Salesforce.PDII-JPN.v2026-05-04.q76

Exam Code:	PDII-JPN
Exam Name:	
Certification Provider:	Salesforce
Free Question Number:	76
Version:	v2026-05-04
# of views:	153
# of Questions views:	760
https://www.freepdfdumps.com/Salesforce.PDII-JPN.v2026-05-04.q76.html	

NEW QUESTION: 1
開発者は、商談のステージが変更された回数を追跡する既存の機能を使用しています。商談のステージが変更されると、ワークフロールールが起動され、項目の値が1増加します。開発者は、項目の値が4から5に変更された際に子レコードを作成するためのafterトリガを作成しました。ユーザーが商談のステージを変更し、カウント項目を手動で4に設定します。カウント項目は5に更新されますが、子レコードは作成されません。なぜこのようなことが起こるのでしょうか？
A. After トリガーはワークフロー ルールの前に実行されます。
B. フィールドの更新後、Trigger.new は変更されません。
C. Trigger.old には、カウント フィールドの更新された値が含まれていません。
D. フィールドの更新後に After トリガーは起動されません。
Answer: (SHOW ANSWER)
12
The issue described is rooted in the Salesforce Order of Execution. When a record is saved, Salesforce follows a very specific sequence of events. First, original triggers (before and after) are executed. Only after these triggers have completed does the system evaluate and execute workflow rules.34 In this scenario, th5e following happens:6
* The user saves the Opportunity with the count set to 4.7
* The After Trigger fires. At this point, the field value i8s still 4. Since the trigger logic only creates a child record when the value "changes from 4 to 5," and it is currently exactly 4 (not yet 5), the condition is not met.
* Next, the Workflow Rule fires and performs a field update, changing the count from 4 to 5.
* According to the platform rules, a workflow field update causes Before and After Triggers to fire one more time to account for the change. However, in this second execution of the after trigger, Trigger.old now contains the value 4 (from the start of the second execution) and Trigger.new contains 5. While the condition old == 4 and new == 5 is now technically

true, many developers fail to realize that the first execution (before the workflow) already bypassed the logic because the workflow hadn't updated the value yet.
If the trigger was written to look for the exact transition, it might fail during the first pass (where the value is
4) and potentially execute during the second pass. However, if the logic is not specifically handled to account for the re-entrant nature of workflow field updates, or if the "After triggers fire before workflow rules" (Option A) is the primary constraint being tested, it explains why the initial save logic didn't see the "5" until it was too late in the initial trigger context.

NEW QUESTION: 2
Visualforce 検索ページで使用される RemoteAction を定義する以下の Apex クラスを検討してください。
Java
global with sharing class MyRemoter {
public String accountName { get; set; }
public static Account account { get; set; }
public MyRemoter() {}
@RemoteAction
global static Account getAccount(String accountName) {
account = [SELECT Id, Name, NumberOfEmployees FROM Account WHERE Name
= :accountName]; return account;
}
}
リモート アクションが正しいアカウントを返したことをアサートするコード スニペットはどれですか。
A. Java
MyRemoter remote = new MyRemoter();
Account a = remote.getAccount('TestAccount');
System.assertEquals('TestAccount', a.Name);
B. Java
MyRemoter remote = new MyRemoter('TestAccount');
Account a = remote.getAccount();
System.assertEquals('TestAccount' , a.Name);
C. Java
Account a = controller.getAccount('TestAccount');
System.assertEquals('TestAccount', a.Name);
D. Java
Account a = MyRemoter.getAccount('TestAccount');
System.assertEquals('TestAccount', a.Name);
Answer: D (LEAVE A REPLY)

In Salesforce Apex, a @RemoteAction method must be defined as static. Static methods belong to the class itself rather than to a specific instance of the class. Therefore, when writing a unit test to verify the behavior of a RemoteAction, the method should be called using the class name notation (ClassName.methodName) rather than by instantiating the class first.
Option D correctly demonstrates the standard way to invoke a static method in Apex: Account a = MyRemoter.getAccount('TestAccount');. Because the method is static, it does not require the new keyword or a constructor to be called. Option A is incorrect because it attempts to call a static method on an instance variable (remote.getAccount), which is considered an illegal or improper reference in Apex. Option B is incorrect because it treats the RemoteAction like an instance method and assumes a non-existent constructor that takes a String. Option C is incorrect because it uses an undefined variable controller.
By calling the method statically in the test, the developer correctly simulates how the Visualforce Remoting engine calls the method from JavaScript at runtime. The assertion System.assertEquals then verifies that the record retrieved from the database matches the expected name, ensuring the search logic is functioning correctly.

NEW QUESTION: 3
開発者は、ユーザーがカスタムLightningページから直接、選択した商品のケースを作成できるLightning Webコンポーネントを作成するという課題を抱えています。コンポーネント内の入力フィールドは、ユーザーがフィールドの意味を理解しやすいように、商品画像の上に非線形に表示されます。開発者は、Lightning Webコンポーネントからケースを作成するために、どの2つのコンポーネントを使用すべきでしょうか？161718
A. ライトニングレコードフォーム192021
B. lightning-record-edit-form222324
C. lightnin25g-入力フィールド2627
D. ライトニング入力2829
Answer: (SHOW ANSWER)
3233
When a requirement specifies a "non-linear" or highl34y customized layout, the high-35level lightning-record- form (A) is unsuitable because it automatically renders fields in a standard grid or based on a page layout. To achieve a custom UI-such as overlaying fields on a product image-the developer needs more granular control.
The lightning-record-edit-form (B) is the ideal container for this scenario. It provides the heavy lifting for data communication (loading and saving records) without imposing a strict visual structure. Within this form, the developer uses lightning-input-field (C) components for each Case field. These components are "context- aware," meaning they automatically inherit metadata like field labels, picklist values, and validation rules directly from the Salesforce object definition.

Because lightning-input-field components can be placed anywhere inside the lightning-record-edit-form tag, the developer can use custom CSS (absolute positioning or CSS Grid) to place them precisely over the product image as requested. Standard lightning-input (D) components could work but would require much more manual code to handle data binding, type conversion, and validation, making the combination of B and C the most efficient and platform-aligned approach.

NEW QUESTION: 4
開発者は、商談ステージの変化に応じて顧客に自動的にメールを送信するソリューションを構築するよう依頼されています。このソリューションは、1日あたり10,000通のメール送信に対応できる拡張性を備えている必要があります。メール送信の基準は、特定の条件が満たされた時点で評価される必要があります。これを実現する最適な方法は何でしょうか？
A. Batch Apex で Apex トリガーを使用します。
B. Flow Builder で電子メールアラートを使用します。
C. Apex トリガーで MassEmailMessage() を使用します。
D. Apex トリガーで SingleEmailMessage() を使用します。
Answer: B (LEAVE A REPLY)
For high-volume, automated standard email communications, Email Alerts with Flow Builder (Option B) is the most scalable and maintainable solution. Salesforce imposes strict limits on the number of "Single" emails sent via Apex (typically 5,000 per 24 hours). However, emails 5sent via Workflow Rules, Process Builder, or Flow6 Builder (Email Alerts) have a much higher daily limit (2,000,000 per org).
To meet the requirement of 10,000 emails per day, an Apex-based solution using SingleEmailMessage() (Option D) would likely hit the daily limit and cause transaction failures. MassEmailMessage() (Option C) is generally intended for manual mass mailing to Contacts or Leads and is not optimized for automated, per- record trigger logic. While Batch Apex (Option A) could process many records, the underlying SingleEmailMessage limit would still apply.
By using an Email Alert triggered by a Record-Triggered Flow, the developer leverages the platform's high- volume email engine. This approach is declarative, requires no custom code, is easier for administrators to maintain, and comfortably handles the 10,000 daily email requirement without risking governor limit exceptions. It also ensures the "criteria evaluation" happens within the standard Flow entry conditions.

NEW QUESTION: 5
ある企業では、複数のカスタムメタデータレコードに参照データを保存しており、それらは特定の地域におけるデフォルト情報と削除動作を表しています。連絡先を挿入する際には、デフォルト情報が設定される必要があります。また、フラグが設定された地域に属する

連絡先を削除しようとすると、エラーメッセージが表示される必要があります。企業の人員リソースの状況に応じて、これを自動化する2つの方法は何ですか？19
A. Apexトリガー
B. リモートアクション
C. フロービルダー
D. Apex呼び出し可能メソッド
Answer: (SHOW ANSWER)
This requirement involves two specific databas26e events: record insertion (for setting defaults) and record deletion (for enforcing business rules/errors). Both Apex Triggers (Option A) and Flow Builder (Option C) are capable of handling these scenarios.
Flow Builder is the declarative (low-code) solution. A Record-Triggered Flow can be set to run "Before Save" to efficiently update the Contact's fields with metadata defaults upon insertion. Furthermore, Flow Builder now supports "Before Delete" triggers and the "Custom Error" element, allowing an administrator to block a deletion and present a specific error message to the user if the Contact belongs to a flagged region.
Apex Triggers are the programmatic solution. A developer can write a before insert trigger to perform the Custom Metadata lookup and assign field values, and a before delete trigger to evaluate the region and call addError() on the record.
Option B (Remote Action) is for Visualforce-to-Apex communication, and Option D (Invocable Method) is a piece of code called by a Flow or Process, not an entry point itself.
Therefore, Triggers and Flows are the two primary automation frameworks for these requirements.

NEW QUESTION: 6
ある企業は、7年経過後に機密情報を自動的に削除する必要があります。そうなると、毎日約100万件のレコードが削除されることになります。どうすれば実現できるでしょうか？
A. 7 年以上前のレコードを照会する @future プロセスをスケジュールし、1,000 件のレコード バッチで自分自身を再帰的に呼び出してそれらを削除します。
B. 集計関数を使用して 7 年以上前のレコードを照会し、AggregateResult オブジェクトを削除します。
C. SOSL ステートメントを実行して、7 年以上前のレコードを検索し、結果セット全体を削除します。
D. 7 年以上前のレコードをクエリして削除するバッチ Apex プロセスを毎日実行するようにスケジュールします。
Answer: D (LEAVE A REPLY)
When dealing with large-scale data maintenance, such as deleting a million records daily, Batch Apex is the most robust and scalable solution provided by the Salesforce platform.
Batch Apex is specifically designed for processing high volumes of records by breaking the total record set into manageable "batches" (defaulting to

200 records per batch). This prevents the transaction from hitting platform governor limits, such as the limit on the total number of DML statements or the maximum number of records processed in a single transaction.
By scheduling a Batch class, the system handles the heavy lifting asynchronously, ensuring that org performance is not negatively impacted during peak hours.
Other options are unsuitable for this scale. Using @future methods for recursive processing is a poor practice because Salesforce prohibits calling a future method from another future method, and it is difficult to monitor or manage the execution flow.
Aggregate functions and AggregateResult o1bjects are used for calculations and data grouping, not for DML operations like deletion. Similarly, SOSL is optimized for text-based searching across multiple objects and returns a limited number of results, making it inappropriate for an exhaustive cleanup of a million records. Batch Apex allows for the use of a QueryLocator, which can handle up to 50 million records, providing the necessary throughput for this high-volume requirement.

NEW QUESTION: 7
開発者はSalesforceプロジェクトの導入プロセスを策定する責任を負っています。このプロジェクトはソース駆動開発アプローチを採用しており、開発者はメタデータの変更に対する効率的な導入とバージョン管理を実現したいと考えています。ソース駆動型導入プロセスを管理するために、どのようなツールまたはメカニズムを活用すべきでしょうか？78
A. データローダー910
B. 変更セット1112
C. Salesforce DX1314 を使用した Salesforce CLI
D. 管理されていないパッケージ1516
Answer: C (LEAVE A REPLY)
1920
Source-driven development shifts th21e "source of truth" from the Salesforce Org to a Version Control System (like Git). To b22ridge the gap between local source code and the Salesforce platform, Salesforce CLI with Salesforce DX (Option C) is the required mechanism.
Salesforce DX (Developer Experience) introduced a source-centric metadata format that is more granular and easier to track in version control than the traditional Metadata API. The Salesforce CLI provides the command-line tools necessary to automate the deployment process, create scratch orgs for isolated testing, and perform "source tracking" to identify exactly which files have changed. This is the foundation of modern CI/CD (Continuous Integration/Continuous Delivery) pipelines in the Salesforce ecosystem.
In contrast, Change Sets (Option B) are org-centric and manual, making them incompatible with automated version control. Data Loader (Option A) is for record data, not metadata. Unmanaged Packages (Option D) are used for distribution but do not support the iterative, source-controlled deployment workflow required for professional project management.

NEW QUESTION: 8
次のコード スニペットを参照してください。
Java
public class LeadController {
public static List<Lead> getFetchLeadList(String searchTerm, Decimal aRevenue) { String safeTerm = '%'+searchTerm.escapeSingleQuotes()+ '%'; return [SELECT Name, Company, AnnualRevenue FROM Lead WHERE AnnualRevenue >= :aRevenue AND Company LIKE :safeTerm LIMIT 20
];
}
}
ある開発者が、Lightning Webコンポーネント（LWC）の一部として、特定の条件が満たされた場合にgetFetchLeadListを呼び出してリードに関する情報を表示するJavaScript関数を作成しました。LWCがセキュリティを維持しながらデータを効率的に表示できるようにするには、上記のApexクラスにどのような3つの変更を加える必要がありますか？
A. SOQL クエリ内で WITH SECURITY_ENFORCED 句を使用します。
B. クラス宣言でwith sharingキーワードを実装します。567
C. Apex メソッドに @AuraEnabled(Cacheable=true) アノテーションを追加します。
D. クラス宣言に without sharing キーワードを実装します。
E. Apex メソッドに @AuraEnabled アノテーションを追加します。
Answer: (SHOW ANSWER)
Compreh17ensive and Detailed 11850 to 250 words of Explanation:
To make an Apex method compatible with a Lightning Web Component's @wire service and ensure it follows security best practices, three specific modifications are required:
* @AuraEnabled(Cacheable=true) (Option C): The @wire service in LWC requires the Apex method to be marked as cacheable. This enables client-side caching via the Lightning Data Service, which significantly improves UI performance by reducing redundant server calls. Note that Cacheable=true is mandatory for @wire but optional for imperative calls.
* with sharing (Option B): In Apex, classes do not enforce sharing rules by default. To ensure the user only sees Leads they have access to according to the organization-wide defaults and sharing model, the class must explicitly use the with sharing keyword.
* WITH SECURITY_ENFORCED (Option A): While with sharing handles record-level access, it does not automatically enforce field-level security (FLS) or object-level security (CRUD). Adding the WITH SECURITY_ENFORCED clause to the SOQL query ensures that if a user does not have permission to view the AnnualRevenue field, the query will throw an exception rather than exposing protected data.
Options D and E are incorrect because without sharing bypasses security, and a simple @AuraEnabled without cacheable=true is insufficient for the LWC @wire service.

