Download Free Salesforce Plat-Dev-301 Exam PDF | PrepBolt

Don't miss out! Download the latest free Salesforce Certified Platform Developer II exam PDF questions. Access real Plat-Dev-301 dumps with verified answers and boost your chances to pass your certification on the first try with PrepBolt Plat-Dev-301 exam pdf questions and answers.

Thank you for Downloading Plat-Dev-301 exam PDF Demo

https://prepbolt.com/Plat-Dev-301.html

QUESTIONS & ANSWERS

DEMO VERSION

(LIMITED CONTENT)

Question 1

Question Type: MultipleChoice

A company wants to allow support managers to see all cases in the org, regardless of who owns them. However, they want to support agents to only see cases they own or cases owned by someone in their role or a subordinate role. Which sharing solution should a developer use to achieve this requirement?

Options:

A- Sharing sets
B- Apex managed sharing
C- Role hierarchy
D- Criteria-based sharing rules

Answer:

C

Explanation:

The requirement describes a standard vertical access pattern that is best handled by the Role Hierarchy (Option C).

In Salesforce, when the Organization-Wide Default (OWD) for an object (like Case) is set to Private, the Role Hierarchy provides the following functionality by default:

Managers see subordinates' data: Users placed higher in the hierarchy automatically gain access to records owned by or shared with users below them. By placing support managers at the top of the hierarchy, they will be able to see all cases owned by the agents.

Agents see restricted data: Agents will only see the records they own or those shared with them. The requirement that they see cases owned by 'someone in their role or a subordinate role' is exactly how the hierarchy facilitates vertical data flow.

While Criteria-based sharing rules (Option D) could potentially share cases with managers, it would be much harder to manage the 'subordinate' logic dynamically as the team grows. Sharing Sets (Option A) are specifically for Experience Cloud (External) users and do not apply to internal support staff. Apex Managed Sharing (Option B) is a 'last resort' for complex, programmatic sharing logic that cannot be achieved via declarative tools. Since the requirement aligns perfectly with the standard manager-subordinate relationship, the Role Hierarchy is the most efficient and standard solution.

Question 2

Question Type: MultipleChoice

A developer has business logic in a trigger that flags high-value opportunities. There is a new requirement to also display high value opportunities in a Lightning web component. Which two steps should the developer take to meet these business requirements, and also prevent the business logic that identifies high-value opportunities from being repeated in more than one place?

Options:

A- Call the trigger from the Lightning web component.
B- Leave the business logic code inside the trigger for efficiency.
C- Create a helper class that fetches the high value opportunities.
D- Use custom metadata to hold the high value amount.

Answer:

C, D

Explanation:

To adhere to the DRY (Don't Repeat Yourself) principle and ensure maintainability, business logic should be decoupled from specific execution contexts like triggers.

Option C is a best practice. By moving the logic into a Service or Helper class, the code becomes reusable. The trigger can call this class during DML operations to flag records, and an Apex controller (invoked by the Lightning Web Component) can call the same class to retrieve the high-value records for display. This ensures that if the criteria for 'high-value' changes, the developer only needs to update the code in one location.

Option D complements this by externalizing the 'High Value Amount' threshold. Instead of hardcoding a value (e.g., $100,000) within the Apex code, storing it in Custom Metadata allows administrators to adjust the threshold without requiring a code deployment. The helper class can dynamically query this metadata.

Option A is technically impossible; triggers are fired by the database system in response to DML, not called directly by UI components. Option B leads to 'Trigger Bloat' and prevents the LWC from accessing the logic, forcing duplication. Together, C and D provide a scalable, configurable, and reusable architecture.

Question 3

Question Type: MultipleChoice

Which code snippet represents the optimal Apex trigger logic for assigning a Lead's Region based on its PostalCode, using a custom Region__c object?

A.

Java

```
Set zips = new Set();
for(Lead l : Trigger.new) {
if(l.PostalCode != Null) {
zips.add(l.PostalCode);
}
}
List regions = [SELECT Zip_Code__c, Region_Name__c FROM Region__c WHERE Zip_Code__c IN :zips];
Map zipMap = new Map();
for(Region__c r : regions) {
zipMap.put(r.Zip_Code__c, r.Region_Name__c);
}
for(Lead l : Trigger.new) {
if(l.PostalCode != Null) {
l.Region__c = zipMap.get(l.PostalCode);
}
}
```

B.

Java

```
Set zips = new Set();
for(Lead l : Trigger.new) {
if(l.PostalCode != Null) {
zips.add(l.PostalCode);
```

```
}
}
for (Lead l : Trigger.new) {
List regions = [SELECT Zip_Code__c, Region_Name__c FROM Region__c WHERE Zip_Code__c IN :zips];
for (Region__c r : regions) {
if(l.PostalCode == r.Zip_Code__c) {
l.Region__c = r.Region_Name__c;
}
}
}
```

C.

