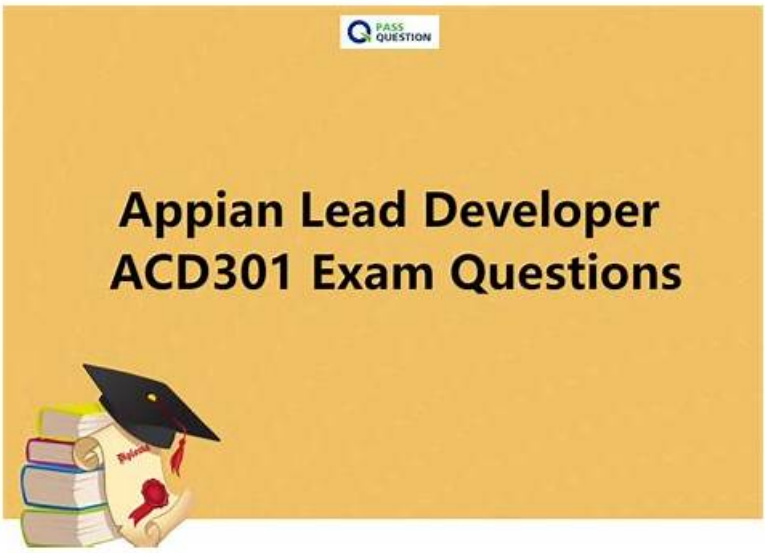
Quiz Appian - Useful ACD301 - Appian Lead Developer Demo Test



BTW, DOWNLOAD part of Prep4sureExam ACD301 dumps from Cloud Storage: https://drive.google.com/open?id=1iueF4e61__blAh3CosACv4uSBNCNwnar

Obtaining a certificate may be not an easy thing for some candidates, choose us, we will help you get the certificate easily. ACD301 learning materials are edited by experienced experts, therefore the quality and accuracy can be guaranteed. In addition, ACD301 exam braindumps contact most of knowledge points for the exam, and you can mater the major knowledge points well by practicing. In order to improve your confidence to ACD301 Exam Materials, we are pass guarantee and money back guarantee. If you fail to pass the exam by using ACD301 exam materials, we will give you full refund.

Appian ACD301 Exam Syllabus Topics:

Topic	Details
Topic 1	• Extending Appian: This section of the exam measures skills of Integration Specialists and covers building and troubleshooting advanced integrations using connected systems and APIs. Candidates are expected to work with authentication, evaluate plug-ins, develop custom solutions when needed, and utilize document generation options to extend the platform's capabilities.
Topic 2	• Application Design and Development: This section of the exam measures skills of Lead Appian Developers and covers the design and development of applications that meet user needs using Appian functionality. It includes designing for consistency, reusability, and collaboration across teams. Emphasis is placed on applying best practices for building multiple, scalable applications in complex environments.
Topic 3	• Data Management: This section of the exam measures skills of Data Architects and covers analyzing, designing, and securing data models. Candidates must demonstrate an understanding of how to use Appian's data fabric and manage data migrations. The focus is on ensuring performance in high-volume data environments, solving data-related issues, and implementing advanced database features effectively.

>> ACD301 Demo Test <<

Latest ACD301 Study Plan, ACD301 Actual Exam

We regard the customer as king so we put a high emphasis on the trust of every users, therefore our security system can protect you both in payment of ACD301 guide braindumps and promise that your computer will not be infected during the process of payment

on our ACD301 Study Materials. Moreover, if you end up the cooperation between us,we have the responsibility to delete your personal information on ACD301 exam prep. In a word, Wwe have data protection act for you to avoid information leakage!

Appian Lead Developer Sample Questions (Q44-Q49):

NEW QUESTION # 44
You are required to configure a connection so that Jira can inform Appian when specific tickets change (using a webhook). Which three required steps will allow you to connect both systems?

- A. Give the service account system administrator privileges.
- B. Create a new API Key and associate a service account.
- C. Configure the connection in Jira specifying the URL and credentials.
- D. Create an integration object from Appian to Jira to periodically check the ticket status.
- E. Create a Web API object and set up the correct security.

