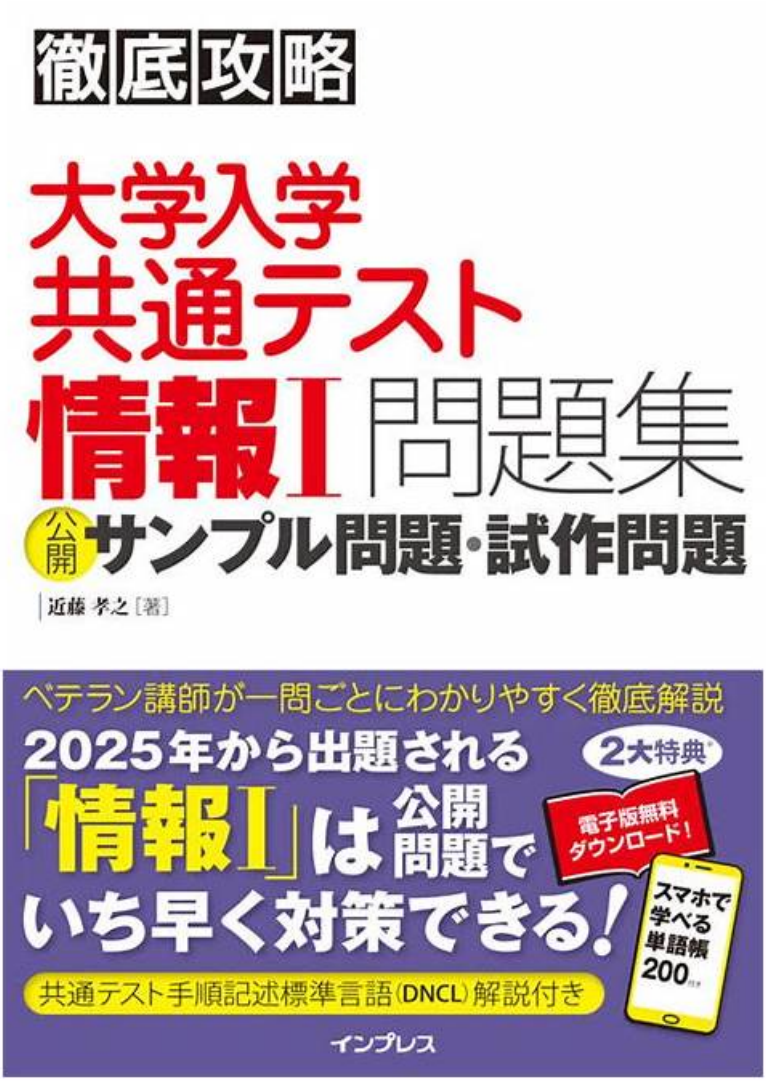
ACD301テスト問題集、ACD301試験問題解説集



P.S. It-PassportsがGoogle Driveで共有している無料かつ新しいACD301ダンプ：https://drive.google.com/open?id=15m-We8ynflWv8OmK3vOAWsqqSQASUryF

It-PassportsのAppianのACD301試験トレーニング資料の知名度が非常に高いことを皆はよく知っています。It-Passportsは世界的によく知られているサイトです。どうしてこのような大きな連鎖反応になりましたか。それはIt-PassportsのAppianのACD301試験トレーニング資料は適用性が高いもので、本当にみなさんが良い成績を取ることを助けられるからです。

おそらく、あなたはゲームをするのに多くの時間を無駄にしたでしょう。関係ありません。変更するのに遅すぎることはありません。過去を後悔する意味はありません。 ACD301試験資料は、希望するACD301認定を取得するのに役立ちます。 ACD301学習教材を学習した後、あなたは大きく変わります。また、あなたは人生について前向きな見方をします。全体として、すべての幻想を捨て、勇敢に現実に立ち向かいます。 ACD301模擬試験が最高のアシスタントになります。あなたは世界で最高でユニークです。新たな挑戦に直面するだけで自信を持ってください！

>> ACD301テスト問題集 <<

ACD301試験問題解説集、ACD301試験資料

人生にはあまりにも多くの変化および未知の誘惑がありますから、まだ若いときに自分自身のために強固な基盤を築くべきです。あなた準備しましたか。It-PassportsのAppianのACD301試験トレーニング資料は最高のト

レーニング資料です。IT職員としてのあなたは切迫感を感じましたか。It-Passportsを選んだら、成功への扉を開きます。頑張ってください。

Appian ACD301 認定試験の出題範囲:

トピック	出題範囲
トピック 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.
トピック 2	• 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.
トピック 3	• 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.
トピック 4	• 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 Lead Developer 認定 ACD301 試験問題 (Q46-Q51):

質問 # 46
An Appian application contains an integration used to send a JSON, called at the end of a form submission, returning the created code of the user request as the response. To be able to efficiently follow their case, the user needs to be informed of that code at the end of the process. The JSON contains case fields (such as text, dates, and numeric fields) to a customer's API. What should be your two primary considerations when building this integration?

