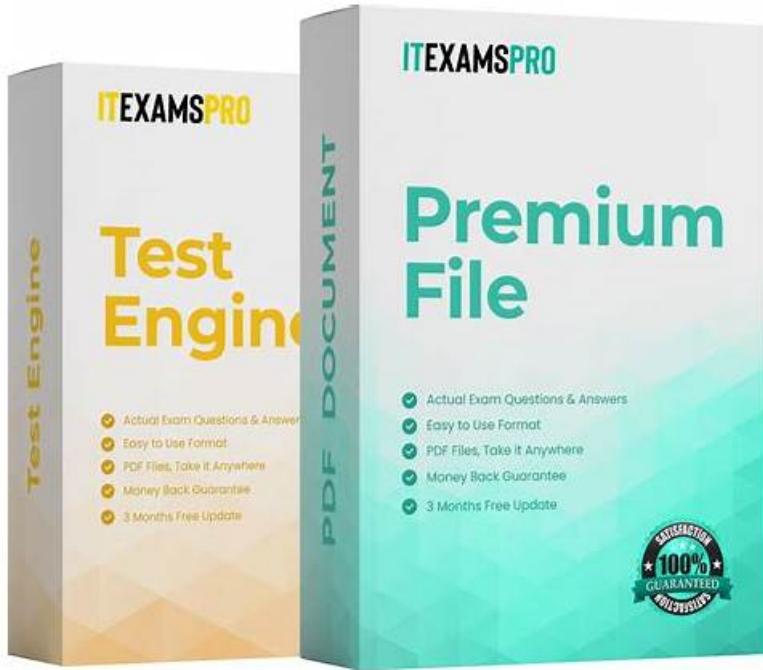


Reliable ACD301 Test Sims & ACD301 Trustworthy Source



P.S. Free & New ACD301 dumps are available on Google Drive shared by DumpStillValid: <https://drive.google.com/open?id=16VMXM9VVU8O0xB9ViRNbIwrI0nKlgh9>

You can write down your doubts or any other question of our Appian Lead Developer test questions. We warmly welcome all your questions. Our online workers are responsible for solving all your problems with twenty four hours service. You still can enjoy our considerate service after you have purchased our ACD301 test guide. If you don't know how to install the study materials, our professional experts can offer you remote installation guidance. Also, we will offer you help in the process of using our ACD301 Exam Questions. Also, if you have better suggestions to utilize our study materials, we will be glad to take it seriously.

For a company with history more than ten years, our ACD301 practice materials have developed into fully academic maturity. All content are arranged legibly. There are three kinds of ACD301 exam braindumps for your reference: the PDF, the Software and the APP online. All these versions of our ACD301 study questions are high-efficient. You can choose either one in accordance with your interests or habits.

>> Reliable ACD301 Test Sims <<

ACD301 Trustworthy Source | Test ACD301 Collection Pdf

Our ACD301 practice materials from our company are invulnerable. And we are consigned as the most responsible company in this area. So many competitors concede our superior position in the market. Besides, we offer some promotional benefits for you. The more times you choose our ACD301 Training Materials, the more benefits you can get, such as free demos of our ACD301 exam dumps, three-version options, rights of updates and so on. So customer orientation is the beliefs we honor.

Appian ACD301 Exam Syllabus Topics:

Topic	Details
Topic 1	<ul style="list-style-type: none">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	<ul style="list-style-type: none"> Platform Management: This section of the exam measures skills of Appian System Administrators and covers the ability to manage platform operations such as deploying applications across environments, troubleshooting platform-level issues, configuring environment settings, and understanding platform architecture. Candidates are also expected to know when to involve Appian Support and how to adjust admin console configurations to maintain stability and performance.
Topic 3	<ul style="list-style-type: none"> 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 4	<ul style="list-style-type: none"> Proactively Design for Scalability and Performance: This section of the exam measures skills of Application Performance Engineers and covers building scalable applications and optimizing Appian components for performance. It includes planning load testing, diagnosing performance issues at the application level, and designing systems that can grow efficiently without sacrificing reliability.

