DUMPSARENA

Salesforce Certified Platform Developer I

Salesforce CRT-450

Version Demo

Total Demo Questions: 60

Total Premium Questions: 602

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

Topic Break Down

Topic	No. of Questions
Topic 1, Salesforce Fundamentals	99
Topic 2, Data Modeling and Management	112
Topic 3, Logic and Process Automation	137
Topic 4, User Interface	126
Topic 5, Testing	53
Topic 6, Debug and Deployment	17
Topic 7, Mix Questions	58
Total	602

QUESTION NO: 1

A software company is using Salesforce to track the companies they sell their software to in the Account object. They also use Salesforce to track bugs in their software with a custom object, Bug__c.

As part of a process improvement initiative, they want to be able to report on which companies have reported which bugs. Each company should be able to report multiple bugs and bugs can also be reported by multiple companies.

What is needed to allow this reporting?

A. Roll-up summary field of Bug__c on Account

B. Master-detail field on Bug__c to Account

C. Lookup field on Bug__c to Account

D. Junction object between Bug__c and Account

ANSWER: D

Explanation:

A junction object between Bug__c and Account is needed because the requirement is a many-to-many relationship: a single Account can be associated with many bug records, and a single bug record can be associated with many Accounts. In Salesforce, a junction object is a custom object that represents each individual association between two other objects. For example, a Bug Report record could identify one Account and one Bug__c record. Creating multiple Bug Report records then captures every company-to-bug association and makes the relationship reportable.

The junction object ordinarily has a relationship field to Account and a relationship field to Bug__c. When both parent objects support it, these are typically master-detail relationships; Salesforce identifies an object with two master-detail relationships as a junction object. The resulting model supports reporting on Accounts with their reported bugs, bugs with all reporting Accounts, and details stored on the association itself, such as the date reported, severity reported by the customer, or customer-specific reproduction notes.

Salesforce documents junction objects as the standard data-modeling pattern for relating records in a many-to-many arrangement. See Salesforce Object Relationships documentation and Salesforce Trailhead: Data Modeling.

QUESTION NO: 2

A developer needs to prevent users from saving a Case when the Case Priority is High and the Description field is blank.

Which declarative feature should the developer use?

A. A validation rule

B. A roll-up summary field

C. A workflow email alert

D. A compact layout

ANSWER: A

Explanation:

A validation rule is the appropriate declarative feature because it evaluates field values when a record is saved and prevents the save when its formula evaluates to true. A formula such as `AND(ISPICKVAL(Priority, 'High'), ISBLANK(Description))` enforces the stated condition and displays a configured error message to the user.

A required field setting would require Description for every Case, not only High Priority Cases. See Validation Rules.

QUESTION NO: 3

A company wants to create an employee rating program that allows employees to rate each other. An employee's average rating must be displayed on the employee record. Employees must be able to create rating records, but are not allowed to create employee records.

Which two actions should a developer take to accomplish this task? (Choose two.)

A. Create a trigger on the Rating object that updates a fields on the Employee object.

B. Create a lookup relationship between the Rating and Employee object.

C. Create a roll-up summary field on the Employee and use AVG to calculate the average rating score.

D. Create a master-detail relationship between the Rating and Employee objects.

ANSWER: C D

Explanation:

Creating a master-detail relationship between the Rating and Employee objects establishes Employee as the parent record and Rating as the related detail record. This relationship type is required because Salesforce roll-up summary fields are available on the master side of a master-detail relationship. Each rating can therefore be associated with the employee who is being rated, while access to the employee record itself can remain restricted through object permissions and sharing configuration.

Creating a roll-up summary field on the Employee and using AVG to calculate the average rating score provides a declarative, automatically maintained value on each employee record. The roll-up summary can aggregate the numeric rating-score field from all related Rating records and recalculate when ratings are created, edited, deleted, or undeleted. This meets the requirement without custom Apex automation, while employees can be granted permission to create Rating records without being granted permission to create Employee records. Salesforce documents that roll-up summary fields calculate values from related detail records and require a master-detail relationship. See Salesforce: Roll-Up Summary Fields and Salesforce: Relationship Considerations.

QUESTION NO: 4

What are two characteristics related to formulas?

Choose 2 answers

A. Formulas are calculated at runtime and are not stored in the database.

B. Formulas can reference themselves.

C. Formulas can reference values in related objects.

D. Fields that are used in a formula field can be deleted or edited without editing the formula.

ANSWER: A C

Explanation:

"Formulas are calculated at runtime and are not stored in the database." is correct because a Salesforce formula field derives its value dynamically whenever Salesforce needs to display, query, report on, or otherwise evaluate the record. Salesforce stores the formula definition rather than a separately persisted value for each record. Consequently, the displayed result reflects the current values of the fields referenced by the formula, without requiring a workflow, Flow, or Apex process to recalculate and save a value.

"Formulas can reference values in related objects." is also correct. Formula fields support cross-object formulas, enabling a record's formula to traverse an established relationship and use fields from a related record. For example, a field on a child record can derive a result from a field on its parent account or other related record, subject to the relationship path and field-access considerations. This lets an organization expose derived business information without duplicating source data across

objects. Salesforce documents formula fields as read-only calculated fields and describes how formulas can reference related-object fields. See Salesforce Help: Custom Field Types and Salesforce Object Reference: Field Types.

QUESTION NO: 5

Universal Containers wants Opportunities to no longer be editable when reaching the Closed/Won stage.

How should a developer accomplish this?

A. Use Flow Builder.

B. Use a validation rule.

C. Use the Process Automation Settings.

D. Mark fields as read-only on the page layout.

ANSWER: B

Explanation:

Use a validation rule. A validation rule evaluates record data whenever a user attempts to save a record and blocks the save when its formula evaluates to true. For this requirement, the rule can identify an update to an Opportunity that was already in the Closed/Won stage, then prevent the update by displaying an error message. For example, the formula can test whether the prior value of StageName was Closed Won and whether relevant values are being changed. This permits the initial transition to Closed/Won while preventing subsequent edits, which is the key distinction in the stated requirement.

The error message can be presented at the top of the Opportunity or beside a chosen field, giving users a clear explanation that closed-won records cannot be modified. Validation rules are declarative, apply consistently to edits made through the Salesforce user interface and supported record-save operations, and do not require Apex code. Salesforce documents that validation rules verify that entered data meets specified standards before a user can save a record, and formula functions such as `PRIORVALUE` provide access to a field's previous value during an update. See Salesforce validation rules documentation and PRIORVALUE function documentation.

QUESTION NO: 6

Given:

Map accountMap = new Map>ID, Account> ([SELECT Id, Name FROM Account]);

What are three valid Apex loop structures for iterating through items in the collection? (Choose three.)

A. for (ID accountID : accountMap.keySet()) {…}

B. for (Account accountRecord : accountMap.values()) {…}

C. for (Integer i = 0; i < accountMap.size(); i++) {…}

D. for (ID accountID : accountMap) {…}

E. for (Account accountRecord : accountMap.keySet()) {…}

ANSWER: A B C

Explanation:

`for (ID accountID : accountMap.keySet()) {…}` is valid because `keySet()` returns a `Set<ID>`. Apex supports enhanced `for` loops over sets, allowing each Account record's unique map key to be processed. This is the appropriate pattern when the loop logic needs an Account ID and can retrieve its related record with `accountMap.get(accountID)`.

`for (Account accountRecord : accountMap.values()) {…}` is also valid because `values()` returns a collection of the map's stored Account records. It permits direct processing of each Account value without a separate lookup.

`for (Integer i = 0; i < accountMap.size(); i++) {…}` is a syntactically valid traditional Apex `for` loop. It executes once for each map entry based on `size()`; in practical code, an index-based loop is commonly paired with an indexed collection, such as a list constructed from the map's keys or values, when indexed access is needed. Salesforce documents traditional and enhanced `for` loop syntax in the Apex Developer Guide. Map collection methods, including `keySet()` and `values()`, are described in the Map Class Reference.

QUESTION NO: 7

A developer creates a Lightning web component that imports a method within an Apex class. When a Validate button is pressed, the method runs to execute complex validations.

In this implementation scenario, which two options are.. of the Controller according to the MVC architecture?

Choose 2 answers

A. HTML file

B. Apex class

C. JavaScript file

D. XML file

ANSWER: B C

Explanation:

The **Apex class** and **JavaScript file** fulfill controller responsibilities in this Lightning web component implementation. The JavaScript file is the client-side controller layer: it handles the Validate button interaction, invokes component methods, manages UI state, and calls server-side Apex when the validation requires Salesforce data or complex business processing. In Lightning Web Components, JavaScript defines the component class and its event handlers, making it the part that coordinates user actions with application behavior.

The Apex class acts as the server-side controller for the validation operation. An Apex method exposed to an LWC with `@AuraEnabled` can be imported into the JavaScript module and invoked when the user presses Validate. Apex is appropriate for complex validation and business logic that must execute securely on the Salesforce platform, including logic that queries or updates Salesforce records. Together, the JavaScript file orchestrates the browser-side interaction and the Apex class performs the server-side operation, which aligns with the controller role of responding to input and coordinating processing. Salesforce documents Apex imports for LWCs and the required `@AuraEnabled` exposure at Import Apex Methods into JavaScript, and describes component JavaScript behavior at JavaScript in Lightning Web Components.

QUESTION NO: 8

A developer can use the debug log to see which three types of information? (Choose three.)

A. HTTP callouts to external systems

B. Database changes

C. Resource usage and limits

D. User login events

E. Actions triggered by time-based workflow

ANSWER: A B C

Explanation:

Salesforce debug logs record detailed execution information for a transaction, making them a core troubleshooting tool for Apex and declarative automation. **HTTP callouts to external systems** are represented by callout log events, which can show request and response details when the relevant log levels are enabled. **Database changes** are captured as DML-related events, allowing a developer to trace inserts, updates, deletes, and related execution activity. **Resource usage and limits** are also available in the log, including cumulative governor-limit consumption such as SOQL queries, DML statements, CPU time, heap usage, and callouts. This information helps developers identify the code path responsible for a transaction and diagnose limit-related failures before deployment or in production troubleshooting. Salesforce's debug-log reference documents the event types used to record execution activity, while the governor-limits documentation explains the per-transaction resources that developers need to monitor. See Salesforce Debug Logs and Apex Governor Limits.

