IT认证电子书

质　量　更　高　　服　务　更　好

半年免费升级服务

http://www.itrenzheng.com

Exam : **DEX-450**

Title : Programmatic Development using Apex and Visualforce in Lightning Experience

Version : DEMO

1.While developing an Apex class with custom search functionality that will be launched from a Lightning Web Component, how can the developer ensure only records accessible to the currently logged in user are displayed?
A. Use the inherited sharing keyword.
B. Use the WITH SECURITY_ENFORCED clause within the SOQL.
C. Use the with sharing keyword.
D. Use the without sharing keyword.
Answer: C
Explanation:
To ensure that only records accessible to the currently logged-in user are displayed when developing an Apex class for custom search functionality launched from a Lightning Web Component (LWC), the developer should use the with sharing keyword.
with sharing Keyword: This keyword enforces the sharing rules of the current user, ensuring that the Apex class respects the user's record-level access permissions. When an Apex class is declared with with sharing, it enforces sharing rules, meaning users can only access records they have permission to see.
"Use the with sharing keyword when declaring a class to enforce the sharing rules that apply to the current user."
— Apex Developer Guide: Using the with sharing or without sharing Keywords
Importance in LWCs: Lightning Web Components invoke Apex methods annotated with @AuraEnabled, which execute in system context by default. Without specifying sharing, these methods can access all records, potentially exposing data that the user shouldn't see.
"By default, Apex code runs in system context; that is, the current user's permissions and field-level security do not apply."
— Apex Developer Guide: Enforcing Sharing Rules
Why Not inherited sharing: While inherited sharing allows an Apex class to run in the sharing context of the caller, in this scenario, since the LWC does not have an inherent sharing context, using with sharing explicitly ensures sharing rules are enforced.
WITH SECURITY_ENFORCED Clause: This SOQL clause enforces field-level security (FLS) and object-level security (OLS), not record-level sharing rules. Therefore, it does not restrict records based on the user's sharing settings.
"The WITH SECURITY_ENFORCED clause applies object-level and field-level security checks to SOQL queries to ensure that users don't see fields or objects they don't have access to."
— SOQL and SOSL
Reference: WITH SECURITY_ENFORCED Clause
Conclusion: By declaring the Apex class with with sharing, the developer ensures that only records the user has access to are returned, aligning with the requirement.

2.While working in a sandbox, an Apex test fails when run in the Test Runner. However, executing the Apex logic in the Execute Anonymous window succeeds with no exceptions or errors.
Why did the method fail in the sandbox test framework but succeed in the Developer Console?
A. The test method is calling an @future method.
B. The test method relies on existing data in the sandbox.
C. The test method has a syntax error in the code.

D. The test method does not use Sysetem.runas= to execute as a specific user.
Answer: B
Explanation:
The Apex test fails in the Test Runner but succeeds in the Execute Anonymous window because the test method relies on existing data in the sandbox.
Isolation of Test Data: By default, Apex tests do not have access to pre-existing data in the organization. Tests are executed in an isolated context to ensure they are not dependent on external data, promoting reliability and repeatability.
"Apex test methods don’t have access to pre-existing data in the organization. The test methods have access to all data that they create and to changes made by other test methods (unless the @IsTest(Isolated=true) annotation is used)."
— Apex Developer Guide: Isolation of Test Data from Organization Data in Unit Tests
Execute Anonymous Window: When code is run in the Execute Anonymous window, it executes in the current user's context and has access to all organization data, including existing records.
"All Apex code runs in system mode. The only exception is anonymous blocks, such as code executed using the Execute Anonymous window or the SOAP API."
— Apex Developer Guide: Enforcing Object and Field Permissions
Solution: To fix the failing test, the developer should create the necessary test data within the test method or use the @IsTest(SeeAllData=true) annotation if access to existing data is absolutely necessary (though this is generally discouraged).
"If you specify @IsTest(SeeAllData=true), the test method has access to all data in the organization, but the test can be slower and less reliable because it might depend on data that changes."
— Apex Developer Guide: Using the SeeAllData Annotation Why Other Options Are Incorrect:
Option A: While calling an @future method requires special handling in tests, it would not cause the method to pass in Execute Anonymous and fail in tests.
Option C: A syntax error would prevent the code from compiling, so it wouldn't run in either context.
Option D: Using System.runAs is only necessary when testing code that behaves differently for users with different profiles or permissions.
Conclusion: The discrepancy arises because the test method is attempting to access data that isn't available in the test context, leading to its failure in the Test Runner but success in the Execute Anonymous window.

3.Universal Containers has implemented an order management application. Each Order can have one or more Order Line items. The Order Line object is related to the Order via a master-detail relationship. For each Order Line item, the total price is calculated by multiplying the Order Line item price with the quantity ordered.
What is the best practice to get the sum of all Order Line item totals on the Order record?
A. Quick action
B. Apex trigger
C. Roll-up summary field
D. Formula field
Answer: C
Explanation:
To calculate the sum of all Order Line item totals on the Order record in a master-detail relationship, the

