

KTest

高品質 より良いサービス



練習問題

<https://www.ktest.jp>

1年間無料アップデート

Exam : DP-800

**Title : Developing AI-Enabled
Database Solutions**

Version : DEMO

1. Topic 1, Contoso Case Study

Existing 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)
VehicleHealthSummary	- IncidentId (int) (primarykey) - VehicleId (int) - 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.

Requirements

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 = gFleetId ORDER BY LastUpdatedUtc DESC;
```

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

vehicleincident Reports often contains details about the weather, traffic conditions, and location.

Analysts report that it is difficult to find similar incidents based on these details.

Planned Changes

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

Security 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 LastModified column to the time of the change.
- Avoid recursive updates.

AI Search, Embedding's, and Vector indexing

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 customer-Feedback 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 generate embeddings to resolve the issues identified by the analysts.

Which column should you use?

- A. vehicleLocation
- B. incidentDescription

- C. incidentType
- D. SeverityScore

Answer: B

Explanation:

The correct column to use for generating embeddings is incidentDescription because embeddings are intended to represent the semantic meaning of rich textual content, not simple categorical, numeric, or location-only values. Microsoft's DP-800 study guide explicitly includes skills such as identifying which columns to include in embeddings, generating embeddings, and implementing semantic vector search for scenarios where users need to find similar records based on meaning rather than exact matches. In this scenario, analysts report that it is difficult to find similar incidents based on details such as weather, traffic conditions, and location. Those are descriptive context elements that are typically captured in a free-text incident description field. An embedding generated from incidentDescription can encode the semantic relationships among these narrative details, making it suitable for similarity search, semantic search, and RAG retrieval. Microsoft documentation on vectors and embeddings explains that embeddings are generated from text data and then stored for vector search to find semantically related items.

The other options are weaker choices:

vehicleLocation is too narrow and usually better handled with geospatial filtering, not embeddings.

incidentType is likely categorical and too low in semantic richness.

SeverityScore is numeric and not appropriate as the primary source for semantic embeddings.

Microsoft also notes that when multiple useful attributes exist, you can either embed each text column separately or concatenate relevant text fields into one textual representation before generating the embedding. But among the options given, the best and most exam-aligned answer is the textual narrative column: incidentDescription.

2.You need to recommend a solution for the development team to retrieve the live metadata. The solution must meet the development requirements.

What should you include in the recommendation?

- A. Export the database schema as a .dacpac file and load the schema into a GitHub Copilot context window.
- B. Add the schema to a GitHub Copilot instruction file.
- C. Use an MCP server
- D. Include the database project in the code repository.

Answer: C

Explanation:

The best recommendation is to use an MCP server. In the official DP-800 study guide, Microsoft explicitly lists skills such as configuring Model Context Protocol (MCP) tool options in a GitHub Copilot session and connecting to MCP server endpoints, including Microsoft SQL Server and Fabric Lakehouse. That makes MCP the exam-aligned mechanism for enabling AI-assisted tools to work with live database context rather than static snapshots.

This also matches the stated development requirement: the team will use Visual Studio Code and GitHub Copilot and needs to retrieve live metadata from the databases. Microsoft's documentation for GitHub Copilot with the MSSQL extension explains that Copilot works with an active database connection, provides schema-aware suggestions, supports chatting with a connected database, and

adapts responses based on the current database context. Microsoft also documents MCP as the standard way for AI tools to connect to external systems and data sources through discoverable tools and endpoints.

The other options do not satisfy the “live metadata” requirement as well:

A .dacpac is a point-in-time schema artifact, not live metadata.

A Copilot instruction file provides guidance, not live database discovery.

Including the database project in the repository helps source control and deployment, but it still does not provide live database metadata by itself.

3.You need to recommend a solution that will resolve the ingestion pipeline failure issues.

Which two actions should you recommend? Each correct answer presents part of the solution. NOTE:

Each correct selection is worth one point.

A. Enable snapshot isolation on the database.

B. Use a trigger to automatically rewrite malformed JSON.

C. Add foreign key constraints on the table.

D. Create a unique index on a hash of the payload.

E. Add a check constraint that validates the JSON structure.

Answer: D, E

Explanation:

The two correct actions are D and E because the ingestion failures are caused by malformed JSON and duplicate payloads, and these two controls address those two problems directly. Microsoft's JSON documentation states that SQL Server and Azure SQL support validating JSON with ISJSON, and Microsoft specifically recommends using a CHECK constraint to ensure JSON text stored in a column is properly formatted.

For the duplicate-payload issue, creating a unique index on a hash of the payload is the appropriate design. Microsoft documents using hashing functions such as HASHBYTES to hash column values, and SQL Server allows a deterministic computed column to be used as a key column in a UNIQUE constraint or unique index. That makes a persisted hash-based computed column plus a unique index a practical and exam-consistent way to reject duplicate payloads efficiently.

The other options do not solve the stated root causes:

Snapshot isolation addresses concurrency behavior, not malformed JSON or duplicate payload detection.

A trigger to rewrite malformed JSON is not the right integrity control and is brittle.

Foreign key constraints enforce referential integrity, not JSON validity or duplicate-payload prevention

4.HOTSPOT

You need to meet the development requirements for the FeedbackJson column

How should you complete the Transact SQL query? To answer, select the appropriate options in the answer area. NOTE: Each correct selection is worth one point.

Answer Area

