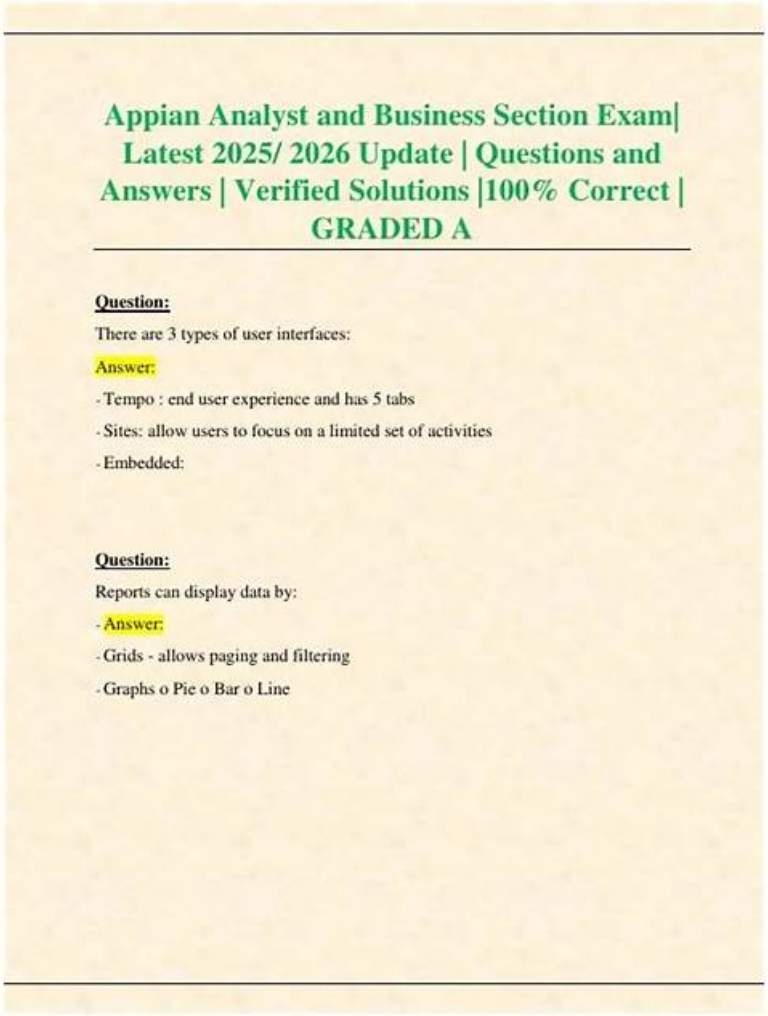
100% Pass 2026 The Best Appian ACD301 Latest Test Simulations



Appian Analyst and Business Section Exam| Latest 2025/ 2026 Update | Questions and Answers | Verified Solutions |100% Correct | GRADED A

Question:

There are 3 types of user interfaces:

Answer:

- Tempo : end user experience and has 5 tabs
- Sites: allow users to focus on a limited set of activities
- Embedded:

Question:

Reports can display data by:

- Answer:
- Grids - allows paging and filtering
- Graphs o Pie o Bar o Line

P.S. Free 2025 Appian ACD301 dumps are available on Google Drive shared by SureTorrent: https://drive.google.com/open?id=19siu-uUMSzd4uLyFGDZUbh7_JWkldHaY

SureTorrent is one of the most reliable platforms to get actual Appian ACD301 dumps. It offers the latest and valid real Appian Lead Developer (ACD301) exam dumps. The product of SureTorrent is available in Appian ACD301 PDF, EXAM CODE desktop practice exam software, and web-based Appian Lead Developer (ACD301) practice test.

Appian ACD301 Exam Syllabus Topics:

Topic	Details
Topic 1	• Project and Resource Management: This section of the exam measures skills of Agile Project Leads and covers interpreting business requirements, recommending design options, and leading Agile teams through technical delivery. It also involves governance, and process standardization.

