信頼できるDP-800最速合格と一番優秀なDP-800試験問題集

あなたは君の初めてのMicrosoftのDP-800認定試験を受ける時に認定試験に合格したいか。Japancertでは、私たちは君のすべての夢を叶えさせて、君の最も早い時間でMicrosoftのDP-800認定試験に合格するということを保証します。JapancertのMicrosoftのDP-800試験トレーニング資料は豊富な経験を持っているIT専門家が研究したもので、問題と解答が緊密に結んでいるものです。Japancertを選ぶなら、絶対に後悔させません。

Japancertは、最新のDP-800試験トレントが能力を強化し、DP-800試験に合格して認定を取得するのに非常に役立つと深く信じています。嫌がらせから抜け出すために、DP-800学習教材は高品質で高い合格率を備えています。ほとんどの時間インターネットにアクセスできない場合、どこかに行く必要がある場合はオフライン状態ですが、DP-800試験のために学習したい場合。当社のウェブサイトは、優れたDP-800試験問題の助けを借りて問題の解決に役立ちます。

>> DP-800最速合格 <<

DP-800試験問題集 & DP-800教育資料

DP-800学習教材が他の学習教材よりも優れた品質を持っているだけでなく、優れた品質を持っていることを知ることは難しくありません。一方で、DP-800学習教材を学習すれば、DP-800試験に簡単に合格することを保証できます。一方、DP-800学習ブレインダンプから多くの有用な知識を学びます。準備はできたか？ 最初に、DP-800学習教材のデモをWebから無料でダウンロードできます。

Microsoft DP-800 認定試験の出題範囲：

トピック	出題範囲

トピック 1	• Implement AI capabilities in database solutions: This domain covers designing and managing external AI models and embeddings, implementing full-text, semantic vector, and hybrid search strategies, and building retrieval-augmented generation (RAG) solutions that connect database outputs with language models.
トピック 2	• Secure, optimize, and deploy database solutions: This domain focuses on implementing data security measures like encryption, masking, and row-level security, optimizing query performance, managing CI • CD pipelines using SQL Database Projects, and integrating SQL solutions with Azure services including Data API builder and monitoring tools.
トピック 3	• Design and develop database solutions: This domain covers designing and building database objects such as tables, views, functions, stored procedures, and triggers, along with writing advanced T-SQL code and leveraging AI-assisted tools like GitHub Copilot and MCP for SQL development.

Microsoft Developing AI-Enabled Database Solutions 認定 DP-800 試験問題 (Q58-Q63):

質問 # 58
You have an Azure SQL database named SalesDB that contains tables named Sales.Orders and Sales.
OrderLines. Both tables contain sales data
You have a Retrieval Augmented Generation (RAG) service that queries SalesDB to retrieve order details and passes the results to a large language model (ILM) as JSON text. The following is a sample of the JSON.
You need to return one 1SON document per order that includes the order header fields and an array of related order lines. The LIM must receive a single JSON array of orders, where each order contains a lines property that is a JSON array of line Items.
Which transact-SQL commands should you use to produce the required JSON shape from the relational tables? To answer, drag the appropriate commands to the correct operations. Each command may be used once, more than once, or not at all. Vou may need to drag the split bar between panes or scroll to view content.
NOTE: Each correct selection is worth one point.

正解：

解説：
Explanation:
* Serialize the order-level JSON : FOR JSON PATH
* Generate a nested lines array : JSON_QUERY
* Extract a single scalar value from the JSON text : JSON_VALUE
The correct mapping is based on how SQL Server and Azure SQL JSON functions are designed to shape relational data into JSON for AI and RAG scenarios.
To serialize the order-level JSON , use FOR JSON PATH . Microsoft documents that FOR JSON PATH gives you full control over the JSON output shape and formats the result as an array of JSON objects . It is the standard way to turn relational query results into the JSON structure needed by downstream consumers such as APIs and LLM-based RAG services. It also supports nested output through subqueries and aliases.
To generate a nested lines array , use JSON_QUERY . Microsoft explains that JSON_QUERY returns a JSON object or array from JSON text, and it is used when you want to preserve a JSON fragment instead of treating it as plain text. In this scenario, the nested lines property must be emitted as a proper JSON array inside each order document, so JSON_QUERY is the correct command to embed that array in the final JSON shape.
To extract a single scalar value from the JSON text , use JSON_VALUE . Microsoft explicitly states that JSON_VALUE extracts a scalar value from a JSON string, while JSON_QUERY is for objects or arrays. So whenever the requirement is to pull out one property such as an order number, currency code, or customer ID from JSON text, JSON_VALUE is the correct function.
The unused commands are not the best fit here:
* OPENJSON is primarily for parsing JSON into rows and columns, not for shaping relational tables into nested output.
* JSON_MODIFY is for updating JSON text, not generating the required output structure.
So the drag-and-drop answers are:
* Serialize the order-level JSON # FOR JSON PATH
* Generate a nested lines array # JSON_QUERY
* Extract a single scalar value from the JSON text # JSON_VALUE