- A. A dictionary that matches the expected request body must be manually constructed.
- B. A process must be built to retrieve the API response afterwards so that the user experience is not impacted.
- C. The request must be a multi-part POST.
- D. The size limit of the body needs to be carefully followed to avoid an error.

正解：A、D

解説：
Comprehensive and Detailed In-Depth Explanation:As an Appian Lead Developer, building an integration to send JSON to a customer's API and return a code to the user involves balancing usability, performance, and reliability. The integration is triggered at form submission, and the user must see the response (case code) efficiently. The JSON includes standard fields (text, dates, numbers), and the focus is on primary considerations for the integration itself. Let's evaluate each option based on Appian's official documentation and best practices:
* A. A process must be built to retrieve the API response afterwards so that the user experience is not impacted:This suggests making the integration asynchronous by calling it in a process model (e.g., via a Start Process smart service) and retrieving the response later, avoiding delays in the UI. While this improves user experience for slow APIs (e.g., by showing a "Processing" message), it contradicts the requirement that the user is "informed of that code at the end of the process." Asynchronous processing would delay the code display, requiring additional steps (e.g., a follow-up task), which isn't efficient for this use case. Appian's default integration pattern (synchronous call in an Integration object) is suitable unless latency is a known issue, making this a secondary-not primary-consideration.
* B. The request must be a multi-part POST:A multi-part POST (e.g., multipart/form-data) is used for sending mixed content, like files and text, in a single request. Here, the payload is a JSON containing case fields (text, dates, numbers)-no files are mentioned. Appian's HTTP Connected System and Integration objects default to application/json for JSON payloads via a standard POST, which aligns with REST API norms. Forcing a multi-part POST adds unnecessary complexity and is incompatible with most APIs

expecting JSON. Appian documentation confirms this isn't required for JSON-only data, ruling it out as a primary consideration.
* C. The size limit of the body needs to be carefully followed to avoid an error:This is a primary consideration. Appian's Integration object has a payload size limit (approximately 10 MB, though exact limits depend on the environment and API), and exceeding it causes errors (e.g., 413 Payload Too Large). The JSON includes multiple case fields, and while "hundreds of thousands" isn't specified, large datasets could approach this limit. Additionally, the customer's API may impose its own size restrictions (common in REST APIs). Appian Lead Developer training emphasizes validating payload size during design-e.g., testing with maximum expected data-to prevent runtime failures. This ensures reliability and is critical for production success.
* D. A dictionary that matches the expected request body must be manually constructed:This is also a primary consideration. The integration sends a JSON payload to the customer's API, which expects a specific structure (e.g., { "field1": "text", "field2": "date" }). In Appian, the Integration object requires a dictionary (key-value pairs) to construct the JSON body, manually built to match the API's schema.
Mismatches (e.g., wrong field names, types) cause errors (e.g., 400 Bad Request) or silent failures.
Appian's documentation stresses defining the request body accurately-e.g., mapping form data to a CDT or dictionary-ensuring the API accepts the payload and returns the case code correctly. This is foundational to the integration's functionality.
Conclusion: The two primary considerations are C (size limit of the body) and D (constructing a matching dictionary). These ensure the integration works reliably (C) and meets the API's expectations (D), directly enabling the user to receive the case code at submission end. Size limits prevent technical failures, while the dictionary ensures data integrity-both are critical for a synchronous JSON POST in Appian. Option A could be relevant for performance but isn't primary given the requirement, and B is irrelevant to the scenario.
References:
* Appian Documentation: "Integration Object" (Request Body Configuration and Size Limits).
* Appian Lead Developer Certification: Integration Module (Building REST API Integrations).
* Appian Best Practices: "Designing Reliable Integrations" (Payload Validation and Error Handling).

