOQL can return either a single sObject or a collection depending on how the query result is assigned, which affects the possible runtime behavior. Because no executable code is supplied, selecting a specific query, DML, or assignment exception would be unsupported. See Salesforce's Apex Governor Limits and SOQL and SOSL Reference for the governing behavior.

QUESTION NO: 44

Which three approaches should a developer implement to obtain the best performance for data retrieval when building a Lightning web component? (Choose three.)

A. Use (Cacheable=true) whenever possible.

B. Use the Lightning Data Service.

C. Use layoutTypes: ['Full'] to display a set of fields.

D. Use lazy load for occasionally accessed data.

E. Use getRecordUi to obtain metadata.

ANSWER: A B D

Explanation:

For efficient Lightning web component data retrieval, use `(Cacheable=true)` whenever possible for Apex methods that only read data. A cacheable Apex method can be called with `@wire`, and Lightning Data Service client-side caching can satisfy repeat requests without a server round trip when the cached result is valid. This reduces latency and avoids unnecessary Apex execution.

Use the Lightning Data Service for record-oriented UI data. Its wire adapters provide a managed, shared client cache and keep record data synchronized across components that use LDS. Request only the fields required by the component so the application transfers and processes the smallest useful payload.

Use lazy load for occasionally accessed data. Load essential data first, then retrieve secondary data only in response to an explicit user action or when the relevant UI is displayed. This improves initial page responsiveness, decreases the number of initial requests, and prevents retrieving data that a user may never view. Salesforce's guidance on client-side caching and Apex wire methods is available in Client-Side Caching of Apex Method Results. Details on LDS record access and its cache are available in Use the Wire Service to Get Data.

QUESTION NO: 45

An org has a Process Builder process on Opportunity that sets a custom field,CommissionBaseAmount__c, when an Opportunity is edited and the Opportunity's Amount changes.

A developer recently deployed an Opportunity before update trigger that uses the CommissionBaseAmount__c and complex logic to calculate a value for a custom field CommissionAmount_c, when an Opportunity stage changes to Closed/Won.

Users report that when they change the Opportunity to Closed/Won and also change the Amount during the same save, the C:rr.i;5icn

A.

T.cur.t c is incorrect.

Which two actions should the developer take to correct this problem? Choose 2 answers

A. Call the trigger from the process.

B. Uncheck the recursion checkbox on the process.

C. Use a static Boolean variable in the trigger.

D. Call the process from the trigger.

ANSWER: B C

Explanation:

Uncheck the recursion checkbox on the process and use a static Boolean variable in the trigger. A Process Builder field update can cause the Opportunity to enter the save order of execution again. Allowing the process to evaluate repeatedly in the same transaction adds avoidable re-entry and makes the sequence of declarative updates and Apex logic harder to control. Disabling repeated process evaluation ensures that the amount-driven update is performed once for that transaction.

The trigger must also explicitly manage its own re-entry. A static Boolean variable is transaction-scoped, so it can retain state across repeated trigger invocations caused by automation on the same Opportunity save. The trigger can use that state to ensure its commission-calculation logic is performed only during the intended execution path, after the required commission-base value is available, rather than recalculating from an intermediate value. This is especially important because a before-update trigger can be invoked again when downstream automation updates the record. Salesforce documents that workflow-style field updates can cause update triggers to execute again, and recommends guarding Apex automation against recursion where appropriate. See Salesforce's order of execution documentation and Apex trigger best practices.

QUESTION NO: 46

A customer requires that when the billing address field on an Account gets updated, the address field on all its related contact records should reflect the same update.

How can this requirement be met with minimal customizations?

A. Create an After Trigger on Account to update its related contact records on update

B. Create a Workflow Rule on Account to update related child Contact records

C. Create a Lightning Process on Account to update related child Contact records

D. Create a scheduled batch job that updates all contact address fields based on the related account record

ANSWER: C

Explanation:

Create a Lightning Process on Account to update related child Contact records because a declarative process can evaluate an Account update and perform an Update Records action against the Contacts related to that Account. The process starts when an Account record is created or edited, uses criteria that detect a change to the relevant billing-address fields, and updates each related Contact's corresponding mailing-address fields with values from the Account. This meets the synchronization requirement without Apex code, scheduled processing, or a separate integration.

For the terminology used by this question, "Lightning Process" refers to a process configured in Process Builder. Salesforce has since retired Process Builder for new automation and recommends record-triggered Flow as the strategic replacement, but the process capability tested here is the declarative ability to update related child records from an Account update. In an implementation using current tooling, the equivalent design would be an after-save record-triggered flow on Account that obtains the related Contacts and updates their address fields. See Salesforce's Process Builder overview and record-triggered flow documentation.