質問 # 59
You have an Azure SQL database that contains tables named dbo.ProduetDocs and dbo.
ProductuocsEnbeddings. dbo.ProductOocs contains product documentation and the following columns:
* Docld (int)
* Title (nvdrchdr(200))
* Body (nvarthar(max))
* LastHodified (datetime2)
The documentation is edited throughout the day. dbo.ProductDocsEabeddings contains the following columns:
* Dotid (int)
* ChunkOrder (int)
* ChunkText (nvarchar(aax))
* Embedding (vector(1536))
The current embedding pipeline runs once per night
Vou need to ensure that embeddings are updated every time the underlying documentation content changes The solution must NOT ' equire a nightly batch process.
What should you include in the solution?

- A. change tracking on dbo.ProductDocs
- B. fixed-size chunking
- C. table triggers
- D. a smaller embedding model

正解：A

解説：
The requirement is to ensure embeddings are updated every time the underlying content changes without relying on a nightly batch job. The right design is to enable change tracking on the source table so an external process can identify which rows changed and regenerate embeddings only for those rows. Microsoft documents that change detection mechanisms are used to pick up new and updated rows incrementally , which is the right pattern when you need near-continuous refresh instead of full nightly rebuilds.
This is better than:
* A. fixed-size chunking , which affects chunk strategy but not change detection.
* B. a smaller embedding model , which affects model cost/latency but not update triggering.
* C. table triggers , which would push embedding-maintenance logic directly into write operations and is generally not the best design for AI-processing pipelines. The question specifically asks for a solution that replaces the nightly batch requirement, not one that performs heavyweight work inline during every transaction.

質問 # 60
You need to enable similarity search to provide the analysts with the ability to retrieve the most relevant health summary reports. The solution must minimize latency.
What should you include in the solution?

- A. a standard nonclustered index on the Fmbeddings (vector (1536)) column
- B. a full-text index on the Fmbeddings (vector (1536)) column
- C. a computed column that manually compares vector values
- D. a vector index on the Embedding* (vector (1536)) column

正解：D

解説：
The correct answer is D because the requirement is to enable similarity search over embedding vectors and to minimize latency . Microsoft documents that CREATE VECTOR INDEX is specifically used to create an index on vector data for approximate nearest neighbor (ANN) search , which is designed to accelerate vector similarity queries compared to exact k-nearest-neighbor scans.
This matches the scenario exactly. The VehicleHealthSummary table already includes an Embeddings (vector (1536)) column. In Microsoft SQL platforms, embeddings are stored in vector columns and queried for semantic similarity. To improve performance and reduce response time, Microsoft recommends a vector index , not a regular B-tree nonclustered index and not a full-text index. A vector index is purpose-built for finding the most similar vectors efficiently.
The other options are not appropriate:
* A would require manual comparison logic and would increase latency rather than minimize it.
* B is incorrect because a standard nonclustered index is not the index type used for vector similarity operations.

* C is incorrect because full-text indexes are for textual token-based search, not numeric vector embeddings.
Microsoft's current documentation is explicit that vector indexes support approximate nearest neighbor search , and that the optimizer can use the ANN index automatically for vector queries. That is the exam- aligned design choice when the goal is fast retrieval of the most relevant health summary reports from an embeddings column.

質問 # 61
You have an Azure SQL database that contains a table named dbo.orders, dbo.orders contains a column named createDate that stores order creation dates.
You need to create a stored procedure that filters Orders by CreateDate for a single calendar day. The solution must be SARGable.
How should you complete the Transact-SQL code? To answer, drag the appropriate values to the correct targets. Each value may be used once, more than once, or not at all. You may need to drag the split bar between panes or scroll to view content.
NOTE: Each correct selection is worth one point.