Java

```
for (Lead l : Trigger.new) {
Region__c reg = [SELECT Region_Name__c FROM Region__c WHERE Zip_Code__c = :l.PostalCode];
l.Region__c = reg.Region_Name__c;
}
```

D.

Java

```
Set zips = new Set();
for(Lead l : Trigger.new) {
if(l.PostalCode != Null) {
zips.add(l.PostalCode);
}
}
for(Lead l : Trigger.new) {
List regions = [SELECT Zip_Code__c, Region_Name__c FROM Region__c WHERE Zip_Code__c IN :zips];
for(Region__c r : regions) {
if(l.PostalCode == r.Zip_Code__c) {
l.Region__c = r.Region_Name__c;
```

```
}

}

}
```

Options:

A- Option A
B- Option B
C- Option C
D- Option D

Answer:

A

Explanation:

The 'optimal' Apex trigger logic is defined by its bulkification and strict adherence to Salesforce governor limits. In Apex, a trigger can process up to 200 records in a single batch. If a developer places a SOQL query inside a loop, the transaction will fail with a System.LimitException if more than 100 records are processed, as the limit is 100 synchronous SOQL queries.

Option A follows the industry-standard 'Bulkify, Query, and Map' design pattern:

Collection: It iterates through the input records once to gather all relevant search criteria (Zip Codes) into a Set.

External Query: It performs a single SOQL query outside of any loop to fetch all matching Region__c records for the entire batch. This is the most critical step for scalability.

Mapping: It organizes the results into a Map<String, String> (Zip Code to Region Name). Maps provide $O(1)$ constant-time lookup performance.

Final Assignment: It iterates through the Leads a second time and assigns the Region using the Map.

Options B, C, and D are all anti-patterns. Option C is the most dangerous, as it executes a query for every single record. Options B and D are slightly better but still redundant, as they execute a query inside a loop that retrieves data already retrieved in previous iterations. Only Option A ensures the code is efficient, bulk-safe, and stays well within governor limits even under heavy data loads.

Question 4

Question Type: MultipleChoice

Refer to the test method below:

Java

```
@isTest

static void testAccountUpdate() {

Account acct = new Account(Name = 'Test');

acct.Integration_Updated__c = false;

insert acct;

CalloutUtil.sendAccountUpdate(acct.Id);

Account acctAfter = [SELECT Id, Integration_Updated__c FROM Account WHERE Id = :acct.Id][0];

System.assert(true, acctAfter.Integration_Updated__c);

}
```

The test method calls a web service that updates an external system with Account information and sets the Account's Integration_Updated__c checkbox to True when it completes. The test fails to execute and exits with an error: "Methods defined as TestMethod do not support Web service callouts." What is the optimal way to fix this?

Options:

A- Add if (!Test.isRunningTest()) around CalloutUtil.sendAccountUpdate.
B- Add Test.startTest() before and Test.stopTest() after CalloutUtil.sendAccountUpdate.
C- Add Test.startTest() before and Test.setMock and Test.stopTest() after CalloutUtil.sendAccountUpdate.
D- Add Test.startTest() and Test.setMock before and Test.stopTest() after CalloutUtil.sendAccountUpdate.

Answer:

D

Explanation:

Salesforce enforces a strict restriction: Actual network callouts are prohibited during unit tests. This is to ensure that tests are deterministic, fast, and do not rely on the availability or state of external third-party systems. When the testing engine encounters a System.Http.send() or a web service call without a mock, it throws the error: 'Methods defined as TestMethod do no1t support Web servi2ce callouts.'34

To resolve this, the developer must provide a Mock Implementation. By using Test.setMock() (Option D), the developer instructs the Apex runtime to intercept any callouts and return a pre-defined

response instead of attempting a real connection. The mock class5 must implement either the HttpCalloutMock interface (for REST) or the WebServiceMock interface (for SOAP).

Furthermore, the call to the mock and the callout method should be wrapped in Test.startTest() and Test.stopTest().6

Test.startTest(): Resets governor limits, providing a fresh context for the specific logic being tested.7

Test.stopTest(): Forces any asynchronous processing (often used in cal8louts, such as @future or Queueable) to complete before the next line of code executes.

In the provided code, Test.setMock must be called before CalloutUtil.sendAccountUpdate for the platform to know which mock to use. Once Test.stopTest() is reached, the mock response is processed, the checkbox is updated, and the subsequent SOQL query and assertion will correctly see the updated data. Option A is a poor practice because it skips the logic entirely, resulting in 0% code coverage for the integration logic.