NEW QUESTION: 9
開発者が、フィールド履歴の追跡機能を利用するAccountHistoryManagerというクラスを作成しました。このクラスには、Accountをパラメータとして受け取り、関連付けられたAccountHistoryオブジェクトのレコードのリストを返すgetAccountHistoryという静的メソッドがあります。以下のテストは失敗します。
Java
@isTest
public static void testAccountHistory(){
Account a = new Account(name = 'test');
insert a;
a.name = a.name + '1';
update a;
List<AccountHistory> ahList = AccountHistoryManager.getAccountHistory(a);
System.assert(ahList.size() > 0);
}
このテストに合格するには何をすべきでしょうか?
A. @isTest(SeeAllData=true) を使用して、組織の履歴データを表示し、AccountHistory レコードを照会します。
B. getAccountHistory() で Test.isRunningTest() を使用して、条件付きで偽の AccountHistory レコードを返します。
C. このコードはテストできないため、テストメソッドを削除する必要があります。
D. テスト セットアップで AccountHistory レコードを手動で作成し、それらを取得するためのクエリを記述します。
Answer: B (LEAVE A REPLY)
A significant challenge in Salesforce unit testing is that system-generated records, such as those in the AccountHistory or OpportunityHistory objects, are not created during the execution of a test method, even if field history tracking is enabled in the organization. Because these records are read-only and managed by the system, a developer cannot manually insert them via DML within a test setup. Consequently, the query in the getAccountHistory method will always return an empty list in a test context, causing the System.assert to fail.
The optimal programmatic solution to this limitation is to implement a "mocking" strategy using Test.
isRunningTest(). By modifying the production code within getAccountHistory, the developer can check if the code is currently being executed by a unit test. If it is, the method can return a manually constructed list of AccountHistory records (stored in memory but not inserted into the database) to satisfy the test's requirements. This allows the test to verify the logic that processes the history data without needing the system to actually generate historical records. While SeeAllData=true (Option A) would allow the test to see real data in the org, it is considered a poor practice because it makes the test dependent on the

specific state of the organization's data, leading to brittle tests that may fail in different environments.12
==========34

NEW QUESTION: 10
ある企業は、カタログとカタログアイテムというカスタムオブジェクトで、自社製品の情報を管理しています。カタログアイテムにはカタログのマスター詳細項目があり、各カタログには最大10万件のカタログアイテムを含めることができます。どちらのカスタムオブジェクトにも、使用する通貨コードを入力するCurrencyIsoCodeテキストフィールドがあります。カタログのCurrencyIsoCodeが変更された場合、そのカタログアイテムのすべてのCurrencyIsoCodeも変更する必要があります。開発者は、カタログのCurrencyIsoCodeが変更された場合に、カタログアイテムのCurrencyIsoCodeを更新するには、どのような方法を使用すればよいでしょうか？
A. カタログの CurrencyIsoCode が異なる場合にカタログ項目を更新する、カタログの更新後トリガー。
B. カタログの CurrencyIsoCode が異なる場合にカタログ アイテムを更新する、カタログ アイテムの挿入後トリガー。
C. Catalog オブジェクトを照会し、Catalog CurrencyIsoCode が異なる場合に Catalog Items を更新する Database.Schedulable および Database.Batchable クラス。
D. カタログ アイテム オブジェクトを照会し、カタログ CurrencyIsoCode が異なる場合にカタログ アイテムを更新する Database.Schedulable および Database.Batchable クラス。
Answer: D (LEAVE A REPLY)
The primary challenge in this scenario is the high volume of child records (up to 100,000 Catalog Items) associated with a single parent Catalog. In Salesforce, synchronous transactions like Apex triggers are subject to strict governor limits, most notably the limit of 10,000 DML rows per transaction. If a developer were to use an "after update" trigger on the Catalog object (Option A) to update 100,000 child items, the transaction would immediately fail with a LimitException as soon as it exceeded that threshold.23 To handle such a massive update succ4essfully, the developer must use an asynchronous approach, specifically Batch Apex. By implementing Database.Batchable, the platf5orm can process the 100,000 records in smaller chunks (batches), each within its own set of governor limits.
Option D is the correct implementation because the Batch class should query the Catalog Item (the child object) where the CurrencyIsoCode does not match the parent Catalog. This targets only the records that need modification.
While Option C suggests batching the parent Catalog, it is more efficient to query the child items directly to ensure every record needing an update is processed across the multiple batches of the execution. This ensures data consistency across the entire 100,000 record set without risking transaction failures.

NEW QUESTION: 11

開発者は、非同期プロセスを含むトリガーが正常に実行されたことをどのように確認すればよいでしょうか?
A. すべてのテスト データを作成し、テスト クラスで @future を使用して、アサーションを実行します。
B. テスト クラスにすべてのテスト データを作成し、System.runAs() を使用してトリガーを呼び出して、アサーションを実行します。
C. テスト クラスにすべてのテスト データを作成し、Test.startTest() と Test.stopTest() を呼び出してアサーションを実行します。
D. Salesforce にレコードを挿入し、seeAllData=true を使用して、アサーションを実行します。
Answer: C (LEAVE A REPLY)
Testing asynchronous Apex (such as @future methods, Batchable, or Queueable classes) requires a specific mechanism to ensure that background tasks complete before the test verifies the results. In a standard execution, asynchronous jobs are queued and run whenever system resources are available, which means a test's assertion statements might execute before the background job has even started. To address this, Salesforce provides the Test.startTest() and Test.stopTest() methods.78 When t9he code that initiates an asynchronous process is placed between Test.startTest() and Test.stopTest(), the platform behaves di10fferently. All asynchronous calls made within this block are collected and held by the system. As soon as the 11code reaches Test.stopTest(), the test execution pauses, and the system forces all queued asynchronous jobs to ru12n immediately and synchronously within the same thread. Once Test.
stopTest() finishes, the execution continues to the next line. By placing assertions immediately after Test.
stopTest(), the developer is guaranteed that the asynchronous logic has finished its work. This is the only supported way to reliably test the side effects of background processes. Other options, like using System.
runAs() or seeAllData=true, do not affect the timing of asynchronous execution and will likely result in failed assertions.

NEW QUESTION: 12
Universal Containersは、既存の社内カスタムWebアプリケーションとの統合を必要としています。このWebアプリケーションは、JSONペイロードを受け取り、商品画像のサイズを変更し、サイズ変更後の画像をSalesforceに送信します。
この統合を実装するために開発者は何を使うべきでしょうか?
A. コールアウトを許可する@futureメソッドを呼び出すApexトリガー910
B. コールアウトを許可する@futureメソッドを呼び出すフロー1112
C. セッションID1314を含む送信メッセージを持つフロー
D. ウェブアプリケーションを呼び出すプラットフォームイベント1516
Answer: A (LEAVE A REPLY)

Comprehensive and 19Detailed 150 to 250 words of Explanation:20
This integration requiremen21t involves two specific needs: sending a custom JSON payload and handling a response that involves updating data in Salesforce. Outbound Messaging (Option C) is a declarative tool, but it is limited to XML/SOAP protocols and cannot send JSON. Therefore, a custom programmatic solution using Apex is required to construct and send the JSON payload to the external REST service.
Since the integration must be triggered by an event in Salesforce (likely the upload or update of a product record), an Apex trigger is the most direct starting point. However, as noted in previous questions, callouts cannot be performed directly within a trigger's execution context because they would block the database transaction. The developer must use asynchronous processing to handle the callout. An @future(callout=true) method is the standard way to achieve this. The trigger captures the necessary data, passes it to the future method, and the future method then performs the HTTP request to the external application.
Once the external application resizes the image, it can use the Salesforce REST API to send the resized file back to Salesforce. While Platform Events (Option D) are a modern alternative for event-driven architectures, they would still require an asynchronous subscriber (like a trigger or a flow) to actually perform the callout, making the Trigger + Future method combination the most straightforward and traditional answer for this PDII scenario.

NEW QUESTION: 13
Lightning Web コンポーネントがロードされたときにカスタムロジックを実行できる手法はどれですか?
A. <aura:handler> init イベントを使用して関数を呼び出します。
B. connectedCallback() メソッドを使用します。
C. renderedCallback() メソッドを使用します。
D. $A.enqueueAction を呼び出し、呼び出すメソッドを渡します。
Answer: B (LEAVE A REPLY)
In Lightning Web Components (LWC), the component lifecycle is managed by standard Web Component lifecycle hooks. To run logic exactly once when the component is inserted into the Document Object Model (DOM), a developer should use the connectedCallback() method (Option B).
The connectedCallback() is the LWC equivalent of Aura's init event. It is the ideal place to:
* Perform initial data fetching (via imperative Apex).
* Initialize internal state or variables.
* Subscribe to a Lightning Message Service (LMS) channel.
* Establish communication with the parent component.
Option A and D are Aura-specific syntax and are not valid in LWC. Option C (renderedCallback()) runs every time the component finishes a render cycle. Because a component can re-render many times (whenever a reactive property changes), placing

initialization logic in renderedCallback can lead to performance issues or infinite loops if not guarded carefully. Therefore, connectedCallback() is the standard, most efficient way to handle "on load" logic in LWC.

NEW QUESTION: 14
以下のマークアップを参照してください。
HTML
<template>
<lightning-record-form
record-id={recordId}
object-api-name="Account"
layout-type="Full">
</lightning-record-form>
</template>
Lightning Webコンポーネントは、取引先名と、オブジェクトに存在する275個のカスタム項目のうち2つを表示します。カスタム項目は正しく宣言され、値も入力されています。しかし、開発者はコンポーネントのパフォーマンスが遅いという苦情を受けています。開発者はパフォーマンスを改善するために何をすればよいでしょうか？
A. layout-type="Full" を fields={fields} に置き換えます。
B. コンポーネントに density="compact" を追加します。
C. layout-type="Full" を layout-type="Partial" に置き換えます。
D. コンポーネントに cache="true" を追加します。
Answer: (SHOW ANSWER)
The lightning-record-form is a powerful, high-level component that simplifies data entry and display.
However, its performance is heavily influenced by the layout-type attribute. When layout-type="Full" is used, the component fetches and renders every single field defined on the object's "Full" page layout in the Salesforce metadata. In this case, the Account object has 275 fields. Fetching and rendering such a large volume of metadata and data causes a significant performance lag, even if the user only cares about three specific fields.
To improve performance, the developer should switch from a layout-based approach to a field-based approach. By removing the layout-type attribute and adding the fields attribute (Option A), the developer can pass an array of only the specific field API names required (e.g., ['Name', 'CustomField1__c',
'CustomField2__c']). This drastically reduces the amount of data requested from the Lightning Data Service (LDS) and minimizes the DOM elements the browser needs to render. Option C is incorrect because "Partial" is not a valid value for layout-type; the only supported values are "Full" and "Compact". Option B only affects the visual spacing of the fields and does not reduce the data load. Option A is the direct and most effective way to optimize component responsiveness by limiting the data scope.

NEW QUESTION: 15
Apexトリガは、営業担当者が商談を獲得するたびにOrder__cレコードを作成します。最近、このトリガによって2つの注文が作成されています。開発者がこれをトラブルシューティングするための最適な方法は何ですか？
A. すべてのフローを無効にし、フローを 1 つずつ再度有効にして、どのフローがエラーの原因であるかを確認します。
B. すべての営業担当者に対してデバッグ ログを設定し、ログでエラーと例外を監視します。
C. Apex トリガーの Apex テストクラスを実行して、コードに十分なコード カバレッジが残っていることを確認します。
D. コードに system.debug() ステートメントを追加し、開発者コンソールのログを使用してコードをトレースします。
Answer: D (LEAVE A REPLY)
When a trigger unexpectedly executes twice (recursion), it is typically due to the Salesforce Order of Execution. A common cause is a workflow rule, process, or flow updating the same record that initiated the transaction, which re-triggers the original Apex trigger. To identify exactly where this second execution is being initiated, the most effective technique is to use system.debug() statements in conjunction with the Developer Console logs (Option D).
By placing debug statements at the start of the trigger (e.g., System.debug('Trigger Fired');), the developer can examine the execution log to see how many times that line appears. More importantly, the execution log provides a hierarchical "trace" of the entire transaction. The developer can look for entries like FLOW_CREATE_INTERVIEW or WF_RULE_EVAL_BEGIN between the two trigger execution blocks.
This trace reveals exactly which automation (Flow, Workflow, or another Trigger) caused the record to be updated a second time.
Option A is inefficient and disruptive to production environments. Option B is too broad and doesn't provide the internal execution context. Option C checks for code validity but does not diagnose runtime logic issues in a live environment. Tracing the execution through logs is the standard programmatic way to debug recursion and side effects in Salesforce.

NEW QUESTION: 16
開発者は、スマートフォンでは1列、タブレット/デスクトップでは2列で表示するLightning Webコンポーネントを必要としています。開発者はコードにどの部分を追加すればよいでしょうか？
A. <lightning-layout-item> 要素に size="12" medium-device-size="6" を追加します。
B. <lightning-layout-item> 要素に size="6" small-device-size="12" を追加します。
C. <lightning-layout-item>要素にsmall-device-size="12"を追加します。
D. <lightning-layout-item>要素にmedium-device-size="6"を追加します。

Answer: (SHOW ANSWER)

Comprehensive and Detailed Explanation:

The lightning-layout grid system uses a 12-column model. To achieve a responsive design, developers use attributes that target different screen breakpoints.

* size: Defines the default size (usually targeting the smallest devices if no other attributes are present).
* medium-device-size: Targets tablets and small desktops.
* large-device-size: Targets large monitors.

To show one column on a phone, an item must occupy all 12 columns (size="12"). To show two columns on a tablet/desktop, each item must occupy 6 columns (medium-device-size="6").

Option A correctly implements this. While small-device-size exists, the size attribute itself typically serves as the baseline for small devices. If you set size="12" and medium-device-size="6", the component will "stack" on phones and sit side-by-side on anything larger.

NEW QUESTION: 17

開発者は、取引先レコードページ用のLightning Webコンポーネントを作成しました。このコンポーネントは、取引先から最近連絡を取った取引先責任者5名を表示します。Apexメソッドは取引先責任者のリストを返し、このリストはコンポーネント内のプロパティに紐付けられます。

Java

```
01:
02: public class ContactFetcher {
03:
04: static List<Contact> getRecentContacts(Id accountId) {
05: List<Contact> contacts = getFiveMostRecent(accountId);
06: return contacts;
07: }
08: private static List<Contact> getFiveMostRecent(Id accountId) {
10: //...implementation...
11: }
12: }
```

Apex メソッドを接続できるようにするには、上記の cod39e のどの 2 行を変更する必要がありますか?
A. 行 04 に public を追加します。
B. 行 08 に @AuraEnabled(cacheable=true) を追加します。
C. 行 09 から private を削除します。
D. 行 03 に @AuraEnabled(cacheable=true) を追加します。
Answer: A,D (LEAVE A REPLY)
To expose an Apex method to a Lightning Web Component (LWC) and specifically use the @wire service, the method must meet two strict criteria: it must be public or global, and it must be annotated with
@AuraEnabled(cacheable=true).
In the provided snippet, line 04 defines the method as static but lacks an access modifier. In Apex, the default access modifier is private, which prevents the LWC framework from accessing the method. Changing line 04 to public static List<Contact> (Option A) makes the method visible to the component.
Additionally, the @wire service requires the method to be "cacheable" to optimize performance by storing results on the client-side. This is achieved by adding the @AuraEnabled(cacheable=true) annotation. Placing this on line 03 (Option D) ensures the method is registered with the Lightning Data Service as an invokable, cacheable action.
Modifying line 08 (Option B) is incorrect because that is a private helper method that the component does not call directly. By making the primary method public and cacheable, the LWC can successfully bind its property to the data returned by the Apex controller.