QUESTION NO: 47

Universal Containers allows customers to log into a Salesforce Community and update their orders via a custom Visualforce page. Universal Containers' sales representatives can edit the orders on the same Visualforce page.

What should a developer use in an Apex test class to test that record sharing is enforced on the Visualforce page?

A. Use System.profllels() to test as an administrator and a community user,

B. Use System.profilels() to test as a sales rep and a community user.

C. Use System.runAs() to test as a sales rep and a community user.

D. Use System.runAs() to test as an administrator and a community user.

ANSWER: C

Explanation:

Use System.runAs() to test as a sales rep and a community user. `System.runAs` lets an Apex test execute code in the context of a specified user, so the test can model the two real audiences accessing the Visualforce page: an internal sales representative and an external community user. The test should create suitable users, establish the relevant Order records and sharing setup, then invoke the page controller or its underlying service logic inside separate `System.runAs` blocks. Assertions can verify that each user can query and update only the Order records granted through the organization's sharing model, sharing rules, ownership, or manual/Apex-managed sharing.

This approach is specifically suited to validating record-level access because Salesforce applies record sharing for the user supplied to `runAs` in a test. The Visualforce controller must also be designed to respect sharing—for example, by using an Apex class declared `with sharing` where appropriate—so that the user-context test reflects the page's intended security behavior. Salesforce notes that `runAs` is used to test behavior under different user sharing contexts; it does not itself enforce object permissions or field-level security. See System.runAs documentation and Apex sharing rules documentation.

QUESTION NO: 48

A business requires that every parent record must have a child record. A developer writes an Apex method with two DML statements to insert a parent record and a child record.

A validation rule blocks child records from being created. The method uses a try/catch block to handle the DML exception.

What should the developer do to ensure the parent always has a child record?

A. Use Database.insert () and set the allorNone parameter to true.

B. Use addError () on the parent record if an error occurs on the child record.

C. Set a database savepoint to rollback if there are errors.

D. Delete the parent record in the catch statement when an error occurs on the child record DML operation.

ANSWER: C

Explanation:

Set a database savepoint to rollback if there are errors. A savepoint lets the Apex transaction explicitly return to a known state when the child insert throws a DML exception. The developer should establish the savepoint before inserting the parent, perform the parent and child DML in the `try` block, and call `Database.rollback(savepoint)` in the `catch` block. When the child is rejected by its validation rule, rolling back removes the previously inserted parent as well. Consequently, the database cannot retain a parent that lacks its required child record.

This pattern is particularly appropriate when exception handling is needed for logging, user messaging, or other controlled recovery logic, while preserving the atomic business requirement across multiple DML operations. Salesforce transaction control supports savepoints and rollbacks within the same transaction; rollback reverses changes made after the savepoint. The developer must place the savepoint before any records whose persistence depends on successful completion of the related work. See Salesforce's Apex Transaction Control documentation and the Database class reference.

QUESTION NO: 49

A developer wants to call an Apex Server-side Controller from a Lightning Aura Component. What are two limitations to the data being returned by the Controller? (Choose two.)

A. A custom Apex Class can be returned, but only the values of public instance properties and methods annotated with @AuraEnabled are serialized and returned.

B. Lists of Custom Apex Classes cannot be returned by Apex Controllers called by Lightning Aura Components.

C. Basic data types are supported, but defaults, such as maximum size for a number, are defined by the objects that they map to.

D. Only Basic data types and sObjects are supported as return types for Apex Controllers called by Lightning Aura Components.

ANSWER: A C

Explanation:

A custom Apex class can be returned to a Lightning Aura component, provided its exposed state is explicitly made available to the framework. The Aura serialization boundary does not automatically expose an entire arbitrary Apex object. Public instance properties and accessors intended for client use must be annotated with `@AuraEnabled`, allowing the framework to serialize the relevant values into the response delivered to JavaScript. This makes a purpose-built wrapper or data-transfer class a common and appropriate return type for controller actions.

Basic data types are also supported, but the values crossing the Aura boundary must conform to the type rules and limits of the Salesforce data model and the objects to which values are mapped. For example, numeric values are interpreted according to their applicable Apex/Salesforce representations and field-level constraints when used with sObject data. Developers should therefore use supported, serializable value types and design returned data with the target object and

client-side consumption in mind. Salesforce documents the use of `@AuraEnabled` to expose Apex members for Lightning clients and the server-side action model in the AuraEnabled reference and Aura server-side actions documentation.

QUESTION NO: 50

A Salesforce org has more than 50,000 contacts. A new business process requires a calculation that aggregates data from all of these contact records. This calculation needs to run once a day after business hours. Which two steps should a developer take to accomplish this?

