DUMPSBOSS.

Designing and Implementing Cloud-Native Applications Using Microsoft Azure Cosmos DB

Microsoft DP-420

Version Demo

Total Demo Questions: 10

Total Premium Questions: 92

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, New Update	41
Topic 2, Case Study 1	2
Topic 3, Case Study 2	2
Topic 4, Mixed Questions	47
Total	92

QUESTION NO: 1

You plan to create an Azure Cosmos DB Core (SQL) API account that will use customer-managed keys stored in Azure Key Vault.

You need to configure an access policy in Key Vault to allow Azure Cosmos DB access to the keys.

Which three permissions should you enable in the access policy? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Wrap Key

B. Get

C. List

D. Update

E. Sign

F. Verify

G. Unwrap Key

ANSWER: A B G

Explanation:

To Configure customer-managed keys for your Azure Cosmos account with Azure Key Vault: Add an access policy to your Azure Key Vault instance:

1. From the Azure portal, go to the Azure Key Vault instance that you plan to use to host your encryption keys. Select Access Policies from the left menu:

2. Select + Add Access Policy.

3. Under the Key permissions drop-down menu, select Get, Unwrap Key, and Wrap Key permissions:

Reference: https://docs.microsoft.com/en-us/azure/cosmos-db/how-to-setup-cmk

QUESTION NO: 2

You have an Azure Cosmos DB for NoSQL account1 that is configured for automatic failover. The account1 account has a single read-write region in West US and a and a read region in East US.

You run the following PowerShell command.

```
Update-AzCosmosDBAccountFailoverPriority -ResourceGroupName "rg1" -Name "account1" -FailoverPolicy @("East US", "West US")
```

What is the effect of running the command?

A. A manual failover will occur.

B. The account will be unavailable to writes during the change.

C. The provisioned throughput for account1 will increase.

D. The account will be configured for multi-region writes.

ANSWER: D

Explanation:

You can use the Set-AzCosmosDBAccountRegion cmdlet to update the regions that an Azure Cosmos DB account uses. You can use this cmdlet to add a region or change the region failover order. The cmdlet requires a resource group name, an Azure Cosmos DB account name, and a list of regions in desired failover order1.

For your scenario, based on the PowerShell command, you are using the Set-AzCosmosDBAccountRegion cmdlet to update the regions for an Azure Cosmos DB account named account1 that is configured for automatic failover. The command specifies two regions: West US and East US. The effect of running the command is that the account will be configured for multi-region writes.

Multi-region writes is a feature of Azure Cosmos DB that allows you to write data to any region in your account and have it automatically replicated to all other regions. This feature provides high availability and low latency for write operations across multiple regions. To enable multi-region writes, you need to specify at least two regions in your account and set them as write regions2. In your command, you are setting both West US and East US as write regions by using the -IsZoneRedundant parameter with a value of $true for both regions.

QUESTION NO: 3

You have an Azure Cosmos DB for NoSQL account named account1 that has a single read-write region and one additional read region. Account1 uses the strong default consistency level.

You have an application that uses the eventual consistency level when submitting requests to account1.

How will writes from the application be handled?

A. Writes will use the strong consistency level.

B. Azure Cosmos DB will reject writes from the application.

C. The write order is not guaranteed during replication.

D. Writes will use the eventual consistency level.

ANSWER: A

Explanation:

This is because the write concern is mapped to the default consistency level configured on your Azure Cosmos DB account, which is strong in this case. Strong consistency ensures that every write operation is synchronously committed to every region associated with your Azure Cosmos DB account. The eventual consistency level that the application uses only applies to the read operations. Eventual consistency offers higher availability and better performance, but it does not guarantee the order or latency of the reads.

QUESTION NO: 4

You need to provide a solution for the Azure Functions notifications following updates to con-product. The solution must meet the business requirements and the product catalog requirements.

Which two actions should you perform? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Configure the trigger for each function to use a different leaseCollectionPrefix

B. Configure the trigger for each function to use the same leaseCollectionName

C. Configure the trigger for each function to use a different leaseCollectionName

D. Configure the trigger for each function to use the same leaseCollectionPrefix

ANSWER: A B

Explanation:

leaseCollectionPrefix: when set, the value is added as a prefix to the leases created in the Lease collection for this Function. Using a prefix allows two separate Azure Functions to share the same Lease collection by using different prefixes.

