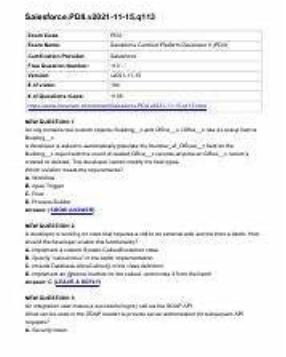
Exam Salesforce PDII Collection Pdf - Exam PDII Overview



BTW, DOWNLOAD part of BraindumpStudy PDII dumps from Cloud Storage: https://drive.google.com/open?id=1-vWLqXgyK8Qn_rCYkaFu-Me9mSPH2rHz

The Salesforce PDF Questions format designed by the BraindumpStudy will facilitate its consumers. Its portability helps you carry on with the study anywhere because it functions on all smart devices. You can also make notes or print out the Platform Developer II (PDII) pdf questions. The simple, systematic, and user-friendly Interface of the Platform Developer II (PDII) PDF dumps format will make your preparation convenient.

Salesforce PDII (Salesforce Certified Platform Developer II) Certification Exam is designed to evaluate the expertise of Salesforce developers who possess advanced skills and knowledge of the platform. Platform Developer II certification is intended for developers who want to take their skills to the next level and work on complex projects. PDII Exam Tests the developer's ability to build customized applications on the Salesforce platform using Apex and Visualforce.

>> Exam Salesforce PDII Collection Pdf <<

Pass Guaranteed Quiz 2026 Salesforce PDII – High-quality Exam Collection Pdf

Many clients worry that after they bought our PDII exam simulation they might find the exam questions are outdated and waste their time, money and energy. There are no needs to worry about that situation because our PDII study materials boost high-quality and it is proved by the high passing rate and hit rate. And we keep updating our PDII learing quiz all the time. We provide the best PDII practice guide and hope our sincere service will satisfy all the clients.

PDII Exam topics

Candidates must know the exam topics before they start of preparation. Because it will really help them in hitting the core. Our **Salesforce PDII Exam Dumps** will include the following topics:

- Performance 7%
- Salesforce Fundamentals 5%
- Testing 12%
- Logic and Process Automation 33%

Salesforce Platform Developer II Sample Questions (Q44-Q49):

NEW QUESTION # 44
A developer wrote the following method to find all the test accounts in the org:

```
public static Account[] searchTestAccounts(){
    List<List<SObject>> search... IN ALL FIELDS
                        RETURNING Account(Name)];
    return (Account[]) searchList[0];
}
```

However, the test method below fails.

```
@isTest
public static void testSearchTestAccounts(){
    Account a = new Account(name='test');
    insert a;
    Account [] accounts = TestAccountFinder.searchTestAccounts();
    System.assert(accounts.size() == 1 );
}
```

What should be used to fix this falling test?

- A. Test. loaddata to set up expected data
- B. Test. fixedSearchResults ( ) method to set up expected data
- C. @isTest (See AllData=true) to access org data for the test
- D. @testSetup method to set up expected data

Answer: C

NEW QUESTION # 45
What is the correct way to fix this?

- A. Add Test_startTest() before and add Test. stopTest() after Line 18 of the code.
- B. Add Test.startTest() before and add Test. stopTest() after both Line 7 and Line 20 of the code.
- C. Change the DataFactory class to create fewer Accounts so that the number of queries in the trigger is reduced.
- D. Use Limits.getlimitQueries() to find the total number of queries that can be issued.

Answer: A

Explanation:
The correct way to fix the test method is to add Test.startTest() before and add Test.stopTest() after Line 18 of the code. Test.startTest() and Test.stopTest() are methods that mark the start and the end of the test code and ensure that all asynchronous processes are completed before the test method finishes. This way, the developer can verify that the batch job is executed and the Account records are updated as expected. The developer should also use the System.assertEquals() method to assert the expected and actual values, instead of System.debug() which only prints the values to the debug log . Reference: [Testing Asynchronous Apex], [Test.startTest()], [System.assertEquals()]

NEW QUESTION # 46
What are three reasons that a developer should write Jest tests for Lightning web components?