Topic 2	• 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	• 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.

Appian ACD301 Exam Questions Are Out: Download And Prepare [2026]

Several advantages we now offer for your reference. On the one hand, our ACD301 learning questions engage our working staff in understanding customers' diverse and evolving expectations and incorporate that understanding into our strategies, thus you can 100% trust our ACD301 Exam Engine. On the other hand, the professional ACD301 study materials determine the high pass rate. According to the research statistics, we can confidently tell that 99% candidates after using our products have passed the ACD301 exam.

Appian Lead Developer Sample Questions (Q35-Q40):

NEW QUESTION # 35
Users must be able to navigate throughout the application while maintaining complete visibility in the application structure and easily navigate to previous locations. Which Appian Interface Pattern would you recommend?

- A. Implement a Drilldown Report pattern to show detailed information about report data.
- B. Implement an Activity History pattern to track an organization's activity measures.
- C. Include a Breadcrumbs pattern on applicable interfaces to show the organizational hierarchy.
- D. Use Billboards as Cards pattern on the homepage to prominently display application choices.

Answer: C

Explanation:
Comprehensive and Detailed In-Depth Explanation:The requirement emphasizes navigation with complete visibility of the application structure and the ability to return to previous locations easily. TheBreadcrumbs patternis specifically designed to meet this need. According to Appian's design best practices, the Breadcrumbs pattern provides a visual trail of the user's navigation path, showing the hierarchy of pages or sections within the application. This allows users to understand their current location relative to the overall structure and quickly navigate back to previous levels by clicking on the breadcrumb links.
* Option A (Billboards as Cards):This pattern is useful for presenting high-level options or choices on a homepage in a visually appealing way. However, it does not address navigation visibility or the ability to return to previous locations, making it irrelevant to the requirement.
* Option B (Activity History):This pattern tracks and displays a log of activities or actions within the application, typically for auditing or monitoring purposes. It does not enhance navigation or provide visibility into the application structure.
* Option C (Drilldown Report):This pattern allows users to explore detailed data within reports by drilling into specific records. While it supports navigation within data, it is not designed for general application navigation or maintaining structural visibility.
* Option D (Breadcrumbs):This is the correct choice as it directly aligns with the requirement. Per Appian's Interface Patterns documentation, Breadcrumbs improve usability by showing ahierarchical path (e.g., Home > Section > Subsection) and enabling backtracking, fulfilling both visibility and navigation needs.
References:Appian Design Guide - Interface Patterns (Breadcrumbs section), Appian Lead Developer Training - User Experience Design Principles.

NEW QUESTION # 36
You have 5 applications on your Appian platform in Production. Users are now beginning to use multiple applications across the platform, and the client wants to ensure a consistent user experience across all applications.

You notice that some applications use rich text, some use section layouts, and others use box layouts. The result is that each application has a different color and size for the header.
What would you recommend to ensure consistency across the platform?

- A. In the common application, create one rule for each application, and update each application to reference its respective rule.
- B. Create constants for text size and color, and update each section to reference these values.
- C. In each individual application, create a rule that can be used for section headers, and update each application to reference its respective rule.
- D. In the common application, create a rule that can be used across the platform for section headers, and update each application to reference this new rule.