best practice is to use a Roll-up Summary Field.
Roll-up Summary Field: This field type allows you to perform calculations on a set of related records, such as summing up the total price of all Order Line items related to an Order.
"Roll-up summary fields allow you to display a value in a master record based on the values of fields in a detail record."
— Salesforce Help: Define Roll-Up Summary Fields
Master-Detail Relationship: Since Order Line items are in a master-detail relationship with Orders, roll-up summary fields can be created on the master object (Order) to summarize data from the detail object (Order Line).
"You can create roll-up summary fields on any custom object that is on the master side of a master-detail relationship."
— Salesforce Developer Guide: Roll-Up Summary Fields Why Not Other Options:
A. Quick Action: Quick actions are used to create records, log calls, send emails, etc., and are not suitable for calculating and displaying aggregate data.
B. Apex Trigger: While triggers can perform the calculation, using them for this purpose is not considered best practice when declarative solutions like roll-up summary fields are available. "Use declarative functionality (roll-up summary fields) for maintaining aggregates instead of triggers."
— Salesforce Developer Guide: Best Practices for Triggers
D. Formula Field: Formula fields cannot aggregate data from child records to parent records unless they are part of a roll-up summary field in a master-detail relationship.
Conclusion: Using a roll-up summary field is the most efficient and recommended approach to sum the total prices of all Order Line items on the Order record.

4.Universal Containers has a Visualforce page that displays a table of every Container___c being rented by a given Account. Recently this page is failing with a view state limit because some of the customers rent over 10,000 containers.
What should a developer change about the Visualforce page to help with the page load errors?
A. Implement pagination with an OffsetController.
B. Use JavaScript remoting with SOQL Offset.
C. Implement pagination with a StandardSetController.
D. Use lazy loading and a transient List variable.
Answer: C
Explanation:
To address the view state limit error caused by loading over 10,000 records in a Visualforce page, the developer should implement pagination with a StandardSetController.
StandardSetController: This controller allows developers to easily paginate large sets of records in Visualforce pages.
"A StandardSetController object contains a reference to a list of records with pagination support, similar to the display in a list view."
— Apex Developer Guide: StandardSetController Class Benefits:
Efficient Data Handling: It retrieves only a subset of records at a time, reducing the amount of data held in view state.
Built-in Pagination: Provides methods to navigate through pages of records (next, previous, etc.).
View State Limit: Visualforce pages have a view state limit of 170KB. Loading all records at once

exceeds this limit.
"To optimize your Visualforce pages, minimize your use of view state. The view state of a Visualforce page is limited to 170KB."
— Visualforce Developer Guide: View State in Visualforce Pages
Why Not Other Options:
A. Implement pagination with an OffsetController: There's no standard OffsetController in Visualforce. Implementing custom pagination with offsets is less efficient and can be complex.
B. Use JavaScript remoting with SOQL Offset: While JavaScript remoting can reduce view state size, using SOQL OFFSET is not efficient for large offsets (over 2,000 records).
"When using OFFSET with a large number of records, performance degrades significantly."
— SOQL and SOSL
Reference: OFFSET Clause
D. Use lazy loading and a transient List variable: Lazy loading can help, but it doesn't solve the issue of handling large datasets efficiently. Transient variables reduce view state but do not manage data retrieval or pagination.
Conclusion: Implementing pagination with a StandardSetController is the recommended solution to handle large datasets efficiently in Visualforce pages and prevent view state limit errors.

5.Universal Containers implemented a private sharing model for the Account object. A custom Account search tool was developed with Apex to help sales representatives find accounts that match multiple criteria they specify. Since its release, users of the tool report they can see Accounts they do not own.
What should the developer use to enforce sharing permissions for the currently logged in user while using the custom search tool?
A. Use the schema describe calls to determine if the logged-in user has access to the Account object,
B. Use the with sharing keyword on the class declaration.
C. Use the without sharing keyword on the class declaration.
D. Use the UserInfo Apex class to filter all SOQL queries to returned records owned by the logged-in user.
Answer: B
Explanation:
Since users can see Accounts they do not own in a private sharing model, the custom Apex code is likely not enforcing sharing rules. To enforce sharing permissions, the developer should use the with sharing keyword on the class declaration.
with sharing Keyword: This enforces the sharing rules of the current user, ensuring that the Apex class respects the user's record-level access permissions.
"Use the with sharing keyword when declaring a class to enforce the sharing rules that apply to the current user."
— Apex Developer Guide: Using the with sharing or without sharing Keywords
Private Sharing Model: In a private sharing model, users should only see records they own or have been shared with them.
"Private: Only the record owner and users above that role in the hierarchy can view, edit, and report on those records."
— Salesforce Help: Organization-Wide Sharing Defaults Why Not Other Options:
A. Use the schema describe calls to determine if the logged-in user has access to the Account object:

Schema describe calls check for object-level access, not record-level sharing.
"Schema describe information provides metadata about object and field properties, but it doesn't enforce record-level access."
— Apex Developer Guide: Schema Namespace
C. Use the without sharing keyword on the class declaration: This runs the class in system context, ignoring sharing rules, which is the opposite of what's needed.
"Classes declared as without sharing or those that do not specify a keyword default to without sharing and don't enforce the sharing rules of the current user."
— Apex Developer Guide: Using the with sharing or without sharing Keywords
D. Use the UserInfo Apex class to filter all SOQL queries to return records owned by the logged-in user: Manually filtering queries is error-prone and not a best practice when with sharing can enforce sharing automatically.
"Avoid hardcoding user or profile IDs and using the UserInfo class to enforce security. Instead, use declarative security features."
— Apex Developer Guide: Enforcing Security in Apex
Conclusion: By declaring the class with with sharing, the Apex code respects the user's sharing rules, ensuring that users only see Accounts they have access to.