Vendor: Microsoft

Exam Code: DP-800

Exam Name: SQL AI Developer Associate

Version: DEMO

QUESTION 1
Case Study 1 - Contoso

Existing Environment
Azure Environment
Contoso has an Azure subscription in North Europe that contains the corporate infrastructure. The current infrastructure contains a Microsoft SQL Server 2017 database. The database contains the following tables.

Table names	Column names
CustomerFeedback	• FeedbackId (int) (primarykey) • FeedbackJson (nvarchar (max))
Fleets	• FleetId (int) (primarykey) • FleetName (nvarchar(100)) • Description (nvarchar(256))
MaintenanceEvents	• MaintenanceId (int) (primarykey) • VehicleId (int) • LastModifiedUTC (datetime2) • Description (nvarchar(256))
SupportTickets	• TicketId (int) (primarykey) • FleetId (int) • CreatedUtc (datetime2)
UserAccounts	• UserId (int) (primarykey) • UserPrincipalName (nvarchar(256)) • JobRole (nvarchar(256)) • StartDate (datetime2)
VehicleIncidentReports	• IncidentId (int) (primarykey) • VehicleId (int) • FleetId (int) • IncidentType (nvarchar(50)) • VehicleLocation (nvarchar(200)) • IncidentDescription (nvarchar(max)) • SeverityScore (int)
Vehicles	• VehicleId (int) (primarykey) • VIN ((nvarchar(50)) • VehicleDescription (nvarchar(256))
VehicleHealthSummary	• VehicleId (int) (primarykey) • FleetId (int) • Summary (nvarchar(2000)) • LastUpdatedUtc (datetime2) • EngineStatus [bit] • EngineStatusLastUpdatedUtc (datetime2) • BatteryHealth (int) • Embeddings (vector (1536))

The `FeedbackJson` column has a full-text index and stores JSON documents in the following

format.

```
{
  "text": "The battery drains too fast when driving uphill.",
  "category": "Battery",
  "metadata": {
    "appVersion": "5.2.1",
    "device": "Android",
    "language": "en-GB"
  }
}
```

The support staff at Contoso never has the `UNMASK` permission.

Problem Statements
Contoso is deploying a new Azure SQL database that will become the authoritative data store for the following:

- AI workloads
- Vector search
- Modernized API access
- Retrieval Augmented Generation (RAG) pipelines

Sometimes the ingestion pipeline fails due to malformed JSON and duplicate payloads.

The engineers at Contoso report that the following dashboard query runs slowly.

```
SELECT VehicleId, LastUpdatedUtc, EngineStatus, BatteryHealth
FROM dbo.VehicleHealthSummary
WHERE FleetId = @FleetId
ORDER BY LastUpdatedUtc DESC;
```

You review the execution plan and discover that the plan shows a clustered index scan.

`VehicleIncidentReports` often contains details about the weather, traffic conditions, and location. Analysts report that it is difficult to find similar incidents based on these details.

Requirements
Planned Changes
Contoso wants to modernize Fleet Intelligence Platform to support AI-powered semantic search over incident reports.

Security Requirements
Contoso identifies the following security requirements:

- Restrict the support staff from viewing Personally Identifiable Information (PII) data, which is full email addresses and phone numbers.
- Enforce row-level filtering so that analysts see only incidents for the fleets to which they are assigned. The analysts can be assigned to multiple fleets.

Database Performance and Requirements
Contoso identifies the following telemetry requirements:

- Telemetry data must be stored in a partitioned table.
- Telemetry data must provide predictable performance for ingestion and retention operations.
- `latitude`, `longitude`, and `accuracy` JSON properties must be filtered by using an index seek.

Contoso identifies the following maintenance data requirements:

- Ensure that any changes to a row in the `MaintenanceEvents` table updates the corresponding value in the `LastModifiedUtc` column to the time of the change.
- Avoid recursive updates.

AI Search, Embeddings, and Vector Indexing
Contoso plans to implement semantic search over incident data to meet the following requirements:

- Embeddings must be stored in dedicated Azure SQL Database tables.
- Embeddings must be generated from rich natural language fields.
- Chunking must preserve semantic coherence.
- Hybrid search must combine the following:
  - Vector similarity
  - Keyword filtering or boosting

Development Requirements
The development team at Contoso will use Microsoft Visual Studio Code and GitHub Copilot and will retrieve live metadata from the databases.