Appian Lead Developer Sample Questions (Q12-Q17):

NEW QUESTION # 12

You are designing a process that is anticipated to be executed multiple times a day. This process retrieves data from an external system and then calls various utility processes as needed. The main process will not use the results of the utility processes, and there are no user forms anywhere.

Which design choice should be used to start the utility processes and minimize the load on the execution engines?

- A. Start the utility processes via a subprocess asynchronously.
- B. Use Process Messaging to start the utility process.
- C. Start the utility processes via a subprocess synchronously.
- D. Use the Start Process Smart Service to start the utility processes.

Answer: A

Explanation:

Comprehensive and Detailed In-Depth Explanation: As an Appian Lead Developer, designing a process that executes frequently (multiple times a day) and calls utility processes without using their results requires optimizing performance and minimizing load on Appian's execution engines. The absence of user forms indicates a backend process, so user experience isn't a concern-only engine efficiency matters. Let's evaluate each option:

* A. Use the Start Process Smart Service to start the utility processes: The Start Process Smart Service launches a new process instance independently, creating a separate process in the Work Queue. While functional, it increases engine load because each utility process runs as a distinct instance, consuming engine resources and potentially clogging the Java Work Queue, especially with frequent executions.

Appian's performance guidelines discourage unnecessary separate process instances for utility tasks, favoring integrated subprocesses, making this less optimal.

* B. Start the utility processes via a subprocess synchronously: Synchronous subprocesses (e.g., a!startProcess with isAsync: false) execute within the main process flow, blocking until completion. For utility processes not used by the main process, this creates unnecessary delays, increasing execution time and engine load. With frequent daily executions, synchronous subprocesses could strain engines, especially if utility processes are slow or numerous. Appian's documentation recommends asynchronous execution for non-dependent, non-blocking tasks, ruling this out.

* C. Use Process Messaging to start the utility process: Process Messaging (e.g., sendMessage() in Appian) is used for inter-process communication, not for starting processes. It's designed to pass data between running processes, not initiate new ones. Attempting to use it for starting utility processes would require additional setup (e.g., a listening process) and isn't a standard or efficient method. Appian's messaging features are for coordination, not process initiation, making this inappropriate.

* D. Start the utility processes via a subprocess asynchronously: This is the best choice. Asynchronous subprocesses (e.g., a!startProcess with isAsync: true) execute independently of the main process, offloading work to the engine without blocking or delaying the parent process. Since the main process doesn't use the utility process results and there are no user forms, asynchronous execution minimizes engine load by distributing tasks across time, reducing Work Queue pressure during frequent executions.

Appian's performance best practices recommend asynchronous subprocesses for non-dependent, utility tasks to optimize engine utilization, making this ideal for minimizing load.

Conclusion: Starting the utility processes via a subprocess asynchronously (D) minimizes engine load by allowing independent execution without blocking the main process, aligning with Appian's performance optimization strategies for frequent, backend

processes.

References:

- * Appian Documentation: "Process Model Performance" (Synchronous vs. Asynchronous Subprocesses).
- * Appian Lead Developer Certification: Process Design Module (Optimizing Engine Load).
- * Appian Best Practices: "Designing Efficient Utility Processes" (Asynchronous Execution).

NEW QUESTION # 13

You have created a Web API in Appian with the following URL to call it:

https://exampleappiancloud.com/suite/webapi/user_management/users?username=john.smith. Which is the correct syntax for referring to the username parameter?