質問 # 47
You are the lead developer for an Appian project, in a backlog refinement meeting. You are presented with the following user story: "As a restaurant customer, I need to be able to place my food order online to avoid waiting in line for takeout." Which two functional acceptance criteria would you consider 'good'?

- A. The user cannot submit the form without filling out all required fields.
- B. The user will receive an email notification when their order is completed.
- C. The user will click Save, and the order information will be saved in the ORDER table and have audit history.
- D. The system must handle up to 500 unique orders per day.

正解：A、C

解説：
Comprehensive and Detailed In-Depth Explanation:
As an Appian Lead Developer, defining "good" functional acceptance criteria for a user story requires ensuring they are specific, testable, and directly tied to the user's need (placing an online food order to avoid waiting in line). Good criteria focus on functionality, usability, and reliability, aligning with Appian's Agile and design best practices. Let's evaluate each option:
A . The user will click Save, and the order information will be saved in the ORDER table and have audit history:
This is a "good" criterion. It directly validates the core functionality of the user story-placing an order online. Saving order data in the ORDER table (likely via a process model or Data Store Entity) ensures persistence, and audit history (e.g., using Appian's audit logs or database triggers) tracks changes, supporting traceability and compliance. This is specific, testable (e.g., verify data in the table and logs), and essential for the user's goal, aligning with Appian's data management and user experience guidelines.
B . The user will receive an email notification when their order is completed:
While useful, this is a "nice-to-have" enhancement, not a core requirement of the user story. The story focuses on placing an order online to avoid waiting, not on completion notifications. Email notifications add value but aren't essential for validating the primary functionality. Appian's user story best practices prioritize criteria tied to the main user need, making this secondary and not "good" in this context.
C . The system must handle up to 500 unique orders per day:
This is a non-functional requirement (performance/scalability), not a functional acceptance criterion. It describes system capacity, not specific user behavior or functionality. While important for design, it's not directly testable for the user story's outcome (placing an order) and isn't tied to the user's experience. Appian's Agile methodologies separate functional and non-functional requirements, making this less relevant as a "good" criterion here.
D . The user cannot submit the form without filling out all required fields:
This is a "good" criterion. It ensures data integrity and usability by preventing incomplete orders, directly supporting the user's ability to place a valid online order. In Appian, this can be implemented using form validation (e.g., required attributes in SAIL interfaces or process model validations), making it specific, testable (e.g., verify form submission fails with missing fields), and critical for a reliable

user experience. This aligns with Appian's UI design and user story validation standards.
Conclusion: The two "good" functional acceptance criteria are A (order saved with audit history) and D (required fields enforced). These directly validate the user story's functionality (placing a valid order online), are testable, and ensure a reliable, user-friendly experience-aligning with Appian's Agile and design best practices for user stories.
Reference:
Appian Documentation: "Writing Effective User Stories and Acceptance Criteria" (Functional Requirements).
Appian Lead Developer Certification: Agile Development Module (Acceptance Criteria Best Practices).
Appian Best Practices: "Designing User Interfaces in Appian" (Form Validation and Data Persistence).

質問 # 48
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.

正解：A

解説：
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 discourage relying 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.
Reference:
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).

質問 # 49

You add an index on the searched field of a MySQL table with many rows (>100k). The field would benefit greatly from the index in which three scenarios?

- A. The field contains many datetimes, covering a large range.
- B. The field contains big integers, above and below 0.
- C. The field contains a textual short business code.
- D. The field contains a structured JSON.
- E. The field contains long unstructured text such as a hash.