QUESTION NO: 9

Universal Containers (UC) uses out-of-the-box order management, that has a Master-Detail relationship between Order and Order Line Item.

UC stores the availability date on each Order Line Item and Orders are only shipped when all of the Order Line Items are available.

Which method should be used to calculate the estimated ship date for an Order?

A. Use a LATEST formula on each of the latest availability date fields.

B. Use a CEILING formula on each of the latest availability date fields.

C. Use a DAYS formula on each of the availability date fields and a COUNT Roll-Up Summary field on the Order.

D. Use a MAX Roll-Up Summary field on the latest availability date fields.

ANSWER: D

Explanation:

Use a MAX Roll-Up Summary field on the latest availability date fields. Because an Order can ship only after every related Order Line Item is available, its estimated ship date must be the latest availability date across all of its line items. A roll-up summary field is appropriate because Order Line Item is in a master-detail relationship with Order. Salesforce lets a roll-up summary field aggregate values from related detail records and store the resulting value directly on the master record.

Configure the roll-up summary on the Order object, select Order Line Item as the summarized object, select the Availability Date field, and use the MAX summary type. For date fields, MAX returns the chronologically greatest date, which represents the final line item becoming available. The value is recalculated by Salesforce as related line items are created, updated, deleted, or undeleted, keeping the Order's estimated ship date aligned with current fulfillment availability without requiring Apex or a flow.

Salesforce documents the master-detail requirement and supported aggregate operations for roll-up summary fields in Roll-Up Summary Fields. See also Define Roll-Up Summary Fields for configuration guidance.

QUESTION NO: 10

A developer needs to include a Visualforce page in the detail section of a page layout for the Account object, but does not see the page as an available option in the Page Layout Editor.

Which attribute must the developer include in the tag to ensure the Visualforce page can be embedded in a page layout?

A. standardController= "Account"

B. extensions= "AccountController"

C. controller= "Account"

D. action= "AccountId"

ANSWER: A

Explanation:

The Visualforce page must declare `standardController="Account"`. Assigning the Account standard controller associates the page with the Account object and makes the page eligible to appear in the Visualforce Pages area of the Account Page Layout Editor. Salesforce uses this object association to determine which Visualforce pages can be embedded in an object's detail page layout. When the page is displayed from an Account record, the standard controller also supplies the record context, including the current record ID and standard Account data behavior. For example, a valid page declaration begins `<apex:page standardController="Account">`. The page can then use standard-controller capabilities and, where needed, an extension to add custom Apex logic while preserving the object association required for layout embedding. After saving the page, the developer can edit the Account page layout and select it from the available Visualforce pages for that object. Salesforce documents that Visualforce pages included in page layouts require a standard controller for the relevant object. See Visualforce Standard Controllers and Salesforce Page Layouts documentation.

QUESTION NO: 11

A developer needs to create a Visualforce page that displays Case data. The page will be used by both support reps and support managers. The Support Rep profile does not allow visibility of the Customer_Satisfaction__c field, but the Support Manager profile does.

How can the developer create the page to enforce Field Level Security and keep future maintenance to a minimum?

A. Create one Visualforce Page for use by both profiles.

B. Use a new Support Manager permission set.

C. Create a separate Visualforce Page for each profile.

D. Use a custom controller that has the with sharing keywords.

ANSWER: A

Explanation:

Create one Visualforce Page for use by both profiles. When a Visualforce page uses a standard controller, Salesforce automatically applies the current user's object permissions, field-level security, and record-sharing access. Consequently, the page can contain the Customer_Satisfaction__c field while safely presenting it only to users who have field-level read access, such as support managers. Users without visibility to that field, such as support reps, do not receive access to it through the standard-controller page. Maintaining one shared page also means that future layout or behavior changes are made once rather than duplicated across profile-specific pages. This approach relies on Salesforce's declarative security model, so administrators can update field permissions through profiles or permission sets without requiring corresponding Visualforce changes. Salesforce documents that standard controllers enforce CRUD, field-level security, and sharing rules; custom controllers require developers to explicitly enforce object and field permissions in Apex. See Salesforce Standard Controllers documentation and Apex security and field-level enforcement documentation.

QUESTION NO: 12

Which code displays the contents of a Visualforce page as a PDF?

```
A. <apex:page contentType="application/pdf">
B. <apex:page renderAs="pdf">
C. <apex:page renderAs="application/pdf">
D. <apex:page contentType="pdf">
```

A. Option A

B. Option B

C. Option C

D. Option D

ANSWER: B

Explanation:

A Visualforce page is rendered as a PDF by setting the `renderAs` attribute on the root `<apex:page>` component to `"pdf"`, for example: `<apex:page renderAs="pdf">`. This instructs the Visualforce renderer to generate PDF output when the page is requested, while allowing the page markup to define the document's content and layout. The generated document is displayed in the browser's PDF viewer or handled according to the user's browser settings. The same page can also conditionally switch between normal HTML and PDF output by binding `renderAs` to a controller expression, which is useful when users need both an interactive page and a printable version. Salesforce supports Visualforce PDF rendering for use cases such as invoices, confirmations, statements, and formatted reports. PDF rendering has specific supported HTML and CSS behavior, so page designs should follow Salesforce's Visualforce PDF rendering guidance when precise print layout is required. See the Salesforce documentation for rendering Visualforce pages as PDFs and the Visualforce PDF quick start.

QUESTION NO: 13

Which three steps allow a custom SVG to be included in a Lightning web component? (Choose three.)

A. Upload the SVG as a static resource.

B. Reference the getter in the HTML template.

C. Import the SVG as a content asset file.

D. Import the static resource and provide a getter for it in JavaScript.

E. Reference the import in the HTML template.

ANSWER: A B D

Explanation:

To use a custom SVG in a Lightning web component, first upload the SVG as a static resource. Static resources are the supported Salesforce mechanism for making custom files, including SVG images, available to component code. The component JavaScript then imports that static resource and exposes the imported URL through a getter. A getter provides a template-accessible property that can be used as the value of an HTML attribute.

Finally, reference the getter in the HTML template, typically as the `src` value of an `img` element, such as `<img src={svgUrl} />`. This follows Lightning Web Components template binding rules: expressions in markup resolve to properties or getters on the component class. Together, uploading the file, importing and exposing its URL through JavaScript, and binding the getter in the template provide the complete supported flow for displaying the SVG. Salesforce documents static-resource imports for Lightning web components and template data binding at Static Resources and Data Binding.

QUESTION NO: 14

A developer has the controller class below.

```
public with sharing class myFooController {
    public integer prop { get; private set;}
}
```

Which code block will run successfully in an execute anonymous window?

A. myFooController m = new myFooController(); System.assert(m.prop !=null);

B. myFooController m = new myFooController(); System.assert(m.prop ==0);

C. myFooController m = new myFooController(); System.assert(m.prop ==null);

D. myFooController m = new myFooController(); System.assert(m.prop ==1);

ANSWER: C

Explanation:

`myFooController m = new myFooController(); System.assert(m.prop ==null);` runs successfully because an `Integer` automatic property that has not been assigned a value has a default value of `null`. Creating an instance invokes the controller's constructor, but without an assignment to `prop`, the property remains null. The property's public getter makes its value accessible from Execute Anonymous, so reading `m.prop` is valid. The assertion therefore evaluates to true and does not throw an assertion exception.

This follows Apex's default initialization behavior: variables and properties that are not explicitly initialized receive type-appropriate default values, and object/reference types—including wrapper types such as `Integer`—default to null. Automatic properties also expose a generated accessor according to the declared access level, allowing the value to be read through normal dot notation. See Salesforce's documentation on defining Apex classes and properties and Apex variables and default values.

QUESTION NO: 15

A Developer Edition org has five existing accounts. A developer wants to add 10 more accounts for testing purposes.

The following code is executed in the Developer Console using the Execute Anonymous window:

How many total accounts will be in the org after this code Is executed?

A. 5

B. 6

C. 10

D. 15

ANSWER: B

Explanation:

`6` is correct when the executed Apex creates and inserts one Account record. The organization starts with five existing Account records, and a successful `insert` operation on one Account adds exactly one persisted record, resulting in six total Accounts. In Apex, constructing an sObject in memory does not save it to the database by itself; a DML statement such as `insert` is what persists the record. If code modifies the fields of the same Account instance repeatedly in a loop, those changes affect that one in-memory object rather than creating separate Account records. Consequently, when that instance is inserted once, only one new Account is added to the original five. Execute Anonymous runs Apex immediately in the current org context, so successful DML performed there commits in the same manner as Apex executed through other supported runtime paths. Salesforce documents that DML statements insert sObjects into the database and that an individual sObject can be passed directly to `insert`. See Salesforce Apex DML examples and Salesforce's Execute Anonymous guidance.

QUESTION NO: 16

Which scenario is valid for execution by unit tests?

A. Load data from a remote site with a callout.

B. Set the created date of a record using a system method.

C. Execute anonymous Apex as a different user.

D. Generate a Visualforce PDF with geccontentAsPDF ().

ANSWER: B

Explanation:

`Set the created date of a record using a system method.` is valid because Apex test code can use `Test.setCreatedDate(recordId, createdDatetime)` to assign a specific CreatedDate to an existing test record. This is particularly useful when the code under test contains date-based logic, such as determining whether a record is old enough for an action, calculating elapsed time, or selecting records created within a defined period. The record must first be inserted in the test context, and the method accepts that record's ID plus a `Datetime` value. Salesforce provides this capability specifically so tests can exercise behavior that depends on the otherwise system-managed CreatedDate field without requiring real time to pass. Since the operation is performed entirely within an isolated test context, it is deterministic and does not rely on external systems or organization data. This helps make unit tests repeatable and ensures the assertions reflect the intended date conditions. Salesforce documents `Test.setCreatedDate` as a test-only method for setting the CreatedDate value of a test record: Salesforce Apex Reference: Test Class Methods. General test isolation and test-data practices are also described in Salesforce Apex Testing.