Answer: B,C,E

Explanation:
Comprehensive and Detailed In-Depth Explanation:Configuring a webhook connection from Jira to Appian requires setting up a mechanism for Jira to push ticket change notifications to Appian in real-time.
This involves creating an endpoint in Appian to receive the webhook and configuring Jira to send the data.
Appian's Integration Best Practices and Web API documentation provide the framework for this process.
* Option A (Create a Web API object and set up the correct security):This is a required step. In Appian, a Web API object serves as the endpoint to receive incoming webhook requests from Jira. You must define the API structure (e.g., HTTP method, input parameters) and configure security (e.g., basic authentication, API key, or OAuth) to validate incoming requests. Appian recommends using a service account with appropriate permissions to ensure secure access, aligning with the need for a controlled webhook receiver.
* Option B (Configure the connection in Jira specifying the URL and credentials):This is essential.
In Jira, you need to set up a webhook by providing the Appian Web API's URL (e.g., https://<appian- site>/suite/webapi/<web-api-name>) and the credentials or authentication method (e.g., API key or basic auth) that match the security setup in Appian. This ensures Jira can successfully send ticket change events to Appian.
* Option C (Create a new API Key and associate a service account):This is necessary for secure authentication. Appian recommends using an API key tied to a service account for webhook integrations. The service account should have permissions to process the incoming data (e.g., write to a process or data store) but not excessive privileges. This step complements the Web API security setup and Jira configuration.
* Option D (Give the service account system administrator privileges):This is unnecessary and insecure. System administrator privileges grant broad access, which is overkill for a webhook integration. Appian's security best practices advocate for least-privilege principles, limiting the service account to the specific objects or actions needed (e.g., executing the Web API).
* Option E (Create an integration object from Appian to Jira to periodically check the ticket status):This is incorrect for a webhook scenario. Webhooks are push-based, where Jira notifies Appian of changes. Creating an integration object for periodic polling (pull-based) is a different approach and not required for the stated requirement of Jira informing Appian via webhook.
These three steps (A, B, C) establish a secure, functional webhook connection without introducing unnecessary complexity or security risks.
References:Appian Documentation - Web API Configuration, Appian Integration Best Practices - Webhooks, Appian Lead Developer Training - External System Integration.
The three required steps that will allow you to connect both systems are:
* A. Create a Web API object and set up the correct security. This will allow you to define an endpoint in Appian that can receive requests from Jira via webhook. You will also need to configure the security settings for the Web API object, such as authentication method, allowed origins, and access control.
* B. Configure the connection in Jira specifying the URL and credentials. This will allow you to set up a webhook in Jira that can send requests to Appian when specific tickets change. You will need to specify the URL of the Web API object in Appian, as well as any credentials required for authentication.
* C. Create a new API Key and associate a service account. This will allow you to generate a unique token that can be used for authentication between Jira and Appian. You will also need to create a service account in Appian that has permissions to access or update data related to Jira tickets.
The other options are incorrect for the following reasons:
* D. Give the service account system administrator privileges. This is not required and could pose a security risk, as giving system administrator privileges to a service account could allow it to perform actions that are not related to Jira tickets, such as modifying system settings or accessing sensitive data.
* E. Create an integration object from Appian to Jira to periodically check the ticket status. This is not required and could cause unnecessary overhead, as creating an integration object from Appian to Jira would involve polling Jira for ticket status changes,

which could consume more resources than using webhook notifications. Verified References: Appian Documentation, section "Web API" and "API Keys".

NEW QUESTION # 45
Your Agile Scrum project requires you to manage two teams, with three developers per team. Both teams are to work on the same application in parallel. How should the work be divided between the teams, avoiding issues caused by cross-dependency?

- A. Allocate stories to each team based on the cumulative years of experience of the team members.
- B. Have each team choose the stories they would like to work on based on personal preference.
- C. Group epics and stories by technical difficulty, and allocate one team the more challenging stories.
- D. Group epics and stories by feature, and allocate work between each team by feature.