Scenario: Use Azure Functions to send notifications about product updates to different recipients.

Trigger the execution of two Azure functions following every update to any document in the con-product container.

Reference: https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-cosmosdb-v2-trigger

QUESTION NO: 5

You have a container named container1 in an Azure Cosmos DB Core (SQL) API account. Upserts of items in container1 occur every three seconds.

You have an Azure Functions app named function1 that is supposed to run whenever items are inserted or replaced in container1.

You discover that function1 runs, but not on every upsert.

You need to ensure that function1 processes each upsert within one second of the upsert.

Which property should you change in the Function.json file of function1?

A. checkpointInterval

B. leaseCollectionsThroughput

C. maxItemsPerInvocation

D. feedPollDelay

ANSWER: D

Explanation:

With an upsert operation we can either insert or update an existing record at the same time.

FeedPollDelay: The time (in milliseconds) for the delay between polling a partition for new changes on the feed, after all current changes are drained. Default is 5,000 milliseconds, or 5 seconds.

Incorrect Answers:

A: checkpointInterval: When set, it defines, in milliseconds, the interval between lease checkpoints. Default is always after each Function call.

C: maxItemsPerInvocation: When set, this property sets the maximum number of items received per Function call. If operations in the monitored collection are performed through stored procedures, transaction scope is preserved when

reading items from the change feed. As a result, the number of items received could be higher than the specified value so that the items changed by the same transaction are returned as part of one atomic batch.

Reference: https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-cosmosdb-v2-trigger

QUESTION NO: 6 - (DRAG DROP)

DRAG DROP

You have an Azure Cosmos DB Core (SQL) API account that is configured for multi-region writes. The account contains a database that has two containers named container1 and container2.

The following is a sample of a document in container1:

{

"customerId": 1234,

"firstName": "John",

"lastName": "Smith",

"policyYear": 2021 }

The following is a sample of a document in container2:

{

"gpsId": 1234,

"latitude": 38.8951,

"longitude": -77.0364 }

You need to configure conflict resolution to meet the following requirements:

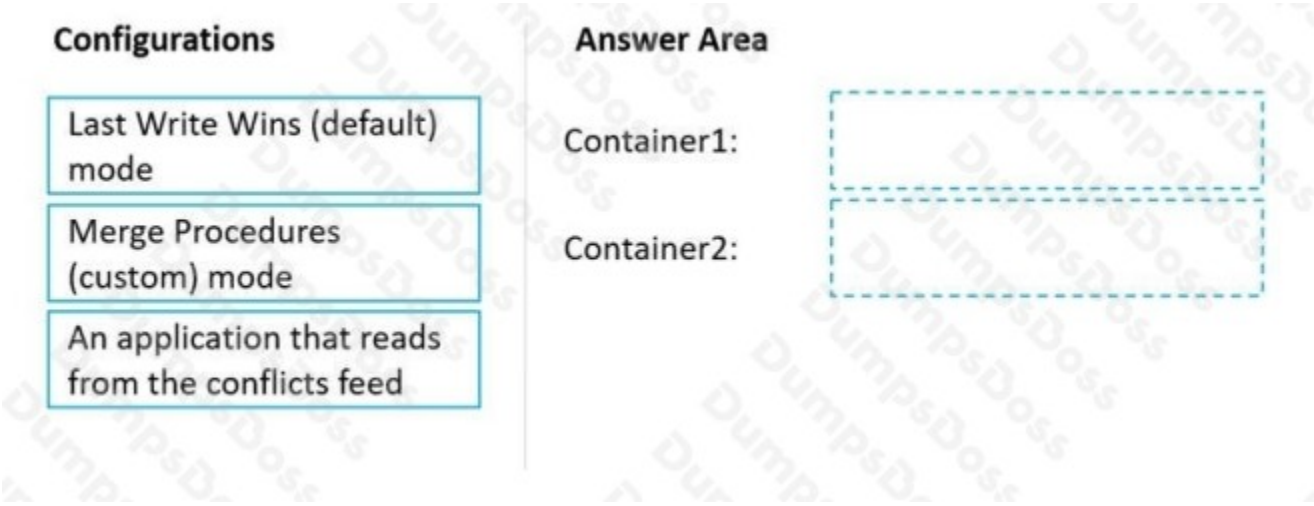
- For container1 you must resolve conflicts by using the highest value for policyYear.