QUESTION NO: 17

Assuming that name is a String obtained by a Visualforce page, which two SOQL queries performed are safe from SOQL injection? (Choose two.)

A.
```
String query = '%' + name + '%';
List results = [SELECT Id FROM Account WHERE Name LIKE :query];
```

B.
```
String query = 'SELECT Id FROM Account WHERE Name LIKE \'%' + name.noQuotes() + '%\'';
List results = Database.query(query);
```

C.
```
String query = 'SELECT Id FROM Account WHERE Name LIKE \'%' +
String.escapeSingleQuotes(name) + '%\'';
List results = Database.query(query);
```

D.
```
String query = 'SELECT Id FROM Account WHERE Name LIKE \'%' + name + '%\'';
List results = Database.query(query);
```

ANSWER: A C

Explanation:

`String query = '%' + name + '%'; List<Account> results = [SELECT Id FROM Account WHERE Name LIKE :query];` is safe because the value is supplied through a SOQL bind variable. Apex sends the complete value of `query` as data for the `LIKE` comparison, rather than interpreting content supplied in `name` as SOQL syntax. Adding wildcard characters before binding does not change that protection.

`String query = 'SELECT Id FROM Account WHERE Name LIKE \'%' + String.escapeSingleQuotes(name) + '%\''; List<Account> results = Database.query(query);` is also safe for this string-literal dynamic SOQL context. `String.escapeSingleQuotes` escapes apostrophes in the externally supplied value before it is concatenated into the quoted SOQL literal. Consequently, an apostrophe in the input remains part of the value being searched instead of terminating the literal and changing the query predicate. Salesforce recommends bind variables where they are available and identifies escaping single quotes as a mitigation when constructing dynamic SOQL strings containing untrusted string input. See Salesforce's SOQL injection prevention guidance and the String.escapeSingleQuotes reference.

QUESTION NO: 18

A developer needs to prevent the creation of Request__c records when certain conditions exist in the system. A RequestLogic class exists that checks the conditions.

What is the correct implementation?

A. apexCopyEdittrigger RequestTrigger on Request__c (before insert) {RequestLogic.validateRecords(Trigger.new);}

B. apexCopyEdittrigger RequestTrigger on Request__c (before insert) {RequestLogic.validateRecords(trigger.new);}

C. apexCopyEdittrigger RequestTrigger on Request__c (before insert) {if (RequestLogic.isValid(Request__c)) {Request.addError('Your request cannot be created at this time.');}}

D. apexCopyEdittrigger RequestTrigger on Request__c (after insert) {if (RequestLogic.isValid(Request__c)) {Request.addError('Your request cannot be created at this time.');}}

ANSWER: B

Explanation:

The implementation `trigger RequestTrigger on Request__c (before insert) { RequestLogic.validateRecords(trigger.new); }` invokes the existing validation logic in a before-insert trigger and passes the complete collection of records currently being created. A before trigger is the appropriate point to enforce a rule that can stop a save, because the records are still editable and have not yet been committed. The `validateRecords` method can evaluate each incoming `Request__c` record against the required system conditions and use `addError()` on any record that must not be saved. Calling `addError()` in this context prevents the affected record from being inserted and returns the error to the user or calling API operation. Passing `trigger.new` also supports bulk processing: the helper receives all records in the transaction, rather than assuming only one record is being created. Salesforce documents that `Trigger.new` provides the list of new records in insert and update trigger contexts, and that `addError()` prevents the associated DML operation from succeeding. See Apex Trigger Context Variables and Apex Exceptions and addError().

QUESTION NO: 19

Which code statement includes an Apex method named updateAccounts in the class AccountController for use in a Lightning web component?

A. import updateAccounts from "AccountControlles';

B. import updateAccounts from "Salesforce/apex/AccountController:';

C. import updateAccounts from "Account@ontroller.updateAccounts';

D. import updateAccounts from "@salesforce/apex/AccountController.updateAccounts";

ANSWER: D

Explanation:

The statement `import updateAccounts from "@salesforce/apex/AccountController.updateAccounts";` is the correct Lightning Web Components import syntax for an Apex method. Lightning Web Components use Salesforce-provided module imports to reference server-side Apex. The module identifier starts with `@salesforce/apex/`, followed by the Apex class name, a period, and the static Apex method name. The local identifier before `from` becomes the JavaScript function that the component can invoke imperatively, such as `updateAccounts({ accountIds: this.selectedIds })`, or pass to an appropriate LWC wire adapter when the Apex method meets wire-service requirements. For this import to work, `AccountController` must be an Apex class accessible to the component, and `updateAccounts` must be declared `static` and annotated with `@AuraEnabled`. If the method is intended for use with `@wire`, it must also be marked `cacheable=true` and must not perform data changes. Salesforce documents this exact module naming pattern for importing Apex methods into LWC JavaScript files. See Import Apex Methods into Lightning Web Components and Call Apex Methods.

QUESTION NO: 20

A developer observes that an Apex test method fails in the Sandbox. To identify the issue, the developer copies the code inside the test method and executes it via the Execute Anonymous tool in the Developer Console. The code then executes with no exceptions or errors. Why did the test method fail in the sandbox and pass in the Developer Console?

A. The test method has a syntax error in the code.

B. The test method relies on existing data in the sandbox.

C. The test method is calling an @future method.

D. The test method does not use System.runAs to execute as a specific user.

ANSWER: B

Explanation:

The test method relies on existing data in the sandbox. Apex tests are isolated from an organization's business data by default: records that happen to exist in a sandbox, such as accounts, contacts, custom-object records, or configuration records created as ordinary data, are not visible to the test unless the test explicitly creates the records it needs. Therefore, a query or later operation in the test can find no records and fail even though matching records are present in the sandbox.

Execute Anonymous is different because it runs in the current sandbox context and can access the existing organization data available to the executing user. Copying the statements from the test into Execute Anonymous can consequently succeed because the queried or referenced records exist there. A reliable unit test should create all required test data within the test setup or test method, then use that data directly. Salesforce documents that tests do not have access to pre-existing organization data by default and should create their own data; see Apex Test Data and Data Access and Apex Testing Examples.

QUESTION NO: 21

Which two conditions cause workflow rules to fire? (Choose two.)

A. An Apex Batch process that changes field values.

B. Updating records using the bulk API

C. Converting leads to person accounts

D. Changing the territory assignments of accounts and opportunities

ANSWER: A B

Explanation:

Workflow rules can be evaluated when a record is created or updated through standard Salesforce data operations, including programmatic and bulk updates. **An Apex Batch process that changes field values.** is correct because Batch Apex performs DML updates on Salesforce records. Those updates enter the normal save process, where applicable workflow rules are evaluated according to their evaluation criteria. This remains true even though Batch Apex processes records asynchronously and in grouped execution chunks.

Updating records using the bulk API is also correct. The Bulk API is designed to process large volumes of inserts, updates, upserts, and deletes, but its record updates still invoke the platform's automation behavior that applies to the relevant DML operation. Consequently, workflow rules whose object, criteria, and reevaluation settings apply to the updated records can fire during a Bulk API job. Developers should account for this behavior when designing integrations and bulk-processing logic, particularly where workflow field updates can cause additional record updates or downstream automation.

Salesforce documents that workflow rules are evaluated as part of the order of execution for record saves, including updates initiated through Apex and APIs. See Salesforce Order of Execution and Bulk API Concepts.

QUESTION NO: 22

In the following example, which sharing context will myMethod execute when it is invoked?

A. Sharing rules will be enforced by the instantiating class.

B. Sharing rules will be enforced for the running user.

C. Sharing rules will not be enforced for the running user.

D. Sharing rules will be inherited from the calling context.

ANSWER: C

Explanation:

Sharing rules will not be enforced for the running user. In the example's intended context, `myClass` has no explicit `with sharing` declaration. For this certification question, the method is therefore evaluated as executing without record-level sharing enforcement. This means that Apex code in `myMethod` can access records based on the code's system context rather than being restricted by the running user's record-sharing access. The sharing keywords control whether Apex applies Salesforce record-sharing rules, such as organization-wide defaults, role hierarchy access, manual sharing, and sharing rules, to database operations performed by the class. They do not independently enforce object-level CRUD permissions or field-level security; those controls must be handled separately when required. Declaring a class explicitly is the recommended practice because it makes the intended record-access behavior clear and avoids ambiguity as code is reused or called from different execution paths. Salesforce documents the behavior and purpose of Apex sharing declarations in its Apex Developer Guide: Using the with sharing, without sharing, and inherited sharing Keywords. For secure Apex design guidance, see Secure Server-Side Development.

QUESTION NO: 23

A company has been adding data to Salesforce and has not done a good Job of limiting the creation of duplicate Lead records. The developer is considering writing an Apex process to identify duplicates and merge the records together.

Which two statements are valid considerations when using merged?

Choose 2 answers

A. The field values on the master record are overwritten by the records being merged.

B. Merge is supported with accounts, contacts, cases, and leads.

C. External ID fields can be used with the merge method.

D. The merge method allows up to three records, including the master and two additional records with the same sObject type, to be merged into the master record.

ANSWER: B D

Explanation:

`merge` is an Apex DML operation designed to consolidate duplicate records of supported standard object types. It is supported for accounts, contacts, cases, and leads, allowing a developer to programmatically designate one record as the master and merge duplicate records into it. This is useful for a lead-cleanup process because the retained master record becomes the surviving record, while Salesforce reparents applicable related records and removes the duplicate records.

A single merge operation can include no more than three records of the same sObject type: one master record and up to two duplicate records. Therefore, an Apex process that identifies a larger duplicate set must handle it through multiple merge operations. Developers can use the `Database.merge` method when they need access to `Database.MergeResult` details, such as the IDs of merged records and errors, rather than relying exclusively on DML exception behavior. These limits and supported-object considerations should be incorporated into both the duplicate-identification logic and its bulk-processing design.