A. Use the @future annotation.

B. Implement the Database.Batchable interface.

C. Implement the Schedulable interface.

D. Implement the Queueable interface.

ANSWER: B C

Explanation:

Implement the Database.Batchable interface to process the contact population in manageable scopes rather than attempting to handle more than 50,000 records in one Apex transaction. Batch Apex uses a QueryLocator to retrieve a very large query result set—up to 50 million records—and executes each scope as a separate transaction with its own governor limits. This makes it appropriate for a daily calculation that must examine and aggregate values across the organization's contacts. The batch job can collect or persist aggregate results as required by the business process.

Implement the Schedulable interface to initiate that batch job at a defined daily, after-hours time. A scheduled Apex class supplies an `execute` method, where it can call `Database.executeBatch` for the batch implementation. The job is then registered through `System.schedule` using a cron expression, allowing the organization to control exactly when the overnight processing begins. This pairing separates timing from scalable record processing and is the standard Apex design for scheduled, high-volume work. See Salesforce's Batch Apex documentation and Scheduled Apex documentation.

QUESTION NO: 51

A developer is asked to find a way to store sacret data with an ability to specify which profiles and users can access which secrets.

What should be used to store this data?

A. system.Cookie class

B. Custom settings

C. Static resources

D. Custom metadata

E. Named Credentials with External Credentials

ANSWER: E

Explanation:

Named Credentials with External Credentials should be used to store sensitive authentication data, such as API keys, passwords, tokens, and client secrets, for outbound integrations. Salesforce encrypts the credential values and keeps them separate from Apex code, so developers do not need to hard-code or expose secrets in configuration records. An External Credential defines the authentication protocol and its secret parameters, while a Named Credential defines the endpoint and associates it with that External Credential.

Access can be controlled through the External Credential's principals. An administrator grants users access to a principal through a permission set, permission set group, or profile, allowing the organization to determine which users are permitted to use a particular credential and its stored secrets. Apex callouts can then reference the Named Credential endpoint using the `callout:` syntax, and Salesforce supplies the appropriate authentication details securely at runtime.

This approach is the Salesforce-supported model for securely managing integration credentials and assigning their use to authorized users. See Salesforce's Named Credentials documentation and External Credential principal access documentation.

QUESTION NO: 52

As part of point-to-point integration, a developer must call an external web service which, due to high demand, takes a long time to provide a response. As part of the request, the developer must collect key Inputs from the end user before making the callout. Which two elements should the developer use to Implement these business requirements?

Choose 2 answers

A. Lightning web component

B. 2 Apex method that returns a Continuation object

C. Screen Flow

D. Process Builder

ANSWER: A B

Explanation:

Lightning web component is appropriate for the interactive client-side portion of this integration. It can present a responsive user interface, collect the required values from the end user, perform client-side validation as needed, and invoke an Apex controller method with those values. This directly supports the requirement that inputs be gathered before the external request is initiated.

2 Apex method that returns a Continuation object provides the scalable asynchronous callout pattern required for an external service with a potentially long response time. A Continuation allows Apex to initiate one or more long-running HTTP requests without holding the application server thread while waiting for the remote system. The method returns the Continuation, and Salesforce later invokes a callback method to process the completed response. For Lightning components, the Apex methods used in this pattern must be exposed for client invocation and annotated with `@AuraEnabled(continuation=true)`; the callback is similarly exposed. Combining the Lightning web component's input collection with Apex Continuation processing delivers a responsive user experience while handling the long-running point-to-point web-service request efficiently. See Salesforce's Use Apex Continuations documentation and the Apex Developer Guide: Continuations.

QUESTION NO: 53

A business process requires sending new Account records to an external system. The Account Name, Id, CreatedDate, and CreatedById must be passed to the external system in near real-time when an Account is inserted without error.

How should a developer achieve this?

A. Use a before insert trigger and a Queueable class

B. Use a before insert trigger and an @future method

C. Use a Process Builder that calls an @InvocableMethod method

D. Use a Workflow rule that calls an @InvocableMethod method

ANSWER: C

Explanation:

Use a Process Builder that calls an @InvocableMethod method. Process Builder evaluates record changes after the Account has been saved successfully, so the inserted record has its server-generated values, including `Id`, `CreatedDate`, and `CreatedById`. The invoked Apex method can accept the Account records as a collection, construct the required outbound payload, and perform the integration logic. This satisfies the requirement to send the data only after a successful insert, rather than attempting to access fields that are not yet finalized during a before-insert context.

An Apex method exposed for declarative invocation uses the `@InvocableMethod` annotation and is designed to receive a list of input values, enabling bulk-safe handling when multiple Accounts are created in the same transaction. The integration implementation should also be designed with platform callout and transaction considerations in mind, such as delegating longer-running work asynchronously where appropriate. Although Flow is Salesforce's current strategic automation tool for new implementations, Process Builder is the mechanism specified in this answer and can invoke Apex actions using an invocable method. See Salesforce's documentation for @InvocableMethod and the platform's order of execution.