- A. `httpRequest.queryParameters.users.username`
- B. `httpRequest.users.username`
- C. `httpRequest.formData.username`
- D. `httpRequest.queryParameters.username`

Answer: D

Explanation:

Comprehensive and Detailed In-Depth Explanation:

In Appian, when creating a Web API, parameters passed in the URL (e.g., query parameters) are accessed within the Web API expression using the `httpRequest` object. The URL https://exampleappiancloud.com/suite/webapi/user_management/users?username=john.smith includes a query parameter `username` with the value `john.smith`. Appian's Web API documentation specifies how to handle such parameters in the expression rule associated with the Web API.

Option D (`httpRequest.queryParameters.username`):

This is the correct syntax. The `httpRequest.queryParameters` object contains all query parameters from the URL. Since `username` is a single query parameter, you access it directly as `httpRequest.queryParameters.username`. This returns the value `john.smith` as a text string, which can then be used in the Web API logic (e.g., to query a user record). Appian's expression language treats query parameters as key-value pairs under `queryParameters`, making this the standard approach.

Option A (`httpRequest.queryParameters.users.username`):

This is incorrect. The `users` part suggests a nested structure (e.g., `users` as a parameter containing a `username` subfield), which does not match the URL. The URL only defines `username` as a top-level query parameter, not a nested object.

Option B (`httpRequest.users.username`):

This is invalid. The `httpRequest` object does not have a direct `users` property. Query parameters are accessed via `queryParameters`, and there's no indication of a `users` object in the URL or Appian's Web API model.

Option C (`httpRequest.formData.username`):

This is incorrect. The `httpRequest.formData` object is used for parameters passed in the body of a POST or PUT request (e.g., form submissions), not for query parameters in a GET request URL. Since the `username` is part of the query string (`?username=john.smith`), `formData` does not apply.

The correct syntax leverages Appian's standard handling of query parameters, ensuring the Web API can process the `username` value effectively.

NEW QUESTION # 14

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

Answer: A,B,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 # 15

You are asked to design a case management system for a client. In addition to storing some basic metadata about a case, one of the client's requirements is the ability for users to update a case. The client would like any user in their organization of 500 people to be able to make these updates. The users are all based in the company's headquarters, and there will be frequent cases where users are attempting to edit the same case. The client wants to ensure no information is lost when these edits occur and does not want the solution to burden their process administrators with any additional effort. Which data locking approach should you recommend?

- A. Design a process report and query to determine who opened the edit form first.
- B. Use the database to implement low-level pessimistic locking.
- C. Allow edits without locking the case CDI.
- D. Add an `@Version` annotation to the case CDT to manage the locking.

Answer: D

Explanation:

Comprehensive and Detailed In-Depth Explanation:

The requirement involves a case management system where 500 users may simultaneously edit the same case, with a need to prevent data loss and minimize administrative overhead. Appian's data management and concurrency control strategies are critical here,

especially when integrating with an underlying database.

Option C (Add an @Version annotation to the case CDT to manage the locking):

This is the recommended approach. In Appian, the @Version annotation on a Custom Data Type (CDT) enables optimistic locking, a lightweight concurrency control mechanism. When a user updates a case, Appian checks the version number of the CDT instance. If another user has modified it in the meantime, the update fails, prompting the user to refresh and reapply changes. This prevents data loss without requiring manual intervention by process administrators. Appian's Data Design Guide recommends @Version for scenarios with high concurrency (e.g., 500 users) and frequent edits, as it leverages the database's native versioning (e.g., in MySQL or PostgreSQL) and integrates seamlessly with Appian's process models. This aligns with the client's no-burden requirement.

Option A (Allow edits without locking the case CDI):

This is risky. Without locking, simultaneous edits could overwrite each other, leading to data loss-a direct violation of the client's requirement. Appian does not recommend this for collaborative environments.

Option B (Use the database to implement low-level pessimistic locking):

Pessimistic locking (e.g., using SELECT ... FOR UPDATE in MySQL) locks the record during the edit process, preventing other users from modifying it until the lock is released. While effective, it can lead to deadlocks or performance bottlenecks with 500 users, especially if edits are frequent. Additionally, managing this at the database level requires custom SQL and increases administrative effort (e.g., monitoring locks), which the client wants to avoid. Appian prefers higher-level solutions like @Version over low-level database locking.