Question 5

Question Type: MultipleChoice

An org has a requirement that addresses on Contacts and Accounts should be normalized to a company standard by Apex code any time that they are saved. What is the optimal way to implement this?

Options:

A- Apex triggers on Contact and Account that normalize the address
B- Apex trigger on Account that calls the Contact trigger to normalize the address
C- Apex trigger on Contact that calls the Account trigger to normalize the address
D- Apex triggers on Contact and Account that call a helper class to normalize the address

Answer:

D

Explanation:

In software engineering, the DRY (Don't Repeat Yourself) principle is fundamental for maintainability. If the logic for normalizing an address is identical for both Accounts and Contacts, that logic should not be duplicated across two separate trigger files.

The optimal implementation (Option D) is to create a single Trigger Helper or Utility class. This class

contains a static method (e.g., AddressService.normalize(List<sObject> records)) that performs the normalization logic. You then create a small trigger on the Account object and a small trigger on the Contact object, both of which simply invoke this shared helper method.

This approach offers several advantages:

Maintainability: If the company standard for addresses changes (e.g., changing 'St.' to 'Street'), the developer only needs to update the code in one place.

Reusability: The helper method can also be called from other contexts, such as an Anonymous Apex script or a Batch job.

Readability: Triggers remain 'thin' and easy to read, acting only as entry points rather than containing complex business logic.

Options B and C are technically impossible; a trigger on one object cannot 'call' a trigger on another object directly. Option A is less optimal because it leads to code duplication and a higher risk of bugs when logic is updated in one trigger but forgotten in the other.

Question 6

Question Type: MultipleChoice

Which two queries are selective SOQL queries and can be used for a large data set of 200,000 Account records?

Options:

A- SELECT Id FROM Account WHERE Id IN (List of Account Ids)
B- SELECT Id FROM Account WHERE Name IN (List of Names) AND Customer_Number__c = 'ValueA'
C- SELECT Id FROM Account WHERE Name != ''
D- SELECT Id FROM Account WHERE Name LIKE '%Partner'

Answer:

A, B

Explanation:

In Large Data Volume (LDV) environments, Salesforce uses the Query Optimizer to determine if a query can use an index. A query is 'selective' if it filters the records using an indexed field and reduces the result set to less than 10% of the first million records.

Option A is highly selective. The Id field is the primary key and is always indexed. Filtering by a list of

IDs is the most efficient way to query records because it allows the database to perform a direct index lookup.

Option B is selective because it filters on the Name field (which is a standard indexed field) and Customer_Number__c. In the context of PDII, a field like 'Customer Number' is typically marked as an External ID, which automatically creates a custom index. Even without the second filter, the IN clause on an indexed field like Name makes the query selective.

Option C is non-selective because negative filters (using !=) and 'not null' checks generally force the database to perform a full table scan. Option D is non-selective because it uses a leading wildcard (%Partner). While the Name field is indexed, the index cannot be used if the search string starts with a wildcard, as the database must inspect the end of every string in the table.

Question 7

Question Type: MultipleChoice

What is the best practice to initialize a Visualforce page in a test class?

Options:

A- Use controller.currentPage.setPage (MyTestPage);
B- Use Test.setCurrentPage.MyTestPage;
C- Use Test.setCurrentPage (Page.MyTestPage);
D- Use Test.currentPage.getParameters.put (MyTestPage);

Answer:

C

Explanation:

When writing unit tests for Visualforce controllers, the code must be executed within a simulated 'Page Context.' Without setting this context, methods that rely on ApexPages.currentPage() (such as retrieving URL parameters or adding page messages) will fail with a null pointer exception.

The standard and correct syntax to set this context is Test.setCurrentPage(Page.MyTestPage); (Option C). The Page reference (e.g., Page.MyTestPage) is a special system variable that represents the URL of the Visualforce page. Passing this into Test.setCurrentPage() tells the Apex testing engine: 'Act as if the browser is currently looking at this specific page.'

Options A and B use incorrect syntax that does not exist in the Apex language. Option D is a secondary step; you use .getParameters().put() after setting the current page to simulate passing

specific values (like an ID) in the URL. By properly initializing the page context using Option C, the developer ensures that the controller's logic---including constructors and action methods---can be tested accurately and reliably.

Thank You for trying Plat-Dev-301 PDF Demo

To try our Plat-Dev-301 practice exam software visit link below

https://prepbolt.com/Plat-Dev-301.html

Start Your Plat-Dev-301 Preparation

Use Coupon “SAVE50” for extra 50% discount on the purchase of Practice Test Software. Test your Plat-Dev-301 preparation with actual exam questions.