QUESTION NO: 54

A company has reference data stored in multiple custom metadata records that represent default information and delete behavior for certain geographic regions.

When a contact is inserted, the default information should be set on the contact from the custom metadata records based on the contact's address information.

Additionally, if a user attempts to delete a contact that belongs to a flagged region, the user must get an error message.

Depending on company personnel resources, what are two ways to automate this?

Choose 2 answers

A. Remote action

B. Flow Builder

C. Apex trigger

D. Apex invocable method

ANSWER: B C

Explanation:

Flow Builder can implement this declaratively with record-triggered flows on Contact. A flow that runs when a Contact is created can use the address fields to locate the applicable custom metadata record and assign its default values to the incoming Contact. A delete-triggered flow can evaluate the applicable geographic-region configuration and use the Custom Error element to prevent the deletion and display a meaningful error to the user. This provides an automation approach that can be maintained by appropriately skilled declarative administrators.

Apex trigger can implement the same requirements in code. A before-insert trigger can query the relevant custom metadata configuration and populate Contact defaults before the record is saved. A before-delete trigger can identify Contacts in regions configured as protected and call `addError()` on the record, which stops the delete operation and returns an error to the user. This approach is appropriate where the organization has developer resources or needs more sophisticated logic, reuse, and automated test coverage. Custom metadata types are designed for application configuration and are accessible from Apex and declarative tools. See Salesforce's Custom Error flow element documentation and Apex trigger documentation.

QUESTION NO: 55

What is the most efficient way in Visualforce to show information based on data filters defined by an end-user for a large volume of data?

A. Use the rendered condition in Visualforce to limit data based on data filters

B. Use filter conditions in a SOQL query to limit data based on data filters

C. Use an Apex controller to refine raw data based on data filters and store the result in a transient variable

D. Use an Apex controller to refine raw data based on data filters and store the result in a static variable

ANSWER: B

Explanation:

Use filter conditions in a SOQL query to limit data based on data filters. Applying the user's filter criteria directly in SOQL lets Salesforce's database query engine return only the records needed for the Visualforce page. This minimizes the number of rows retrieved into Apex, reduces heap consumption, and avoids unnecessary CPU time that would otherwise be spent loading and processing records that will not be displayed. For large data volumes, the query should use selective filter fields where possible, ideally fields supported by appropriate indexes, and should retrieve only the fields required by the page. When the result set can be large, the controller can also use pagination mechanisms such as `StandardSetController` or `OFFSET` where appropriate, so the page handles records in manageable groups. Salesforce's SOQL guidance emphasizes filtering queries effectively and keeping queries selective, particularly when working with large objects or tables. Visualforce pages invoke Apex to obtain their data, so querying only the filtered data is the most scalable design before rendering occurs. See Salesforce's SOQL and SOSL Reference and Working with Very Large SOQL Queries.

QUESTION NO: 56

A developer is creating a page in App Builder that will be used in the Salesforce mobile app.

Which two practices should the developer follow to ensure the page operates with optimal performance?

Choose 2 answers

A. Limit five visible components on the page.

B. Limit 25 fields on the record detail page.

C. Limit the number of Tabs and Accordion components.

D. Analyze the page with Performance Analysis for App Builder.

ANSWER: B D

Explanation:

Limit 25 fields on the record detail page. is a Salesforce mobile-performance guideline. Record detail pages require the client to render each displayed field and retrieve supporting record data. Keeping the field count to 25 or fewer reduces rendering work and the amount of data that must be processed, which is particularly important on mobile devices where network latency, memory, and CPU resources can vary significantly. Developers should prioritize the fields that users need for the mobile task and make less frequently used information available through other focused experiences when appropriate.

Analyze the page with Performance Analysis for App Builder. is also a recommended practice because the App Builder tool evaluates a Lightning record page against Salesforce performance guidance and identifies page-design conditions that can affect load time and responsiveness. It provides actionable feedback while the page is being designed, allowing the developer to resolve performance concerns before activation. This creates a repeatable validation step rather than relying only on manual testing across devices and network conditions. See Salesforce's Performance Analysis for App Builder documentation and the Salesforce Lightning Web Components performance guidance.

QUESTION NO: 57

What is a consideration when testing batch Apex? (Choose two.)

A. Test methods must execute the batch with a scope size of less than 200 records

B. Test methods must call the batch execute() method once

C. Test methods must use the @isTest (SeeAllData=true) annotation

D. Test methods must run the batch between Test.startTest() and Test.stopTest()

ANSWER: A D

Explanation:

When testing Batch Apex, the batch should be designed and invoked so that the test causes only one batch execution. A test context is limited to one `Database.executeBatch` invocation, and the test data and scope should therefore be kept within a single batch. Using a scope size below 200 records is a practical way to meet that constraint when the test dataset is sized appropriately. This lets the test exercise the batch's query, processing, and completion behavior without exceeding the asynchronous Apex limits that apply during tests.