- For container2 you must resolve conflicts by accepting the distance closest to latitude: 40.730610 and longitude: -73.935242. - Administrative effort must be minimized to implement the solution.

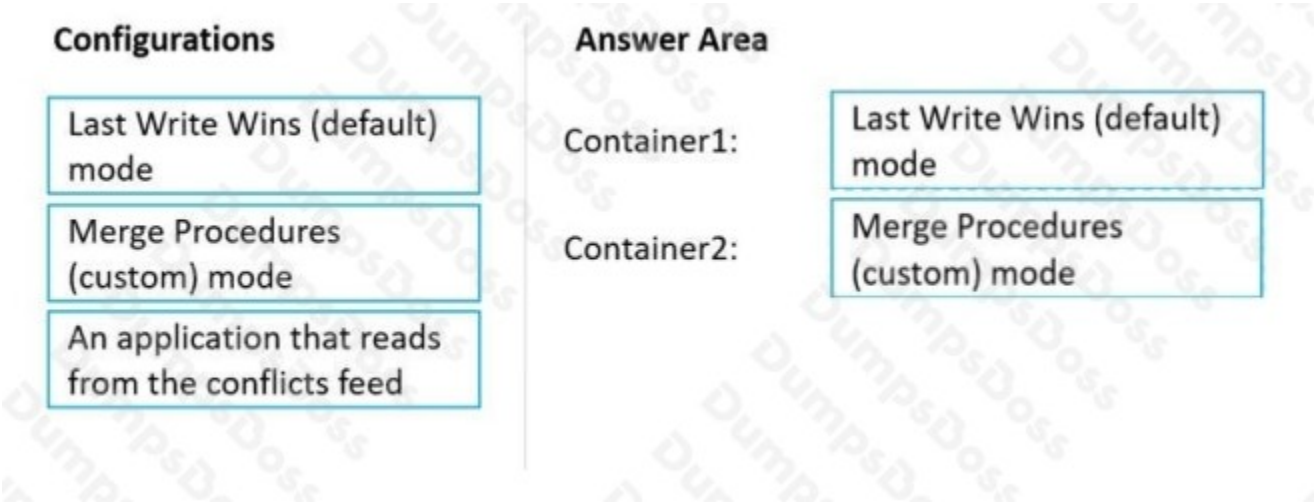
What should you configure for each container? To answer, drag the appropriate configurations to the correct containers. Each configuration may be used once, more than once, or not at all. You may need to drag the split bar between panes or scroll to view content.

NOTE: Each correct selection is worth one point.

Select and Place:



ANSWER:



Explanation:

Box 1: Last Write Wins (LWW) (default) mode

Last Write Wins (LWW): This resolution policy, by default, uses a system-defined timestamp property. It's based on the time-synchronization clock protocol.

Box 2: Merge Procedures (custom) mode

Custom: This resolution policy is designed for application-defined semantics for reconciliation of conflicts. When you set this policy on your Azure Cosmos container, you also need to register a merge stored procedure. This procedure is automatically invoked when conflicts are detected under a database transaction on the server. The system provides exactly once guarantee for the execution of a merge procedure as part of the commitment protocol.

Reference: https://docs.microsoft.com/en-us/azure/cosmos-db/conflict-resolution-policies https://docs.microsoft.com/en-us/azure/cosmos-db/sql/how-to-manage-conflicts

QUESTION NO: 7

You need to create a database in an Azure Cosmos DB for NoSQL account. The database will contain three containers named coll1, coll2 and coll3. The coll1 container will have unpredictable read and write volumes. The col!2 and coll3 containers will have predictable read and write volumes. The expected maximum throughput for coll1 and coll2 is 50,000 request units per second (RU/s) each.

How should you provision the collection while minimizing costs?

A. Create a provisioned throughput account. Set the throughput for coll1 to Manual. Set the throughput for coll2 and coll3 to Autoscale.

B. Create a provisioned throughput account. Set the throughput for call1 to Autoscale. Set the throughput for call2 and coll3 to Manual.