Contoso identifies the following requirements for querying data in the `FeedbackJson` column of the `CustomerFeedback` table:

- Extract the customer feedback text from the JSON document.
- Filter rows where the JSON text contains a keyword.
- Calculate a fuzzy similarity score between the feedback text and a known issue description.
- Order the results by similarity score, with the highest score first.

You need to recommend a solution to resolve the slow dashboard query issue. What should you recommend?

A. Create a clustered index on LastUpdatedUtc.
B. On FleetId, create a nonclustered index that includes LastUpdatedUtc, EngineStatus, and BatteryHealth.
C. On LastUpdatedUtc, create a nonclustered index that includes FleetId.
D. On FleetId, create a filtered index where LastUpdatedUtc > DATEADD(DAY, -7, SYSUTCDATETIME()).

Answer: B
Explanation:
Scenario:
The engineers at Contoso report that the following dashboard query runs slowly.

```
SELECT VehicleId, LastUpdatedUtc, EngineStatus, BatteryHealth
FROM dbo.VehicleHealthSummary
WHERE FleetId = @FleetId
ORDER BY LastUpdatedUtc DESC;
```

You review the execution plan and discover that the plan shows a clustered index scan.

To optimize this query, you should create a covering nonclustered index that handles both the filtering (WHERE) and the sorting (ORDER BY) requirements while including the remaining columns to avoid expensive lookups.

Recommended Index Strategy
Creating an index with FleetID as the first key column and LastUpdateUtc as the second key column will allow SQL Server to perform an Index Seek to find the specific fleet and then retrieve those rows in the pre-sorted order required by your ORDER BY clause.

T-SQL Implementation:

CREATE NONCLUSTERED INDEX IX_VehicleHealth_Fleet_Update
ON dbo.VehicleHealthSummary (FleetID, LastUpdateUtc DESC)
INCLUDE (EngineStatus, BatteryHealth);
Use code with caution.

Why this works
Eliminates Clustered Index Scan: The current plan scans the entire table because no index exists that starts with FleetID. This new index allows the engine to jump directly to the relevant rows.

Avoids a Sort Operator: By including LastUpdateUtc DESC in the index key, the data is already physically ordered. SQL Server can read the index and return the results immediately without needing a costly in-memory sort.

Fully Covers the Query: Using the INCLUDE clause for EngineStatus and BatteryHealth ensures all data required by the SELECT statement is present in the index. This prevents "Key Lookups," where the engine would otherwise have to go back to the original table for those specific values.

Reference:
https://www.mssqltips.com/sqlservertip/8192/sql-server-uses-non-clustered-index-rather-than-clustered-index

QUESTION 2
Drag and Drop Question

You have an Azure SQL database that contains a table named dbo.Orders.

You have an application that calls a stored procedure named dbo.usp_CreateOrder to insert rows into dbo.Orders.