```

SELECT
    f.FeedbackId,
    f.VehicleId,

    CONTAINS(FeedbackJson, @Keyword)
    EDIT_DISTANCE( JSON_VALUE(f.FeedbackJson, '$.details.comment'), @Keyword) < 3
    EDIT_DISTANCE(JSON_VALUE(f.FeedbackJson, '$.text'), @Keyword) < 3
    JSON_QUERY(f.FeedbackJson, '$.text', @KnownIssueDescription) AS FeedbackText
    JSON_VALUE(f.FeedbackJson, '$.text') AS FeedbackText
    SimilarityScore
    dbo.CustomerFeedback f

    JSON_VALUE(f.FeedbackJson, '$.text'),
    @KnownIssueDescription
) AS SimilarityScore
FROM
    dbo.CustomerFeedback f
WHERE

```

```

CONTAINS(FeedbackJson, @Keyword)
EDIT_DISTANCE( JSON_VALUE(f.FeedbackJson, '$.details.comment'), @Keyword) < 3
EDIT_DISTANCE(JSON_VALUE(f.FeedbackJson, '$.text'), @Keyword) < 3
JSON_QUERY(f.FeedbackJson, '$.text', @KnownIssueDescription) AS FeedbackText
JSON_VALUE(f.FeedbackJson, '$.text') AS FeedbackText
SimilarityScore

```

```

CONTAINS(FeedbackJson, @Keyword)
EDIT_DISTANCE( JSON_VALUE(f.FeedbackJson, '$.details.comment'), @Keyword) < 3
EDIT_DISTANCE(JSON_VALUE(f.FeedbackJson, '$.text'), @Keyword) < 3
JSON_QUERY(f.FeedbackJson, '$.text', @KnownIssueDescription) AS FeedbackText
JSON_VALUE(f.FeedbackJson, '$.text') AS FeedbackText
SimilarityScore

```

Answer:

Answer Area

```

SELECT
    f.FeedbackId,
    f.VehicleId,

    CONTAINS(FeedbackJson, @Keyword)
    EDIT_DISTANCE( JSON_VALUE(f.FeedbackJson, '$.details.comment'), @Keyword) < 3
    EDIT_DISTANCE(JSON_VALUE(f.FeedbackJson, '$.text'), @Keyword) < 3
    JSON_QUERY(f.FeedbackJson, '$.text', @KnownIssueDescription) AS FeedbackText
    JSON_VALUE(f.FeedbackJson, '$.text') AS FeedbackText
    SimilarityScore
    dbo.CustomerFeedback f

    JSON_VALUE(f.FeedbackJson, '$.text'),

    @KnownIssueDescription

) AS SimilarityScore

FROM

    dbo.CustomerFeedback f

WHERE

```

```

CONTAINS(FeedbackJson, @Keyword)
EDIT_DISTANCE( JSON_VALUE(f.FeedbackJson, '$.details.comment'), @Keyword) < 3
EDIT_DISTANCE(JSON_VALUE(f.FeedbackJson, '$.text'), @Keyword) < 3
JSON_QUERY(f.FeedbackJson, '$.text', @KnownIssueDescription) AS FeedbackText
JSON_VALUE(f.FeedbackJson, '$.text') AS FeedbackText
SimilarityScore

```