C. Create a serverless account.

ANSWER: B

Explanation:

Azure Cosmos DB offers two different capacity modes: provisioned throughput and serverless1. Provisioned throughput mode allows you to configure a certain amount of throughput (expressed in Request Units per second or RU/s) that is provisioned on your databases and containers. You get billed for the amount of throughput you've provisioned, regardless of how many RUs were consumed1. Serverless mode allows you to run your database operations without having to configure any previously provisioned capacity. You get billed for the number of RUs that were consumed by your database operations and the storage consumed by your data1.

To create a database that minimizes costs, you should consider the following factors:

Based on these factors, one possible option that you could choose is B. Create a provisioned throughput account. Set the throughput for coll1 to Autoscale. Set the throughput for coll2 and coll3 to Manual.

This option has the following advantages:

This option also has some limitations, such as:

Depending on your specific use case and requirements, you may need to choose a different option. For example, you could use a serverless account if all containers have low or sporadic traffic that does not require predictable performance or geo-distribution1. Alternatively, you could use a provisioned throughput account with Manual for all containers if all containers have stable and consistent traffic that requires predictable performance or geo-distribution1.

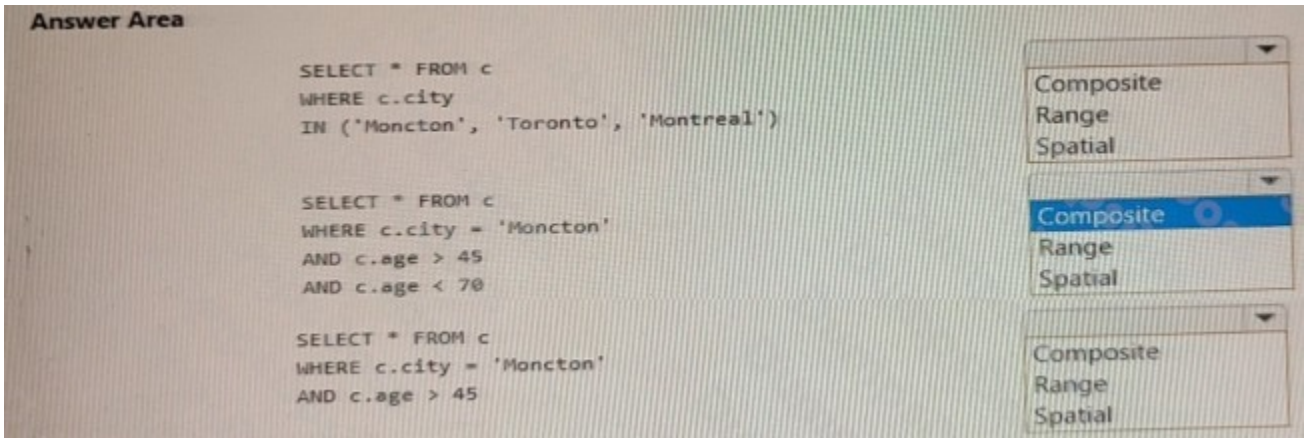
QUESTION NO: 8 - (HOTSPOT)

You have an Azure Cosmos DB for NoSQL account that frequently receives the same three queries.

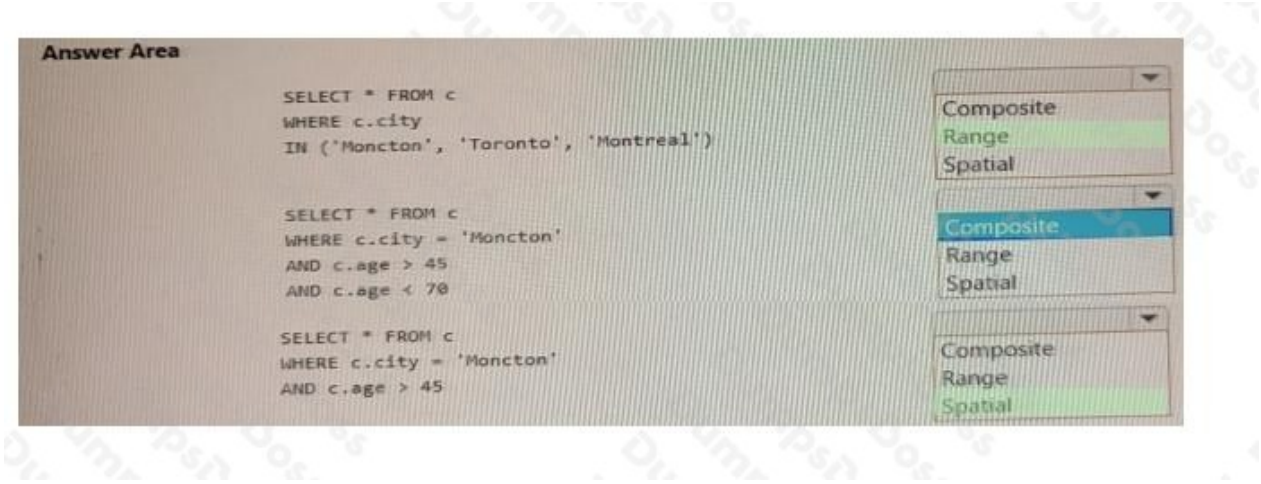
You need to configure indexing to minimize RUs consumed by the queries.