The batch should also be started between `Test.startTest()` and `Test.stopTest()`. Batch Apex runs asynchronously, and `Test.stopTest()` forces asynchronous work queued after `Test.startTest()` to finish before assertions run. Consequently, the test can reliably query and verify the records updated by the batch, as well as outcomes produced in the `finish` method. Salesforce documents the one-batch limitation for tests and recommends using the test boundaries when testing asynchronous Apex. See Using Batch Apex and Testing Asynchronous Apex.

QUESTION NO: 58

When testing batch classes, what must a developer do? (Choose two.)

A. Use seeAllData=true

B. Encapsulate code in Test.startTest() and Test.stopTest()

C. Call the class' "execute" method

D. Limit the amount of records you test to < 200

ANSWER: B D

Explanation:

Batch Apex is asynchronous, so the call that enqueues the batch job should be placed after `Test.startTest()`, and `Test.stopTest()` must be called before making assertions. Calling `Test.stopTest()` causes asynchronous work submitted during the test context, including the batch job, to run synchronously before the test method continues. This lets the test reliably query the records changed by the batch and verify the expected outcome.

Apex tests are limited to executing one batch scope for a Batch Apex job. Therefore, the test data must be limited to no more than 200 records when using the default batch scope size. This ensures that all test records are processed in the single batch execution available to the test. A developer can create the necessary test data in the test itself and then invoke the batch through `Database.executeBatch`, enclosing that invocation in the test boundary. Salesforce documents the asynchronous-testing pattern at Testing Asynchronous Apex and the one-batch limitation in its Using Batch Apex guidance.

QUESTION NO: 59

Which of the following variables are not transmitted in the view state? (Choose two.)

A. Private

B. Transient

C. Public

D. Static

ANSWER: B D

Explanation:

`Transient` and `Static` variables are not transmitted in a Visualforce page's view state. Visualforce uses view state to preserve controller instance data between requests, allowing values to remain available when a user submits a form or performs another postback. Marking an instance variable as `transient` explicitly excludes it from view-state serialization. This is useful for data that can be recalculated, retrieved again when needed, or is only required during the current request. It can also reduce view-state size, which helps page performance.

`Static` variables are class-level values rather than state belonging to a particular controller instance. Because view state serializes applicable instance state for the specific page interaction, static fields are not included or restored through the view-state mechanism. Their values should therefore not be used to retain page-specific state across requests. Salesforce's Visualforce controller guidance identifies both transient and static variables as excluded from view state. See Visualforce Controller Variables and Visualforce View State.

QUESTION NO: 60

Refer to the code snippet below:

```
01 public void createChildRecord (String externalIdentifier){
02   Account parentAccount;
03
04   Contact newContact = new contact ();
05   newContact.Account = parentAccount ;
06
07   insert (newContact);
08 }
```

As part of an integration development effort, a developer is tasked to create an Apex method that solely relies on the use of foreign identifiers in order to relate new contact records to existing Accounts in Salesforce. The account object contains a field marked as an external ID, the API Name of this field is Legacy_Id_c.

What is the most efficient way to instantiate the parentAccount variable on line 02 to ensure the newly created contact is properly related to the Account?

A. Account parentAccount = new Account(Legacy_Id_c = externalIdentifier);

B. Account parentAccount = [SELECT Id FROM Account WHERE Legacy_Id_c = :externalIdentifier].Id;

C. Account parentAccount = [SELECT Id FROM Account WHERE Legacy_Id_c = :externalIdentifier];

D. Account parentAccount = new Account(); parentAccount.Id = externalIdentifier;

ANSWER: A

Explanation:

`Account parentAccount = new Account(Legacy_Id_c = externalIdentifier);` is the appropriate foreign-key reference pattern. Apex can represent a related parent record as an unsaved sObject containing an external ID field value rather than its Salesforce record ID. When that parent sObject is assigned to the contact's relationship field and the contact is inserted, Salesforce resolves the external ID value to the existing Account and establishes the relationship. This avoids a preliminary SOQL query, making the integration more efficient and allowing the operation to rely solely on the foreign identifier supplied by the external system.

This pattern is particularly useful for integrations because external IDs provide a stable, unique business key that can be used to identify Salesforce records without exposing or persisting Salesforce-generated IDs in the external application. The Account field used for this purpose must be configured as an External ID and its value must identify the intended Account. Salesforce documents this relationship approach as using an sObject reference populated with an external ID field for relationship resolution during DML. See Apex Developer Guide: Foreign Key Relationships and Salesforce Object Reference: Relationships Among Objects.