Answer: D

Explanation:
Comprehensive and Detailed In-Depth Explanation:As an Appian Lead Developer, ensuring a consistent user experience across multiple applications on the Appian platform involves centralizing reusable components and adhering to Appian's design governance principles. The client's concern about inconsistent headers (e.g., different colors, sizes, layouts) across applications using rich text, section layouts, and box layouts requires a scalable, maintainable solution. Let's evaluate each option:
* A. Create constants for text size and color, and update each section to reference these values:Using constants (e.g., cons!TEXT_SIZE and cons!HEADER_COLOR) is a good practice for managing values, but it doesn't address layout consistency (e.g., rich text vs. section layouts vs. box layouts).
Constants alone can't enforce uniform header design across applications, as they don't encapsulate layout logic (e.g., a!sectionLayout() vs. a!richTextDisplayField()). This approach would require manual updates to each application's components, increasing maintenance overhead and still risking inconsistency. Appian's documentation recommends using rules for reusable UI components, not just constants, making this insufficient.
* B. In the common application, create a rule that can be used across the platform for section headers, and update each application to reference this new rule:This is the best recommendation. Appian supports a
"common application" (often called a shared or utility application) to store reusable objects like expression rules, which can define consistent header designs (e.g., rule!CommonHeader(size:
"LARGE", color: "PRIMARY")). By creating a single rule for headers and referencing it across all 5 applications, you ensure uniformity in layout, color, and size (e.g., using a!sectionLayout() or a!
boxLayout() consistently). Appian's design best practices emphasize centralizing UI components in a common application to reduce duplication, enforce standards, and simplify maintenance-perfect for achieving a consistent user experience.
* C. In the common application, create one rule for each application, and update each application to reference its respective rule:This approach creates separate header rules for each application (e.g., rule!
App1Header, rule!App2Header), which contradicts the goal of consistency. While housed in the common application, it introduces variability (e.g., different colors or sizes per rule), defeating the purpose. Appian's governance guidelines advocate for a single, shared rule to maintain uniformity, making this less efficient and unnecessary.
* D. In each individual application, create a rule that can be used for section headers, and update each application to reference its respective rule:Creating separate rules in each application (e.g., rule!
App1Header in App 1, rule!App2Header in App 2) leads to duplication and inconsistency, as each rule could differ in design. This approach increases maintenance effort and risks diverging styles, violating the client's requirement for a"consistent user experience." Appian's best practices discourage duplicating UI logic, favoring centralized rules in a common application instead.
Conclusion: Creating a rule in the common application for section headers and referencing it across the platform (B) ensures consistency in header design (color, size, layout) while minimizing duplication and maintenance. This leverages Appian's application architecture for shared objects, aligning with Lead Developer standards for UI governance.
References:
* Appian Documentation: "Designing for Consistency Across Applications" (Common Application Best Practices).
* Appian Lead Developer Certification: UI Design Module (Reusable Components and Rules).
* Appian Best Practices: "Maintaining User Experience Consistency" (Centralized UI Rules).
The best way to ensure consistency across the platform is to create a rule that can be used across the platform for section headers. This rule can be created in the common application, and then each application can be updated to reference this rule. This will ensure that all of the applications use the same color and size for the header, which will provide a consistent user experience.
The other options are not as effective. Option A, creating constants for text size and color, and updating each section to reference these values, would require updating each section in each application. This would be a lot of work, and it would be easy to make mistakes. Option C, creating one rule for each application, would also require updating each application. This would be less work than option A, but it would still be a lot of work, and it would be easy to make mistakes. Option D, creating a rule in each individual application, would not ensure consistency across the platform. Each application would have its own rule, and the rules could be different. This would not provide a consistent user experience.
Best Practices:

* When designing a platform, it is important to consider the user experience. A consistent user experience will make it easier for users to learn and use the platform.
* When creating rules, it is important to use them consistently across the platform. This will ensure that the platform has a consistent look and feel.
* When updating the platform, it is important to test the changes to ensure that they do not break the user experience.

NEW QUESTION # 37
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. Group epics and stories by technical difficulty, and allocate one team the more challenging stories.
- C. Have each team choose the stories they would like to work on based on personal preference.
- 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.

NEW QUESTION # 38
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. 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.
- 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. Run an explain statement on the views, identify critical areas of improvement that can be remediated without business knowledge.

Answer: D

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.
References:Appian Documentation - Performance Troubleshooting Guide, Appian Lead Developer Training
- Database Optimization, MySQL/PostgreSQL Documentation - EXPLAIN Statement.