NEW QUESTION: 18
Java
@isTest
static void testUpdateSuccess() {
Account acet = new Account(Name = 'test');
insert acet;
// Add code here
extension.inputValue = 'test';
PageReference pageRef = extension.update();
System.assertNotEquals(null, pageRef);
}
テスト用のコントローラー拡張を作成するには、上記の単体テストのセットアップの指定された場所に何を追加する必要がありますか?
A. AccountControllerExt 拡張機能 = 新しい AccountControllerExt(acet);
B. ApexPages.StandardController sc = 新しい ApexPages.StandardController(acet);
AccountControllerExt extension = 新しい AccountControllerExt(sc);

C. ApexPages.StandardController sc = 新しい ApexPages.StandardController(acet.Id);
AccountControllerExt extension = 新しい AccountControllerExt(sc);
D. AccountControllerExt 拡張機能 = 新しい AccountControllerExt(acet.Id);
Answer: B (LEAVE A REPLY)
123
To properly test a Visualforce Controller Extension in Apex, the developer must replicate the instantiation process used by the Lightning Platform at runtime. A4 controller extension is a class whose constructor is specifically defined to accept a single parameter of the type ApexPages.StandardController. 5The standard controller acts as the bridge b6etween the custom extension logic and the underlying Salesforce record (in this case, an Account).
In the provided test method, a record acet has been inserted into the database. To create the extension object, the developer must first instantiate the StandardController by passing the acet SObject record into its constructor. Once the standard controller instance (e.g., sc) is created, it is then passed into the constructor of the AccountControllerExt class. Option B is the correct implementation of this two-step pattern. Option C is incorrect because the StandardController constructor requires a full SObject record, not just an ID string.
Options A and D are incorrect because extension classes do not support constructors that take an SObject or an ID directly; they specifically require the StandardController object to provide the necessary context for standard actions and fields. Following the correct pattern ensures the test accurately simulates the page execution context.

NEW QUESTION: 19
ある企業は、関連オブジェクトを通じて収益を追跡したいと考えています。約10万件の商談について、複雑なロジックに基づいて収益レコードを作成し、一度だけデータをシードする必要があります。これを自動化する最適な方法は何でしょうか？
A. Database.executeBatch() を使用して、Database.Batchable クラスを呼び出します。
B. System.enqueueJob() を使用して Queueable クラスを呼び出します。
C. Database.executeBatch() を使用して Queueable クラスを呼び出します。
D. System.scheduleJob() を使用して、Database.Scheduleable クラスをスケジュールします。
Answer: A (LEAVE A REPLY)
For high-volume data processing (such as seeding 100,000 records) involving complex logic, Batch Apex (Option A) is the standard and most robust solution. Batch Apex is designed to handle up to 50 million records by breaking the total set into smaller, manageable chunks (defaulting to 200 records per batch).
Each batch execution gets its own set of governor limits. This is crucial for "complex logic," as it prevents the transaction from hitting CPU time or heap size limits that would occur if one tried to process all 100,000 records in a single synchronous transaction. Batch Apex also provides built-in state management and error handling (via Database.RaisesPlatformEvents or the finish method).

Option B (Queueable) is better suited for smaller, chained tasks; while it can handle some volume, it is not optimized for 100,000 records in the same way the QueryLocator in Batch Apex is. Option C is a syntactical impossibility as executeBatch only accepts Batchable classes. Option D (Schedulable) is used for timing, but the actual heavy lifting for 100,000 records would still need to be handed off to a Batch class to avoid timeout errors. Therefore, Database.Batchable is the optimal tool for large-scale data seeding.

NEW QUESTION: 20
ある企業は、多数のメソッドを含むテストクラスのユニットテストで、前提条件となる参照データ用のレコードを多数作成する処理が遅いことに気づきました。開発者はこの問題を解決するために何ができるでしょうか？
A. テストを実行するときに、トリガー、フロー、検証をオフにします。
B. 前提条件となる参照データの設定をテスト クラスのコンストラクターに移動します。
C. 前提条件となる参照データのセットアップをテスト クラスの @testSetup メソッドに移動します。
D. 前提条件となる参照データの設定を TestDataFactory に移動し、各テスト メソッドからそれを呼び出します。
Answer: (SHOW ANSWER)
The most effective way to optimize test performance when multiple test methods require the same set of records is to use the @testSetup annotation (Option C).
When a method is marked with @testSetup, Salesforce executes it exactly once before any test methods in the class run. The platform then creates a "database snapshot" of those records. For every individual test method in the class, Salesforce simply rolls back the database to that snapshot rather than re-executing the DML operations required to create the records. This dramatically reduces the total number of DML statements and CPU time consumed during the test run.
In contrast, calling a TestDataFactory (Option D) from within every test method results in the same records being inserted and deleted repeatedly for every test case. If a class has 50 test methods, the records are created
50 times. With @testSetup, they are created once. Option A is not possible (you cannot programmatically disable these during a test run), and Option B is incorrect as test classes do not use constructors in this manner. Using @testSetup is the platform-recommended pattern for efficient, high-performance unit testing.

NEW QUESTION: 21
どのコードスニペットが最もメモリ効率の高い方法でレコードを処理し、ガバナ制限を回避しますか？
「Apex ヒープ サイズが大きすぎます」?
A. Java

Map<Id, Opportunity> opportunities = new Map<Id, Opportunity>([SELECT Id, Amount from Opportunity]); for(Id oppId: opportunities.keySet()){
// perform operation here
}
B. Java
for(Opportunity opp: [SELECT Id, Amount from Opportunity]){
// perform operation here
}
C. Java
List<Opportunity> opportunities = Database.query('SELECT Id, Amount from Opportunity');
for(Opportunity opp: opportunities){
// perform operation here
}
D. Java
List<Opportunity> opportunities = [SELECT Id, Amount from Opportunity]; for(Opportunity opp: opportunities){
// perform operation here
}
Answer: B (LEAVE A REPLY)
When processing large data volumes in Apex, the "Heap Size" limit (6MB for synchronous, 12MB for asynchronous) is a common bottleneck. This limit tracks the amount of memory consumed by objects and variables currently in scope.
Option B utilizes the SOQL For Loop pattern. This is the most memory-efficient approach because it processes records in "chunks" of 200 at a time using internal query locators. Instead of loading the entire result set (potentially thousands of records) into a single collection in memory, the SOQL For Loop iterates through the records without ever holding the full set in the heap. This prevents the "Apex heap size too large" error when dealing with high-volume queries.
In contrast, Options A, C, and D all involve assigning the entire query result to a collection (a Map or List) before the loop begins. This immediately consumes heap space proportional to the number of records and fields retrieved. If the query returns a large number of records, these collections will quickly exceed the heap limit before the first line of the loop even executes. Therefore, for bulk data processing where memory efficiency is the priority, the inline SOQL For Loop (Option B) is the standard best practice.

NEW QUESTION: 22
次の包含階層があるとします。
HTML
<template>
<c-my-child-components></c-my-child-components>
</template>

プロパティが my-child-component 内で定義されている場合、「passthrough」という名前のプロパティの新しい値を my-parent-component に伝達する正しい方法は何ですか?
A. let cEvent = new CustomEvent('passthrough', { detail: 'this.passthrough' });
this.dispatchEvent(cEvent);
B. let cEvent = new CustomEvent('passthrough'); this.dispatchEvent(cEvent);
C. let cEvent = new CustomEvent('passthrough', { detail: this.passthrough });
this.dispatchEvent(cEvent);
D. let cEvent = new CustomEvent($passthrough); this.dispatchEvent(cEvent);
Answer: (SHOW ANSWER)
In Lightning Web Components (LWC), data flows "down" via properties and "up" via events. When a child component needs to communicate a change in a property or state to its parent, it must dispatch a CustomEvent. To pass specific data-such as the new value of the passthrough property-along with the event, the developer must use the detail property within the event initialization object.
Option C is the correct syntax. It creates a new CustomEvent named 'passthrough' and assigns the current value of the component's property (this.passthrough) to the detail key. The parent component can then listen for this event (using onpassthrough={handleEvent}) and access the value via event.detail. Option A is incorrect because it wraps the variable in quotes, passing the literal string "this.passthrough" instead of the actual data. Option B creates an event but fails to include the data payload, meaning the parent would know an event occurred but wouldn't receive the new value. Option D uses incorrect syntax for event naming and variable referencing. Using the standard CustomEvent constructor with the detail property is the platform- standard way to ensure robust, typed data communication between component layers in the Shadow DOM.

NEW QUESTION: 23
開発者が Lightning Web コンポーネントの Jest テストを記述する必要がある 3 つの理由は何ですか?
A. 基本的なユーザーインタラクションをテストする
B. コンポーネントの非公開プロパティをテストする
C. 複数のコンポーネントがどのように連携して動作するかをテストする
D. コンポーネントのDOM出力を検証する
E. イベントが期待通りに発生することを確認する
Answer: A,D,E (LEAVE A REPLY)
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: 24
開発者が外部Webサービスへの呼び出しを必要とするコードを記述しています。非同期メソッドで呼び出しを行う必要があるシナリオはどれですか？
A. コールアウトが完了するまでに 60 秒以上かかる可能性があります。
B. コールアウトは Apex トリガーで実行されます。
C. 1 回のトランザクションで 10 回を超えるコールアウトが行われます。
D. コールアウトは REST API を使用して行われます。
Answer: B (LEAVE A REPLY)
In Salesforce, a critical restriction is that synchronous callouts cannot be made from within an Apex trigger.
This architectural limitation is in place because triggers execute as part of a database transaction. Allowing a synchronous callout-which waits for a response from an external system-would keep the database connection and transaction locks open for an indeterminate amount of time. This could lead to severe performance bottlenecks and row-locking issues across the entire organization. Therefore, if a developer needs to initiate an external integration based on a record change (trigger), the logic must be handed off to an asynchronous process, such as a @future(callout=true) method, a Queueable class, or a Platform Event.
Regarding the other options: A callout that takes longer than the maximum timeout (120 seconds) will fail regardless of whether it is synchronous or asynchronous, though async is often preferred for longer-running tasks to avoid blocking the user UI. Salesforce allows up to 100 callouts in a single transaction, so making more than 10 (Option C) does not inherently force an asynchronous approach unless the 100-callout limit is exceeded. Finally, the use of the REST API (Option D) is simply a protocol choice and does not dictate the execution context. Consequently, the presence of an Apex trigger is the only scenario listed that programmatically mandates the use of asynchronous execution to perform a callout.

NEW QUESTION: 25
開発者がテストクラス内から組織データにアクセスしようとしています。テストクラスに (seeAllData=true) アノテーションが必要な sObject 型はどれですか？
A. レコードタイプ
B. レポート
C. ユーザー
D. プロフィール
Answer: (SHOW ANSWER)
Comprehensive and Detailed Explanation:
Salesforce enforces test isolation, meaning unit tests do not have access to the data in the organization by default. However, some objects are considered "metadata-like" or "setup objects" and are always visible to tests without special annotations. These include Users (Option C), Profiles (Option D), and RecordTypes (Option A).
Conversely, some objects are considered "data" but cannot be easily created within a test method due to their complexity or nature. Reports (Option B), along with Pricebooks and Folders, fall into a category where access to existing org data is often required because the platform does not support DML operations to create them in a test context. To query for an existing Report record in a unit test, the test class or method must be annotated with @isTest(seeAllData=true).
Note: It is a best practice to avoid seeAllData=true whenever possible to ensure tests are deterministic and independent of the environment, but it remains a requirement for Report-related logic.

NEW QUESTION: 26
次のコードがあるとします。
Java
for (Contact c : [SELECT Id, LastName FROM Contact WHERE CreatedDate = TODAY])
{
Account a = [SELECT Id, Name FROM Account WHERE CreatedDate = TODAY LIMIT 5];
c.AccountId = a.Id; update c;
}
今日、連絡先が 10 件、アカウントが 5 件作成されたと仮定すると、どのような結果が期待されますか?
A. System.QueryException: リストにアカウントの割り当ての行が複数あります
B. System.LimitException: 連絡先に対するSOQLクエリが多すぎます
C. System.LimitException: アカウントのSOQLクエリが多すぎます
D. System.QueryException: 連絡先に DML ステートメント エラーが多すぎます
Answer: A (LEAVE A REPLY)
2

The primary issue in this code snippet is a violation of basic variable assignment rules in Apex when dealing with SOQL results. In the line Account a = [SELECT Id, Name FROM Account ...], the code attempts to assign the result of a query directly 3to a single SObject variable (Account a).
In Apex, a SOQL query assigned to a single SObject variable must return exactly one record. If the query returns zero records, it throws a System.QueryException: List has no rows for assignment. If the query returns more than one record, it throws a System.QueryException: List has more than one row for assignment (Option A). Since the prompt explicitly states that five Accounts were created today and the query is not filtered to a single unique ID, the query will return five records. Even though there is a LIMIT 5 clause, that only caps the results at five; it does not ensure only one is returned. To fix this, the result should be assigned to a List<Account> or the query should be filtered to return exactly one row.
While the code also violates the "SOQL in a loop" best practice, it would not hit the LimitException (Option C) in this specific case because the loop only runs 10 times (10 contacts), and the limit is 100 queries. The runtime assignment error occurs before any governor limits are breached.

NEW QUESTION: 27
開発者は、関連する商談が高価値と判断された際に取引先評価を更新する商談トリガを作成しました。現在、高価値と判断される商談の基準は、金額が100万ドル以上であることとなっています。ただし、この基準値は時間の経過とともに変更される可能性があります。Lightning Webコンポーネントにも高価値商談を表示するという新たな要件があります。これらのビジネス要件を満たし、高価値商談を取得するビジネスロジックが複数の場所で重複して使用されないようにするために、開発者はどの2つのアクションを実行する必要がありますか？2021
A. 効率を上げるために、ビジネスロジックコードをトリガー内に残します。23
B. カスタムメタデータを使用して、高額の金額を保持します。24
C. Lightning web25 コンポーネントからトリガーを呼び出します。
D. 高価値の機会を取得するヘルパー クラスを作成します。
Answer: B,D (LEAVE A REPLY)
To build a maintainable and scalable solution, a developer must separate business logic from trigger execution and avoid hard-coding threshold values.
First, the $1,000,000 threshold should be stored in Custom Metadata (Option B). Custom Metadata is superior to hard-coding or even Custom Settings in this context because the records are deployable and can be queried efficiently without counting against standard SOQL limits. If the business decides to lower the "high value" threshold to $500,000, an administrator can simply update the Custom Metadata record without requiring any code changes or redeployments.

Second, the logic to identify these opportunities should be moved to a Helper Class (Option D). This follows the "Separation of Concerns" principle. Instead of writing the SOQL query and logic inside the trigger, the trigger calls a method in the helper class. Similarly, the Lightning Web Component's Apex controller can call the exact same method in the helper class. This ensures that the definition of a "high value opportunity" is managed in one single place in the codebase. If the logic becomes more complex (e.g., adding Industry filters), the developer only needs to update the helper class once to satisfy both the trigger and the UI component. Option A leads to "spaghetti code," and Option C is impossible as triggers cannot be called directly from JavaScript or other Apex classes.

NEW QUESTION: 28
開発者は、様々なデバイスでレスポンシブな Lightning Web コンポーネントを作成するという課題を抱えています。この目標を達成するために役立つ 2 つのコンポーネントはどれですか。
A. Lightning入力場所
B. Lightningレイアウト
C. ライトニングナビゲーション
D. Lightningレイアウト項目
Answer: (SHOW ANSWER)
To build a responsive user interface in Lightning Web Components (LWC), developers rely on the layout system provided by the Lightning Design System (SLDS). This system is primarily implemented through two tightly coupled components: lightning-layout and lightning-layout-item.
lightning-layout acts as a flexible container (a flexbox wrapper). It allows developers to arrange child components in a row or column and manage horizontal and vertical alignment. It provides the overall structure for the grid. lightning-layout-item is the child component that resides within a lightning-layout. It is responsible for defining the actual proportions of the content across different breakpoints (small, medium, and large devices) using attributes like size, small-device-size, medium-device-size, and large-device-size. Together, these two components allow a developer to specify exactly how much space a piece of content should occupy depending on the user's screen size. For example, a sidebar might occupy 100% of the width on a phone but only 25% on a desktop. lightning-input-location is a specific input field for geolocation data, and lightning-navigation is a service used for page routing; neither is involved in the visual responsiveness or structural layout of the component. Therefore, options B and D are the essential building blocks for responsive design.