Option D (Design a process report and query to determine who opened the edit form first):

This is impractical and inefficient. Building a custom report and query to track form opens adds complexity and administrative overhead. It doesn't inherently prevent data loss and relies on manual resolution, conflicting with the client's requirements.

The @Version annotation provides a robust, Appian-native solution that balances concurrency, data integrity, and ease of maintenance, making it the best fit.

NEW QUESTION # 16

You are just starting with a new team that has been working together on an application for months. They ask you to review some of their views that have been degrading in performance. The views are highly complex with hundreds of lines of SQL. What is the first step in troubleshooting the degradation?

- A. Go through the entire database structure to obtain an overview, ensure you understand the business needs, and then normalize the tables to optimize performance.
- B. Run an explain statement on the views, identify critical areas of improvement that can be remediated without business knowledge.
- C. Go through all of the tables one by one to identify which of the grouped by, ordered by, or joined keys are currently indexed.
- D. Browse through the tables, note any tables that contain a large volume of null values, and work with your team to plan for table restructure.

Answer: B

Explanation:

Comprehensive and Detailed In-Depth Explanation:

Troubleshooting performance degradation in complex SQL views within an Appian application requires a systematic approach. The views, described as having hundreds of lines of SQL, suggest potential issues with query execution, indexing, or join efficiency. As a new team member, the first step should focus on quickly identifying the root cause without overhauling the system prematurely. Appian's Performance Troubleshooting Guide and database optimization best practices provide the framework for this process. Option B (Run an explain statement on the views, identify critical areas of improvement that can be remediated without business knowledge):

This is the recommended first step. Running an EXPLAIN statement (or equivalent, such as EXPLAIN PLAN in some databases) analyzes the query execution plan, revealing details like full table scans, missing indices, or inefficient joins. This technical analysis can identify immediate optimization opportunities (e.g., adding indices or rewriting subqueries) without requiring business input, allowing you to address low-hanging fruit quickly. Appian encourages using database tools to diagnose performance issues before involving stakeholders, making this a practical starting point as you familiarize yourself with the application.

Option A (Go through the entire database structure to obtain an overview, ensure you understand the business needs, and then normalize the tables to optimize performance):

This is too broad and time-consuming as a first step. Understanding business needs and normalizing tables are valuable but require collaboration with the team and stakeholders, delaying action. It's better suited for a later phase after initial technical analysis.

Option C (Go through all of the tables one by one to identify which of the grouped by, ordered by, or joined keys are currently indexed):

Manually checking indices is useful but inefficient without first knowing which queries are problematic. The EXPLAIN statement provides targeted insights into index usage, making it a more direct initial step than a manual table-by-table review.

Option D (Browse through the tables, note any tables that contain a large volume of null values, and work with your team to plan for table restructure):

Identifying null values and planning restructures is a long-term optimization strategy, not a first step. It requires team input and may not address the immediate performance degradation, which is better tackled with query-level diagnostics.

Starting with an EXPLAIN statement allows you to gather data-driven insights, align with Appian's performance troubleshooting methodology, and proceed with informed optimizations.

NEW QUESTION # 17

If you are now determined to go to research, there is still a little hesitation in product selection. ACD301 exam prep offers you a free trial version! You can choose one or more versions that you are most interested in, and then use your own judgment. ACD301 Exam Materials really hope that every user can pick the right ACD301 study guide for them. If you really lack experience, you do not know which one to choose. You can consult our professional staff.

ACD301 Trustworthy Source: <https://www.dumpstillvalid.com/ACD301-prep4sure-review.html>

BONUS!!! Download part of DumpStillValid ACD301 dumps for free: <https://drive.google.com/open?id=16VMXM9VVU8O0xB9ViVrnB1wrl0nKlgh9>