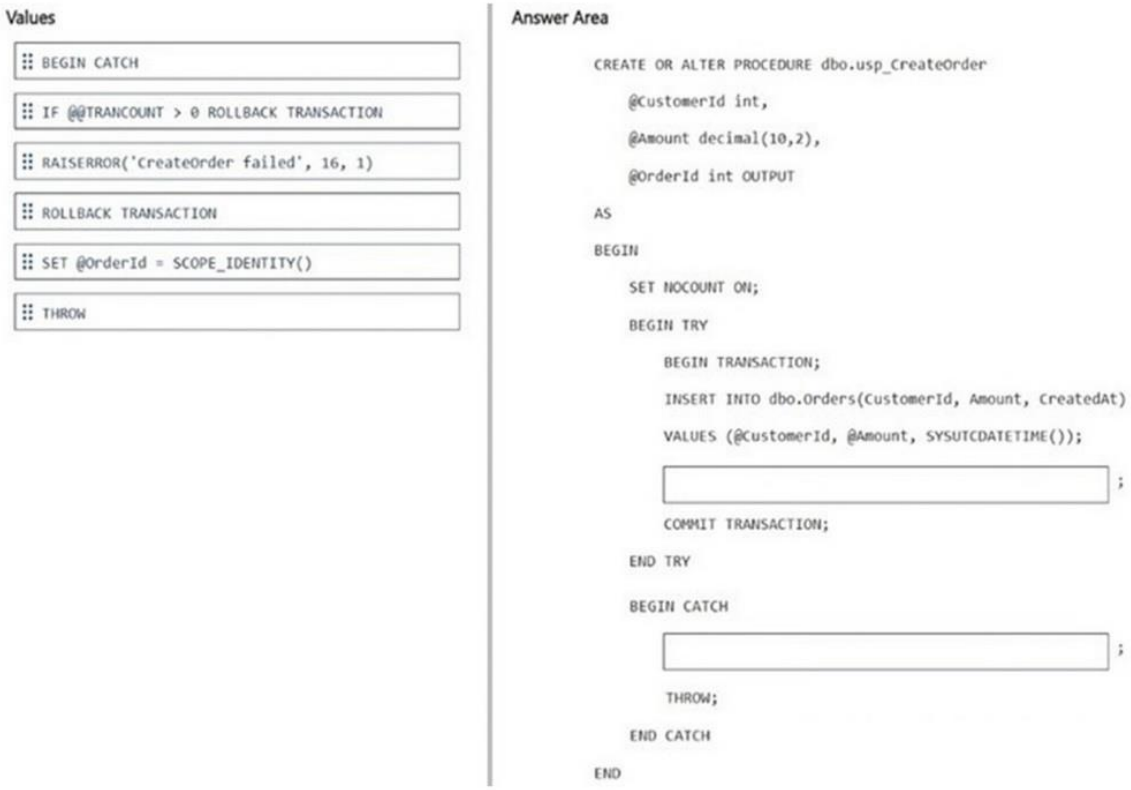
When an insert fails, the application receives inconsistent error details.

You need to implement error handling to ensure that any failures inside the procedure abort the transaction and return a consistent error to the caller.

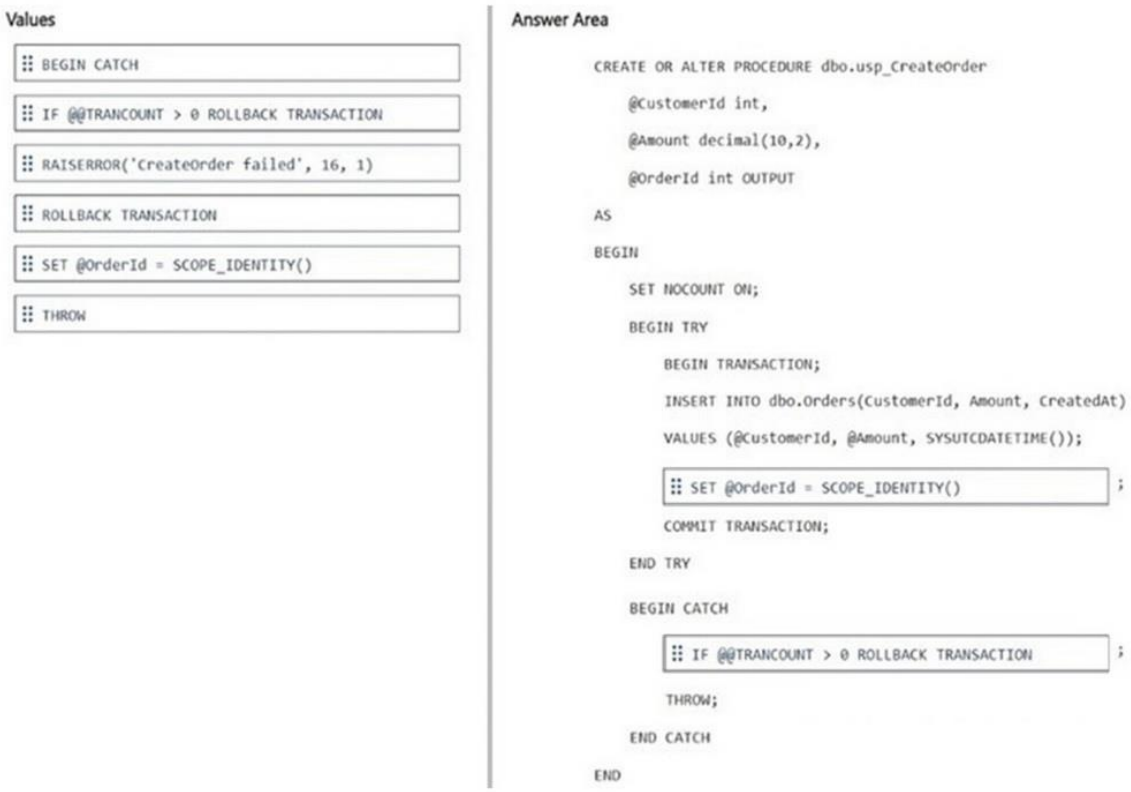
How should you complete the stored procedure? To answer, drag the appropriate values to the correct targets. Each value may be used once, more than once, or not at all. You may need to