NEW QUESTION: 29
Universal Containersは、Convention_Attendee__cに非公開共有モデルを実装しています。参照フィールドEvent_Reviewer__cが作成されています。経営陣は、イベントレビュー担当

者に、担当するすべてのレコードへの読み取り/書き込みアクセス権を自動的に付与したいと考えています。最適なアプローチは何でしょうか？
A. Convention Attendee カスタム オブジェクトに after insert トリガーを作成し、Apex Sharing Reasons と Apex Managed Sharing を使用します。
B. Convention Attendee カスタム オブジェクトに before insert トリガーを作成し、Apex Sharing Reasons と Apex Managed Sharing を使用します。
C. コンベンション参加者カスタム オブジェクトに条件に基づく共有ルールを作成し、イベント レビュー担当者とレコードを共有します。
D. コンベンション参加者カスタム オブジェクトに条件に基づく共有ルールを作成し、イベント レビュー担当者のグループとレコードを共有します。
Answer: (SHOW ANSWER)
In a private sharing model, access is restricted to the owner and those above them in the hierarchy. When a specific user is identified in a lookup field (like Event_Reviewer__c), standard sharing rules cannot dynamically grant access to that specific individual because sharing rules only target Roles, Public Groups, or Territories.1 The solution is Apex Managed Sharing (Option A). By using an after insert (and likely after update) trigger, the developer can pro2grammatically create records in the object's "Share" table (e.g., Convention_Attendee__Share). Each share record specifies the UserOrGroupId (from the lookup field), the AccessLevel ('Edit' for Read/Write), and a RowCause (Apex Sharing Reason).
The "After" trigger is required because the ID of the Convention_Attendee__c record must exist before a Share record can be associated with it. "Before" triggers (Option B) are used for field updates on the record itself, not for creating related sharing records. Options C and D are not viable because they cannot dynamically reference a User ID stored within a field on the record itself. Apex Managed Sharing provides the most granular and automated way to handle this dynamic security requirement.

NEW QUESTION: 30
選択的 SOQL クエリであり、200,000 件のアカウント レコードの大規模なデータ セットに使用できるクエリはどれですか。
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: (SHOW ANSWER)
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.

NEW QUESTION: 31
Ajax 動作を開始して部分的なページ更新を実行するために使用できる 3 つの Visualforce コンポーネントはどれですか。
A. <apex:form>
B. <apex:actionStatus>
C. <apex:コマンドボタン>
D. <apex:actionSupport>
E. <apex:commandLink>
Answer: C,D,E (LEAVE A REPLY)
Comprehensive and Detailed Explanation:
In Visualforce, partial page updates (AJAX) are achieved using the reRender attribute. This attribute allows a component to refresh only a specific part of the page identified by an ID, rather than performing a full browser reload.
* <apex:commandButton> (Option C) and <apex:commandLink> (Option E) are standard action components. When their reRender attribute is populated, they perform an asynchronous postback.
* <apex:actionSupport> (Option D) is an auxiliary component that adds AJAX functionality to other components that do not natively support it. For example, you can place an actionSupport inside an inputText to trigger a partial refresh on the onchange event.
Option A (<apex:form>) is a container and does not initiate AJAX behavior itself. Option B (<apex:
actionStatus>) is used to display the status of an AJAX request (e.g., a loading spinner) but does not initiate the request.

NEW QUESTION: 32
開発者は、フィルター機能を備えた Lightning Web コンポーネントにカスタム データ テーブルを実装しました。
しかしながら、フィルターを変更すると読み込み時間が長くなるという件で、ユーザーからサポートチケットが提出されています。このコンポーネントは、選択されたフィルターに基づいてレコードをクエリするために呼び出されるApexメソッドを使用しています。開発者はコンポーネントのパフォーマンスを改善するために何をすべきでしょうか？
A. SOSL を使用して、フィルターの変更時にレコードを照会します。1
B. Apex メソッドで setStorable() を使用して、応答をクライアント側キャッシュに保存します。2
C. コンポーネントが作成されると、すべてのレコードがリストに返され、JavaScript で配列がフィルタリングされます。3
D. カスタム インデックスを使用して a4 選択的 SOQL クエリを使用します。
Answer: D (LEAVE A REPLY)
When a data table experiences slow performance during filtering, the root cause is often "non-selective" queries that result in full table scans. In Salesforce, a query is selective when it uses an indexed field in the WHERE clause and filters down to a small percentage of the total records (typically 10% of the first million).
Option D is the correct strategy. By ensuring that the filter fields (such as a Status or Category) are either standard indexed fields or custom fields marked as "External ID" or "Unique" (which creates a custom index), the Query Optimizer can quickly locate the relevant rows. This significantly reduces the database processing time and the "Time to First Byte" (TTFB) returned to the Lightning component.
Option C might seem attractive for small datasets, but it is not scalable; if the org has 50,000 records, returning all of them to the client will crash the browser's heap memory.
Option B (setStorable) is an Aura- specific concept and is replaced by @AuraEnabled(cacheable=true) in LWC, but even caching doesn't help if the initial query itself is slow. Option A (SOSL) is for text searching across multiple objects and is generally less efficient than a targeted SOQL query for specific field filtering. Using a selective SOQL query ensures the component remains fast as data volume grows.

NEW QUESTION: 33

開発者は、システムに入力されたメールアドレスとカスタムオブジェクト「Survey_Response__c」が、ブロック対象ドメインのリストに含まれていないことを確認する必要があります。ブロック対象ドメインのリストは、ユーザーによるメンテナンスを容易にするため、カスタムオブジェクトに保存されています。Survey_Response__c オブジェクトへの入力は、カスタムVisualforceページから行います。これを実装する最適な方法は何でしょうか？

A. Contact および Survey_Response__c オブジェクトの検証ルールにロジックを実装します。

B. 連絡先の Apex トリガーおよびカスタム Visualforce ページ コントローラから呼び出されるヘルパー クラスにロジックを実装します。

C. 連絡先の Apex トリガーにロジックを実装し、カスタム Visualforce ページ コントローラ内にもロジックを実装します。

Answer: (SHOW ANSWER)

The requirement involves enforcing a business rule across two different objects (Contact and Survey_Response__c) and through different entry points: standard UI/Triggers for Contacts and a custom Visualforce page for Survey Responses. Implementing the logic separately in a trigger and a controller (Option C) is a poor architectural choice because it duplicates code, leading to increased maintenance effort and a higher risk of inconsistency if the logic needs to change in the future. Validation rules (Option A) are also unsuitable here because they cannot natively perform a dynamic lookup against a list of records in a separate "Blocked Domains" object without complex and unscalable workarounds.

The optimal approach is to centralize the validation logic within a single "Helper" or "Service" class. This follows the DRY (Don't Repeat Yourself) principle of software development. By creating a shared method in a helper class, the developer can ensure that both the Contact trigger and the Visualforce controller use the exact same logic to verify email domains. This centralized method queries the custom object containing the blocked domains and returns a validation result. This architecture simplifies testing, ensures uniform enforcement of business rules across the entire platform, and allows administrators to update the blocked domains list without requiring any further code modifications.

NEW QUESTION: 34

開発者は、商談トリガから呼び出される複雑なコミッション計算エンジンをApexで構築しています。QA中に、計算結果に誤りがあると報告されました。開発者は、開発者サンドボックスに代表的なテストデータと合格したテストメソッドを保有しています。コードを実行し、重要な行で一時停止して様々なApex変数の値を視覚的に確認するために、開発者が使用できるツールまたはテクニックを3つ挙げてください。

A. Apex インタラクティブデバッガー

B. Apex リプレイデバッガー

C. ワークベンチ

D. 開発者コンソール
E. ブレークポイント
Answer: A,B,E (LEAVE A REPLY)
To perform traditional "step-through" debugging in Salesforce-where you can pause execution and inspect the state of variables in real-time or near-real-time-developers use the Salesforce extensions for VS Code.
The Apex Replay Debugger (Option B) is a free tool that allows developers to "replay" a transaction using a debug log. By setting Breakpoints (Option E) in their code and then executing the logic (or running a test), the developer can generate a log file. The Replay Debugger then parses this log, allowing the developer to step through the code line-by-line in VS Code as if it were happening live1.2 The Apex Interactive Debugger (Option A) is a specia3lized tool (typically requiring a paid add-on for sandboxes) that allows for live, real-time debugging on a sandbox or scratch org. Unlike the Replay Debugger, it pauses the actual execution on the Salesforce server, giving the developer a live view of the environment.
The Developer Console (Option D) and Workbench (Option C) provide log viewing and query capabilities, but they do not support the ability to "pause" execution at specific lines for visual inspection in the manner of a traditional IDE debugger. Therefore, the combination of a debugger tool and the use of breakpoints is the standard requirement for this debugging pattern.

NEW QUESTION: 35
Visualforce リモート オブジェクトと比較した JavaScript リモート処理の利点は何ですか?
A. 複雑なサーバー側アプリケーションロジックをサポートします
B. Apexコードは必要ありません
C. JavaScriptコードは必要ありません
D. 指定された再レンダリングターゲットを許可します
Answer: (SHOW ANSWER)
Visualforce provides two primary ways to perform AJAX-style operations: JavaScript Remoting and Remote Objects. The primary benefit of JavaScript Remoting (Option A) is its ability to execute complex server- side logic.
JavaScript Remoting works by calling an Apex method annotated with @RemoteAction. Because this is a standard Apex method, it can perform advanced calculations, complex SOQL queries with multi-object joins, DML operations across various objects, and even call out to external web services.
In contrast, Remote Objects are purely declarative. They allow you to perform basic CRUD operations (Create, Read, Update, Delete) on a single object directly from JavaScript without writing any Apex. While Remote Objects are easier to set up for simple data entry, they cannot handle "business logic" (like calculating a discount based on customer history) during the data retrieval process.

Option B and C describe Remote Objects, not Remoting. Option D is a feature of standard Visualforce tags like <apex:commandButton>, but not a specific differentiator for Remoting. Therefore, when your application needs "brains" on the server, JavaScript Remoting is the superior choice.

NEW QUESTION: 36
開発チームは、カスタムインターフェースの一部として、様々な新しいLightning Webコンポーネントを作成しました。各コンポーネントは、トーストメッセージを使用してエラーを処理します。受け入れテスト中に、コンポーネントの読み込み中にエラーが発生した際に表示されるトーストメッセージの長い連鎖について、ユーザーから苦情が寄せられました。ユーザーエクスペリエンスを向上させるために、開発者はどの2つの手法を実装すべきでしょうか？
A. Lightning Web コンポーネントを使用して、すべてのエラーを集約して表示します。
B. エラーメッセージを表示するには、window.alert() メソッドを使用します。23
C. <template>タグを使用して、インプレースエラーメッセージを表示します。24
D. 各コンポーネントのパブリックプロパティを使用してエラーメッセージを表示します。25
Answer: A,C (LEAVE A REPLY)
When multiple components on a single page all fail simultaneously (for example, due to a shared service failure), individual toast messages stack up, creating a poor user experience. To resolve this, developers should shift away from "ephemeral" notifications like toasts toward more structured error handling.
Option A involves creating a dedicated error-handling component. This component can act as a listener or a central repository for errors across the page. Instead of each component firing its own toast, they can communicate their error state to this central component, which aggregates the messages into a single, clean display. This reduces visual clutter and allows the user to read all errors in one place.
Option C refers to "in-place" error handling. Using conditional rendering (the lwc:if or template if:true directives), a component can hide its normal UI and display an error message exactly where the component would have been. This provides immediate context to the user about which specific part of the page failed without interrupting the overall flow with pop-ups.
Option B (window.alert) is considered a legacy practice that blocks the browser thread and is generally avoided in modern web development. Option D (public properties) is a mechanism for component communication but doesn't solve the display problem itself. Combining A and C provides a modern, professional, and less intrusive error-handling strategy.

NEW QUESTION: 37

開発者が外部Webサービスへの呼び出しを必要とするコードを記述しています。非同期メソッドで呼び出しを行う必要があるシナリオはどれですか？
A. コールアウトが完了するまでに 60 秒以上かかる可能性があります。
B. コールアウトは REST API を使用して行われます。
C. コールアウトは Apex トリガーで実行されます。
D. 1 回のトランザクションで 10 回を超えるコールアウトが行われます。
Answer: (SHOW ANSWER)
In Salesforce, a callout cannot be made directly from an Apex trigger (Option C). This is a fundamental architectural restriction of the platform. Triggers execute as part of a database transaction. If a trigger were allowed to make a synchronous callout, the database transaction-along with all its associated row locks- would remain open while waiting for a response from the external system. This could lead to severe performance bottlenecks and transaction timeouts.
To perform a callout based on a trigger event, the developer must move the callout logic into an asynchronous context, such as a @future(callout=true) method, a Queueable class, or by publishing a Platform Event.
Option A is incorrect because while asynchronous methods allow for higher CPU time, they do not extend the maximum individual callout timeout (120 seconds). Option B is incorrect because REST is simply a protocol and does not dictate synchronicity. Option D is incorrect because the limit for callouts in a single transaction is 100, so 10 callouts would still be permissible in a synchronous context (assuming they aren't in a trigger). Consequently, the presence of a trigger is the only factor listed that programmatically mandates an asynchronous approach.

NEW QUESTION: 38
開発者は、次のように Queueable インターフェースを実装するクラスを作成しました。
ジャワ
共有なしのパブリッククラス OrderQueueableJob は Queueable を実装します {
パブリック void 実行(QueueableContext コンテキスト) {
// 実装ロジック
System.enqueueJob(新しい FollowUpJob());
}
}
デプロイメントプロセスの一環として、開発者は対応するテストクラスを作成する必要があります。テストクラスを正常に実行するために、開発者が実行すべき2つのアクションはどれですか？1
A. Queueable ジョブが一括モードで実行できるようにするには、seeAllData=true を実装します。2
B. テスト実行中にジョブの連鎖を防ぐには、Te3st.isRunningTest() を実装します。

C. テスト クラスの実行ユーザーが、少なくとも Order オブジェクトに対する 「すべて表示」権限を持っていることを確認します。
D. System.enqueueJob(new OrderQueueableJob()) を Test.startTest と Test.stopTest() で囲みます。
Answer: (SHOW ANSWER)
Testing asynchronous Apex, such as the Queueable interface, requires specific handling to ensure the job actually executes during the test run and adheres to platform limits.
First, Salesforce enforces a strict limit on chaining Queueable jobs during unit tests. Specifically, you cannot chain jobs (enqueue a job from another job) within a test. Since the OrderQueueableJob attempts to enqueue FollowUpJob, the test will throw an error. To resolve this, the developer should use Test.isRunningTest() (Option B) within the execute method to conditionally bypass the System.enqueueJob call during test runs.
Second, asynchronous jobs are queued by the system and do not run immediately. To force the execution of a Queueable job so that results can be asserted, the System.enqueueJob call must be wrapped within Test.
startTest() and Test.stopTest() (Option D). When Test.stopTest() is called, the execution pauses until all queued asynchronous jobs in that block finish running.
Option A is incorrect because seeAllData does not affect asynchronous processing modes.
Option C is incorrect because the class is defined as without sharing, meaning it bypasses sharing rules regardless of the running user's permissions. By combining recursion/chaining guards and the startTest/stopTest block, the developer can reliably test the logic within the execute method.

NEW QUESTION: 39
開発者は、高価値商談にフラグを立てるトリガーにビジネスロジックを組み込んでいます。Lightning Webコンポーネントにも高価値商談を表示するという新しい要件があります。これらのビジネス要件を満たし、高価値商談を識別するビジネスロジックが複数の場所で重複して使用されないようにするために、開発者はどの2つの手順を実行する必要がありますか？
A. Lightning Web コンポーネントからトリガーを呼び出します。
B. 効率性を高めるために、ビジネス ロジック コードをトリガー内に残します。
C. 高価値の機会を取得するヘルパー クラスを作成します。
D. カスタム メタデータを使用して、高額の金額を保持します。
Answer: (SHOW ANSWER)
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.