正解：A、B、C

解説：
Comprehensive and Detailed In-Depth Explanation:Adding an index to a searched field in a MySQL table with over 100,000 rows improves query performance by reducing the number of rows scanned during searches, joins, or filters. The benefit of an index depends on the field's data type, cardinality (uniqueness), and query patterns. MySQL indexingbest practices, as aligned with Appian's Database Optimization Guidelines, highlight scenarios where indices are most effective.
* Option A (The field contains a textual short business code):This benefits greatly from an index. A short business code (e.g., a 5-10 character identifier like "CUST123") typically has high cardinality (many unique values) and is often used in WHERE clauses or joins. An index on this field speeds up exact-match queries (e.g., WHERE business_code = 'CUST123'), which are common in Appian applications for lookups or filtering.
* Option C (The field contains many datetimes, covering a large range):This is highly beneficial.
Datetime fields with a wide range (e.g., transaction timestamps over years) are frequently queried with range conditions (e.g., WHERE datetime BETWEEN '2024-01-01' AND '2025-01-01') or sorting (e.g., ORDER BY datetime). An index on this field optimizes these operations, especially in large tables, aligning with Appian's recommendation to index time-based fields for performance.
* Option D (The field contains big integers, above and below 0):This benefits significantly. Big integers (e.g., IDs or quantities) with a broad range and high cardinality are ideal for indexing. Queries like WHERE id > 1000 or WHERE quantity < 0 leverage the index for efficient range scans or equality checks, a common pattern in Appian data store queries.
* Option B (The field contains long unstructured text such as a hash):This benefits less. Long unstructured text (e.g., a 128-character SHA hash) has high cardinality but is less efficient for indexing due to its size. MySQL indices on large text fields can slow down writes and consume significant storage, and full-text searches are better handled with specialized indices (e.g., FULLTEXT), not standard B-tree indices. Appian advises caution with indexing large text fields unless necessary.
* Option E (The field contains a structured JSON):This is minimally beneficial with a standard index.
MySQL supports JSON fields, but a regular index on the entire JSON column is inefficient for large datasets (>100k rows) due to its variable structure. Generated columns or specialized JSON indices (e.
g., using JSON_EXTRACT) are required for targeted queries (e.g., WHERE JSON_EXTRACT (json_col, '$.key') = 'value'), but this requires additional setup beyond a simple index, reducing its immediate benefit.
For a table with over 100,000 rows, indices are most effective on fields with high selectivity and frequent query usage (e.g., short codes, datetimes, integers), making A, C, and D the optimal scenarios.
References:Appian Documentation - Database Optimization Guidelines, MySQL Documentation - Indexing Strategies, Appian Lead Developer Training - Performance Tuning.

質問 # 50
You are reviewing the Engine Performance Logs in Production for a single application that has been live for six months. This application experiences concurrent user activity and has a fairly sustained load during business hours. The client has reported performance issues with the application during business hours.
During your investigation, you notice a high Work Queue - Java Work Queue Size value in the logs. You also notice unattended process activities, including timer events and sending notification emails, are taking far longer to execute than normal.
The client increased the number of CPU cores prior to the application going live.
What is the next recommendation?

- A. Add execution and analytics shards
- B. Add more application servers.
- C. Add more engine replicas.
- D. Optimize slow-performing user interfaces.

正解：C

解説：
As an Appian Lead Developer, analyzing Engine Performance Logs to address performance issues in a Production application

requires understanding Appian's architecture and the specific metrics described. The scenario indicates a high "Work Queue - Java Work Queue Size," which reflects a backlog of tasks in the Java Work Queue (managed by Appian engines), and delays in unattended process activities (e.g., timer events, email notifications). These symptoms suggest the Appian engines are overloaded, despite the client increasing CPU cores. Let's evaluate each option:
A . Add more engine replicas:
This is the correct recommendation. In Appian, engine replicas (part of the Appian Engine cluster) handle process execution, including unattended tasks like timers and notifications. A high Java Work Queue Size indicates the engines are overwhelmed by concurrent activity during business hours, causing delays. Adding more engine replicas distributes the workload, reducing queue size and improving performance for both user-driven and unattended tasks. Appian's documentation recommends scaling engine replicas to handle sustained loads, especially in Production with high concurrency. Since CPU cores were already increased (likely on application servers), the bottleneck is likely the engine capacity, not the servers.
B . Optimize slow-performing user interfaces:
While optimizing user interfaces (e.g., SAIL forms, reports) can improve user experience, the scenario highlights delays in unattended activities (timers, emails), not UI performance. The Java Work Queue Size issue points to engine-level processing, not UI rendering, so this doesn't address the root cause. Appian's performance tuning guidelines prioritize engine scaling for queue-related issues, making this a secondary concern.
C . Add more application servers:
Application servers handle web traffic (e.g., SAIL interfaces, API calls), not process execution or unattended tasks managed by engines. Increasing application servers would help with UI concurrency but wouldn't reduce the Java Work Queue Size or speed up timer/email processing, as these are engine responsibilities. Since the client already increased CPU cores (likely on application servers), this is redundant and unrelated to the issue.
D . Add execution and analytics shards:
Execution shards (for process data) and analytics shards (for reporting) are part of Appian's data fabric for scalability, but they don't directly address engine workload or Java Work Queue Size. Shards optimize data storage and query performance, not real-time process execution. The logs indicate an engine bottleneck, not a data storage issue, so this isn't relevant. Appian's documentation confirms shards are for long-term scaling, not immediate performance fixes.
Conclusion: Adding more engine replicas (A) is the next recommendation. It directly resolves the high Java Work Queue Size and delays in unattended tasks, aligning with Appian's architecture for handling concurrent loads in Production. This requires collaboration with system administrators to configure additional replicas in the Appian cluster.
Reference:
Appian Documentation: "Engine Performance Monitoring" (Java Work Queue and Scaling Replicas).
Appian Lead Developer Certification: Performance Optimization Module (Engine Scaling Strategies).
Appian Best Practices: "Managing Production Performance" (Work Queue Analysis).