See Salesforce's Apex merge statement documentation and the Database.MergeResult reference.

QUESTION NO: 24

A developer must create an Apex REST service that receives JSON data from an external application.

Which two statements are true?

Choose 2 answers

A. The class can use the @RestResource annotation.

B. A method can use the @HttpPost annotation.

C. The class must implement HttpCalloutMock.

D. The method must be annotated with @AuraEnabled.

ANSWER: A B

Explanation:

An Apex REST class is exposed by annotating the class with `@RestResource` and defining a URL mapping. The service can use a method annotated with `@HttpPost` to handle an HTTP POST request. The method can access the request body through `RestContext.request` and deserialize JSON into Apex types.

The class does not use `@AuraEnabled`, which is intended to expose Apex methods to Lightning components. Salesforce documents Apex REST classes and HTTP method annotations in Expose Apex Classes as REST Web Services.

QUESTION NO: 25

A developer creates a Lightning web component that retrieves Account data by using the wire service.

The component must refresh the displayed data after an imperative Apex method updates an Account record.

What should the developer use to refresh the wired data?

A. refreshApex()

B. connectedCallback()

C. getRecordNotifyChange()

D. renderedCallback()

ANSWER: A

Explanation:

Use `refreshApex()` with the value provisioned by the wire service. The developer stores the wired result and passes it to `refreshApex()` after the imperative Apex update completes. This causes the wire service to retrieve current data and update the component.

Calling the Apex update method alone does not automatically refresh data returned by a wired Apex method. Salesforce documents refreshing wired Apex data in Client-Side Caching of Apex Method Results.

QUESTION NO: 26

Universal Containers wants to assess the advantages of declarative development versus programmatic customization for specific use cases in its Salesforce implementation.

What are two characteristics of declarative development over programmatic customization?

Choose 2 answers

A. Declarative development does not require Apex test classes.

B. Declarative development has higher design limits and query limits,

C. Declarative development can be done using the Setup menu.

D. Declarative code logic does not require maintenance or review.

ANSWER: A C

Explanation:

Declarative development does not require Apex test classes because declarative tools are configured through Salesforce metadata and point-and-click interfaces rather than deployed as Apex code. Apex classes and triggers require automated test coverage before they can be deployed to production, whereas standard declarative configuration such as validation rules, page layouts, and Flow configuration does not have that Apex test-class requirement. Salesforce still provides testing and debugging capabilities for declarative automation, including Flow Builder's Debug feature, so administrators and developers should validate behavior before activation and deployment.

Declarative development can be done using the Setup menu because Setup is Salesforce's central interface for configuring the platform without writing code. From Setup, teams can create and manage objects and fields, validation rules, record types, permission sets, Lightning record pages, and automation such as flows. This enables faster implementation for requirements supported by the platform's built-in capabilities and allows changes to be managed as metadata. Salesforce describes its low-code tools and the role of Flow in automating business processes in the Salesforce Flow documentation. Deployment requirements for Apex code, including test coverage, are documented in the Apex testing guide.

QUESTION NO: 27

Universal Containers (UC) is developing a process for their sales teams that requires all sales reps to go through a set of scripted steps with each new customer they create.

In the first step of collecting information, UC's ERP system must be checked via a REST endpoint to see if the customer exists. If the customer exists, the data must be presented to the sales rep in Salesforce.

Which two should a developer implement to satisfy the requirements?

Choose 2 answers

A. Flow

B. Invocable method

C. Future method

D. Trigger

ANSWER: A B

Explanation:

Flow is appropriate for implementing the guided, scripted interaction required for sales representatives. In particular, a screen flow can present each required step in sequence, collect user input, and display values returned during the process before the representative continues or creates the customer. This makes the process interactive and ensures the ERP lookup occurs as part of the user's workflow.

An **Invocable method** enables the flow to invoke custom Apex. The Apex implementation can make the synchronous HTTP callout to the ERP REST endpoint, deserialize the response, and return output values that the flow can store in variables and show on a screen. To make the callout, the ERP host should be configured through a Named Credential or the applicable endpoint authorization configuration. Together, the flow supplies the guided user experience, while the invocable Apex action supplies the REST integration and returned customer data needed immediately by the representative.

Salesforce documents how to expose Apex methods to Flow using `@InvocableMethod` and invocable variables in its InvocableMethod documentation. See also Salesforce's Apex HTTP callouts documentation for the REST callout implementation model.

QUESTION NO: 28

Universal Containers recently transitioned from Classic to Lighting Experience. One of its business processes requires certain value from the opportunity object to be sent via HTTP REST callout to its external order management system based on a user-initiated action on the opportunity page. Example values are as follow

Which two methods should the developer implement to fulfill the business requirement? (Choose 2 answers)

A. Create a Lightning component that performs the HTTP REST callout, and use a Lightning Action to expose the component on the Opportunity detail page.

B. Create a Process Builder on the Opportunity object that executes an Apex immediate action to perform the HTTP REST callout whenever the Opportunity is updated.

C. Create an after update trigger on the Opportunity object that calls a helper method using @Future(Callout=true) to perform the HTTP REST callout.

D. Create a Visualforce page that performs the HTTP REST callout, and use a Visualforce quick action to expose the component on the Opportunity detail page.

ANSWER: A D

Explanation:

Creating a Lightning component that performs the HTTP REST callout and exposing it with a Lightning Action on the Opportunity detail page provides a native Lightning Experience interaction that the user explicitly initiates. The action places the custom interface in the Opportunity record context, so it can obtain the current record ID and relevant Opportunity values. The component can invoke an Apex controller method, and that Apex method can make the REST request to the external order-management service. This design also allows the interface to provide immediate status or error feedback after the user selects the action.

Creating a Visualforce page that performs the HTTP REST callout and exposing it through a Visualforce quick action also meets the requirement. Visualforce quick actions are supported in Lightning Experience and provide a user-launched record-page action. A Visualforce controller or controller extension can use the current Opportunity context, perform the callout through Apex, and present the result to the user. This is especially appropriate when an organization has existing Visualforce or Apex controller functionality it wants to retain while operating in Lightning Experience. Salesforce documents record quick actions for user-initiated record operations and the use of Apex for HTTP callouts: Salesforce Developer Guide: Quick Actions and Salesforce Developer Guide: Call Apex Methods.

QUESTION NO: 29

A company's engineering department is conducting a month-long test on the scalability of an in-house-developed software that requires a cluster of 100 or more servers. Which of the following models is the best to use?

A. PaaS

B. SaaS

C. BaaS

D. IaaS

ANSWER: D

Explanation:

IaaS is the appropriate model because the engineering team needs a large, temporary pool of configurable server resources to run a month-long scalability test for software it developed itself. Infrastructure as a Service provides on-demand compute capacity, networking, and storage while allowing the customer to provision and manage the operating systems, runtime environment, and application deployment. This model is well suited to creating a cluster of one hundred or more servers without purchasing, installing, and later retiring equivalent physical hardware. The team can scale the cluster up for peak test loads and release the capacity when the test concludes, aligning cost with the short-lived workload. The U.S. National Institute of Standards and Technology defines IaaS as the capability to provision processing, storage, networks, and other fundamental computing resources on which consumers can deploy and run arbitrary software, including operating systems and applications. That level of infrastructure control is important when testing an in-house application under specific server, network, and load conditions. See the NIST Definition of Cloud Computing and Salesforce Trailhead's Cloud Computing Basics for cloud-service-model context.

QUESTION NO: 30

A development team wants to use a deployment script lo automatically deploy lo a sandbox during their development cycles.

Which two tools can they use to run a script that deploys to a sandbox?

Choose 2 answers

A. Ant Migration Tool

B. SFDX CLI

C. Change Sets

D. Developer Console

ANSWER: A B

Explanation:

Ant Migration Tool and **SFDX CLI** can both be used in scripted, repeatable metadata deployments to a Salesforce sandbox. The Ant Migration Tool uses the Metadata API and an `build.xml` file to define deployment tasks. This makes it suitable for automated development processes and continuous-integration environments, where a build server or local automation script can authenticate to a target sandbox and execute deployments without requiring interactive use of the Setup UI.

The SFDX CLI (now distributed as the Salesforce CLI) also supports command-line metadata deployment workflows. A script can authenticate to a sandbox org and invoke deployment commands for source or metadata, allowing teams to incorporate validation, test execution, and deployment steps into CI/CD pipelines. Command-line automation makes deployments consistent across development cycles and enables the same process to be run by developers or automated build agents. Salesforce documents the Ant-based Metadata API migration process at Ant Migration Tool Guide and provides current CLI deployment guidance at Salesforce CLI Project Commands Reference.

QUESTION NO: 31

Which two statements are true about Apex code executed in Anonymous Blocks? (Choose two.)

A. The code runs with the permissions of the user specified in the runAs() statement.

B. The code runs with the permissions of the logged in user.

C. The code runs in system mode having access to all objects and fields.

D. All DML operations are automatically rolled back.

E. Successful DML operations are automatically committed.

ANSWER: C E

Explanation:

Apex executed through an Anonymous Block runs in system mode. This elevated execution context means that Apex isn't constrained by the executing user's object-level and field-level permissions in the way standard user-interface actions are. Consequently, the code can access objects and fields that the user might not ordinarily be able to access through their profile or permission sets. Developers should therefore use care when running anonymous code, particularly when querying, changing, or deleting production data.

Anonymous Apex is also executed as a normal transaction. When a DML statement succeeds and the transaction completes without an unhandled exception or another condition that causes rollback, Salesforce commits the changes automatically. This makes Execute Anonymous useful for administrative scripts, data corrections, and one-time maintenance tasks, but it also means that data changes are persistent unless the transaction is rolled back. Salesforce's Apex documentation describes the default system-mode behavior and the transactional behavior of Apex DML operations. See Salesforce Apex Security and Salesforce Apex DML Statements.

QUESTION NO: 32

The sales management team at Universal Containers requires that the Lead Source field of the Lead record be populated when a Lead is converted What should be done to ensure that a user populates the Lead Source field prior to converting a Lead?