NEW QUESTION: 40
Lightning レコードページのボタンをクリックしてモーダルダイアログに表示されるようにするには、Aura コンポーネントでどのインターフェースを実装する必要がありますか?
A. lightning:editAction
B. lightning:クイックアクション
C. 強制:lightningEditAction
D. 強制:lightningクイックアクション
Answer: D (LEAVE A REPLY)
In the Aura Component framework, "Quick Actions" are used to extend the functionality of the Salesforce UI, allowing developers to surface custom components in modal pop-ups from the "Buttons, Links, and Actions" section of an object.
To make an Aura component available as a Quick Action, it must implement the force: lightningQuickAction (Option D) interface.2 When this interface is added to the <aura:component> tag, the component becomes selectable when creating a new "Action" for an object. When a user clicks the corresponding button on the record page, Salesforce automatically wraps the component in a standard modal dialog box.
If the requirement specifically calls for the modal to occupy more screen real estate, the developer can also use force:lightningQuickActionWithoutHeader, which removes the standard "Save" and "Cancel" buttons and the header, providing a blank canvas within the modal.3 Option B is incorrect as the prefix must be force:, which refers to the library of interfaces provided by the Lightning Experience container. Option A and C are not standard interfaces for this purpose. Implementing force:lightningQuickAction is the standard way to bridge custom Aura logic with the native Salesforce record page button functionality.

NEW QUESTION: 41
ユニバーサル・チャリティーズ（UC）は、Salesforceを使用して、個人および法人からクレジットカードによる電子寄付を徴収しています。カスタマーサービス担当者がクレジットカード情報を入力すると、寄付を処理するために、その情報はサードパーティの決済代行

業者に送信する必要があります。UCは、個人向けと法人向けにそれぞれ異なる決済代行業者を使用しています。システム管理者が導入後に必要に応じて設定を変更できるよう、開発者は各決済代行業者の設定をどのような方法で保存すればよいでしょうか？
A. 階層のカスタム設定
B. カスタムラベル
C. カスタムメタデータ
D. カスタム設定をリストする
Answer: C (LEAVE A REPLY)
For storing application configurations and integration settings that need to be easily modified by administrators and deployed across environments, Custom Metadata Types are the preferred solution. Unlike Custom Settings (Options A and D), records within a Custom Metadata Type are considered metadata rather than data. This is a critical distinction for the development lifecycle because these records can be included in Change Sets or deployment packages. This eliminates the manual overhead and risk associated with re- creating configuration records in production after a sandbox deployment.
In this scenario, UC needs to manage settings for two different payment processors. A Custom Metadata Type can be created with fields for API endpoints, merchant IDs, and security keys. An administrator can then create and edit the specific records for the "Individual" and "Corporate" processors directly in the Setup menu.
Furthermore, Custom Metadata queries are efficient and do not count against standard SOQL governor limits in many contexts. While Custom Labels (Option B) are useful for translating text, they are not intended for complex, structured configuration data. Hierarchy Custom Settings are designed for user-specific overrides, which is not applicable here. Therefore, Custom Metadata provides the most robust, deployable, and administrator-friendly way to manage external service configurations.

NEW QUESTION: 42
以下のコード スニペットを参照してください。
ジャワ
public static void updateCreditMemo(String customerId, Decimal newAmount)
{ List<Credit_Memo__c> toUpdate = new List<Credit_Memo__c>(); for(Credit_Memo__c creditMemo : [Select Id, Name, Amount__c FROM Credit_Memo__c WHERE Customer_Id__c = :customerId LIMIT 50]) { creditMemo.Amount__c = newAmount; toUpdate.add(creditMemo);
}
データベースを更新します(toUpdate、false)。
}
Salesforce環境にCredit_Memo__cというカスタムオブジェクトが存在します。新機能開発の一環として、開発者はApexトランザクション内でレコードセットが変更される際に競合状態が発生しないようにする必要があります。上記のApexコードにおいて、SOQL機能を

使用してトランザクション内で競合状態が発生しないようにするには、クエリステートメントをどのように変更すればよいでしょうか？
A. [Credit_Memo__c から Id、Name、Amount__c を選択 (Customer_Id__c = :customerId、更新の制限は 50)]
B. [Customer_Id__c = :customerId で、スコープ制限 50 を使用して、Credit_Memo__c から Id、Name、Amount__c を選択]
C. [Credit_Memo__c から Id、Name、Amount__c を選択 (Customer_Id__c = :customerId、参照用に 50 を制限)]
D. [Credit_Memo__c から Id、Name、Amount__c を選択 (Customer_Id__c = :customerId、表示制限 50)]
Answer: A (LEAVE A REPLY)
Co6mprehensive and Detailed 150 to 250 words of Explanation:
To prevent race conditions in Salesforce, where two or more concurrent transactions attempt to update the same record simultaneously, a developer must implement record locking. In SOQL, this is achieved using the FOR UPDATE keyword.
When a query includes the FOR UPDATE clause, the platform places an exclusive lock on the returned records for the duration of the transaction. If another process attempts to update these records or lock them with its own FOR UPDATE query, it will be forced to wait until the first transaction completes (commits or rolls back). If the lock is not released within 10 seconds, the second process will receive a QueryException.
Option A is the correct syntax to ensure that the Credit_Memo__c records are "reserved" for the current logic, preventing other threads from interfering with the newAmount calculation or update. Option B (USING SCOPE) is used for filtering records based on predefined ownership scopes like "My Terrace." Options C and D (FOR REFERENCE and FOR VIEW) are used to update the "Last Referenced" or "Last Viewed" timestamps on records, respectively, and do not provide the row-level locking required to prevent data concurrency issues or race conditions.

NEW QUESTION: 43
マイオポチュニティ.js
JavaScript
import { LightningElement, api, wire } from 'lwc';
import getOpportunities from
'@salesforce/apex/OpportunityController.findMyOpportunities'; export default class
MyOpportunities extends LightningElement {
@api userId;
@wire(getOpportunities, {oppOwner: '$userId'})
opportunities;
}
OpportunityController.cls

Java
public with sharing class OpportunityController {
@AuraEnabled
public static List<Opportunity> findMyOpportunities(Id oppOwner) {
return [
SELECT Id, Name, StageName, Amount
FROM Opportunity
WHERE OwnerId = :oppOwner
];
}
}
開発者がLightning Webコンポーネントで問題を抱えています。このコンポーネントは、現在ログインしているユーザーが所有する商談に関する情報を表示する必要があります。コンポーネントをレンダリングすると、「データ取得エラー」というメッセージが表示されます。これを接続可能にするには、Apexメソッドでどのアクションを実行する必要がありますか？
A. ApexメソッドでContinuation=true属性を使用します。14
B. Apexクラスでwithout sharingキーワードを使用するようにコードを編集します。15
C. ApexメソッドでCacheable=true属性を使用します。16
D. OpportunityオブジェクトのOWDがPublicであることを確認します。17
Answer: (SHOW ANSWER)
To use the @wire service in a Lightning Web Component (LWC) to retrieve data from an Apex method, the Apex method must be marked as cacheable. In the provided OpportunityController class, the findMyOpportunities method is annotated with @AuraEnabled, but it lacks the mandatory cacheable=true property.
The @wire service is part of the Lightning Data Service (LDS) reactive framework. Salesforce requires wireable methods to be cacheable to improve performance and ensure that data can be stored in the client-side cache. Without (cacheable=true), the @wire decorator will fail to execute, resulting in the "Error retrieving data" message or a similar runtime failure.
Option C is the correct fix: the developer must update the annotation to @AuraEnabled(cacheable=true).
Option A (Continuation) is only for long-running external callouts. Option B (without sharing) might affect visibility but isn't a requirement for the wire service itself. Option D (OWD) affects whether records are visible based on security, but the technical failure described is rooted in the component-to-Apex binding requirement. Once marked as cacheable, the LDS can properly manage the data lifecycle for the component.

NEW QUESTION: 44
Aura コンポーネントから呼び出される以下の Apex コントローラを考えてみましょう。

5行目 @AuraEnabled
6行目 public List<String> getStringArray() {
7行目 String[] arrayItems = new String[]{ 'red', 'green', 'blue' };
8行目 return arrayItems;
9行目 }
10行目}
このコードの何が問題なのでしょうか?
A. 1行目と6行目: クラスとメソッドはグローバルである必要があります
B. 1行目: クラスはグローバルである必要があります
C. 6行目: メソッドは静的である必要があります
D. 8行目: メソッドはリストを返す前にJSONにシリアル化する必要があります
Answer: C (LEAVE A REPLY)
When building Apex controllers for Aura Components (or Lightning Web Components), any method annotated with @AuraEnabled must be defined as static. The snippet provided defines the method getStringArray() as an instance method (public List<String>...) rather than a static method (public static List<String>...).
The Lightning Component framework does not instantiate an object of the Apex class; instead, it calls the method statically. If the method is not static, the framework cannot locate or execute it, resulting in an error.
While global was required in older versions or for managed packages, public is sufficient for code within the same namespace. Apex handles the serialization of standard types like Lists automatically, so manual JSON serialization is not required.

NEW QUESTION: 45
ある企業では、iOSネイティブの注文受付アプリをSalesforceに接続し、様々なオブジェクトからJSON形式で統合された情報を取得する必要があります。Salesforceでこれを実装する最適な方法はどれでしょうか？
A. Apex SOAP Webサービス
B. Apex SOAPコールアウト
C. Apex RESTコールアウト
D. Apex REST Webサービス
Answer: (SHOW ANSWER)
When an external application needs to request data from Salesforce, you must expose an endpoint. Since the requirement specifically mentions JSON format and a native mobile app (iOS), an Apex REST web service (Option D) is the optimal choice. REST (Representational State Transfer) is the industry-standard architecture for mobile-to-cloud integrations because it is lightweight, uses standard HTTP methods, and natively supports JSON.
By using the @RestResource and @HttpGet annotations in an Apex class, a developer can create a custom endpoint that performs complex logic, such as querying multiple objects (Accounts, Orders, Line Items), and returns a single, consolidated JSON response.

This is far more efficient than the standard REST API if the mobile app would otherwise need to make multiple sequential calls to gather the same information.
Options B and C are incorrect because "callouts" are used when Salesforce requests data from an external system, not the other way around. Option A (SOAP) is a heavier, XML-based protocol that is less efficient for mobile development and does not use the JSON format natively. Therefore, a custom Apex REST service provides the best performance and flexibility for mobile integrations.

NEW QUESTION: 46
Visualforce ページで transient キーワードを使用すると、どのパフォーマンスの問題の解決に役立ちますか?
A. ビューステートを減らす
B. クエリパフォーマンスが向上します
C. 読み込み時間を短縮します
D. ページ転送を改善します
Answer: (SHOW ANSWER)
In Visualforce, the "View State" is a hidden form field that maintains the state of the page (and the controller's variables) across postbacks to the server. If the View State becomes too large, it can slow down page loads and eventually hit the Salesforce View State limit (170KB), causing the page to crash.
The transient keyword is used to declare instance variables in Apex controllers that should not be saved in the View State. When a variable is marked as transient, its value is discarded after the request finishes and is not transmitted back to the client. This effectively reduces the size of the View State. It is commonly used for data that is needed only for the duration of the current request (like a large list of records displayed in a read-only table) and can be easily requeried or recalculated if needed.

NEW QUESTION: 47
開発者は、サンドボックスではテストの失敗を確認しましたが、本番環境では確認しませんでした。サンドボックスの作成以降、どちらの環境でもコードやメタデータの変更は行われていません。この問題を解決するには、どの点を確認すべきでしょうか？
A. テストクラスがSeeAllData = trueを使用していることを確認します。15

B. Apex クラスが同じ API バージョンであることを確認します。16
C. サンドボックスが本番環境と同じリリースであることを確認します。17
D. 「並列Apexテストを無効にする」のチェックが外れていることを確認します。18
Answer: (SHOW ANSWER)
When code behavior differs between two environments that theoretically share the same codebase, the root cause is often the underlying platform version. Salesforce releases updates (e.g., Spring, Summer, Winter) on a staggered schedule. Sandbox instances are frequently updated to the "Preview" version of the upcoming release weeks before the Production environment.
If a sandbox is on a newer release (Option C), it may include changes to the Apex compiler, new platform governor limits, or modifications to how standard objects behave. If a test relies on a specific behavior that changed in the new release, it might fail in the sandbox while still passing in the older Production environment.
Option A (SeeAllData=true) is a poor practice and would likely cause more issues rather than resolve them.
Option B is unlikely because API versions are fixed in the metadata of the class and wouldn't change spontaneously. Option D relates to execution speed and resource contention but rarely causes a functional test to pass in one environment and fail in another unless there are significant race conditions. Checking the release version is the standard first step when diagnosing environment-specific discrepancies in the absence of code changes.

NEW QUESTION: 48
ある組織では、顧客の注文情報を配送先住所の項目を持つカスタムオブジェクト「Order__c」に記録しています。開発者は、配送先住所の地域に関連付けられた定額の送料に基づいて注文の送料を計算するコードを追加するよう指示されています。変更を本番環境にデプロイする際に、追加の手順を必要とせずに計算を実行できるように、開発者は地域別の送料をどのように保存すればよいでしょうか？3132
A. カスタムメタデータタイプ3334
B. カスタムリスト設定3536
C. カスタムオブジェクト3738
D. カスタム階層設定3940
Answer: A (LEAVE A REPLY)
Comprehensive and Detailed 43150 to 250 words of Explanation:44
The critical requirement in this scenario is that "no additional steps are needed45" upon deployment. This refers to the ability to migrate not just the logic, but also the configuration data (the rates) across environments (e.g., from Sandbox to Production).
Custom Metadata Types (Option A) are the only option among those listed that are treated as metadata rather than data. This means that individual records within a Custom Metadata Type can be included in Change Sets, unlocked packages, or via the Metadata

API. When the developer deploys the code, they include the metadata records for the regions and rates in the same package. Once the deployment finishes, the code can immediately query those rates and perform calculations without any manual record creation in the production environment.
In contrast, Custom List Settings (Option B), Custom Objects (Option C), and Hierarchy Settings (Option D) store their records as "data." Data is not deployable via Change Sets. If a developer used these options, they would have to manually re-enter all the regional rates in Production after the code was deployed, which violates the requirement for a seamless, automated deployment process.

NEW QUESTION: 49
ある会社のサポートプロセスでは、ケースが 解決できませんでした」というステータスでクローズされるたびに、エンジニアリングレビューカスタムオブジェクトレコードを作成し、ケース、連絡先、およびケースに関連付けられている製品の情報を入力することが規定されています。Apexトリガーを使用してこれを自動化する正しい方法は何ですか？12
A. エンジニアリングレビューレコードを作成して挿入するケースの更新前トリガー3456
B. エンジニアリングレビューレコードを作成し、それを挿入するケースのアフターアップセットトリガー
C. 11 エンジニアリングレビューレコードを作成して挿入するケースの事前アップセットトリガー1213
D. エンジニアリングレビューレコードを作成し、それをCa15seに挿入する更新後トリガー14
Answer: (SHOW ANSWER)
In Salesforce Apex triggers, choosing the correct context (before vs. after) is critical for performance and data integrity. When the business requirement involves creating or modifying related records (records other than the one that fired the trigger), an after trigger is the platform-standard choice.
An after update trigger (Option D) is ideal here because it ensures that the Case record has been successfully validated and saved to the database (obtaining a valid timestamp and state) before the child Engineering_Review__c record is generated. Furthermore, an after trigger provides access to the read-only Trigger.new and Trigger.old maps, which contain the final field values after system validation rules and flows have run. Performing DML operations to insert a new object within a before trigger is technically possible but discouraged, as it can lead to issues if the primary record fails a subsequent system validation, potentially leaving orphaned data or causing complicated rollback scenarios.
Options B and C are incorrect because "upsert" is not a valid trigger event; the valid events are insert, update, delete, and undelete. Option A is incorrect because before triggers should primarily be used for updating fields on the same record that initiated the trigger to avoid redundant DML calls. Therefore, an after update trigger provides the most stable and scalable context for cross-object record creation.