Which type of index should you use for each query? To answer, select the appropriate options in the answer area.

NOTE: Each correct selection is worth one point.



ANSWER:



Explanation:

Box 1 = Range Azure Cosmos DB supports three types of indexes: range, spatial and composite. For the query you provided, which is an equality query on a single property, the best type of index to use is range. Range index is based on an ordered tree-like structure and it is used for equality queries, range queries and checking for the presence of a property1. Range index also supports any string or number2.

Box 2 = Composite

Azure Cosmos DB supports three types of indexes: range, spatial and composite. For the query you provided, which is an order by query on two properties, the best type of index to use is composite. Composite index is used for optimizing order by queries on multiple properties1. Composite index allows you to specify a list of property paths and sort orders that are used for ordering items2.

Box 3 = spatial

Azure Cosmos DB supports three types of indexes: range, spatial and composite. For the query you provided, which is a spatial query on a point property, the best type of index to use is spatial. Spatial index is used for querying items based on their location or proximity to a given point1. Spatial index supports point, polygon and linestring data types2.

QUESTION NO: 9 - (HOTSPOT)

HOTSPOT

You have an Azure Cosmos DB Core (SQL) account that has a single write region in West Europe.

You run the following Azure CLI script.

```
az cosmosdb update -n $accountName -g $resourceGroupName \
   --locations regionName='West Europe' failoverPriority=0 isZoneRedundant-False \
   --locations regionName='North Europe' failoverPriority=1 isZoneRedundant-False

az cosmosdb failover-priority-change -n $accountName -g $resourceGroupName \
   --failover-policies 'North Europe=0' 'West Europe=1'
```

For each of the following statements, select Yes if the statement is true. Otherwise, select No.

NOTE: Each correct selection is worth one point.

Hot Area:

Answer Area

Statements	Yes	No
After running the script, there will be an instance of Azure Cosmos DB in North Europe that is writable	○	○
After running the script, the Azure Cosmos DB instance in West Europe will be writable	○	○
The cost of the Azure Cosmos DB account is unaffected by running the script	○	○

ANSWER:

Answer Area

Statements	Yes	No
After running the script, there will be an instance of Azure Cosmos DB in North Europe that is writable	○	○
After running the script, the Azure Cosmos DB instance in West Europe will be writable	○	○
The cost of the Azure Cosmos DB account is unaffected by running the script	○	○

Explanation:

Box 1: Yes

The Automatic failover option allows Azure Cosmos DB to failover to the region with the highest failover priority with no user action should a region become unavailable.

Box 2: No

West Europe is used for failover. Only North Europe is writable.

To Configure multi-region set UseMultipleWriteLocations to true.

Box 3: Yes

Provisioned throughput with single write region costs $0.008/hour per 100 RU/s and provisioned throughput with multiple writable regions costs $0.016/per hour per 100 RU/s.

Reference:

https://docs.microsoft.com/en-us/azure/cosmos-db/sql/how-to-multi-master https://docs.microsoft.com/en-us/azure/cosmos-db/optimize-cost-regions

QUESTION NO: 10

You have an Azure Cosmos DB Core (SQL) API account that is used by 10 web apps.

You need to analyze the data stored in the account by using Apache Spark to create machine learning models. The solution must NOT affect the performance of the web apps.

Which two actions should you perform? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. In an Apache Spark pool in Azure Synapse, create a table that uses cosmos.olap as the data source.