A. Create an after trigger on Lead.

B. Use Lead Conversion field mapping.

C. Use a formula field.

D. Use a validation rule.

ANSWER: D

Explanation:

Use a validation rule. Lead conversion changes the Lead record's converted status, and validation rules evaluate the record before Salesforce saves that change. A rule can therefore prevent conversion when Lead Source has not been supplied, while allowing users to save unconverted leads that do not yet have a value. For example, the rule can test whether the record is being converted and whether `LeadSource` is blank, such as `AND(ISCHANGED(IsConverted), IsConverted, ISBLANK(TEXT(LeadSource)))`. When the formula evaluates to true, Salesforce displays the configured error message and does not complete the conversion. This directly enforces the business requirement at the point where it matters and works regardless of whether the user converts the lead through the standard user interface, API, or other supported process that performs the record update. The error message should clearly instruct the user to select or enter a Lead Source before converting. Salesforce describes validation rules as formulas that check entered data and display an error when the data does not meet defined criteria. See Salesforce Validation Rules documentation and Salesforce Lead object reference.

QUESTION NO: 33

Assuming that name is a String obtained by a Visualforce page, which two SOQL queries performed are safe from SOQL injection? (Choose two.)

A. String query = '%' + name + '%'; List results = [SELECT Id FROM Account WHERE Name LIKE :query];

B. String query = 'SELECT Id FROM Account WHERE Name LIKE \'%' + name.noQuotes() + '%\''; List results = Database.query(query);

C. String query = 'SELECT Id FROM Account WHERE Name LIKE \'%' + String.escapeSingleQuotes(name) + '%\''; List results = Database.query(query);

D. String query = 'SELECT Id FROM Account WHERE Name LIKE \'%' + name + '%\''; List results = Database.query(query);

ANSWER: A C

Explanation:

`String query = '%' + name + '%'; List<Account> results = [SELECT Id FROM Account WHERE Name LIKE :query];` is safe because it uses a static SOQL bind variable. The value of `query`, including any quotes or other characters supplied through the Visualforce page, is passed as data for the `LIKE` comparison rather than interpreted as SOQL syntax. Salesforce recommends bind variables as the preferred protection against SOQL injection.

`String query = 'SELECT Id FROM Account WHERE Name LIKE \'%' + String.escapeSingleQuotes(name) + '%\''; List<Account> results = Database.query(query);` is also safe for this string-literal dynamic-SOQL use case. Dynamic SOQL is needed when query text must be assembled at runtime, and `String.escapeSingleQuotes` escapes quote characters in the untrusted input before that value is concatenated into the query. Consequently, user-provided quotes remain part of the searched text and cannot terminate the intended string literal to introduce additional query logic.

In production code, prefer the static query with a bind whenever its query structure is known at compile time. When dynamic SOQL is necessary, apply the appropriate controls to every dynamic portion of the statement; escaping is specifically relevant here because the external value is inserted inside a quoted string. See Salesforce's guidance on dynamic SOQL and String.escapeSingleQuotes.

QUESTION NO: 34

A developer needs to have records with specific field values in order to test a new Apex class.

What should the developer do to ensure the data is available to the test?

A. Use SOQL to query the org for the required data.

B. Use Anonymous Apex to create the required data.

C. Use Test.loadData() and reference a CSV file.

D. Use Test.loadData() and reference a static resource.

ANSWER: D

Explanation:

`Test.loadData()` and a static resource is the appropriate approach when a test needs a known set of records with precise field values. The developer first creates a CSV file whose column headers correspond to fields on the target sObject and whose rows define the required test records. That CSV is uploaded as a Salesforce static resource. In the test method, `Test.loadData(Account.sObjectType, 'MyTestAccounts')`, for example, loads the CSV data and inserts the resulting records into the test's isolated data context. The method also returns the inserted records, allowing the test to access their IDs or validate setup details when needed.

This provides repeatable, source-controlled fixture data without relying on records that happen to exist in an org. It is particularly useful for larger or more structured datasets, including data with relationship references supported by the CSV format. Test-created data is available for the test execution and is rolled back afterward, so the test remains self-contained and does not alter persistent organization data. Salesforce documents `Test.loadData` as the mechanism for loading test records from a CSV static resource; see the Apex Developer Guide: Loading Test Data and the System.Test methods reference.

QUESTION NO: 35

A developer is asked to create a custom Visualforce page that will be used as a dashboard component. Which three are valid controller options for this page? (Choose three.)

A. Use a standard controller.

B. Use a standard controller with extensions.

C. Use a custom controller with extensions.

D. Do not specify a controller.

E. Use a custom controller.

ANSWER: A B E

Explanation:

For a Visualforce page embedded as a dashboard component, the valid supported controller models are a standard controller, a standard controller with extensions, and a custom controller. A standard controller is appropriate when the component is centered on the behavior and data context of a Salesforce standard or custom object. Adding a controller extension allows the developer to supplement that standard controller with Apex logic, such as additional queries, calculations, or actions, while retaining the standard controller's built-in behavior. A custom controller is appropriate when the dashboard component requires fully tailored logic and does not need the conventions supplied by a standard object controller.

These supported models let a dashboard component obtain and prepare the data it displays while following the Visualforce dashboard integration rules. The controller design should also account for the fact that dashboard viewers can differ in access to records and fields, so Apex should respect sharing and enforce appropriate security for data returned to the page. Salesforce documents the requirements and considerations for adding Visualforce pages to dashboards in Visualforce Pages in Dashboards. The available Visualforce controller patterns are described in the Visualforce Controllers documentation.

QUESTION NO: 36

In which three areas can a Lightning component be used in the Lightning Experience? (Choose three.)

A. Lightning Report page

B. Lightning Connect page

C. Lightning Record Page

D. Lightning Community Page

E. Lightning Home page

ANSWER: C D E

Explanation:

Lightning components can be exposed for placement in the standard Lightning Experience page types supported by Lightning App Builder. A component intended for a record page uses the `flexipage:availableForRecordHome` interface, which makes it available for record-oriented Lightning pages. A component intended for the Lightning Home page uses `flexipage:availableForAllPageTypes` or the applicable Home-page interface, allowing an administrator to add it through Lightning App Builder. For an Experience Cloud site, formerly called a Community, an Aura component can implement `forceCommunity:availableForAllPageTypes` so that it is available in Experience Builder and can be placed on a community page. These interfaces define where a component appears as an available custom component; the component must also be configured with `access="global"` where required for use outside its namespace. Thus, Lightning Record Page, Lightning Community Page, and Lightning Home page are the supported areas identified here. Salesforce documents the Lightning page interfaces for Aura components in its Lightning Component Reference and documents Experience Builder component availability through the forceCommunity:availableForAllPageTypes interface.

QUESTION NO: 37

Which two statements are true about Getter and Setter methods? (Choose two.)

A. Setter methods always have to be declared global.

B. Setter methods are required to pass a value from a page to a controller.

C. There is no guarantee for the order in which Getter or Setter methods are executed.

D. Getter methods can pass a value from a controller to a page.

ANSWER: C D

Explanation:

There is no guarantee for the order in which Getter or Setter methods are executed. Visualforce evaluates expressions while rendering the page and while processing requests, so a getter can be invoked multiple times and in an order that the developer must not rely on. Consequently, getters should avoid side effects such as changing data, performing DML, or depending on a particular sequence of method calls. This is an important design consideration for controller properties used by Visualforce markup.

Getter methods can pass a value from a controller to a page. A Visualforce expression such as `{!accountName}` reads the corresponding controller property; Apex supplies that property's value through its getter method, whether it is explicitly implemented or generated through an Apex property definition. This is the standard controller-to-page data flow used to display field values, calculated values, and other controller state in Visualforce. Salesforce's Visualforce controller guidance describes getters as methods that return values to Visualforce pages and cautions that they can be called repeatedly during page processing. See Visualforce Getter Methods and Apex Properties.

QUESTION NO: 38

Which Apex class contains methods to return the amount of resources that have been used for a particular governor, such as the number of DML statements?

A. Exception

B. Messaging

C. OrgLimits

D. Limits

ANSWER: D

Explanation:

`Limits` is the Apex system class used to inspect governor-limit consumption in the current transaction. It provides methods that report both the amount already consumed and the maximum permitted for a resource. For example, `Limits.getDmlStatements()` returns the number of DML statements that have been issued so far, and `Limits.getLimitDmlStatements()` returns the transaction's DML-statement limit. This lets Apex code monitor resource use while it executes, which is useful when processing records in bulk or deciding whether additional work can safely occur within the same transaction. The class also exposes corresponding methods for resources such as SOQL queries, query rows, DML rows, CPU time, callouts, heap size, and email invocations. Governor limits apply at the transaction level, so the values returned by these methods represent usage accumulated during the active Apex transaction. Salesforce documents the available methods in the Apex Reference Guide's `Limits` class reference, and its governor-limit documentation explains the transaction-scoped resource model. See Salesforce Apex Reference: Limits Class and Salesforce Developer Guide: Apex Governor Limits.

QUESTION NO: 39

Universal Containers has a support process that allows users to request support from its engineering team using a custom object,

Engineering Support c.

Users should be able to associate multiple Engineering Support __c records to a single Opportunity record. Additionally, aggregate information

about the Engineering Support _c records should be shown on the Opportunity record.

Which relationship field should be implemented to support these requirements?

A. Lookup field from Opportunity to Engineering _support__c

B. Master-detail field from Engineering Support__c to Opportunity

C. Master-detail field from Opportunity to Engineering Support__c

D. Lookup field from Engineering Support __c to Opportunity

ANSWER: B

Explanation:

Master-detail field from Engineering Support__c to Opportunity is correct because Engineering Support__c is the child object and Opportunity is the parent object in the required one-to-many relationship. A master-detail relationship permits many Engineering Support__c records to be related to one Opportunity, which directly meets the association requirement. Crucially, it also enables Salesforce roll-up summary fields on the Opportunity record. Roll-up summaries can calculate aggregate values from related Engineering Support__c records, such as the number of support requests, a sum of estimated engineering hours, or the earliest or latest support-request date. These summary values are stored and displayed on the parent Opportunity, satisfying the requirement to show aggregate support information there. Salesforce's relationship model requires the master-detail field to be created on the detail, or child, object and point to its master parent. After that relationship exists, administrators can create roll-up summary fields on Opportunity to summarize its related Engineering Support__c records. See Salesforce's Roll-Up Summary Fields documentation and Relationship Fields reference for the relationship and aggregation behavior.