QUESTION NO: 61

A developer is building a complex commission calculation engine in Apex that is called from an Opportunity trigger. During QA it was reported that the calculations are incorrect. The developer has representative test data and passing test methods in their developer sandbox. Which three tools or techniques could the developer use to execute the code and pause it at key lines to visually inspect values of various Apex variables?

A. Apex Interactive Debugger

B. Apex Replay Debugger

C. Workbench

D. Developer Console

E. Breakpoints

ANSWER: A B E

Explanation:

Apex Interactive Debugger, **Apex Replay Debugger**, and **Breakpoints** together support source-level investigation of an Apex transaction. The Apex Interactive Debugger provides a live debugging session for eligible sandbox environments. A developer can start the relevant test or execution, place breakpoints in the commission logic, suspend execution when a breakpoint is reached, and inspect the current call stack and variable values before stepping through subsequent statements. Salesforce documents this as its interactive debugging capability for Apex in Visual Studio Code.

The Apex Replay Debugger supports equivalent step-through analysis after execution by replaying a captured debug log locally in Visual Studio Code. The developer obtains a sufficiently detailed log for the representative transaction or test, opens it in the debugger, and uses breakpoints to stop replay at the relevant source lines. At each stop, the debugger exposes the variables and execution context recorded in the log, which is especially useful for reproducing an issue without rerunning the transaction repeatedly.

Breakpoints are the technique that identifies the key executable lines at which either live execution or log replay should suspend. They enable focused inspection of intermediate commission values, inputs, and branching state. See Salesforce's Apex Interactive Debugger documentation and Apex Replay Debugger documentation.

QUESTION NO: 62

Which three actions must be completed in a Lightning web component for a JavaScript file In a static resource to be loaded? Choose 3 answers

A. Import the static resource.

B. Reference the static resource in a

C. Call loadScript.

D. Import a method from the platformResourceLoader.

E. Append the static resource to the DOM.

ANSWER: A C D

Explanation:

To load a JavaScript library stored as a static resource in a Lightning web component, the component must first import the static resource URL using the `@salesforce/resourceUrl` scoped module. This makes the deployed static-resource location available to the component's JavaScript code. The component must also import the `loadScript` method from `lightning/platformResourceLoader`. Salesforce provides this module specifically for loading external JavaScript and CSS resources in Lightning web components while maintaining Lightning platform security and component lifecycle behavior.

Finally, the component calls `loadScript(this, resourceUrl)`, commonly from `renderedCallback()`, to request and load the library. Because rendering can occur more than once, production components commonly use a Boolean flag to ensure the loading operation is initiated only once. `loadScript` returns a Promise, so code that depends on the external library should run after the Promise resolves. This standard resource-loader pattern is documented by Salesforce for third-party JavaScript libraries and static resources. See Use Third-Party JavaScript Libraries and Access Static Resources.

QUESTION NO: 63

A developer created the code to perform an HTP GET request to an external system.

```
public class ERPCatalog {
  private final ERP_CATALOG_URL = 'http//sampleCatalog.com/cat';

  public String getERPCatalogContents() {
    Http h = new Http();
    HttpRequest req = new HttpRequest();
    req.setEndpoint(ERP_CATALOG_URL);
    req.setMethod('GET');

    HttpResponse res = h.send(req);
    return res.getBody();
  }
}
```

When the code is executed, the callout is unsuccessful and the following error appears within the Developer Console:System.CalloutException: Unauthorized endpoint

Which recommended approach should the developer implement to the callout exception?

A. create a remote site setting configuration that includes the endpoint.

B. Annotate the getERPCatalogContents method With @Future (Callout-true)

C. use the setHeader () method to specify Basic Authentication.

D. Change the access modifier for ERPCatelog from Public to global

ANSWER: A

Explanation:

The correct approach is to create a remote site setting configuration that includes the endpoint. Salesforce requires an outbound HTTP callout target to be explicitly authorized before Apex can send a request to it. A `System.CalloutException: Unauthorized endpoint` occurs when the endpoint's protocol and host have not been allowlisted in Remote Site Settings, or when the configured URL does not match the callout URL sufficiently. The developer should add a Remote Site Setting with the external system's base URL—for example, the scheme and host used by the request—and mark it active. Once authorized, Salesforce can initiate the GET request to that host.

Remote Site Settings are the appropriate configuration when the callout code uses a direct endpoint URL. In a production-quality integration, a Named Credential can also be used to centralize endpoint and authentication configuration; however, for the stated exception and the available answer, authorizing the endpoint through a Remote Site Setting directly resolves the platform-level endpoint authorization requirement. Salesforce documents this requirement for Apex HTTP callouts in its Remote Site Settings documentation and provides setup guidance in Configure Remote Site Settings.