Answer: D

Explanation:
Comprehensive and Detailed In-Depth Explanation:In an Agile Scrum environment with two teams working on the same application in parallel, effective work division is critical to avoid cross-dependency, which can lead to delays, conflicts, and inefficiencies. Appian's Agile Development Best Practices emphasize team autonomy and minimizing dependencies to ensure smooth progress.
* Option B (Group epics and stories by feature, and allocate work between each team by feature):
This is the recommended approach. By dividing the application's functionality into distinct features (e.
g., Team 1 handles customer management, Team 2 handles campaign tracking), each team can work independently on a specific domain. This reduces cross-dependency because teams are not reliant on each other's deliverables within a sprint. Appian's guidance on multi-team projects suggests feature- based partitioning as a best practice, allowing teams to own their backlog items, design, and testing without frequent coordination. For example, Team 1 can develop and test customer-related interfaces while Team 2 works on campaign processes, merging their work during integration phases.
* Option A (Group epics and stories by technical difficulty, and allocate one team the more challenging stories):This creates an imbalance, potentially overloading one team and underutilizing the other, which can lead to morale issues and uneven progress. It also doesn't address cross-dependency, as challenging stories might still require input from both teams (e.g., shared data models), increasing coordination needs.
* Option C (Allocate stories to each team based on the cumulative years of experience of the team members):Experience-based allocation ignores the project's functional structure and can result in mismatched skills for specific features. It also risks dependencies if experienced team members are needed across teams, complicating parallel work.
* Option D (Have each team choose the stories they would like to work on based on personal preference):This lacks structure and could lead to overlap, duplication, or neglect of critical features. It increases the risk of cross-dependency as teams might select interdependent stories without coordination, undermining parallel development.
Feature-based division aligns with Scrum principles of self-organization and minimizes dependencies, making it the most effective strategy for this scenario.
References:Appian Documentation - Agile Development with Appian, Scrum Guide - Multi-Team Coordination, Appian Lead Developer Training - Team Management Strategies.

NEW QUESTION # 46
You need to generate a PDF document with specific formatting. Which approach would you recommend?

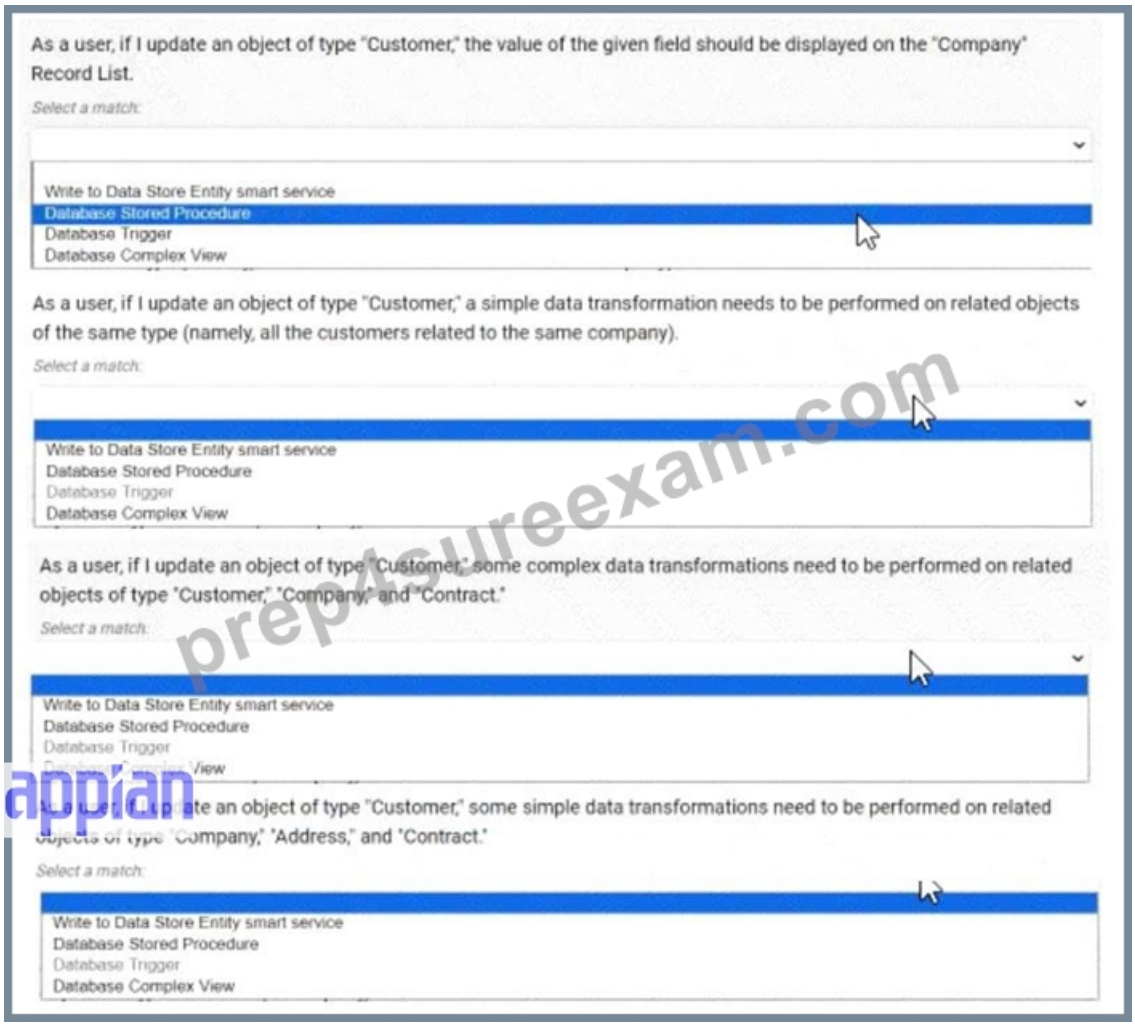
- A. Use the PDF from XSL-FO Transformation smart service to generate the content with the specific format.
- B. Create an embedded interface with the necessary content and ask the user to use the browser "Print" functionality to save it as a PDF.
- C. Use the Word Doc from Template smart service in a process model to add the specific format.
- D. There is no way to fulfill the requirement using Appian. Suggest sending the content as a plain email instead.