QUESTION NO: 40

A developer is creating a page that allows users to create multiple Opportunities. The developer is asked to verify the current user's default Opportunity record type, and set certain default values based on the record type before inserting the record.

How can the developer find the current user's default record type?

A. Use the Schema.userInfo.Opportunity.getDefaultRecordType() method.

B. Query the Profile where the ID equals userInfo.getProfileID() and then use the profile.Opportunity.getDefaultRecordType() method.

C. Create the opportunity and check the opportunity.recordType, which will have the record ID of the current user's default record type, before inserting.

D. Use Opportunity.SObjectType.getDescribe().getRecordTypeInfos() to get a list of record types, and iterate through them until isDefaultRecordTypeMapping() is true.

ANSWER: D

Explanation:

Use Opportunity.SObjectType.getDescribe().getRecordTypeInfos() to get a list of record types, and iterate through them until isDefaultRecordTypeMapping() is true. The Describe API exposes record-type metadata for an sObject through `Schema.DescribeSObjectResult`, and `getRecordTypeInfos()` returns the available `Schema.RecordTypeInfo` entries. For each entry, `isDefaultRecordTypeMapping()` identifies whether that record type is the default mapping for the running user. Once the matching `RecordTypeInfo` is found, the developer can obtain its record type Id with

`getRecordTypeId()` and assign it to the new Opportunity's `RecordTypeId`. The page logic can then apply the appropriate field defaults before performing DML. This approach is user-context aware: it uses the record type mappings available to the current user rather than assuming a fixed organization-wide record type. It also avoids a query and relies on Salesforce's describe metadata, which is appropriate when code needs to inspect record type availability and default mappings dynamically. Salesforce documents `RecordTypeInfo.isDefaultRecordTypeMapping()` and the related describe methods in the Schema.RecordTypeInfo Apex reference and describes record type metadata access in the DescribeSObjectResult Apex reference.

QUESTION NO: 41

A developer is creating a Lightning web component to show a list of sales records.

The Sales Representative user should be able to see the commission field on each record. The Sales Assistant user should be able to see all fields on the record except the commission field.

How should this be enforced so that the component works for both users without showing any errors?

A. Use WITH SECURITY_ENFORCED In the SOQL that fetches the data for the component,

B. Use Security.stripInaccessible Le to remove fields inaccessible to the current user.

C. Use Lightning Locker Service to enforce sharing rules and field-level security.

D. Use Lightning Data Service to get the collection of sales records.

ANSWER: B

Explanation:

Use Security.stripInaccessible Le to remove fields inaccessible to the current user. When an Apex controller queries sales records for a Lightning web component, field-level security is not applied automatically merely because the data is returned to the component. Calling `Security.stripInaccessible` with read access checks evaluates the current user's field permissions and sanitizes the queried records before they are serialized and sent to the client. For a Sales Representative who has read access to the commission field, that field remains available in the returned records. For a Sales Assistant without that access, the commission value is removed, while the fields that user can read remain available. This lets the same Apex-backed component serve both user types safely, provided the component renders fields that are present rather than assuming every field is returned. The method also supplies information about removed fields through the returned `SObjectAccessDecision`, which can be useful for diagnostics or conditional UI behavior. Salesforce documents `stripInaccessible` as the Apex mechanism for enforcing object- and field-level security while gracefully sanitizing records: Security.stripInaccessible documentation. See also Salesforce's Secure Apex Classes guidance.

QUESTION NO: 42

A developer wrote Apex code that calls out to an external system. How should a developer write the test to provide test coverage?

A. Write a class that extends HttpCalloutMock.

B. Write a class that implements the HttpCalloutMock interface.

C. Write a class that implements the WebserviceMock interface.

D. Write a class that extends WebserviceMock

ANSWER: B

Explanation:

Write a class that implements the HttpCalloutMock interface. Apex tests cannot make real HTTP callouts, so the test must register a mock response that Salesforce returns when the production code invokes `Http.send()`. A mock class

implementing `HttpCalloutMock` defines the `respond` method, which receives the request and returns an `HTTPResponse` configured with the desired status code, headers, and response body. The test then uses `Test.setMock(HttpCalloutMock.class, new MyMock())` before executing the code that performs the callout. This lets the test exercise the callout-handling logic deterministically, without depending on the availability, credentials, data, or behavior of the external system. It also enables test cases for successful responses, error status codes, and malformed or empty response payloads. Salesforce documents `HttpCalloutMock` as the mechanism for testing HTTP callouts in Apex; see Testing HTTP Callouts by Using Mock Callouts and the HttpCalloutMock interface reference.

QUESTION NO: 43

Which two statements are true about using the @testSetup annotation in an Apex test class? (Choose two.)

A. The @testSetup annotation cannot be used when the @isTest(SeeAllData=True) annotation is used.

B. Test data is inserted once for all test methods in a class.

C. Records created in the @testSetup method cannot be updates in individual test methods.

D. The @testSetup method is automatically executed before each test method in the test class is executed.

ANSWER: A B

Explanation:

The @testSetup annotation is intended for creating common test data shared by the test methods in one Apex test class. Salesforce creates the setup data once, which avoids repeating the same insert operations in every test method and helps keep tests faster, more maintainable, and consistent. Each test method can query and use the records created by the setup method; changes made during a test do not persist beyond that test's transaction.

The @testSetup annotation is supported only when the test class uses its default isolated data access, which is `SeeAllData=false`. Consequently, it cannot be combined with a test class annotated as `@isTest(SeeAllData=true)`. This preserves test isolation and ensures that setup data is managed by the test framework rather than relying on, or mixing with, organization data. Salesforce's Apex Developer Guide describes test setup methods as a way to create common records for all tests in a class and documents the restriction involving `SeeAllData=true`. See Using Test Setup Methods and Test Data Access.

QUESTION NO: 44

A Lightning component has a wired property, searchResults, that stores a list of Opportunities. Which definition of the Apex method, to which the searchResults property is wired, should be used?

A. @AuraEnabled(cacheable=true)
public static List search(String term) { /* implementation*/ }

B. @AuraEnabled(cacheable=true) public List search(String term) { /*implementation*/ }

C. @AuraEnabled(cacheable=false) public static List search(String term) { /*implementation*/ }

D. @AuraEnabled(cacheable=false) public List search(String term) { /*implementation*/ }

ANSWER: A

Explanation:

`@AuraEnabled(cacheable=true)`
`public static List<Opportunity> search(String term) { /* implementation*/ }` is the appropriate method definition for supplying Opportunity data through a wired property. When Lightning components invoke Apex through the wire service, the Apex method must be exposed with `@AuraEnabled`, declared `static`, and marked `cacheable=true`. The cacheable designation tells the platform that the method is used to retrieve data without performing

data mutations, enabling client-side caching and allowing the wire service to manage invocation and refresh behavior efficiently.

The method's return type, `List<Opportunity>`, directly represents the collection that the `searchResults` wired property is intended to hold. The `term` parameter can be supplied reactively from a component property, so a change to that property can cause the wire service to retrieve an updated result set. Salesforce documents that Apex methods used with `@wire` must be static and cacheable, and that cacheable Apex methods are designed for read operations. See Wire Apex Methods to Lightning Web Components and Client-Side Caching of Apex Method Results.

QUESTION NO: 45

As part of new feature development, a developer is asked to build a responsive application capable of responding to touch events, that will be executed on stateful clients.

Which two technologies are built on a framework that fully supports the business requirement? Choose 2 answers

A. Lightning Web Components

B. Visualforce Components

C. Visualforce Pages

D. Aura Components

ANSWER: A D

Explanation:

Lightning Web Components and Aura Components are appropriate because both support building rich, responsive client-side user interfaces on the Salesforce Lightning platform. Their component frameworks run substantial application behavior in the browser, allowing client-side state to be maintained and updated as users interact with the application rather than requiring every interaction to be handled as a full server-rendered page request.

Both technologies use web-oriented event models that can support touch-driven interactions on mobile and touch-enabled devices. Components can respond to browser events, update displayed data dynamically, and use responsive styling through Salesforce Lightning Design System patterns and CSS. This makes them suitable for experiences that must work across desktop and mobile form factors while retaining an interactive, stateful client experience.

Lightning Web Components is Salesforce's modern standards-based component model and is designed for performant UI development. Aura Components remains a supported component framework for Lightning applications and provides the same broad client-side, event-driven application architecture needed by the requirement. Salesforce documents the Lightning component framework as the basis for building dynamic web applications, and its LWC documentation describes the standards-based programming model used for Lightning UI development. See Lightning Web Components introduction and Aura Components overview.

QUESTION NO: 46

The Review__c object has a lookup relationship up to the Job_Application__c object. The Job_Application__c object has a master-detail relationship up to the Position__c object. The relationship field names are based on the auto-populated defaults.

What is the recommended way to display field data from the related Position__c record on a Visualforce page for a single Review__c record?

A. Use the Standard Controller for Review__c and cross-object Formula Fields on the Position__c object to display Position__c data.

B. Use the Standard Controller for Job_Application__c and a Controller Extension to query for Position__c data.

C. Use the Standard Controller for Job_Application__c and cross-object Formula Fields on the Review__c object to display Position__c data.

D. Use the Standard Controller for Review__c and expression syntax in the Page to display related Position__c data through the Job_Application__c object.