drag the split bar between panes or scroll to view content.

NOTE: Each correct selection is worth one point.



Answer:



QUESTION 3
Hotspot Question

You have an Azure SQL database that contains a table named knowledge_base. knowledge_base stores human resources (HR) policy documents and contains columns named title, content, category, and embedding.

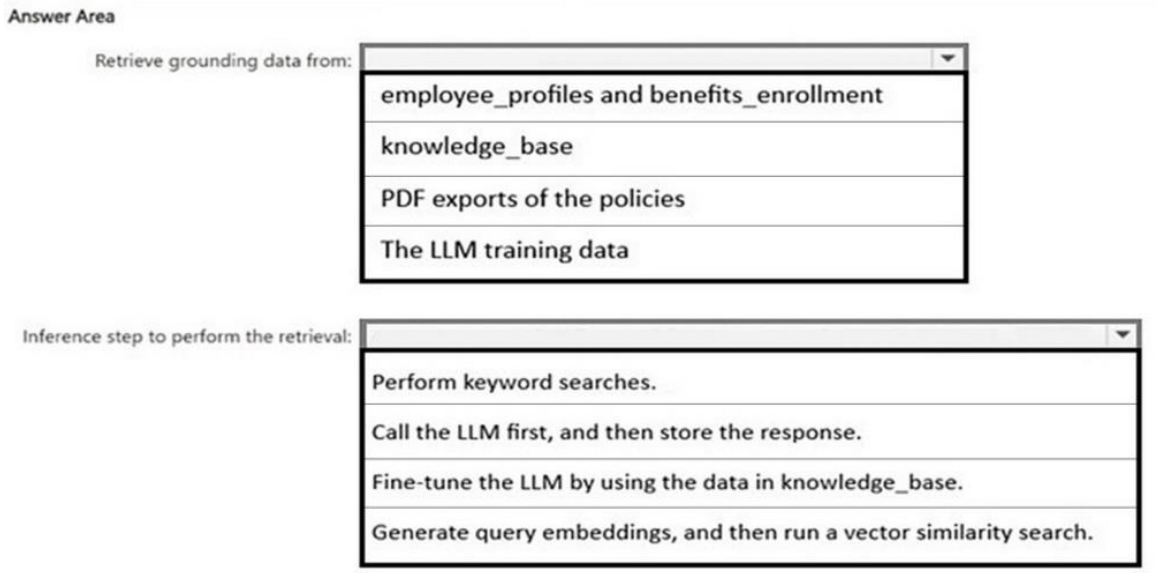
You have an application named App1. App1 queries two relational tables named employee_profiles and benefits_enrollment that contain HR data. App1 hosts a chatbot that calls a large language model (LLM) directly.

Users report that the chatbot answers general HR questions correctly but provides outdated or incorrect answers when policies change. The chatbot also fails to answer questions that reference internal policy documents by title or category.