Answer: A

Explanation:
Comprehensive and Detailed In-Depth Explanation:As an Appian Lead Developer, generating a PDF with specific formatting is a common requirement, and Appian provides several tools to achieve this. The question emphasizes "specific formatting," which implies precise control over layout, styling, and content structure.
Let's evaluate each option based on Appian's official documentation and capabilities:
* A. Create an embedded interface with the necessary content and ask the user to use the browser "Print" functionality to save it as a PDF:This approach involves designing an interface (e.g., using SAIL components) and relying on the browser's native print-to-PDF feature. While this is feasible for simple content, it lacks precision for "specific formatting." Browser rendering varies across

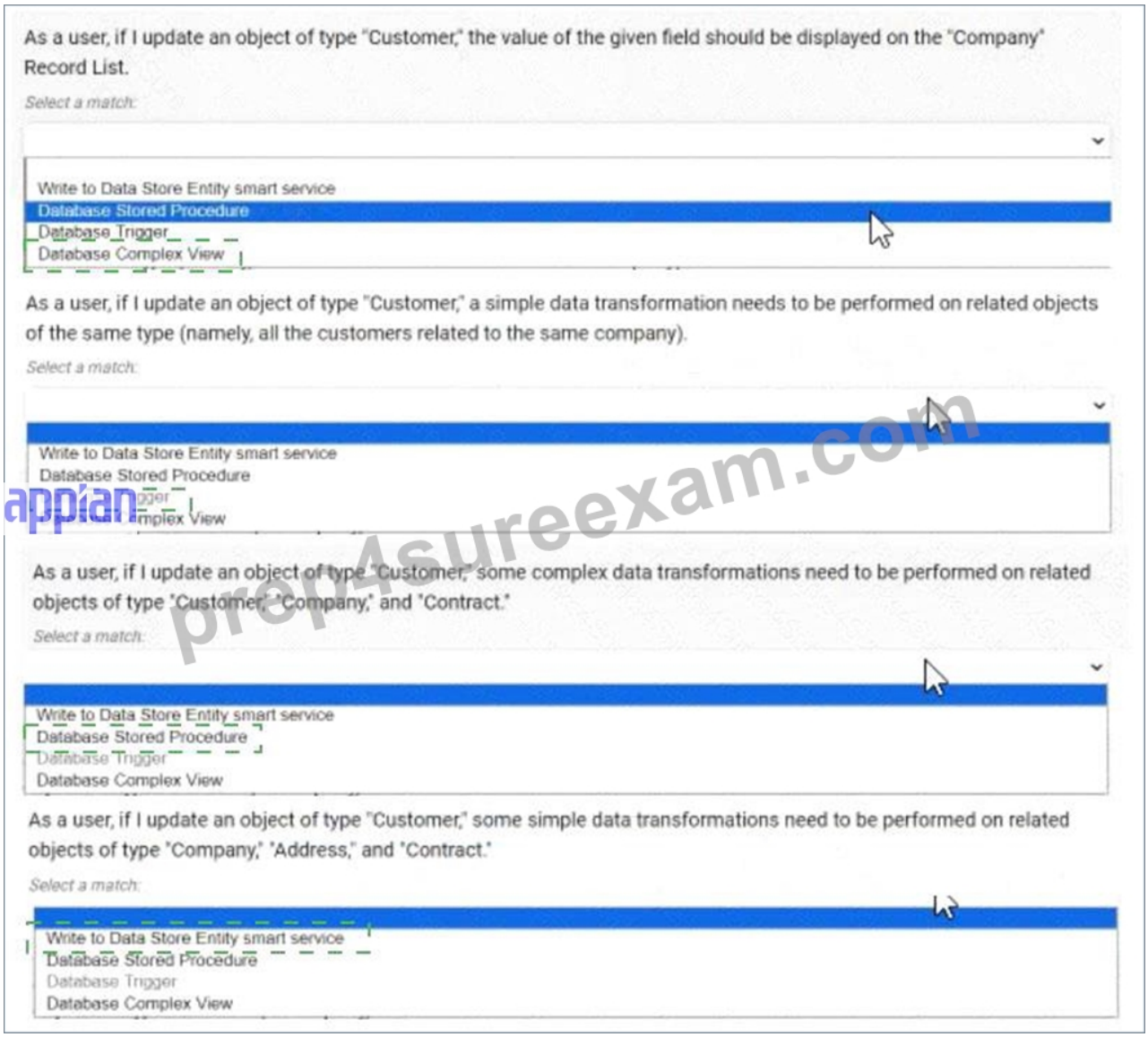
devices and browsers, and print styles (e.g., CSS) are limited in Appian's control. Appian Lead Developer best practices discouragerelying on client-side functionality for critical document generation due to inconsistency and lack of automation. This is not a recommended solution for a production-grade requirement.
* B. Use the PDF from XSL-FO Transformation smart service to generate the content with the specific format:This is the correct choice. The "PDF from XSL-FO Transformation" smart service (available in Appian's process modeling toolkit) allows developers to generate PDFs programmatically with precise formatting using XSL-FO (Extensible Stylesheet Language Formatting Objects). XSL-FO provides fine- grained control over layout, fonts, margins, and styling-ideal for "specific formatting" requirements.
In a process model, you can pass XML data and an XSL-FO stylesheet to this smart service, producing a downloadable PDF. Appian's documentation highlights this as the preferred method for complex PDF generation, making it a robust, scalable, and Appian-native solution.