ANSWER: D

Explanation:

Use the Standard Controller for Review__c and expression syntax in the Page to display related Position__c data through the Job_Application__c object. A standard controller for `Review__c` is appropriate because the page is rendering one Review record, so it supplies that record as the page's standard object and makes its fields available through Visualforce expressions. Relationship expressions can then traverse from the Review lookup relationship to `Job_Application__r`, and from that record's master-detail relationship to `Position__r`, before accessing the desired Position field. For example, a page can render a Position name with an expression such as `{!Review__c.Job_Application__r.Position__r.Name}`, assuming the standard controller's object is exposed under its normal name. This approach directly uses the relationship data model and avoids creating redundant formula fields or writing a controller extension merely to retrieve fields that are reachable through the record relationships. Salesforce documents that Visualforce expressions can access related-object fields using relationship notation, and standard controllers provide access to the current record in a page request. See Visualforce Expression Language and Standard Controllers.

QUESTION NO: 47

A developer executes the following query in Apex to retrieve a list of contacts for each account:

List accounts = [Select ID, Name, (Select ID, Name from Contacts) from Account] ; Which two exceptions may occur when it executes? (Choose two.)

A. CPU limit exception due to the complexity of the query.

B. SOQL query row limit exception due to the number of contacts.

C. SOQL query limit exception due to the number of contacts.

D. SOQL query row limit exception due to the number of accounts.

ANSWER: B D

Explanation:

The correct answers are *SOQL query row limit exception due to the number of contacts.* and *SOQL query row limit exception due to the number of accounts.* Apex enforces a governor limit on the total number of records returned by SOQL queries in a transaction. A parent-to-child relationship query returns both the Account records from the outer query and the Contact records returned by the nested Contacts subquery. Consequently, either a sufficiently large number of Account rows or a sufficiently large combined result set caused by Contact rows can make the transaction exceed the synchronous SOQL row limit of 50,000 records and raise a limit exception.

The nested Contacts query is not an additional independently executed Apex SOQL statement for each Account. It is part of one relationship query, so the number of contacts returned does not cause the transaction to exceed the standard limit on the number of SOQL queries merely because many Contact records exist. Salesforce documents both the 50,000-record SOQL retrieval limit and the way parent-to-child relationship queries retrieve child records. See Apex Governor Limits and SOQL Relationship Queries.

QUESTION NO: 48

The sales team at Universal Containers would like to see a visual indicator appear on both Account and Opportunity page layouts to alert salespeople when an Account is late making payments or has entered the collections process.

What can a developer implement to achieve this requirement without having to write custom code?

A. Formula Field

B. Workflow Rule

C. Quick Action

D. Roll-up Summary Field

ANSWER: A

Explanation:

Formula Field is the appropriate declarative solution because formula fields can calculate and display a value dynamically on a page layout without Apex or other custom code. On the Account, a formula can evaluate fields that represent payment status or whether the account is in collections. It can then return a visual indicator, such as an image via the `IMAGE()` formula function, or a text-based warning. Formula fields are evaluated whenever the record is accessed, so the indicator reflects the current underlying Account data without needing a scheduled update or user action.

A corresponding formula field can be created on Opportunity and can reference fields on its parent Account through the standard Account relationship. This lets the Opportunity page layout show the same warning based on the related Account's payment or collections status. Each formula field can then be added to its respective Account or Opportunity page layout, making the alert visible to salespeople in both record contexts. Salesforce documents formula fields as read-only fields that derive their values from expressions, including values from related records, and documents `IMAGE()` for displaying an image from a formula. See Salesforce custom field types and Salesforce IMAGE() function reference.

QUESTION NO: 49

A developer wants to get access to the standard price book in the org while writing a test class that covers an OpportunityLineItem trigger.

Which method allows access to the price book?

A. Use Test.getStandardPricebookId() to get the standard price book ID.

B. Use @isTest(SeeAllData=true) and delete the existing standard price book.

C. Use @TestVisible to allow the test method to see the standard price book.

D. Use Test.loadData() and a static resource to load a standard price book.

ANSWER: A

Explanation:

Use `Test.getStandardPricebookId()` to obtain the ID of the org's standard price book from Apex test code. Salesforce makes this method available specifically so tests can create the pricing data required for products and opportunity line items while remaining isolated from ordinary organization data. The returned ID can be assigned to a `PricebookEntry` record's `Pricebook2Id`, after which the test can insert an active price book entry for a test product and use that entry when constructing an `OpportunityLineItem`. This supports deterministic, self-contained tests and avoids coupling test outcomes to data that happens to exist in a particular organization. The standard price book is a special record: although Apex tests normally cannot access pre-existing business data, Salesforce provides this dedicated test utility for the standard price book use case. Salesforce's Apex reference documents `getStandardPricebookId()` as returning the standard price book ID for use in test methods. See the System.Test methods reference and Salesforce's Apex testing documentation.

QUESTION NO: 50

Which feature allows a developer to create test records for use in test classes?

A. Documents

B. WebServiceTests

C. HttpCalloutMocks

D. Static Resources

E. @testSetup

ANSWER: E

Explanation:

The `@testSetup` annotation is the Salesforce feature designed to create common test data for a test class. A method marked `@testSetup` runs once before that class's test methods execute, and it can insert the records required by multiple tests. This makes test code cleaner because the class establishes shared baseline data in one location rather than duplicating record creation in every test method.

Although the setup method's records are available to each test method, Salesforce provides test isolation: changes made to those records by one test method don't persist for another test method. Test methods can query the setup data and use it as the starting point for their own scenarios. The annotated method must be static, take no arguments, and return no value. This pattern supports reliable, maintainable Apex tests while avoiding dependence on existing organization data. Salesforce documents `@testSetup` as the mechanism for creating test records once and making them available throughout the test class. See Using @testSetup Methods and Apex Testing.

QUESTION NO: 51

What is a benefit of using a trigger framework?

A. Reduces trigger execution time

B. Allows functional code to be tested by a test class

C. Increases trigger governor limits

D. Simplifies addition of context-specific logic

ANSWER: D

Explanation:

Simplifies addition of context-specific logic is correct because a trigger framework separates the minimal trigger entry point from handler logic that is organized around the trigger execution context. Rather than placing all behavior in one trigger body, the framework can route work to methods intended for before-insert, before-update, after-delete, and other relevant contexts. This gives developers a predictable location for each kind of behavior and makes it easier to extend the automation later without turning the trigger into a long chain of context checks.

Salesforce recommends keeping trigger logic in handler classes and using a single trigger per object, an approach that helps maintain clear ordering and organization as automation grows. A framework formalizes this pattern by providing a consistent dispatcher and lifecycle structure. For example, when new update-only validation or post-insert processing is required, developers can add the behavior to the appropriate context handler while preserving the trigger's simple entry point. This supports maintainability and makes the intended execution context explicit. See Salesforce's Apex trigger best practices and the Apex Triggers Trailhead module.

QUESTION NO: 52

A developer is asked to create a PDF quote document formatted using the company's branding guidelines, and automatically save it to the Opportunity record.

Which two ways should a developer create this functionality? (Choose two.)

A. Install an application from the AppExchange to generate documents.

B. Create a Visualforce page with custom styling.

C. Create an email template and use it in Process Builder.

D. Create a visual flow that implements the company's formatting.

ANSWER: A B

Explanation:

Install an application from the AppExchange to generate documents. is appropriate because AppExchange document-generation products can produce branded quote documents from Salesforce record data. These managed solutions commonly provide merge templates, PDF output, and automation capabilities that store the generated document as a Salesforce File or attach it to the relevant Opportunity. This is a practical implementation path when the organization can use a packaged solution rather than building the entire document-generation service.

Create a Visualforce page with custom styling. is also appropriate because Visualforce supports server-side PDF rendering through the `renderAs="pdf"` attribute. A developer can use Visualforce markup and CSS to precisely implement branding elements such as logos, fonts, layout, tables, and quote line-item formatting. Apex can invoke the page as a PDF, create a file from the rendered content, and relate that file to the Opportunity, including when initiated by automation. Salesforce documents PDF rendering for Visualforce pages at Visualforce PDF Rendering. Packaged document-generation solutions can be evaluated through the Salesforce AppExchange.

QUESTION NO: 53

An org tracks customer orders on an Order object and the line items of an Order on the Line Item object. The Line Item object has a Master/Detail relationship to the Order object. A developer has a requirement to calculate the order amount on an Order and the line amount on each Line Item based on quantity and price.

What is the correct implementation?

A. Write a single before trigger on the Line Item that calculates the item amount and updates the order amount on the Order.

B. Write a process on the Line Item that calculates the item amount and order amount and updates the fields on the Line Item and the Order.

C. Implement the line amount as a numeric formula field and the order amount as a roll-up summary field.

D. Implement the line amount as a currency field and the order amount as a SUM formula field.

ANSWER: C

Explanation:

Implement the line amount as a numeric formula field and the order amount as a roll-up summary field. A formula field on Line Item can calculate each line's amount directly from its Quantity and Price fields, such as `Quantity__c * Price__c`. Because formulas are calculated dynamically, the displayed line amount remains current whenever either input value changes and does not require Apex, Flow, or stored field updates.

The master-detail relationship enables a roll-up summary field on Order. The roll-up can use the SUM operation over the Line Item line-amount formula field, producing the aggregate amount for all related line items. Salesforce maintains this summary automatically as line items are created, edited, deleted, or reparented where applicable. This declarative design uses the platform relationship model as intended, keeps the calculation logic transparent, and avoids unnecessary automation and DML operations. Salesforce documents that roll-up summary fields are available on the master side of a master-detail relationship and can calculate aggregate values from related detail records. See Salesforce Help: Roll-Up Summary Fields and Salesforce Help: Formula Fields.

QUESTION NO: 54

Which three code lines are required to create a Lightning component on a Visualforce page? (Choose three.)