NEW QUESTION # 39
Your team has deployed an application to Production with an underperforming view. Unexpectedly, the production data is ten times that of what was tested, and you must remediate the issue. What is the best option you can take to mitigate their performance concerns?

- A. Introduce a data management policy to reduce the volume of data.
- B. Create a table which is loaded every hour with the latest data.
- C. Bypass Appian's query rule by calling the database directly with a SQL statement.
- D. Create a materialized view or table.

Answer: D

Explanation:
Comprehensive and Detailed In-Depth Explanation:
As an Appian Lead Developer, addressing performance issues in production requires balancing Appian's best practices, scalability, and maintainability. The scenario involves an underperforming view due to a significant increase in data volume (ten times the tested amount), necessitating a solution that optimizes performance while adhering to Appian's architecture. Let's evaluate each option:
A . Bypass Appian's query rule by calling the database directly with a SQL statement:
This approach involves circumventing Appian's query rules (e.g., a!queryEntity) and directly executing SQL against the database. While this might offer a quick performance boost by avoiding Appian's abstraction layer, it violates Appian's core design principles. Appian Lead Developer documentation explicitly discourages direct database calls, as they bypass security (e.g., Appian's row-level security), auditing, and portability features. This introduces maintenance risks, dependencies on database-specific logic, and potential production instability-making it an unsustainable and non-recommended solution.
B . Create a table which is loaded every hour with the latest data:
This suggests implementing a staging table updated hourly (e.g., via an Appian process model or ETL process). While this could reduce query load by pre-aggregating data, it introduces latency (data is only fresh hourly), which may not meet real-time requirements typical in Appian applications (e.g., a customer-facing view). Additionally, maintaining an hourly refresh process adds complexity and overhead (e.g., scheduling, monitoring). Appian's documentation favors more efficient, real-time solutions over periodic refreshes unless explicitly required, making this less optimal for immediate performance remediation.
C . Create a materialized view or table:
This is the best choice. A materialized view (or table, depending on the database) pre-computes and stores query results, significantly improving retrieval performance for large datasets. In Appian, you can integrate a materialized view with a Data Store Entity, allowing a!queryEntity to fetch data efficiently without changing application logic. Appian Lead Developer training emphasizes leveraging database optimizations like materialized views to handle large data volumes, as they reduce query execution time while

keeping data consistent with the source (via periodic or triggered refreshes, depending on the database). This aligns with Appian's performance optimization guidelines and addresses the tenfold data increase effectively.
D . Introduce a data management policy to reduce the volume of data:
This involves archiving or purging data to shrink the dataset (e.g., moving old records to an archive table). While a long-term data management policy is a good practice (and supported by Appian's Data Fabric principles), it doesn't immediately remediate the performance issue. Reducing data volume requires business approval, policy design, and implementation-delaying resolution. Appian documentation recommends combining such strategies with technical fixes (like C), but as a standalone solution, it's insufficient for urgent production concerns.
Conclusion: Creating a materialized view or table (C) is the best option. It directly mitigates performance by optimizing data retrieval, integrates seamlessly with Appian's Data Store, and scales for large datasets-all while adhering to Appian's recommended practices. The view can be refreshed as needed (e.g., via database triggers or schedules), balancing performance and data freshness. This approach requires collaboration with a DBA to implement but ensures a robust, Appian-supported solution.
Reference:
Appian Documentation: "Performance Best Practices" (Optimizing Data Queries with Materialized Views).
Appian Lead Developer Certification: Application Performance Module (Database Optimization Techniques).
Appian Best Practices: "Working with Large Data Volumes in Appian" (Data Store and Query Performance).

NEW QUESTION # 40
.....

Why do so many people determine to take part in Appian ACD301 exam? Owing a nice certification will not only testify your professional skills and qualification but also show your knowledge and ability, it will be useful for your career. ACD301 New Test Bootcamp materials will be valid and useful for your test. If you get a certification, you will be regards as knowledgeable expert. Now there is a large demand for these skillful senior engineers.