QUESTION NO: 64

A developer exposes an Apex method to a Lightning web component. The method queries Case records and returns a wrapper containing Case fields. The company requires the method to enforce both object-level and field-level security for the running user.

Which two actions should the developer take? Choose 2 answers

A. Declare the SOQL query with `WITH USER_MODE`.

B. Declare the Apex class `with sharing`.

C. Use `System.runAs()` in the Apex method.

D. Mark the method `cacheable=true`.

ANSWER: A B

Explanation:

Declare the SOQL query with `WITH USER_MODE`. User-mode database operations enforce the running user's object permissions, field-level security, and sharing rules. A query using `WITH USER_MODE` ensures that the method does not retrieve Case records or fields the user is not allowed to access.

Declare the Apex class `with sharing`. The `with sharing` keyword ensures that record-level sharing rules are respected by the class. Although user mode also enforces sharing for supported operations, declaring the class explicitly provides clear sharing behavior for other queries or logic that might be added to the class. The developer should design the entire entry point to enforce the required access model rather than relying on client-side visibility. See Enforce User Mode for Database Operations and Using the with sharing, without sharing, and inherited sharing Keywords.

QUESTION NO: 65

A comply his reference data stored m multiple custom metadata records that represent del auh information and delete behavior for certain geographic regions.

When a contact is inserted, the default information should be set on the contact from the custom metadata records based on the contact's address information. Additionally, if a user attempts to delete a contact that belongs to a flagged region, the user must get an error message.

Depending on company personnel resources, what are two ways to

automate this?

Choose 2 answers

A. Apex mvocaWe method

B. Remote action

C. Flow Builder

D. Apex trigger

ANSWER: C D

Explanation:

Flow Builder is a valid declarative solution when the organization has limited Apex development resources. Record-triggered flows can run when a Contact is created to retrieve the applicable custom metadata record based on the address or region and populate the required default field values. A record-triggered flow running for deletion can evaluate the region's configured delete behavior and use a Custom Error element to prevent the delete operation and present an error to the user. Salesforce documents record-triggered flow behavior and supported operations at Record-Triggered Flows.

Apex trigger is also appropriate where a programmatic implementation is preferred or the mapping and validation logic is complex. A before-insert trigger can obtain the relevant configuration from custom metadata and set Contact defaults before the record is saved. A before-delete trigger can evaluate the configured regional restriction and call `addError()` on the Contact, which stops deletion and returns an error message to the user. Custom metadata types are designed for deployable application configuration, and Apex can access custom metadata records without consuming SOQL-query limits through generated accessors such as `getAll()`. See Salesforce's Custom Metadata Type Apex Methods.

QUESTION NO: 66

Choose the correct definition for .

A. Allows controller methods to be called directly from JavaScript. Must be encapsulated in tags.

B. Sends an AJAX request according to the time interval you specify. If this ever gets re-rendered, it resets

C. Adds AJAX support to another component (e.g. onClick, onMouseUp, onFocus, etc.)

D. Can be associated with an AJAX request (actionFunction/actionSupport/actionPoller) and shows content conditionally depending on the status of the request (in progress/complete). Use the "id" field to specify name; use "status" field on related components to connect them

E. Signifies which components should be processed by the server when an AJAX request is generated

ANSWER: A

Explanation:

`<apex:actionFunction>` defines a JavaScript function that invokes a specified Apex controller action through an AJAX request. Salesforce generates the client-side function from the component's `name` attribute, so page JavaScript can call that function when needed rather than requiring the user to interact with a particular Visualforce component. The component must be placed inside an `<apex:form>`, because the AJAX request uses the form context to submit data and execute the server-side action. It can also use attributes such as `rerender`, `oncomplete`, and `status` to refresh page regions and coordinate client-side behavior after the request completes. This is particularly useful when custom JavaScript, rather than a standard Visualforce event handler, needs to trigger controller logic. Salesforce's Visualforce component reference identifies `actionFunction` as a way to invoke a controller action from JavaScript, and its AJAX guidance describes how Visualforce AJAX components submit requests and rerender designated content. See the Visualforce actionFunction component reference and Visualforce AJAX documentation.

QUESTION NO: 67

A developer needs to store variables to control the style and behavior of a Lightning Web Component. Which feature should be used to ensure that the variables are testable in both Production and all Sandboxes?

A. Custom Metadata

B. Custom Object

C. Custom Setting

D. Custom Variable

ANSWER: A

Explanation:

Custom Metadata is appropriate because it stores application configuration as metadata rather than ordinary org data. A developer can define a custom metadata type with fields representing the Lightning Web Component's style and behavior controls, then create metadata records containing the configuration values. Those records can be queried by Apex and exposed to the component through an Apex controller or other supported access pattern, allowing the component's behavior to be driven by declarative, environment-independent configuration.