NEW QUESTION: 50
アカウントオブジェクトには、アカウントに必要な監査の種類を指定するための Audit_Code__c項目と、割り当てられた監査人であるユーザーへの参照であるAuditor__c 項目があります。アカウントが最初に作成される際、ユーザーはAudit_Code__cを指定します。組織内の各ユーザーには、アカウントのAuditor__c項目に適切なユーザーを自動的に割り当てるために使用される、固有のテキスト項目Audit_Code__cがあります。
トリガー：
ジャワ
trigger AccountTrigger on Account (before insert) {
AuditAssigner.setAuditor(Trigger.new);
}
Apex Class:
Java
public class AuditAssigner {
public static void setAuditor(List<Account> accounts) {
for (User u : [SELECT Id, Audit_Code__c FROM User]) {
for (Account a : accounts) {
if (u.Audit_Code__c == a.Audit_Code__c) {
a.Auditor__c = u.Id;
}
}
}
}
}
コードの効率を最も最適化するには何を変更する必要がありますか?
A. 監査コードでフィルタリングするには、SOQL クエリに WHERE 句を追加します。
B. 監査コードからアカウントへの Map<String, List<Account>> を構築します。
C. コードを監査するためのアカウント ID の Map<Id, List<String>> を構築します。
D. すべての異なる監査コードを取得するための初期 SOQL クエリを追加します。
Answer: A (LEAVE A REPLY)
The current implementation suffers from a significant performance bottleneck due to an unfiltered SOQL query and a nested loop ($O(N \times M)$ complexity). The query [SELECT Id, Audit_Code__c FROM User] retrieves every single user in the Salesforce organization. In large enterprises with thousands of users, this results in excessive heap memory usage and consumes unnecessary CPU time, even if only a few specific users are needed for the current batch of accounts.
The most critical optimization to improve efficiency is to add a WHERE clause (Option A) to the SOQL query. By first collecting the Audit_Code__c values from the accounts list into a Set<String> and then using that set in the query (e.g., WHERE Audit_Code__c

IN :auditCodeSet), the developer ensures that Salesforce only retrieves the relevant User records from the database. This significantly reduces the data volume transferred from the database to the application server and avoids hitting governor limits like the "Total number of SOQL records retrieved." While building a Map (Option B) is also a best practice to eliminate the nested loop and move toward $O (N+M)$ complexity, the primary efficiency gain in terms of platform resources and governor limit safety comes from making the query selective. A selective query is the foundation of a "bulkified" and performant trigger handler.

NEW QUESTION: 51
開発者は、どのプロファイルとユーザーがどのシークレットにアクセスできるかを指定できるシークレットデータを保存する方法を見つけるよう求められています。このデータの保存には何を使用すべきでしょうか？
A. 静的リソース
B. カスタムメタデータ
C. カスタム設定
D. System.Cookie クラス
Answer: (SHOW ANSWER)
When a requirement involves storing configuration or "secret" data that needs to vary based on the specific user or profile, Hierarchy Custom Settings (Option C) are the correct choice. Unlike Custom Metadata (Option B), which is available to the entire organization regardless of the user, Hierarchy Custom Settings use a built-in logic to provide values based on the "most specific" level defined.
The hierarchy follows a specific order: User > Profile > Organization. If a secret key is defined at the User level, that value is returned for that specific user. If not, the system looks for a value defined at the user's Profile level, and finally falls back to the Organization-wide default. This allows a developer to store sensitive keys or flags and restrict or vary them dynamically based on the current user's identity.
Custom Metadata is better suited for application-wide configurations that need to be deployable via packages, but it lacks the granular per-user/per-profile override capability inherent to Hierarchy Custom Settings. Static Resources (Option A) are public to anyone with access to the resource, and Cookies (Option D) are stored on the client side, making them insecure for "secret" data. Therefore, Custom Settings provide the best balance of security and hierarchical flexibility for this use case.

NEW QUESTION: 52
開発者がカスタム設定を使用して、時々変更される設定データを保存していました。しかし、最近更新した一部のサンドボックスでテストが失敗するようになりました。今後この問題を解消するにはどうすればよいでしょうか？
A. カスタム設定の設定タイプをリストに設定します。

B. カスタム設定を静的リソースに置き換えます。1
C. カスタム設定をカスタムメタデータに置き換えます。2
D. カスタム設定の設定タイプを階層に設定します。3
Answer: (SHOW ANSWER)
5
The iss6ue described is a direct result of how Salesforce treats data during sandbox refreshes and unit testing.
Custom Settings (both List and Hierarchy) store their information as data records. By default, sandbox refreshes do not include data records from Production unless it is a Full Sandbox. Furthermore, Apex unit tests are isolated from organization data; they cannot "see" Custom Setting records unless the records are explicitly created within the test or the @isTest(SeeAllData=true) annotation is used (which is a poor practice).
Custom Metadata (Option C) solves this by treating configuration records as metadata rather than data.
Because they are metadata, the records are automatically included in every sandbox refresh and are deployable via Change Sets or the Salesforce CLI. Most importantly, Custom Metadata is visible to Apex unit tests without needing to create setup data or bypass test isolation. This ensures that configuration-dependent logic remains consistent across all environments and that tests pass reliably after a refresh. Replacing Custom Settings with Custom Metadata is the modern Salesforce standard for environment-independent configuration management.

NEW QUESTION: 53
リアルタイム通知が不要な場合に、開発者がユーザーが最後に Salesforce にログインした日時を確認できる Salesforce 機能はどれですか。
A. 非同期データキャプチャイベント
B. カレンダーイベント
C. イベント監視ログ
D. 開発者ログ
Answer: C (LEAVE A REPLY)
To perform historical auditing and analysis of user behavior-such as tracking the last login time-Salesforce provides Event Monitoring (Option C). This feature captures a wide range of "Event Types," including logins, logouts, URI (page) clicks, and API calls.
These events are stored in EventLogFile objects. Each day, Salesforce generates log files for the previous day's activity. A developer can query these files or use tools like the Salesforce Event Log File Browser to see a comprehensive history of user logins. This is the correct choice when real-time alerts are not necessary and the goal is to analyze patterns over time.
Option A refers to Change Data Capture, which is for tracking record changes, not user sessions. Option B is for the standard Salesforce Calendar. Option D (Developer Log) is

used for debugging Apex and system execution in the short term and does not provide a reliable or long-term history of user login events. Event Monitoring provides the high-fidelity audit trail required for compliance and security analysis.

NEW QUESTION: 54
アーカイブされたタスクを含み、削除されたタスクを除いた、12～24 か月前までのタスクを取得するには、どの SELECT ステートメントを挿入する必要がありますか?
ジャワ
日付 initialDate = System.Today().addMonths(-24);
日付 endDate = System.Today().addMonths(-12);
A. [SELECT ... FROM Task WHERE What.Type = 'Account' AND isArchived=true AND CreatedDate
=> :initialDate ...]
B. [SELECT ... FROM Task WHERE What.Type = 'Account' AND CreatedDate
=> :initialDate ... ALL ROWS]
C. [SELECT ... FROM Task WHERE What.Type = 'Account' AND isDeleted=false AND CreatedDate =>
:initialDate ... ALL ROWS]
D. [SELECT ... FROM Task WHERE What.Type = 'Account' AND isArchived=true AND CreatedDate
=> :initialDate ... ALL ROWS]
Answer: B (LEAVE A REPLY)
This query requires a specific understanding of how Salesforce handles old records. Salesforce "archives" Tasks that are older than 365 days to maintain performance. These archived records are not returned by standard SOQL queries.
To include archived records in a query, the developer must use the ALL ROWS keywords at the end of the query. However, ALL ROWS also returns records that are in the Recycle Bin (deleted records). To satisfy the requirement of "including archived" but "excluding deleted," the developer simply needs to rely on the fact that standard WHERE clause filters apply to the results of ALL ROWS.
Option B is the most accurate. In SOQL, isDeleted=false is implicit unless you are specifically looking for deleted records. The crucial part is the ALL ROWS keyword which brings the archived tasks into scope.
Options A and D are incorrect because isArchived=true would only return the archived tasks and exclude those that are not yet archived. Option C is redundant because isDeleted=false is the default behavior. The syntax => in the prompt options is also technically a typo for >=. Option B represents the standard way to query across the "active" and "archived" boundary while ignoring the Recycle Bin.

NEW QUESTION: 55

Visualforceページは、表示されるデータ量が多いため、読み込みが遅くなります。開発者はどのような戦略を用いてパフォーマンスを改善できますか？
A. カスタム コントローラーで使用されるリスト変数に transient キーワードを使用します。
B. コントローラーのコンストラクターではなく、遅延読み込みを使用して、要求に応じてデータを読み込みます。
C. JavaScript を使用して、データ処理をコントローラーではなくブラウザーに移動します。
D. ページ内で <apex:actionPoller> を使用して、すべてのデータを非同期的に読み込みます。
Answer: B (LEAVE A REPLY)
When a Visualforce page is slow because it attempts to load a massive dataset all at once, the bottleneck is often the time taken by the controller constructor to query and process those records before the page can even begin to render. Lazy Loading (Option B) is a strategy where the developer avoids loading all data during the initial page initialization. Instead of querying 10,000 records in the constructor, the developer can load only the structural elements of the page first. Then, using an action method (triggered by <apex:actionFunction> or a window.onload script), the data is fetched in the background. This allows the user to see the page layout immediately while the data "populates" shortly after. This significantly improves the Time to First Byte (TTFB) and the perceived performance of the page.
Option A (transient) is excellent for reducing the View State size and speeding up postbacks (button clicks), but it does not inherently speed up the initial load time of a large dataset. Option C (JavaScript processing) can help but doesn't solve the data retrieval bottleneck. Option D (actionPoller) is used for periodic updates and is not an efficient way to load an initial large dataset. Lazy loading ensures that the "heavy lifting" is decoupled from the initial page load, providing a more responsive experience.

NEW QUESTION: 56
ある組織では、取引先には必ず1人の連絡先がプライマリとして登録されている必要があるという要件があります。そのため、1人の連絡先を選択すると、他の連絡先は選択解除されます。クライアントは、この機能を制御するために、連絡先に 「プライマリ」というチェックボックスを追加したいと考えています。また、すべての連絡先の姓がすべて大文字で保存されるようにしたいと考えています。これらの要件を実装する最適な方法は何でしょうか？12345
A. 連絡先にIs Primaryロジックの検証ルールを記述し、連絡先に姓ロジックの更新前トリガーを記述します。8910
B. Is Primary ロジック用に Contact に更新後トリガーを記述し、姓ロジック用に Contact に別の更新前トリガーを記述します。

C. Is Primary ロジック用に Account に更新後トリガーを記述し、姓ロジック用に Contact に更新前トリガーを記述します。
D. 更新後と更新前の両方に対してContactに単一のトリガーを記述し、各ロジックセットを処理するヘルパークラスを呼び出します。14
Answer: D (LEAVE A REPLY)
16
The optimal architectural ap17proach for complex logic on a single object is to use a single trigger per object that delegates responsibilities to a helper class. This requirement involves two distinct types of logic: field manipulation (uppercase names) and cross-record updates (ensuring only one primary contact).
In Salesforce, Before triggers are ideal for updating fields on the record that initiated the trigger, as the changes are saved to the database without an additional DML statement. Therefore, converting the LastName to uppercase should occur in a before update and before insert context. Conversely, the "Is Primary" logic requires updating other records (de-selecting other contacts on the same account). This must happen in an After trigger context to ensure the primary record has been successfully updated and to avoid recursion or conflicts with the initial save.
Option D is the best practice because it follows the "One Trigger Per Object" design pattern. By using a single trigger with multiple context variables (isBefore, isAfter, isUpdate), the developer can cleanly route the "Last Name" logic to a before-save helper and the "Is Primary" cross-record update logic to an after-save helper.
This centralized control prevents multiple triggers from firing in an unpredictable order and simplifies maintenance and debugging.

NEW QUESTION: 57
ある企業は、アカウントが挿入された際に、住所フィールドがまだ設定されていない場合に、サードパーティのウェブサービスを導入して住所フィールドを設定したいと考えています。これを実現する最適な方法は何でしょうか？
A. Before Save フローを作成し、そこから Queueable ジョブを実行し、Queueable ジョブからコールアウトを実行します。
B. Apex クラスを作成し、そこから Batch Apex ジョブを実行し、Batch Apex ジョブからコールアウトを行います。
C. Apex トリガーを作成し、そこから Queueable ジョブを実行し、Queueable ジョブからコールアウトを行います。
D. Apex クラスを作成し、そこから Future メソッドを実行し、Future メソッドからコールアウトを行います。
Answer: C (LEAVE A REPLY)
The requirement is to perform a callout when a record is inserted. Because Salesforce prohibits synchronous callouts after DML operations in the same transaction, this must be handled asynchronously.

Queueable Apex (Option C) is the optimal choice for this integration. When an Account is inserted, a trigger captures the ID and enqueues a Queueable job. Queueable is superior to Future methods (Option D) because it supports complex data types (not just primitives), allows for job chaining, and provides a Job ID that can be used for monitoring via AsyncApexJob.
Before-Save Flows (Option A) cannot directly perform callouts, and while they can trigger asynchronous paths, the programmatic control offered by a Trigger-to-Queueable pattern is more robust for complex third- party integrations. Batch Apex (Option B) is designed for bulk processing of existing records and is "overkill" for a real-time, record-by-record trigger requirement. By using a Trigger with Queueable, the developer ensures the address validation happens nearly in real-time without blocking the user's initial save transaction or hitting concurrent request limits.

NEW QUESTION: 58
Apexトリガーは、商談が成立/成立とマークされるたびに契約レコードを作成します。履歴レコードは読み込む必要がありますが、この一括読み込み中に契約レコードは作成しないでください。これを実現するためにApexトリガーを更新する最も拡張可能な方法は何ですか？
A. データをロードするユーザーのプロファイル ID をトリガーに追加します。
B. データ ロード ユーザーによる作成を防ぐために、契約に検証ルールを追加します。
C. リストのカスタム設定を使用して、データをロードするユーザーのトリガーを無効にします。
D. 階層カスタム設定を使用して、データをロードするユーザーに対してトリガー内のロジックの実行をスキップします。
Answer: D (LEAVE A REPLY)
When managing trigger logic that needs to be conditionally bypassed (often called a "Trigger Kill Switch"), using Hierarchy Custom Settings (Option D) is the best practice for scalability and maintainability.
A developer can create a Hierarchy Custom Setting named Trigger_Settings__c with a checkbox field Disable_Opportunity_Trigger__c. In the Apex trigger, the code should first check this setting:
if (Trigger_Settings__c.getInstance().Disable_Opportunity_Trigger__c) return; Because it is a hierarchy setting, an administrator can enable this checkbox specifically for the User or Profile performing the data load without affecting the daily operations of other sales reps. Once the data load is complete, the setting can be toggled off. This approach is "extendable" because it doesn't require hardcoding IDs (Option A), which change between environments (Sandbox vs. Production).
Option C (List Custom Settings) is less flexible because it doesn't support the Profile-to-User hierarchy.