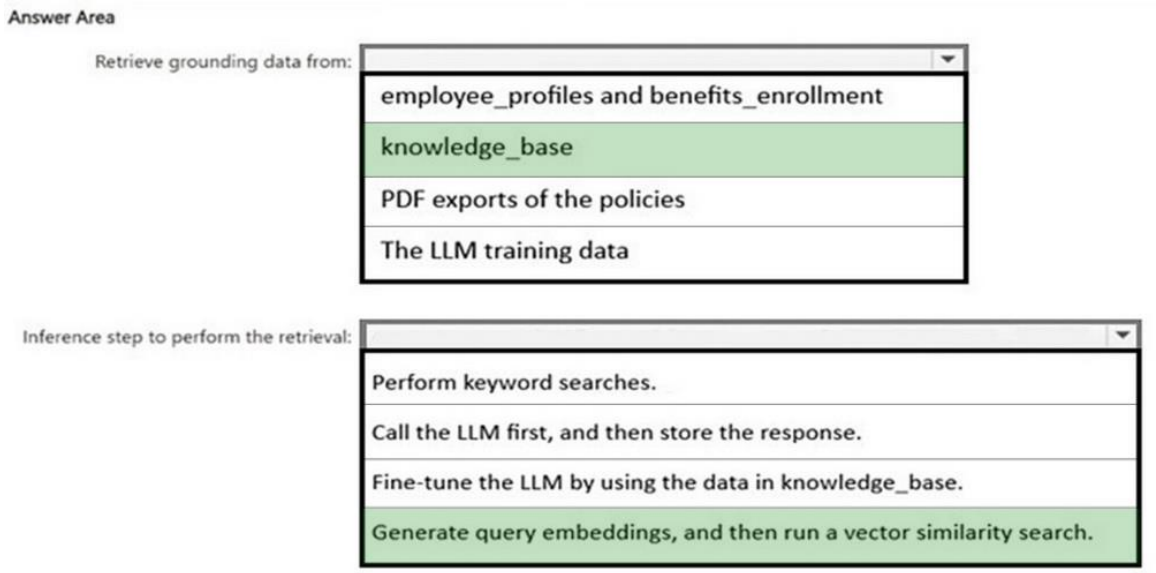
You need to recommend a Retrieval Augmented Generation (RAG) solution to resolve the chatbot issues.

What should you recommend? To answer, select the appropriate options in the answer area.

NOTE: Each correct selection is worth one point.



Answer:



QUESTION 4
You have an Azure SQL database.

You need to create a scalar user-defined function (UDF) that returns the number of whole years between an input parameter named @OrderDate and the current date/time as a single positive integer. The function must be created in Azure SQL Database.

You write the following code.

```
01 CREATE FUNCTION dbo.ufnYearsSinceOrder (@OrderDate datetime2)
02 RETURNS int
03 AS
04 BEGIN
05
06 END
```

What should you insert at line 05?

A. RETURN DATEDIFF(year, GETDATE(), @OrderDate);
B. DATEDIFF(month, @orderdate, GETDATE()) / 12
C. DATEPART(year, GETDATE()) - DATEPART(year, @orderdate)
D. RETURN DATEDIFF(year, @OrderDate, GETDATE());

Answer: D
Explanation:
Use RETURN to produce the scalar value of the function.

In an Azure SQL Database scalar function (a user-defined function that returns a single value), you must use the RETURN statement to return the scalar value.
The RETURN statement immediately terminates the function's execution and returns the value specified in its argument to the calling statement or procedure. The value returned must be of the data type specified in the RETURNS clause of the function definition.

The second argument to DATEDIFF should be @OrderDate as it is the start date, while the third argument is the end date, which is the current date.

Note:
DATEDIFF (Transact-SQL)

This function returns the count (as a signed integer value) of the specified datepart boundaries crossed between the specified startdate and enddate.

Syntax
DATEDIFF (datepart , startdate , enddate)

Arguments
datepart
Specifies the units in which DATEDIFF reports the difference between the startdate and enddate. Commonly used datepart units include month or second.

Reference:
https://learn.microsoft.com/en-us/sql/t-sql/functions/datediff-transact-sql

QUESTION 5
You have an Azure SQL database.

You deploy Data API builder (DAB) to Azure Container Apps by using the mcr.microsoft.com/azure-databases/data-api-builder:latest image.

You have the following Container Apps secrets:

```
- MSSQL_CONNECTION_STRING that maps to the SQL connection string
```

```
- DAB_CONFIG_BASE64 that maps to the DAB configuration
```

You need to initialize the DAB configuration to read the SQL connection string.

Which command should you run?

A. dab init --database-type mssql --connection-string “secretref:DAB_CONFIG_BASE64” --host-mode Production --config dab-config.json
B. dab init --database-type mssql --connection-string “@env(‘MSSQL_CONNECTION_STRING’)” --host-mode Production --config dab-config.json
C. dab init --database-type mssql --connection-string “secretref:mssql-connection-string” --host-mode Production --config dab-config.json
D. dab init --database-type mssql --connection-string “@env(‘DAB_CONFIG_BASE64’)” --host-mode Production --config dab-config.json

Answer: B
Explanation:
To initialize the Data API builder (DAB) configuration to read the SQL connection string from your Container Apps secret, use the following dab init command:

dab init --database-type mssql --connection-string "@env('MSSQL_CONNECTION_STRING')"

Why this command works
--database-type mssql: Specifies that you are connecting to an Azure SQL or SQL Server database.