* C. Use the Word Doc from Template smart service in a process model to add the specific format:This option uses the "Word Doc from Template" smart service to generate a Microsoft Word document from a template (e.g., a .docx file with placeholders). While it supports formatting defined in the template and can be converted to PDF post-generation (e.g., via a manual step or external tool), it's not a direct PDF solution. Appian doesn't natively convert Word to PDF within the platform, requiring additional steps outside the process model. For "specific formatting" in a PDF, this is less efficient and less precise than the XSL-FO approach, as Word templates are better suited for editable documents rather than final PDFs.
* D. There is no way to fulfill the requirement using Appian. Suggest sending the content as a plain email instead:This is incorrect. Appian provides multiple tools for document generation, including PDFs, as evidenced by options B and C. Suggesting a plain email fails to meet the requirement of generating a formatted PDF and contradicts Appian's capabilities. Appian Lead Developer training emphasizes leveraging platform features to meet business needs, ruling out this option entirely.
Conclusion: The PDF from XSL-FO Transformation smart service (B) is the recommended approach. It provides direct PDF generation with specific formatting control within Appian's process model, aligning with best practices for document automation and precision. This method is scalable, repeatable, and fully supported by Appian's architecture.
References:
* Appian Documentation: "PDF from XSL-FO Transformation Smart Service" (Process Modeling > Smart Services).
* Appian Lead Developer Certification: Document Generation Module (PDF Generation Techniques).
* Appian Best Practices: "Generating Documents in Appian" (XSL-FO vs. Template-Based Approaches).