Option B (Validation Rules) would stop the record creation but would result in a DML error in the trigger, potentially failing the entire data load transaction rather than simply skipping the logic. Hierarchy Custom Settings provide a clean, metadata-driven way to control execution flow across different user contexts.

NEW QUESTION: 59
以下のコンポーネント コードと要件を参照してください。
HTML
<lightning:layout multipleRows="true">
<lightning:layoutItem size="12">{!v.account.Name}</lightning:layoutItem>
<lightning:layoutItem size="12">{!v.account.AccountNumber}</lightning:layoutItem>
<lightning:layoutItem size="12">{!v.account.Industry}</lightning:layoutItem>
</lightning:layout>
要件：
* モバイル デバイスの場合、情報は 3 行に表示されます。
* デスクトップとタブレットの場合、情報は 1 行に表示されます。
要件2が期待どおりに表示されません。デスクトップとタブレットの要件を満たす正しいコンポーネントコードを持つオプションはどれですか？
A. HTML
<lightning:layout multipleRows="true">
<lightning:layoutItem size="12" mediumDeviceSize="6">{!
v.account.Name}</lightning:layoutItem>
<lightning:layoutItem size="12" mediumDeviceSize="6">{!
v.account.AccountNumber}</lightning:
layoutItem>
<lightning:layoutItem size="12" mediumDeviceSize="6">{!
v.account.Industry}</lightning:layoutItem>
</lightning:layout>
B. <lightning:layout multipleRows="true"></lightning:layout>1213
C. 1415
HTML
<lightning:layout multipleRows="true">
<lightning:layoutItem size="12" largeDeviceSize="4">{!
v.account.Name}</lightning:layoutItem>
<lightning:layoutItem size="12" largeDeviceSize="4">{!
v.account.AccountNumber}</lightning:
layoutItem>
<lightning:layoutItem size="12" largeDeviceSize="4">{!
v.account.Industry}</lightning:layoutItem>
</lightning:layout>
D. HTML

<lightning:layout multipleRows="true">
<lightning:layoutItem size="12" mediumDeviceSize="4" largeDeviceSize="4"> {!
v.account.Name} <
/lightning:layoutItem>
<lightning:layoutItem size="12" mediumDeviceSize="4" largeDeviceSize="4"> {!v.account.
AccountNumber} </lightning:layoutItem>
<lightning:layoutItem size="12" mediumDeviceSize="4" largeDeviceSize="4"> {!
v.account.Industry}
</lightning:layoutItem>
</lightning:layout>

Answer: D (LEAVE A REPLY)

To achieve a responsive layout in Salesforce Aura components, the lightning:layout and lightning:layoutItem components utilize a 12-column grid system. The size attribute defines the default behavior, which targets the smallest devices (mobile). In the original code, size="12" is applied to all three items. Since each item occupies the full 12 columns, they stack vertically in three rows.

To meet Requirement 2 (displaying in a single row for tablets and desktops), the three items must share the

12-column horizontal space. Mathematically, $12 \div 3 = 4$. Therefore, each item must be assigned a size of

4 for larger screen breakpoints.

Option D is the correct implementation. It maintains size="12" for mobile (Requirement 1) and sets mediumDeviceSize (tablets) and largeDeviceSize (desktops) to 4. This ensures that on any screen larger than a phone, the three items will sit side-by-side in a single row.

Option A is incorrect because mediumDeviceSize="6" would only allow two items per row ($6 + 6 = 12$), forcing the third item to a second row. Option C is incomplete as it only specifies the largeDeviceSize, potentially leaving tablets in a stacked configuration. Option D follows the best practice of explicitly defining the span for all relevant device categories to ensure a consistent user experience.

NEW QUESTION: 60

開発者は、次の HTML スニペットを使用して、sObject Lightning ページに存在する再利用可能な Aura コンポーネントを開発しています。

HTML

<aura:component implements="force:hasRecordId,flexipage:availableForAllPageTypes">
こんにちは！
</aura:component>

追加のテスト カバレッジを必要とせずに、コンポーネントのコントローラが sObject が存在する Lightning ページのコンテキストを取得するにはどうすればよいでしょうか?

A. デザイン属性を作成し、App Builder 経由で構成します。

B. sObject タイプをコンポーネント属性として設定します。
C. Apex クラスで getSObjectType() メソッドを使用します。
D. implements 属性に force:hasSObjectName を追加します。
Answer: D (LEAVE A REPLY)
In Aura components, Salesforce provides specific interfaces that "inject" context into the component automatically. The developer is already using force:hasRecordId, which automatically populates a recordId attribute with the ID of the current record.
To get the API Name of the sObject (e.g., "Account" or "Contact") without writing Apex code or requiring manual configuration in the App Builder, the developer should implement force:hasSObjectName (Option D). When this interface is added to the implements attribute, the framework automatically provides an attribute named sObjectName to the component.
This is the most efficient solution because:
* It is purely declarative and handled by the Lightning framework.
* It requires zero Apex code, meaning there is no need for additional unit tests or code coverage (satisfying the requirement "without requiring additional test coverage").
* It makes the component truly reusable; you can drop it on an Account page, a Contact page, or any custom object page, and it will immediately know which object it is currently displaying.
Options A and B require manual configuration or hardcoding, which reduces reusability.
Option C requires an Apex call, which adds complexity and necessitates writing test classes for coverage.

NEW QUESTION: 61
開発者は、スケジュールされたApexがDML制限に達しているという問題の調査を依頼されました。調査の結果、スケジュールされたApexによって処理されるレコード数が最近10,000件を超えていることが分かりました。この制限例外エラーを解消するには、開発者はどのように対処すればよいでしょうか？
A. Batchable インターフェースを実装します。
B. プラットフォーム イベントを使用します。
C. @future アノテーションを使用します。
D. Queueable インターフェースを実装します。
Answer: A (LEAVE A REPLY)
The issue described involves hitting DML limits due to a high volume of records (over 10,000) being processed in a single transaction. In Salesforce, a single synchronous transaction or a standard Scheduled Apex job has strict governor limits on the number of DML statements (150) and the total number of records processed (10,000 for DML rows).
To resolve this, the developer should implement the Database.Batchable interface. Batch Apex is specifically designed to handle large data volumes by breaking the processing into smaller, manageable chunks (batches) of records (default 200). Each batch runs within its

own transaction context, resetting the governor limits for that specific batch. This prevents the "Too many DML rows" exception. While Queueable (Option D) and @future (Option C) provide asynchronous processing, they are not inherently designed to iterate over large datasets in the same robust, governor-limit-safe manner as Batch Apex, particularly when the record count exceeds the thousands.

NEW QUESTION: 62
システムにはLightning Webコンポーネントが存在し、レコードに関する情報をコンテキストに応じてモーダルとして表示します。Salesforce管理者は、Lightning App Builder内でこのコンポーネントを使用する必要があります。開発者は、XMLリソースファイル内でどの2つの設定を行う必要がありますか？
A. 'IsVisible' 属性を 'true' に設定します。
B. 'IsExposed' 属性を 'true' に設定します。
C. 'target' を 'lightning__RecordPage' に指定します。
D. 'target' を 'lightning__AppPage' に指定します。
Answer: B,C (LEAVE A REPLY)
To make a Lightning Web Component (LWC) available for use in the Lightning App Builder, the developer must modify the component's configuration file (the .js-meta.xml file).
First, the isExposed tag must be set to true. By default, this is false, meaning the component is only available to other components within the namespace but not to the App Builder or Experience Builder. Setting it to true exposes it to the tools.
Second, the developer must specify where the component can be dragged and dropped by defining targets.
Since the question states the component displays information about the "record in context," it implies the component is designed to live on a Record Page. Therefore, adding <target>lightning__RecordPage</target> is essential. While lightning__AppPage (Option D) is a valid target, the context of "record in context" strongly points to the Record Page target. There is no IsVisible attribute in the LWC configuration schema.

NEW QUESTION: 63

ある企業では、すべての親レコードに必ず子レコードが存在することが求められています。開発者は、親レコードと子レコードを挿入するための2つのDML文を含むApexメソッドを作成しました。入力規則により、子レコードの作成がブロックされています。メソッドは、DML例外を処理するためにtry/catchブロックを使用しています。親レコードが常に子レコードを持つことを保証するには、開発者は何をすべきでしょうか？
A. Database.insert() を使用し、allOrNone パラメータを true に設定します。
B. 子レコードの DML 操作でエラーが発生した場合、catch ステートメントで親レコードを削除します。
C. エラーが発生した場合にロールバックするためのデータベース セーブポイントを設定します。
D. 子レコードでエラーが発生した場合は、親レコードで addError() を使用します。
Answer: C (LEAVE A REPLY)
1516
In Salesforce Apex, each DML statement is its own mini-transaction unless specific measures are taken to group them. If a developer performs an insert parent; followed by an insert child;, and the child insertion fails due to a validation rule, the parent record remains in the database. This leads to "orphaned" parent records, violating the bu17siness requireme18nt that every parent must have a child.
To ensure "all-or-nothing" behavior across multiple DML statements, the developer should use Database.
setSavepoint() and Database.rollback(). By setting a savepoint before the parent is inserted, the developer creates a "marker" for the database state. If the child insertion fails and throws an exception, the code enters the catch block. Within the catch block, the developer can call Database.rollback(sp), which reverts the database to the state before the parent was ever inserted. This ensures that either both records are created successfully or neither is created.
Option A is incorrect because allOrNone only applies to the records within a single Database.insert() call; it cannot link the success of two separate DML calls for different objects. Option B is a manual cleanup approach that is less reliable than a system-level rollback. Option D is typically used in triggers to prevent saving, but it doesn't provide the transactional control needed in a custom method with multiple DML steps.
Using savepoints is the standard best practice for managing multi-object transaction atomicity.

NEW QUESTION: 64
テストクラスで Visualforce ページを初期化するためのベストプラクティスは何ですか?
A. controller.currentPage.setPage(MyTestPage); を使用します。
B. Test.setCurrentPage.MyTestPage を使用します。
C. Test.setCurrentPage(Page.MyTestPage); を使用します。
D. Test.currentPage.getParameters.put (MyTestPage); を使用します。

Answer: C (LEAVE A REPLY)
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.

NEW QUESTION: 65
開発者が作成したテストクラスでは、モックコールアウトを行う前にテストデータを作成していましたが、「コミットされていない作業が保留中です。コールアウトを行う前にコミットまたはロールバックしてください」というエラーが発生します。このエラーを解決するには、どのような手順を踏む必要がありますか？
A. 挿入とモック呼び出しの両方が Test.stopTest() の後に行われることを確認します。1
B. レコードが Test.startTest() ステートメントの前に挿入され、モックコールアウトが @testSetup.2 でアノテーションされたメソッド内で発生することを確認します。
C. レコードが Test.startTest() ステートメントの前に挿入され、モックコールアウトが Test.startTest() の後に発生することを確認します。45
D. 挿入とモックコールアウトの両方が Test.startTest() の後に行われることを確認します。67
Answer: C (LEAVE A REPLY)
Comprehensive and Deta10iled 150 to 250 words of E11xplanation:
This error occurs because Salesforce prohibits making a callout (even a mock one) in the same transaction after a DML operation has been performed. When you insert test data, it creates a "pending" transaction in the database. If you then attempt to execute a callout immediately, the platform throws the CalloutException to prevent data inconsistency issues.
To resolve this in a test context, you must separate the DML operation from the callout using the Test.
startTest() and Test.stopTest() methods. When Test.startTest() is called, Salesforce provides a fresh set of governor limits and effectively creates a new transaction context for the code that follows. By inserting the records before Test.startTest() and performing the

logic that triggers the callout after Test.startTest(), the developer ensures that the DML operation is "committed" to the test database context before the callout is initiated.
Option C correctly identifies this pattern. Option B is incorrect because @testSetup is used for global data creation across all test methods and does not specifically address the callout transaction split. Options A and D do not solve the problem because they either keep the DML and callout in the same context or place the DML in a context where it would still interfere with the subsequent callout request.

NEW QUESTION: 66
Salesforce Platform開発者が、新しいアプリケーションを本番環境に導入するチームを率いています。チームはソース駆動開発を採用しており、アプリケーションの導入が確実に成功することを目標としています。導入が成功したことを確認するには、どのようなツールやメカニズムを使用すべきでしょうか？
A. Force.com 移行ツール
B. Apex テスト実行
C. Salesforce DX CLI
D. Salesforce インスペクター
Answer: C (LEAVE A REPLY)
56
In a source-driven development workflow, the "source of truth" is a version control system, and the primary interface for interacting with the Salesforce platform is the Salesf7orce DX CLI (Option C).8 When a deployment is initiated (using commands like project deploy start or the legacy force:source:deploy), the CLI provides real-time feedback on the status of the deployment. To verify a successful deployment, the CLI returns a detailed summary of the metadata components successfully created or updated, as well as the results of any Apex tests that were required to run as part of the production deployment. If any part of the deployment fails, the CLI provides specific error messages and line numbers, and the entire transaction is rolled back.
Option B (Apex Test Execution) is a component of the verification process, but the CLI is the tool that manages and reports on the overall success. Option A (Migration Tool) is a legacy Ant-based tool that is not the standard for modern source-driven DX projects.
Option D is a browser extension for data and metadata inspection but is not a deployment or verification tool. The Salesforce CLI is the backbone of modern CI/CD pipelines and the essential mechanism for verifying deployment integrity.

NEW QUESTION: 67
開発者がVisualforceコンポーネントとLightningコンポーネントのどちらを作成するか検討しています。Visualforceの使用が必要なシナリオはどれですか？
A. サードパーティのアプリケーションを使用せずに画面を PDF としてレンダリングする必要がありますか?

B. (コンテキストでオプションが提供されていません)
C. 画面では JavaScript フレームワークが使用されますか?
D. (質問ロジックのコンテキストでオプションAに一致)
Answer: A (LEAVE A REPLY)
While Salesforce highly recommends Lightning Web Components (LWC) or Aura components for modern UI development, there are specific legacy use cases where Visualforce is still the required technology. The most prominent of these is rendering a page as a PDF.
Visualforce provides the renderAs="pdf" attribute on the <apex:page> tag, which utilizes a server-side engine to convert the HTML markup into a downloadable PDF document. Neither Lightning Web Components nor Aura components have a native, built-in capability to render directly to PDF on the server side; they rely on client-side rendering or third-party libraries/services. Therefore, if the requirement is native PDF generation, Visualforce is the correct choice.

NEW QUESTION: 68
セーブポイントに関して正しい記述はどれですか?
A. 任意の順序で作成された任意のセーブポイント変数にロールバックできます。
B. 静的変数はロールバック中に元に戻されません。
C. セーブポイントは、DML ステートメント ガバナーの制限によって制限されません。
D. セーブポイントへの参照はトリガー呼び出しをまたぐことができます。
Answer: B (LEAVE A REPLY)
In Salesforce Apex, a Database.Savepoint allows a developer to define a point in a transaction that can be returned to if an error occurs, effectively undoing database changes. However, it is critical to understand what is-and is not-affected by a rollback. According to the platform specifications, while database records (DML operations) are reverted to their state at the time the savepoint was set, static variables are not reverted (Option B). Static variables persist across the entire transaction regardless of rollbacks, which is why they are commonly used as "recursion guards" to track if a trigger has already fired.
Other options are incorrect due to the strict "stack" nature of savepoints. Savepoints must be rolled back in reverse chronological order; rolling back to an earlier savepoint invalidates any savepoints created after it (Option A). Furthermore, a savepoint variable is only valid within the specific execution context in which it was created. You cannot pass a savepoint reference across different trigger invocations or asynchronous boundaries (Option D). Finally, while the rollback itself isn't a DML statement, the operations performed between setting a savepoint and rolling back still count toward DML and CPU governor limits (Option C).
Understanding that memory state (static variables) remains unchanged during a database rollback is essential for debugging complex transactional logic.