@env('MSSQL_CONNECTION_STRING'): This is the built-in DAB function that tells the runtime to substitute the value of the specified environment variable at load time. Since your Container Apps secret is mapped to MSSQL_CONNECTION_STRING, DAB will resolve it automatically when the container starts.

--connection-string: This flag sets the data source connection. By using the @env() syntax here, you ensure the secret remains out of the static configuration file.

Reference:
https://learn.microsoft.com/en-us/azure/data-api-builder/command-line/dab-init

QUESTION 6
You have a SQL database in Microsoft Fabric that contains a nvarchar (max) column named MessageText. An ID is always contained within the first paragraph of MessageText.

You need to write a Transact-SQL query that uses REGEXP_SUBSTR to extract the ID from MessageText.

What should you include in the query?

A. Apply STRING_ESCAPE(MessageText, ‘json’) before calling REGEXP_SUBSTR.
B. Cast MessageText to nvarchar (4000) before calling REGEXP_SUBSTR.
C. Add a COLLATE Latin1_General_CS_AS clause to MessageText before calling REGEXP_SUBSTR.
D. Run TRY_CONVERT(varchar(max), MessageText) before calling REGEXP_SUBSTR.

Answer: A
Explanation:

To extract an ID (e.g., alphanumeric) from the first paragraph of an nvarchar(max) column in Microsoft Fabric using STRING_ESCAPE, use STRING_SPLIT or CHARINDEX to isolate the first paragraph, apply STRING_ESCAPE, and a regex pattern.

Note: T-SQL does not natively support a REGEXP_SUBSTR function like Oracle/Snowflake. The solution below uses STRING_ESCAPE followed by pattern matching via PATINDEX and SUBSTRING to extract a typical alphanumeric ID.

Reference:
https://learn.microsoft.com/en-us/sql/t-sql/queries/contains-transact-sql

QUESTION 7
You have an Azure SQL database that contains database-level Data Definition Language (DDL) triggers, including a trigger named ddl_Audit.

You need to prevent ddl_Audit from firing during the next deployment. The trigger object must remain in place.

Which Transact-SQL statement should you use?

A. ALTER TRIGGER
B. ALTER DATABASE
C. ALTER SERVER AUDIT SPECIFICATION
D. DISABLE TRIGGER
E. ALTER DATABASE AUDIT SPECIFICATION

Answer: D
Explanation:
The DISABLE TRIGGER Transact-SQL statement is the correct and appropriate solution for this scenario.

Solution Breakdown
To prevent a specific database-level DDL trigger from firing without removing the object, you can use the following syntax:

DISABLE TRIGGER [TriggerName] ON DATABASE;

Key Considerations
Object Retention: A disabled trigger remains in the database as an object and is visible in catalog views like sys.triggers, but it will not execute when its programmed events occur.

Reactivation: You can re-enable the trigger after your deployment is complete using the ENABLE TRIGGER statement.

Permissions: To execute this command on a database-scoped DDL trigger in Azure SQL, you must have at least ALTER ANY DATABASE DDL TRIGGER permission.

Reference:
https://learn.microsoft.com/en-us/sql/t-sql/statements/disable-trigger-transact-sql

Thank You for Trying Our Product

Lead2pass Certification Exam Features:

★ More than **99,900** Satisfied Customers Worldwide.

★ Average **99.9%** Success Rate.

★ **Free Update** to match latest and real exam scenarios.

★ **Instant Download** Access! No Setup required.

★ Questions & Answers are downloadable in **PDF** format and **VCE** test engine format.

★ Multi-Platform capabilities - **Windows, Laptop, Mac, Android, iPhone, iPod, iPad**.

★ **100%** Guaranteed Success or **100%** Money Back Guarantee.

★ **Fast**, helpful support **24x7**.

View list of all certification exams: http://www.lead2pass.com/all-products.html

10% Discount Coupon Code: **ASTR14**