NEW QUESTION # 47
For each scenario outlined, match the best tool to use to meet expectations. Each tool will be used once Note: To change your responses, you may deselected your response by clicking the blank space at the top of the selection list.



Answer:

Explanation:



Explanation:

* As a user, if I update an object of type "Customer", the value of the given field should be displayed on the "Company" Record List. # Database Complex View
* As a user, if I update an object of type "Customer", a simple data transformation needs to be performed on related objects of the same type (namely, all the customers related to the same company). # Database Trigger
* As a user, if I update an object of type "Customer", some complex data transformations need to be performed on related objects of type "Customer", "Company", and "Contract". # Database Stored Procedure
* As a user, if I update an object of type "Customer", some simple data transformations need to be performed on related objects of type "Company", "Address", and "Contract". # Write to Data Store Entity smart service Comprehensive and Detailed In-Depth Explanation:Appian integrates with external databases to handle data updates and transformations, offering various tools depending on the complexity and context of the task.

The scenarios involve updating a "Customer" object and triggering actions on related data, requiring careful selection of the best tool. Appian's Data Integration and Database Management documentation guides these decisions.

* As a user, if I update an object of type "Customer", the value of the given field should be displayed on the "Company" Record List # Database Complex View:This scenario requires displaying updated customer data on a "Company" Record List, implying a read-only operation to join or aggregate data across tables. A Database Complex View (e.g., a SQL view combining "Customer" and "Company" tables) is ideal for this. Appian supports complex views to predefine queries that can be used in Record Lists, ensuring the updated field value is reflected without additional processing. This tool is best for read operations and does not involve write logic.
* As a user, if I update an object of type "Customer", a simple data transformation needs to be performed on related objects of the same type (namely, all the customers related to the same company) # Database Trigger:This involves a simple transformation (e.g., updating a flag or counter) on related "Customer" records after an update. A Database Trigger, executed automatically on the database side when a "Customer" record is modified, is the best fit. It can perform lightweight SQL updates on related records (e.g., via a company ID join) without Appian process overhead. Appian recommends triggers for simple, database-level automation, especially when transformations are confined to the same table type.

* As a user, if I update an object of type "Customer", some complex data transformations need to be performed on related objects of type "Customer", "Company", and "Contract" # Database Stored Procedure:This scenario involves complex transformations across multiple related object types, suggesting multi-step logic (e.g., recalculating totals or updating multiple tables). A Database Stored Procedure allows you to encapsulate this logic in SQL, callable from Appian, offering flexibility for complex operations. Appian supports stored procedures for scenarios requiring transactional integrity and intricate data manipulation across tables, making it the best choice here.
* As a user, if I update an object of type "Customer", some simple data transformations need to be performed on related objects of type "Company", "Address", and "Contract" # Write to Data Store Entity smart service:This requires simple transformations on related objects, which can be handled within Appian's process model. The "Write to Data Store Entity" smart service allows you to update multiple related entities (e.g., "Company", "Address", "Contract") based on the "Customer" update, using Appian's expression rules for logic. This approach leverages Appian's process automation, is user-friendly for developers, and is recommended for straightforward updates within the Appian environment.
Matching Rationale:
* Each tool is used once, covering the spectrum of database integration options: Database Complex View for read/display, Database Trigger for simple database-side automation, Database Stored Procedure for complex multi-table logic, and Write to Data Store Entity smart service for Appian-managed simple updates.
* Appian's guidelines prioritize using the right tool based on complexity and context, ensuring efficiency and maintainability.
References:Appian Documentation - Data Integration and Database Management, Appian Process Model Guide - Smart Services, Appian Lead Developer Training - Database Optimization.