正解：

解説：
Explanation:
The correct SARGable pattern for filtering a single calendar day is to use a half-open date range :
o.CreateDate > = @StartDate
AND o.CreateDate < @EndDate
with:
SET @EndDate = DATEADD(day, 1, @StartDate)
This is the correct design because it keeps the function off the column and applies it only to the parameter.
That allows SQL Server and Azure SQL to use an index on CreateDate efficiently, which is the key requirement for a SARGable predicate. Microsoft documents DATEADD as the standard function for adding one day to a date value, which makes it the right way to derive the exclusive upper boundary for the next day.
The incorrect choices are the ones that wrap CreateDate in CONVERT(...), because expressions like:
CONVERT(char(10), CreateDate, 121) = ...
make the predicate non-SARGable and typically prevent efficient seeks on an index over CreateDate.
So the completed procedure is:
CREATE PROCEDURE dbo.usp_SearchOrders
@StartDate date
AS
BEGIN
SET NOCOUNT ON;
DECLARE @EndDate date;
SET @EndDate = DATEADD(day, 1, @StartDate);
SELECT o.CreateDate,
o.OrderId,
o.ShipDate
FROM dbo.Orders AS o
WHERE o.CreateDate > = @StartDate
AND o.CreateDate < @EndDate;
END;

質問 # 62
You have an Azure SQL database that has Query Store enabled
Query Performance Insight shows that one stored procedure has the longest runtime. The procedure runs the following parameterized query.
The dbo.orders table has approximately 120 million rows. Customer-id is highly selective, and orderOate is used for range filtering and sorting.
Vou have the following indexes:
* Clustered index: PK_Orders on (Orderld)
* Nonclustered index: lx_0rders_order-Date on (OrderDate) with no included columns An actual execution plan captured from Query Store for slow runs shows the following:
* An index seek on ixordersorderDate followed by a Key Lookup (Clustered) on PKOrders for customerid, status, and TotalAnount

* A sort operator before top (50), because the results are ordered by orderDate DESC For each of the following statements, select Yes if the statement is true. Otherwise, select No. NOTE: Each correct selection is worth one point.

正解：

解説：

Explanation:

The first statement is Yes . The query filters on CustomerId, applies a range predicate on OrderDate, and sorts by OrderDate DESC. Microsoft's index design guidance recommends putting equality predicates first in the key, followed by columns used for ordering/range access, because the order of key columns determines seek and sort support. A nonclustered index on (CustomerId, OrderDate DESC) can support an ordered seek for this query and avoid the explicit sort. Including Status and TotalAmount helps cover the query, and OrderId is already available because the clustered key is stored with nonclustered index rows.

The second statement is No . Adding CustomerId as an included column to IX_Orders_OrderDate does not make it part of the index's navigational structure. Microsoft states that included columns are nonkey columns used to cover queries; they do not provide the seek and ordering characteristics that key columns do. So an index keyed only on OrderDate still is not the right ordered access path for WHERE CustomerId =

@CustomerId ... ORDER BY OrderDate DESC.

The third statement is Yes . The described actual plan shows an index seek on the wrong access path for the workload, followed by clustered key lookups and an explicit sort before TOP (50). That is characteristic of a suboptimal query/index plan . Query Store and Query Performance Insight are designed to surface plan- related performance regressions, while locking/blocking problems are typically identified through waits

/DMVs and blocking-session indicators, not from a plan shape like seek + lookup + sort alone.

質問 # 63

.....

- DP-800日本語版と英語版 □ DP-800試験勉強書 □ DP-800全真模擬試験 □ □ www.mogiexam.com □から ➥ DP-800 □を検索して、試験資料を無料でダウンロードしてくださいDP-800模擬試験
- vinnylqju282075.blog4youth.com, macieeoki888392.plpwiki.com, www.stes.tyc.edu.tw, elijahzqwi917071.thebindingwiki.com, indexedbookmarks.com, minibookmarking.com, anniesemq362269.luwebs.com, ok-social.com, umarrfjq356717.blogdemls.com, www.stes.tyc.edu.tw, Disposable vapes