質問 # 51
.....

24時間年中無休のサービスオンラインサポートサービスを提供しており、専門スタッフにリモートアシスタンスを提供しています。また、ACD301実践教材の請求書が必要な場合は、請求書情報を指定してメールをお送りください。また、購入前にACD301トレーニングエンジンの試用版を無料でダウンロードできます。この種のサービスは、当社のACD301学習教材に関する自信と実際の強さを示しています。また、当社のウェブサイト購入プロセスにはセキュリティ保証がありますので、ACD301試験問題をダウンロードしてインストールする必要はありません。

ACD301試験問題解説集: https://www.it-passports.com/ACD301.html

- ACD301模擬対策 ☃ ACD301模擬対策 □ ACD301合格体験談 □ 【 ACD301 】を無料でダウンロード「 www.goshiken.com 」ウェブサイトを入力するだけACD301ブロンズ教材
- ACD301練習問題集 □ ACD301対応受験 □ ACD301合格対策 □ ➽ www.goshiken.com □で使える無料オンライン版➽ ACD301 □ の試験問題ACD301合格体験談
- ACD301技術問題 □ ACD301受験対策 □ ACD301練習問題 □ 【 jp.fast2test.com 】の無料ダウンロード▹ ACD301 ◃ページが開きますACD301技術問題
- 効果的なACD301テスト問題集試験-試験の準備方法-ハイパスレートのACD301試験問題解説集 □ 今すぐ ➽ www.goshiken.com □を開き、▸ ACD301 ◂を検索して無料でダウンロードしてくださいACD301難易度
- ACD301専門トレーリング □ ACD301練習問題 □ ACD301再テスト □ ✔ www.mogiexam.com □✔□の無料ダウンロード□ ACD301 □ページが開きますACD301日本語版試験解答
- ACD301専門トレーリング □ ACD301日本語版試験解答 □ ACD301ブロンズ教材 □ “ www.goshiken.com ”で使える無料オンライン版➡ ACD301 □□□ の試験問題ACD301技術問題
- ACD301模擬対策 □ ACD301合格対策 □ ACD301復習対策書 □ ウェブサイト☀ www.topexam.jp □☀□を開き、（ ACD301 ）を検索して無料でダウンロードしてくださいACD301模擬対策

- ACD301最新問題 □ ACD301最新問題 □ ACD301模擬対策 □ ウェブサイト□ www.goshiken.com □を開き、➠ ACD301 □を検索して無料でダウンロードしてくださいACD301勉強資料
- 検証するACD301テスト問題集試験-試験の準備方法-素敵なACD301試験問題解説集 □ ▹ www.goshiken.com ◃を入力して➠ ACD301 □を検索し、無料でダウンロードしてくださいACD301技術問題
- 最高のACD301テスト問題集のみがAppian Lead Developerの合格率を提供できます □【 www.goshiken.com 】から簡単に《 ACD301 》を無料でダウンロードできますACD301受験トレーリング
- ACD301 PDF □ ACD301勉強資料 ❣ ACD301ブロンズ教材 □☀ ACD301 □☀□を無料でダウンロード☀ www.goshiken.com □☀□で検索するだけACD301練習問題集
- www.stes.tyc.edu.tw, study.stcs.edu.np, www.stes.tyc.edu.tw, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, www.stes.tyc.edu.tw, www.stes.tyc.edu.tw, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, ncon.edu.sa, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, Disposable vapes

P.S. It-PassportsがGoogle Driveで共有している無料かつ新しいACD301ダンプ：https://drive.google.com/open?id=15m-We8ynflWv8OmK3vOAWsqqSQASUryF