NEW QUESTION # 48
As part of your implementation workflow, users need to retrieve data stored in a third-party Oracle database on an interface. You need to design a way to query this information.
How should you set up this connection and query the data?

- A. Configure a timed utility process that queries data from the third-party database daily, and stores it in the Appian business database. Then use a!queryEntity using the Appian data source to retrieve the data.
- B. Configure a Query Database node within the process model. Then, type in the connection information, as well as a SQL query to execute and return the data in process variables.
- C. Configure an expression-backed record type, calling an API to retrieve the data from the third-party database. Then, use a!queryRecordType to retrieve the data.
- D. In the Administration Console, configure the third-party database as a "New Data Source." Then, use a!queryEntity to retrieve the data.

Answer: D

Explanation:
Comprehensive and Detailed In-Depth Explanation:
As an Appian Lead Developer, designing a solution to query data from a third-party Oracle database for display on an interface requires secure, efficient, and maintainable integration. The scenario focuses on real-time retrieval for users, so the design must leverage Appian's data connectivity features. Let's evaluate each option:
A . Configure a Query Database node within the process model. Then, type in the connection information, as well as a SQL query to execute and return the data in process variables:
The Query Database node (part of the Smart Services) allows direct SQL execution against a database, but it requires manual connection details (e.g., JDBC URL, credentials), which isn't scalable or secure for Production. Appian's documentation discourages using Query Database for ongoing integrations due to maintenance overhead, security risks (e.g., hardcoding credentials), and lack of governance. This is better for one-off tasks, not real-time interface queries, making it unsuitable.
B . Configure a timed utility process that queries data from the third-party database daily, and stores it in the Appian business database. Then use a!queryEntity using the Appian data source to retrieve the data:
This approach syncs data daily into Appian's business database (e.g., via a timer event and Query Database node), then queries it with a!queryEntity. While it works for stale data, it introduces latency (up to 24 hours) for users, which doesn't meet real-time needs on an interface. Appian's best practices recommend direct data source connections for up-to-date data, not periodic caching, unless latency is acceptable-making this inefficient here.
C . Configure an expression-backed record type, calling an API to retrieve the data from the third-party database. Then, use a!queryRecordType to retrieve the data:
Expression-backed record types use expressions (e.g., a!httpQuery()) to fetch data, but they're designed for external APIs, not direct database queries. The scenario specifies an Oracle database, not an API, so this requires building a custom REST service on the Oracle side, adding complexity and latency. Appian's documentation favors Data Sources for database queries over API calls when direct access is available, making this less optimal and over-engineered.
D . In the Administration Console, configure the third-party database as a "New Data Source." Then, use a!queryEntity to retrieve

the data:
This is the best choice. In the Appian Administration Console, you can configure a JDBC Data Source for the Oracle database, providing connection details (e.g., URL, driver, credentials). This creates a secure, managed connection for querying via a!queryEntity, which is Appian's standard function for Data Store Entities. Users can then retrieve data on interfaces using expression-backed records or queries, ensuring real-time access with minimal latency. Appian's documentation recommends Data Sources for database integrations, offering scalability, security, and governance-perfect for this requirement.
Conclusion: Configuring the third-party database as a New Data Source and using a!queryEntity (D) is the recommended approach. It provides direct, real-time access to Oracle data for interface display, leveraging Appian's native data connectivity features and aligning with Lead Developer best practices for third-party database integration.
Reference:
Appian Documentation: "Configuring Data Sources" (JDBC Connections and a!queryEntity).
Appian Lead Developer Certification: Data Integration Module (Database Query Design).
Appian Best Practices: "Retrieving External Data in Interfaces" (Data Source vs. API Approaches).

NEW QUESTION # 49
.....

We believe that getting the newest information about the exam will help all customers pass the ACD301 exam easily. If you purchase our study materials, you will have the opportunity to get the newest information about the ACD301 exam. More importantly, the updating system of our company is free for all customers. If you decide to buy and use the ACD301 Training Materials from our company, it will be very easy for you to pass the exam without doubt. We sincerely hope that you can achieve your dream in the near future by the ACD301 latest questions of our company.