NEW QUESTION: 69
Universal Containers は、Lightning Web コンポーネントとその Apex コントローラを分析しています。コードスニペットに基づいて、Lightning Web コンポーネントで取引先責任者の郵送先住所を表示するには、どのような変更を加える必要がありますか？
Apex コントローラクラス:
ジャワ
共有クラス AccountContactsController を持つパブリック {
@AuraEnabled
パブリック静的リスト<連絡先> getAccountContacts(String accountId) {
[SELECT Id, Name, Email, Phone FROM Contact WHERE AccountId = :accountId] を返します。
}
}
A. Apex コントローラ クラスに新しいメソッドを追加して、郵送先住所を個別に取得します。
B. getAccountContacts メソッドの SOQL クエリを変更して、MailingAddress フィールドを含めます。
C. lightning-datatable コンポーネントを拡張して、MailingAddress フィールドの列を追加します。
D. getAccountContacts メソッドの SOQL クエリを変更して MailingAddress フィールドを含め、JavaScript ファイルの columns 属性を更新して郵送先住所フィールドを追加します。
Answer: D (LEAVE A REPLY)
To display new data in a lightning-datatable, changes are required at both the Data Retrieval (Apex) layer and the UI Definition (JavaScript) layer.
First, the SOQL query in the Apex controller must be updated to include the address fields. While MailingAddress is a compound field in Salesforce, in a SOQL query intended for a datatable, it is often best to retrieve the individual components (e.g., MailingStreet, MailingCity, MailingState).
Second, the JavaScript controller for the LWC must be updated. The columns array in the JS file defines which data properties the lightning-datatable looks for. Even if the Apex method returns the address data, the datatable will not render it unless there is a corresponding column definition with the correct fieldName property.
Option D correctly identifies this two-step process. Option B is incomplete because the UI wouldn't know to show the new data. Option C is incorrect because you don't "extend" the component to add columns; you simply pass a configuration array. Option A is inefficient as it creates extra server round-trips for data that should be fetched in a single query. By aligning the SOQL fields with the datatable column definitions, the mailing address can be displayed seamlessly.

NEW QUESTION: 70
開発者は、Lightning Web コンポーネントからの入力に基づいてアカウントを更新する Apex クラスを作成しました。
アカウントの更新は、まだ登録されていない場合にのみ行ってください。
ジャワ
account = [SELECT Id, Is_Registered__c FROM Account WHERE Id = :accountId]; if (! account.Is_Registered__c) { account.Is_Registered__c = true;
// ...他のアカウント フィールドを設定します...
アカウントを更新します。
}
複数のユーザーが同時に同じアカウントを更新した場合に、互いの更新が上書きされないようにするには、開発者は何をすべきでしょうか?
A. 更新の周囲に try/catch ブロックを追加します。
B. 更新の代わりに upsert を使用します。
C. SOQL クエリで FOR UPDATE を使用します。
D. 最近更新されていないことを確認するために、クエリに LastModifiedDate を含めます。
Answer: (SHOW ANSWER)
In a multi-user environment, a "Race Condition" occurs when two users query the same record at the same time, see that Is_Registered__c is false, and both proceed to update it. To prevent this, developers must implement Row-Level Locking.
In Salesforce SOQL, the FOR UPDATE (Option C) keyword is used for this exact purpose. When a query includes FOR UPDATE, the platform places an exclusive lock on the records returned. If a second transaction attempts to query the same records using FOR UPDATE, it will be forced to wait until the first transaction is completed (either committed or rolled back). If the lock isn't released within 10 seconds, the second transaction throws a QueryException.
By using FOR UPDATE, the first user who executes the query "reserves" the Account. The second user's request will wait, and by the time their query actually executes and retrieves the data, Is_Registered__c will have been updated to true by the first user, thus failing the if (!account.Is_Registered__c) check and preventing a duplicate registration. Option D is a manual "optimistic locking" approach that is less reliable than the platform's native locking mechanism. Options A and B do not address the timing issue of the initial check.

NEW QUESTION: 71
開発者は、Salesforce からデータを取得してレコード プロパティに割り当てる Lightning Web コンポーネントを構築しています。
JavaScript
'lwc' から { LightningElement、api、wire } をインポートします。
'lightning/uiRecordApi' から { getRecord } をインポートします。
エクスポートのデフォルトクラス Record は LightningElement を拡張します {

@api フィールド;
@api レコードID;
記録;
}
Salesforce からデータを取得するには、コンポーネントで何を行う必要がありますか?
A. 上記のレコードに @api(getRecord, { recordId: '$recordId' }) を追加します。
B. レコードの上に @wire(getRecord, { recordId: '$recordId' }) を追加します。
C. レコードの上に @api(getRecord, { recordId: '$recordId', fields: '$fields' }) を追加します。
D. レコードの上に @wire(getRecord, { recordId: '$recordId', fields: '$fields' }) を追加します。
Answer: D (LEAVE A REPLY)
To retrieve record data in a Lightning Web Component using the Lightning Data Service (LDS), the developer must use the @wire decorator with the getRecord wire adapter. Option D provides the correct syntax. The @wire decorator requires the adapter name (getRecord) and a configuration object. The configuration must include the recordId and either the fields or layoutTypes to be retrieved. By using the $ prefix (e.g., '$recordId'), the developer makes the property "reactive." This means that whenever the value of recordId or fields changes, the wire service automatically re-provisions the data from Salesforce. Options A and C are incorrect because @api is used to expose public properties, not for wiring data. Option B is incorrect because getRecord requires a field list or layout type to know which data points to fetch from the server. Using the reactive $fields ensures that the component remains flexible and can load different field sets as defined by its parent component.

NEW QUESTION: 72
次のコード スニペットを検討してください。
ジャワ
HttpRequest req = 新しい HttpRequest();
エンドポイントを 'https://TestEndpoint.example.com/some_path' に設定します。
req.setMethod('GET');
Blob headerValue = Blob.valueOf('myUserName' + ':' + 'strongPassword'); String authorizationHeader = 'BASIC ' + EncodingUtil.base64Encode(headerValue); req.setHeader('Authorization', authorizationHeader); Http http = new Http(); HTTPResponse res = http.send(req); コードを変更せずにエンドポイントと資格情報を変更するための柔軟性を追加するには、開発者が実行する必要がある 2 つの手順はどれですか?1
A. req.setEndpoint(Label.endPointURL); を使用します。
B. コールアウトリクエスト内で req.setEndpoint('callout:endPoint_NC'); を使用します。2
C. エンドポイントの URL を endPointURL.4 というカスタムラベルに保存します。

D. エンドポイントと資格情報を格納するための名前付き資格情報 endPoint_NC を作成します。5
Answer: B,D (LEAVE A REPLY)
7
The provided code uses hard-coded strings for the URL and authe8ntication headers, which is a security risk and makes maintenance difficult. To add flexibility and security, Salesforce provides Named Credentials (Option D). A Named Credential acts as a single definition that encapsulates both the endpoint URL and the required authentication (Basic, OAuth, etc.) in a single Setup record.
By using the callout: protocol in the setEndpoint method (Option B), the developer instructs the Apex runtime to look up the configuration stored in the Named Credential named endPoint_NC. This approach provides several key benefits:
* Security: Credentials like passwords are encrypted and stored safely in the platform's metadata, not in the code.
* Simplified Code: The logic for building the Authorization header and base64 encoding (as seen in the original snippet) is handled automatically by the platform.
* Maintainability: If the endpoint URL or the password changes, an administrator can update the Named Credential record via the UI without requiring any code deployment.
Custom Labels (Option A and C) can store URLs, but they cannot securely handle authentication logic or headers. Therefore, the combination of a Named Credential and the callout: syntax is the optimal platform- native solution for flexible and secure integrations.

NEW QUESTION: 73
ある企業はERPシステムで顧客の注文を受け付けており、SalesforceにOrder__cレコードとして統合する必要があります。このレコードは、Accountへの参照項目を持ちます。Accountオブジェクトには、ERP_Customer_ID__cという外部ID項目があります。適切なAccountに自動的に関連付けられる新しいOrder__cレコードを作成するには、どのような連携方法を使用すればよいでしょうか？1234
A. Order__c オブジェクトに Upsert し、Account 関係に ERP_Customer_ID__c を指定します。5678
B. Account に Upsert し、relations9hip.101112 に ERP_Customer_ID__c を指定します。
C. Order__c オブジェクトをマージし、Account 関係に ERP_Customer_ID__c を指定します。
D. Order__c オブジェクトに挿入し、その後 Order__c オブジェクトを更新します。
Answer: A (LEAVE A REPLY)
In Salesforce integration, the Upsert operation is highly powerful because it allows you to create or update records and relate them to other records using External IDs instead of Salesforce record IDs ($001...$).
When the ERP system sends an order, it may not know the 18-character Salesforce ID for the Account, but it does know the ERP_Customer_ID__c. By performing an Upsert on the

Order__c object (Option A), the integration can leverage "Foreign Key" syntax in the API payload. This essentially tells Salesforce: "Create this Order and link its Account__c lookup field to the Account that has this specific ERP_Customer_ID__c." This approach is the "Gold Standard" for integrations because:
* It eliminates the need for the external system to perform a "lookup" query in Salesforce to find an ID before creating a record.
* It reduces the total number of API calls, saving on governor limits.
* It handles "idempotency," ensuring that if the same order is sent twice, it is updated rather than duplicated.
Options B, C, and D are less efficient. Upserting the Account (Option B) doesn't create the Order. Merging (Option C) is for deduplication. The Insert-then-Update approach (Option D) is an anti-pattern that doubles the required API calls and transaction overhead.

NEW QUESTION: 74
ある企業では、ケースの所有者に関係なく、サポート マネージャーが組織内のすべてのケースを閲覧できるようにしたいと考えています。
しかし、エージェントが自分の担当ケース、または自分のロールまたは下位ロールの担当者が担当するケースのみを表示できるようにしたいと考えています。この要件を満たすには、開発者はどの共有ソリューションを使用すべきでしょうか？
A. 共有セット
B. Apex 管理共有
C. ロール階層
D. 基準に基づく共有ルール
Answer: (SHOW ANSWER)
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.

NEW QUESTION: 75
開発者は、Salesforceの任意の2つのオブジェクトレコードのName項目を比較する汎用Apexメソッドを作成したいと考えています。例えば、取引先と商談のName項目、または取引先と取引先責任者のName項目を比較したいとします。Name項目が存在する場合、開発者はどのようにこれを実行すればよいでしょうか？4
A. String.replace() メソッドを使用して各 Name フィールドの内容を解析し、結果を比較します。567
B. SalesforceメタデータAPIを使用して各オブジェクトの値を抽出し、名前フィールドを比較します。910
C. 各オブジェクトをsObjectにキャストし、sObject.get('Name')を使用してNameフィールドを比較します。1112
D. Schema.describe() 関数を呼び出して、各 Name フィールドの値を比較します。
Answer: C (LEAVE A REPLY)
To write "generic" code in Apex-meaning code that can handle different types of objects (Accounts, Contacts, Custom Objects) interchangeably-the developer should use the base sObject class. All Salesforce objects inherit from this base class.
The sObject class provides a method called .get(fieldName) (Option C). This method allows you to retrieve the value of any field by passing its string name as an argument. By casting the input records (e.g., an Account and an Opportunity) to the generic sObject type, the developer can call .get('Name') on both and compare the returned objects (typically strings). This bypasses the need to know the specific object type at compile-time and avoids complex conditional logic.
Option A is incorrect because String.replace is for string manipulation, not data retrieval. Option B is incorrect because the Metadata API is for modifying org structure, not retrieving record field values. Option D is incorrect because Schema.describe provides information about the field's definition (type, label, length), but it does not provide the actual data value stored in a specific record. Using the generic sObject.get() method is the standard, most efficient way to achieve polymorphism in Apex data handling.

NEW QUESTION: 76
Apex トリガーと Apex クラスは、ケースが変更されるたびにカウンター `Edit_Count__c` を増分します。
```java
public class CaseTriggerHandler {
public static void handle(List<Case> cases) {
for (Case c : cases) {
c.Edit_Count__c = c.Edit_Count__c + 1;
```

}
}
}
trigger on Case(before update) {
CaseTriggerHandler.handle(Trigger.new);
}
```
ケースオブジェクトに、ケースの作成または更新時に実行される保存前レコードトリガフローを本番環境に新しく追加しました。このプロセスを追加して以来、ケースの編集時に「Edit_Count__c」が複数回増加しているという報告を受けています。この問題を修正するApexコードはどれでしょうか？
**A.** ```java
public class CaseTriggerHandler {
public static Boolean firstRun = true;
public static void handle(List<Case> cases) {
for (Case c : cases) {
**B.** Edit_Count__c = c.Edit_Count__c + 1;
}
}
}
trigger on Case(before update) {
CaseTriggerHandler.firstRun = true;
if (CaseTriggerHandler.firstRun) {
CaseTriggerHandler.handle(Trigger.newMap);
}
CaseTriggerHandler.firstRun = false;
}
```
C. ```java
public class CaseTriggerHandler {
public static Boolean firstRun = true;
public static void handle(List<Case> cases) {
for (Case c : cases) {
D. Edit_Count__c = c.Edit_Count__c + 1;
}
}
}
trigger on Case(before update) {
if (CaseTriggerHandler.firstRun) {
CaseTriggerHandler.handle(Trigger.new);
}

CaseTriggerHandler.firstRun = false;
}
```

**E.** ```java
public class CaseTriggerHandler {
Boolean firstRun = true;
public static void handle(List<Case> cases) {
if (firstRun) {
for (Case c : cases) {
**F.** Edit_Count__c = c.Edit_Count__c + 1;
}
}
firstRun = false;
}
}
trigger on Case(before update) {
CaseTriggerHandler.handle(Trigger.new);
}
```

G. ```java
trigger on Case(before update) {
Boolean firstRun = true;
if (firstRun) {
CaseTriggerHandler.handle(Trigger.newMap);
}
firstRun = false;
}
```

**Answer: B (LEAVE A REPLY)**

This scenario illustrates a classic trigger recursion issue triggered by the Salesforce Order of Execution. When a record is updated, Salesforce runs through a specific sequence: first, it executes "Before-Save" Record- Triggered Flows, then "Before" triggers, followed by "After" triggers, and finally "After-Save" flows and processes. If a process or flow updates the same record that initiated the transaction, the entire cycle of Apex triggers can be re-invoked within that single transaction.

To prevent logic from running multiple times during these re-entrant cycles, developers implement a static boolean variable as a "recursion guard." Static variables in Apex persist for the entire duration of a single transaction. By checking the value of a static boolean at the start of the trigger, the code can determine if it has already processed the current set of records.
```

Option B is the correct implementation of this pattern. It defines `public static Boolean firstRun = true;` in the handler class. The trigger checks if `firstRun` is true, executes the handler logic, and then immediately sets
`firstRun` to false. Any subsequent execution of the Case trigger within the same transaction will find the variable set to false and skip the increment logic.
Option A is incorrect because it resets `firstRun` to true every time the trigger starts, defeating the guard.
Option C incorrectly uses an instance variable (non-static), which is recreated for every call. Option D uses a local variable within the trigger body, which is re-initialized to true every time the trigger fires, providing no protection against recursion.

Valid PDII-JPN Dumps shared by Actual4test.com for Helping Passing PDII-JPN Exam! Actual4test.com now offer the **newest PDII-JPN exam dumps**, the Actual4test.com PDII-JPN exam **questions have been updated** and **answers have been corrected** get the **newest** Actual4test.com PDII-JPN dumps with Test Engine here: https://www.actual4test.com/PDII-JPN_examcollection.html (**163** Q&As Dumps, **30%OFF Special Discount: Freepdfdumps**)