```

CONTAINS(FeedbackJson, @Keyword)
EDIT_DISTANCE( JSON_VALUE(f.FeedbackJson, '$.details.comment'), @Keyword) < 3
EDIT_DISTANCE(JSON_VALUE(f.FeedbackJson, '$.text'), @Keyword) < 3
JSON_QUERY(f.FeedbackJson, '$.text', @KnownIssueDescription) AS FeedbackText
JSON_VALUE(f.FeedbackJson, '$.text') AS FeedbackText
SimilarityScore

```

Explanation:

JSON_VALUE(f.FeedbackJson, '\$.text') AS FeedbackText

CONTAINS(FeedbackJson, @Keyword)

SimilarityScore

These three selections are the correct way to complete the query because they align exactly with the stated requirements for the FeedbackJson column.

First, to extract the customer feedback text from the JSON document, the correct expression is JSON_VALUE(f.FeedbackJson, '\$.text') AS FeedbackText. Microsoft documents that JSON_VALUE is

used to extract a scalar value from JSON, while JSON_QUERY is used for returning an object or array. Since \$.text is the textual feedback string, JSON_VALUE is the correct function.

Second, to filter rows where the JSON text contains a keyword, the best choice is CONTAINS(FeedbackJson, @Keyword). The scenario explicitly states that FeedbackJson already has a full-text index, and Microsoft documents that CONTAINS is the full-text predicate used in the WHERE clause to search full-text indexed character data. That makes it more appropriate than using EDIT_DISTANCE for keyword filtering.

Third, to order the results by similarity score, highest first, the correct item is SimilarityScore in the ORDER BY clause, which would be paired with DESC in the query. This matches the requirement to sort by the computed fuzzy similarity value. The DP-800 study guide specifically includes writing queries that use fuzzy string matching functions such as EDIT_DISTANCE, which supports the earlier computed SimilarityScore expression in the query.

5.DRAG DROP

You need to meet the database performance requirements for maintenance data

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.

Values

⋮ i.MaintenanceId IS NOT NULL

⋮ m.LastModifiedUtc <> i.LastModifiedUtc

⋮ m.MaintenanceId = i.MaintenanceId

⋮ m.VehicleId = i.VehicleId

Answer Area

```
CREATE TRIGGER dbo.trgMaintenanceEvents_UpdateTimestamp
ON dbo.MaintenanceEvents
AFTER UPDATE
AS
BEGIN
UPDATE m
SET LastModifiedUtc = SYSUTCDATETIME()
FROM dbo.MaintenanceEvents m
INNER JOIN inserted i
ON
WHERE
END;
GO
```

Answer:

Values

Answer Area

```
CREATE TRIGGER dbo.trgMaintenanceEvents_UpdateTimestamp
ON dbo.MaintenanceEvents
AFTER UPDATE
AS
BEGIN
    UPDATE m
    SET LastModifiedUtc = SYSUTCDATETIME()
    FROM dbo.MaintenanceEvents m
    INNER JOIN inserted i
    ON
    WHERE
    END;
GO
```

Explanation:

ON → m.maintenanceId = i.maintenanceId

WHERE → m.LastModifiedUtc <> i.LastModifiedUtc

The correct drag-and-drop completion is:

ON m.maintenanceId = i.maintenanceId

WHERE m.LastModifiedUtc <> i.LastModifiedUtc

This satisfies the requirement to ensure that when a row in MaintenanceEvents changes, the corresponding LastModifiedUtc value is updated to the current system time, while also helping avoid unnecessary repeat updates.

The inserted pseudo-table in a SQL Server AFTER UPDATE trigger contains the rows that were just updated. To update the matching row in the base table correctly, the trigger must join the target table row to the corresponding row in inserted by the table's primary key. In this schema, MaintenanceId is the primary key for MaintenanceEvents, so the correct join is m.maintenanceId = i.maintenanceId. Joining on VehicleId would be incorrect because multiple maintenance rows could exist for the same vehicle, which could update unintended rows. Microsoft's trigger documentation explains that inserted and deleted are used to work with the affected rows and that multi-row logic should be based on proper key matching. The WHERE m.LastModifiedUtc <> i.LastModifiedUtc predicate is used to prevent the trigger from re-updating rows where the timestamp already matches the value in inserted. That reduces redundant writes and supports the requirement to avoid recursive or repeated update behavior. In practice, this means the trigger updates only rows whose current stored timestamp differs from the just-updated version. This is the exam-appropriate pattern for a self-updating timestamp column in an AFTER UPDATE trigger.