B. Create a private endpoint connection to the account.

C. In an Azure Synapse Analytics serverless SQL pool, create a view that uses OPENROWSET and the CosmosDB provider.

D. Enable Azure Synapse Link for the account and Analytical store on the container.

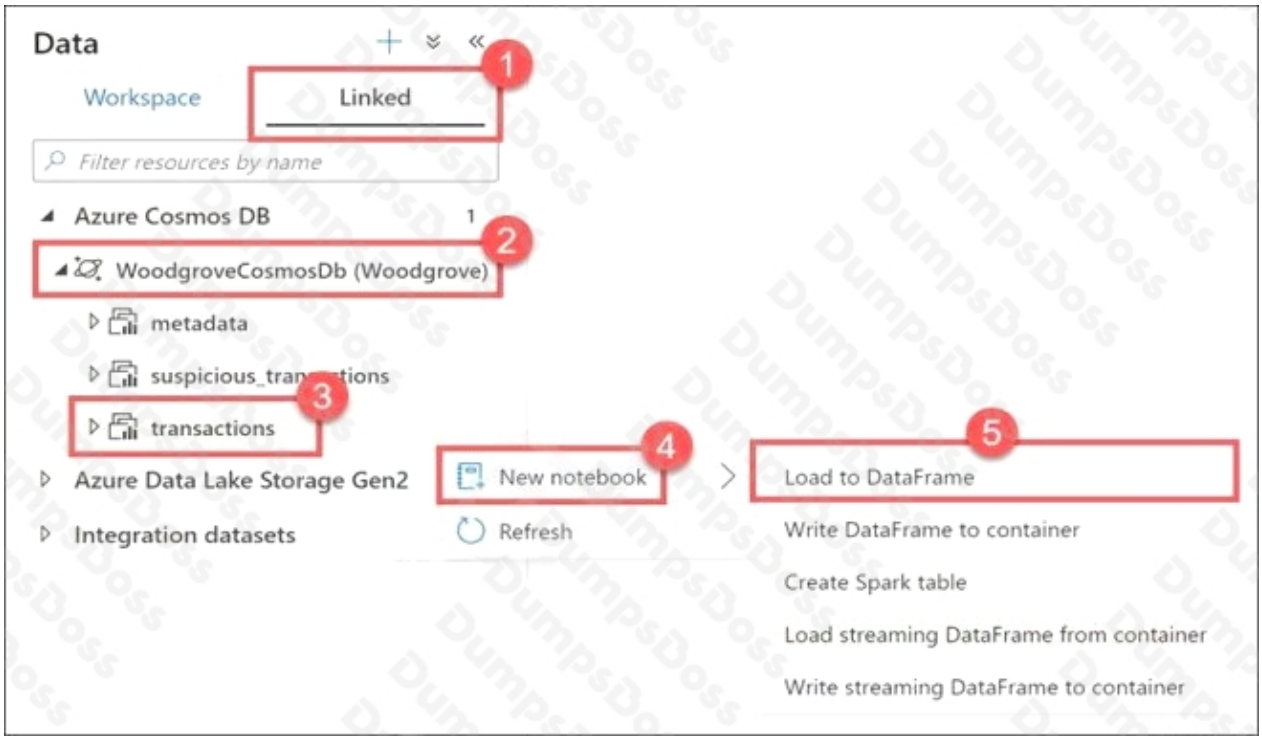
E. In an Apache Spark pool in Azure Synapse, create a table that uses cosmos.oltp as the data source.

ANSWER: A D

Explanation:

Explore analytical store with Apache Spark 1. Navigate to the Data hub.

2. Select the Linked tab (1), expand the Azure Cosmos DB group (if you don't see this, select the Refresh button above), then expand the WoodgroveCosmosDb account (2). Right-click on the transactions container (3), select New notebook (4), then select Load to DataFrame (5).



3. In the generated code within Cell 1 (3), notice that the spark.read format is set to cosmos.olap. This instructs Synapse Link to use the container's analytical store. If we wanted to connect to the transactional store, like to read from the change feed or write to the container, we'd use cosmos.oltp instead.

Reference: https://github.com/microsoft/MCW-Cosmos-DB-Real-Time-Advanced-Analytics/blob/main/Hands-on%20lab/HOL%20step-by%20step%20-%20Cosmos%20DB%20real-time%20advanced%20analytics.md