- A. To test how multiple components work together
- B. To test basic user interaction
- C. To verify the DOM output of a component

- D. To test a component's non-public properties
- E. To verify that events fire when expected

Answer: B,C,E

Explanation:
Jest is a powerful JavaScript testing framework used for Lightning Web Components (LWC) to ensure individual units of code function correctly in isolation. One of the primary reasons to use Jest is to verify the DOM output of a component (D). Since LWC is a UI framework, Jest allows developers to inspect the rendered HTML to ensure that elements, classes, and data are displayed as intended after a state change.
Furthermore, Jest is essential for testing basic user interaction (A). Developers can simulate events like button clicks or text input and then assert that the component's state or UI updates accordingly.
Another critical use case is to verify that events fire when expected (E). Components often communicate with parents via custom events; Jest allows you to "spy" on these events to ensure they are dispatched with the correct detail payload at the right time.
Conversely, Jest is not intended for testing how multiple components work together (C)-this is the domain of integration tests or end-to-end tests (like UTAM). Additionally, Jest tests should focus on the component's public API and observable behavior; testing non-public (private) properties (B) is generally discouraged as it leads to brittle tests that break upon internal refactoring. By focusing on DOM output, interactions, and events, Jest provides a fast, reliable way to maintain component quality.

NEW QUESTION # 47
A developer wrote a test class that successfully asserts a trigger on Account, It fires and updates data correctly In a sandbox environment.
A Salesforce admin with a custom profile attempts to deploy this trigger via a change set into the production environment, but the test class falls with an Insufficient privileges error.
What should a developer do to fix the problem?

- A. Configure the production environment to enable "Run All Tests as Admin User."
- B. Add seeallData=true to the test class to work within the sharing model for the production environment.
- C. Add System.runAs() to the test class to execute the trigger as a user with the correct object permissions.
- D. Verify that Test. startTest() Is not Inside a For loop in the test class,

Answer: C

Explanation:
When a test class fails with an "Insufficient privileges" error during deployment, it indicates that the user profile under which the tests are being executed doesn't have the necessary permissions to perform the actions required by the test. In a sandbox, the test might have been running with a different set of permissions compared to the production environment.
By using System.runAs(), you can specify a user context in which the test should run, which allows you to simulate the appropriate permissions. For this to work, you need to create a User instance in your test class with the profile that has the necessary permissions and then enclose the logic of your test within a System.
runAs() block with this user.
This is a preferred solution over seeAllData=true, which would give the test access to all data in the production environment, potentially leading to tests that are not isolated and thus less reliable. It's also preferred over changing organization-wide settings or relying on specific setup in test classes, such as making sure Test.startTest() is not inside a loop.
References:
Salesforce Documentation on Using the runAs Method: Testing with the runAs Method Salesforce Help Article on System Permissions: Profiles and Permissions

NEW QUESTION # 48
A developer wrote a trigger on Opportunity that will update a custom Last Sold Date field on the Opportunity's Account whenever an Opportunity is closed. In the test class for the trigger, the assertion to validate the Last Sold Date field fails.
What might be causing the failed assertion?

- A. The test class has not implemented seealldata=true in the test method.
- B. The test class has not re-queried the Account record after updating the Opportunity.
- C. The test class has not defined an Account owner when inserting the test data.
- D. The test class is not using System. runs () to run tests as a Salesforce administrator.

Answer: B

Explanation:
The test class may not be re-querying the Account record after updating the Opportunity, which is necessary to verify the updated field values. If the test does not query the database to get the most recent data after the trigger runs, it will not see the changes made by the trigger.
References: Apex Developer Guide - Testing Best Practices

NEW QUESTION # 49
.....