The key requirement is that the configuration be available and testable consistently in Production and sandbox environments. Salesforce treats custom metadata type definitions and their records as deployable metadata, so they can be included in source-driven development, change sets, packages, and other deployment processes. This lets a team version the configuration alongside the component and deploy the same configuration to test environments before releasing it to Production. Salesforce also supports creating custom metadata records for test scenarios, helping Apex tests validate logic that depends on configuration without relying on mutable business data. See Salesforce's Custom Metadata Types in Apex documentation and the Custom Metadata Types overview.

QUESTION NO: 68

A developer built a Component to be used at the front desk for quests to self-register upon arrival at a kiosk. The developer is now asked to create a Component for the Utility Tray to alert Users whenever a guest has arrived at the front desk.

What should be used?

A. Application Event

B. DML Operation

C. Component Event

D. ChangeLog

ANSWER: A

Explanation:

Application Event is correct because the kiosk component and the Utility Tray component are separate components that need to communicate across the application rather than through a direct parent-child containment relationship. An Aura application event is designed for this broader communication pattern: a component fires the event when the guest registers, and an appropriate handler in the utility component can respond by displaying an alert or updating its UI. Application events are propagated throughout the application, allowing independently located components to participate without requiring the utility component to be a parent or child of the kiosk component. This creates a loosely coupled design, which is particularly useful for utility-bar functionality that can remain available while users navigate elsewhere in the app. The event definition can include attributes such as the guest's name, arrival time, or record ID, enabling the Utility Tray component to present a meaningful notification. Salesforce's Aura event documentation distinguishes application-level event communication from containment-based component event communication, and its utility bar documentation describes utility items as independently available workspace tools. See Salesforce Aura Application Events and Salesforce Lightning Utility Bar documentation.

QUESTION NO: 69

Universal Containers needs to integrate with their own, existing, internal custom web application. The web application accepts JSON payloads, resizes product images, and sends the resized images back to Salesforce.

What should the developer use to implement this integration?

A. An Apex trigger that calls an @future method that allows callouts

B. A platform event that makes a callout to the web application

C. A flow that calls an @future method that allows callouts

D. A flow with an outbound message that contains a session ID

ANSWER: A

Explanation:

An Apex trigger that calls an @future method that allows callouts is appropriate when the image-resizing request must begin in response to a Salesforce record change. Apex triggers cannot directly perform HTTP callouts, so the trigger delegates the

work to an asynchronous Apex method annotated with `@future(callout=true)`. That method can construct and send the required JSON payload to the internal web application using the Apex HTTP callout classes.

The asynchronous method can then read the HTTP response containing the resized image and persist it in Salesforce, for example by creating or updating a Salesforce File through `ContentVersion`, or by updating related record data. This separates the external network operation from the trigger transaction, avoiding a dependency on the remote application's response time while preserving a programmatic integration path for both JSON serialization and response handling.

The endpoint should be configured through a Named Credential where possible. Named Credentials centralize the endpoint and authentication configuration and allow Apex to use the named endpoint rather than hardcoding connection details. Salesforce documents asynchronous callouts from future methods and the `callout=true` requirement in the Apex Developer Guide. For HTTP request and response handling, see HTTP Callouts in Apex.

QUESTION NO: 70

A company has an Apex process that makes multiple extensive database operations and web service callouts. The database processes and web services can take a long time to run and must be run sequentially. How should the developer write this Apex code without running into governor limits and system limitations? 123

A. Use Apex Scheduler to schedul4e each process.56

B. Use Limits class to stop entire pr7ocess once governor limits are reached.8

C. Use multip9le @future methods for each process and callout.

D. Use Queueable Apex to chain the jobs to run sequentially.

ANSWER: D

Explanation:

Use Queueable Apex to chain the jobs to run sequentially. A Queueable class can perform asynchronous database work and can make HTTP callouts when it implements `Database.AllowsCallouts`. At the end of a queueable job's `execute` method, it can enqueue the next queueable job. This creates an ordered workflow: the next unit of work is submitted only after the current unit has completed, preserving the required sequence.

Chaining also divides a large process into separate asynchronous transactions. Each queued job receives its own asynchronous governor-limit context, allowing the developer to keep database processing and callouts in manageable units rather than attempting all work in one transaction. The implementation should persist or pass only the state needed by the following job, check for callout and DML outcomes as appropriate, and enqueue at most one successor from an asynchronous queueable execution. This pattern provides explicit programmatic control over the sequence while keeping individual transactions within platform execution constraints. Salesforce documents Queueable Apex and its ability to enqueue another job for chaining at Queueable Apex. For callouts from asynchronous Apex, see Asynchronous Callouts.