A. $Lightning.useComponent
B. <apex:slds/>
C. $Lightning.use
D. <apex:includeLightning/>
E. $Lightning.createComponent

A. Option A

B. Option B

C. Option C

D. Option D

E. Option E

ANSWER: C D E

Explanation:

Embedding an Aura Lightning component in a Visualforce page uses Lightning Out. The Visualforce page must include the Lightning Out JavaScript support with `<apex:includeLightning />`. It must then initialize a Lightning Out application by calling `$Lightning.use()`; the application provides the dependency context for the component being loaded. Finally, the page creates and renders the Aura component through `$Lightning.createComponent()`, providing the component descriptor, any attribute values, and the DOM element ID where the component is inserted. Together, these statements load the Lightning framework, establish the app context, and instantiate the component in the Visualforce document. The Lightning Out application itself must be defined as an Aura application that extends `ltng:outApp`, with the required component dependency declared. Salesforce documents this Visualforce integration pattern, including the use of `apex:includeLightning`, `$Lightning.use()`, and `$Lightning.createComponent()`, in the Lightning Components in Visualforce Pages guide and its Lightning Out documentation.

QUESTION NO: 55

Managers at Universal Containers want to ensure that only decommissioned containers are able to be deleted in the system. To meet the business requirement a Salesforce developer adds "Decommissioned" as a picklist value for the Status _c custom field within the Container _c object.

Which two approaches could a developer use to enforce only Container records with a status of "Decommissioned" can be deleted?

Choose 2 answers

A. Before record-triggered flow

B. Apex trigger

C. After record-triggered flow

D. Validation rule

ANSWER: A B

Explanation:

Before record-triggered flow and **Apex trigger** can enforce this requirement because each can execute before Salesforce completes a deletion and can add an error that blocks the operation. A before-delete record-triggered flow can inspect the Container record's Status__c value and use an Add Error element when its value is not Decommissioned. The error is returned to the user or calling process, and the deletion is not committed. This is a declarative solution that is appropriate when the rule is straightforward and does not require complex code.

An Apex trigger can implement the same safeguard in a `before delete` trigger. The trigger iterates through the records in `Trigger.old`, checks each Status__c value, and calls `addError()` on any record that is not decommissioned. Because the error is added during the before-delete context, Salesforce cancels deletion of the invalid record. Apex is especially useful where the check must support bulk deletes, involve more complex logic, or be reused with programmatic deletion operations. Salesforce documents the use of `addError()` to prevent DML and the supported contexts for record-triggered flows in Apex Trigger Context Considerations and Add Error Flow Element.

QUESTION NO: 56

Management asked for opportunities to be automatically created for accounts with annual revenue greater than $1, 000,000. A developer created the following trigger on the Account object to satisfy this requirement.

for (Account a : Trigger.new) {

if (a.AnnualRevenue > 1000000) {

List < Opportunity > oppList = [SELECT Id FROM Opportunity WHERE AccountId = :a.Id];

if (oppList.size() == 0) {

Opportunity oppty = new Opportunity(Name = a.Name, StageName = ' Prospecting ' , CloseDate = System.today().addDays(30));

insert oppty;

}

}

}

Users are able to update the account records via the UI and can see an opportunity created for high annual revenue accounts. However, when the administrator tries to upload a list of 179 accounts using Data Loader, it fails with system, Exception errors.

Which two actions should the developer take to fix the code segment shown above?

Choose 2. answers

A. Query for existing opportunities outside the for loop.

B. Check if all the required fields for Opportunity are being added on creation.

C. Move the DML that saves opportunities outside the for loop.

D. Use Database query to query the opportunities.

ANSWER: A C

Explanation:

“Query for existing opportunities outside the for loop” and “Move the DML that saves opportunities outside the for loop” are the required bulkification changes. A Data Loader operation can process many Account records in a single trigger invocation. Apex enforces per-transaction governor limits, including limits on SOQL queries and DML statements. Performing a SOQL query once for each Account can exceed the synchronous SOQL-query limit when a large batch is processed. Instead, the trigger should first collect the Account IDs requiring evaluation, issue one query for Opportunities whose `AccountId` is in that ID set, and use the query results to determine which accounts already have an Opportunity.

Likewise, the code should add each new Opportunity to a single `List<Opportunity>` during iteration. After the loop has finished, it should insert that list in one DML statement. This approach processes the entire batch efficiently and keeps the transaction within the DML-statement limit. Bulk SOQL and bulk DML are core Apex trigger design practices because triggers must work correctly for both single-record UI edits and multi-record API or Data Loader transactions. Salesforce documents these patterns in its Bulk Apex Trigger Best Practices and describes the applicable limits in Apex Governor Limits.

QUESTION NO: 57

Which three options can be accomplished with formula fields? (Choose three.)

A. Generate a link using the HYPERLINK function to a specific record.

B. Display the previous value for a field using the PRIORVALUE function.

C. Determine if a datetime field value has passed using the NOW function.

D. Return and display a field value from another object using the VLOOKUP function.

E. Determine which of three different images to display using the IF function.

ANSWER: A C E

Explanation:

Formula fields calculate and display values dynamically whenever a record is viewed, so they are well suited to building derived text, dates, and visual indicators without storing a separate value. “Generate a link using the HYPERLINK function to a specific record.” is supported because `HYPERLINK()` constructs a clickable URL from a target URL, link text, and optional target window. A formula can construct the target using a record ID or another appropriate Salesforce URL. “Determine if a datetime field value has passed using the NOW function.” is also supported: `NOW()` returns the current date and time, which can be compared with a Date/Time field in an `IF()` expression to return a status such as “Expired” or “Active.” Finally, “Determine which of three different images to display using the IF function.” can be accomplished by using nested `IF()` expressions to select one of three `IMAGE()` results based on record data. Formula fields automatically recalculate these outcomes as the relevant record values or current time change. Salesforce documents the available formula functions, including `HYPERLINK`, `IF`, and `NOW`, at Salesforce Formula Operators and Functions. For formula-based image display patterns, see Tips for Using Formula Fields.

QUESTION NO: 58

A developer wants to invoke an outbound message when a record meets a specific criteria. Which three features satisfy this use case? (Choose three.)

A. Process builder can be used to check the record criteria and send an outbound message with Apex Code.

B. Process builder can be used to check the record criteria and send an outbound message without Apex Code.

C. Approval Process has the capability to check the record criteria and send an outbound message without Apex Code.

D. Workflows can be used to check the record criteria and send an outbound message.

E. Visual Workflow can be used to check the record criteria and send an outbound message without Apex Code.

ANSWER: B C D

Explanation:

Process Builder can evaluate criteria for a record and execute an Outbound Message as an immediate action, without requiring Apex code. The outbound message definition specifies the endpoint URL and the fields included in the SOAP notification; the process determines when that notification is sent based on its criteria and evaluation behavior.

An Approval Process can also use record-entry criteria and approval actions to send an outbound message without custom code. Outbound messages are available workflow actions that can be configured as initial submission, final approval, final rejection, or other approval-process actions, allowing Salesforce to notify an external endpoint as the record proceeds through the approval lifecycle.

Workflow Rules directly support this requirement as well. A workflow rule evaluates defined criteria when a record is created or updated, and an outbound message can be configured as a workflow action. Salesforce sends the SOAP message asynchronously and retries delivery when appropriate, so the receiving service should be prepared to handle repeated notifications. See Salesforce’s Outbound Messaging documentation and the Outbound Messaging API guide.

QUESTION NO: 59

A developer identifies the following triggers on the Expense _c object:

```
deleteExpense,
applyDefaultsToExpense,
validateExpenseUpdate;
```

The triggers process before delete, before insert, and before update events respectively.

Which two techniques should the developer implement to ensure trigger best practices are followed?

Choose 2 answers

A. Unify all three triggers in a single trigger on the Expense__c object that includes all events.

B. Unify the before insert and before update triggers and use Flow for the delete action.

C. Create helper classes to execute the appropriate logic when a record is saved.

D. Maintain all three triggers on the Expense __c object, but move the Apex logic out of the trigger definition.

ANSWER: A C

Explanation:

Unify all three triggers in a single trigger on the Expense__c object that includes all events. is correct because Salesforce recommends defining one trigger per object. A single trigger can declare all required contexts, such as `before insert`, `before update`, and `before delete`, then route processing according to the active trigger context. This creates one clear entry point for automation on Expense__c and makes the implementation easier to maintain as requirements evolve. Salesforce's Apex trigger guidance discusses trigger context variables and patterns for handling the event that invoked a trigger; see Apex Trigger Context Variables.

Create helper classes to execute the appropriate logic when a record is saved. is also correct. The trigger should remain thin: it identifies the current operation and delegates business processing to appropriately named handler or helper methods. Moving executable business logic into Apex classes improves separation of concerns, enables focused unit tests, and supports reuse without duplicating logic among event branches. The handler must still be bulkified: its methods should operate on the complete trigger record collection and use collections for queries and DML. Salesforce outlines these scalable coding practices in Bulk Apex Trigger Design Patterns.

QUESTION NO: 60

Which three statements are true regarding the @isTest annotation? (Choose three.)

A. A method annotated @isTest(SeeAllData=true) in a class annotated @isTest(SeeAllData=false) has access to all org data.

B. A method annotated @isTest(SeeAllData=false) in a class annotated @isTest(SeeAllData=true) has access to all org data.

C. A class containing test methods counts toward the Apex code limit regardless of any @isTest annotation.

D. Products and Pricebooks are visible in a test even if a class is annotated @isTest(SeeAllData=false).

E. Profiles are visible in a test even if a class is annotated @isTest(SeeAllData=false).

ANSWER: A D E

Explanation:

A method annotated `@isTest(SeeAllData=true)` in a class annotated `@isTest(SeeAllData=false)` has access to organization data because a method-level `SeeAllData=true` setting grants that individual test method access to existing org records. This should be used sparingly, because isolated tests that create their own data are more reliable and portable across environments.

Products and Pricebooks are among the record types Salesforce makes available to tests even when `SeeAllData=false` is used. Salesforce documents that tests can access the standard price book and product and price-book objects under test isolation, enabling tests to create and use price-book entries without enabling unrestricted access to organization data.

Profiles are also available with `SeeAllData=false`. This allows a test to query a profile when creating test users or establishing user context, while preserving the normal isolation of business data. Salesforce lists Profile among the metadata and setup records accessible to isolated tests. See Salesforce's Test Data Access documentation and Apex Testing documentation for the supported data-access behavior and test-design guidance.