Free Questions for DP-420 by vceexamstest

Shared by Dorsey on 15-04-2024

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

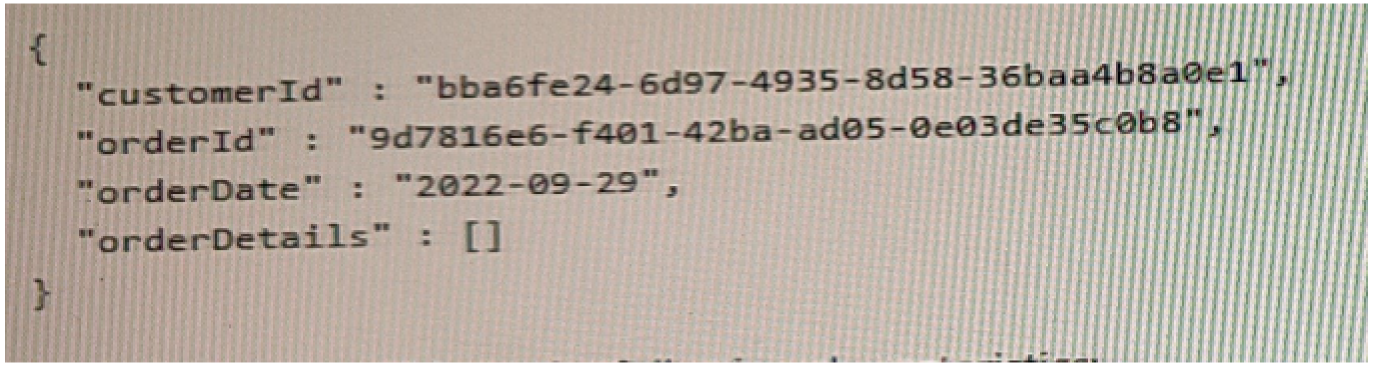
Question Type: MultipleChoice

You have a database named db1 in an Azure Cosmos DB for NoSQL

You are designing an application that will use dbl.

In db1, you are creating a new container named coll1 that will store in coll1.

The following is a sample of a document that will be stored in coll1.



```
{
    "customerId" : "bba6fe24-6d97-4935-8d58-36baa4b8a0e1",
    "orderId" : "9d7816e6-f401-42ba-ad05-0e03de35c0b8",
    "orderDate" : "2022-09-29",
    "orderDetails" : []
}
```

The application will have the following characteristics:

* New orders will be created frequently by different customers.

* Customers will often view their past order history.

You need to select the partition key value for coll1 to support the application. The solution must minimize costs.

To what should you set the partition key?

Options:

A- id

B- customerId

C- orderDate

D- orderId

Answer:

C

Explanation:

Based on the characteristics of the application and the provided document structure, the most suitable partition key value for coll1 in the given scenario would be the customerId, Option B.

The application frequently creates new orders by different customers and customers often view their past order history. Using customerId as the partition key would ensure that all orders associated with a particular customer are stored in the same partition. This enables efficient querying of past order history for a specific customer and reduces cross-partition queries, resulting in lower costs and improved performance.

a partition key is a JSON property (or path) within your documents that is used by Azure Cosmos DB to distribute data among multiple partitions3.A partition key should have a high cardinality, which means it should have many distinct values, such as hundreds or thousands1.A partition key should also align with the most common query patterns of your application, so that you can efficiently retrieve data by using the partition key value1.

Based on these criteria, one possible partition key that you could use for coll1 isB. customerId.

This partition key has the following advantages:

It has a high cardinality, as each customer will have a unique ID3.

It aligns with the query patterns of the application, as customers will often view their past order history3.

It minimizes costs, as it reduces the number of cross-partition queries and optimizes the storage and throughput utilization1.

This partition key also has some limitations, such as:

It may not be optimal for scenarios where orders need to be queried independently from customers or aggregated by date or other criteria3.

It may result in hot partitions or throttling if some customers create orders more frequently than others or have more data than others1.

It may not support transactions across multiple customers, as transactions are scoped to a single logical partition2.

Depending on your specific use case and requirements, you may need to adjust this partition key or choose a different one.For example, you could use a synthetic partition key that concatenates multiple properties of an item2, or you could use a partition key with a random or pre-calculated suffix to distribute the workload more evenly2.

Question 2

Question Type: MultipleChoice

You plan to store order data in Azure Cosmos DB for NoSQL account. The data contains information about orders and their associated items.

You need to develop a model that supports order read operations. The solution must minimize the number or requests.

Options:

A- Create a single database that contains one container. Store orders and order items in separate documents in the container.

B- Create a single database that contains one container. Create a separate document for each order and embed the order items into the order documents.

C- Create a database for orders and a database for order items.

D- Create a single database that contains a container for order and a container for order items.

Answer:

B

Explanation:

Azure Cosmos DB is a multi-model database that supports various data models, such as documents, key-value, graph, and column-family3.The core content-model of Cosmos DB's database engine is based on atom-record-sequence (ARS), which allows it to store and query different types of data in a flexible and efficient way3.

To develop a model that supports order read operations and minimizes the number of requests, you should consider the following factors:

The size and shape of your data

The frequency and complexity of your queries

The latency and throughput requirements of your application

The trade-offs between storage efficiency and query performance

Based on these factors, one possible model that you could implement isB. Create a single database that contains one container. Create a separate document for each order and embed the order items into the order documents.

This model has the following advantages:

It stores orders and order items as self-contained documents that can be easily retrieved by order ID1.

It avoids storing redundant data or creating additional containers for order items1.

It allows you to view the order history of a customer with simple queries1.

It leverages the benefits of embedding data, such as reducing the number of requests, improving query performance, and simplifying data consistency2.

This model also has some limitations, such as:

It may not be suitable for some order items that have data that is greater than 2 KB, as it could exceed the maximum document size limit of 2 MB2.

It may not be optimal for scenarios where order items need to be queried independently from orders or aggregated by other criteria2.

It may not support transactions across multiple orders or customers, as transactions are scoped to a single logical partition2.

Depending on your specific use case and requirements, you may need to adjust this model or choose a different one.For example, you could use a hybrid data model that combines embedding and referencing data2, or you could use a graph data model that expresses entities and relationships as vertices and edges.

Question 3

Question Type: MultipleChoice

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?

Options:

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:

The read and write volumes of your containers

The predictability and variability of your traffic

The latency and throughput requirements of your application

The geo-distribution and availability needs of your data

Based on these factors, one possible option that you could choose isB. 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:

It allows you to handle unpredictable read and write volumes for coll1 by using Autoscale, which automatically adjusts the provisioned throughput based on the current load1.

It allows you to handle predictable read and write volumes for coll2 and coll3 by using Manual, which lets you specify a fixed amount of provisioned throughput that meets your performance needs1.

It allows you to optimize your costs by paying only for the throughput you need for each container1.

It allows you to enable geo-distribution for your account if you need to replicate your data across multiple regions1.

This option also has some limitations, such as:

It may not be suitable for scenarios where all containers have intermittent or bursty traffic that is hard to forecast or has a low average-to-peak ratio1.

It may not be optimal for scenarios where all containers have low or sporadic traffic that does not justify provisioned capacity1.

It may not support availability zones or multi-master replication for your account1.

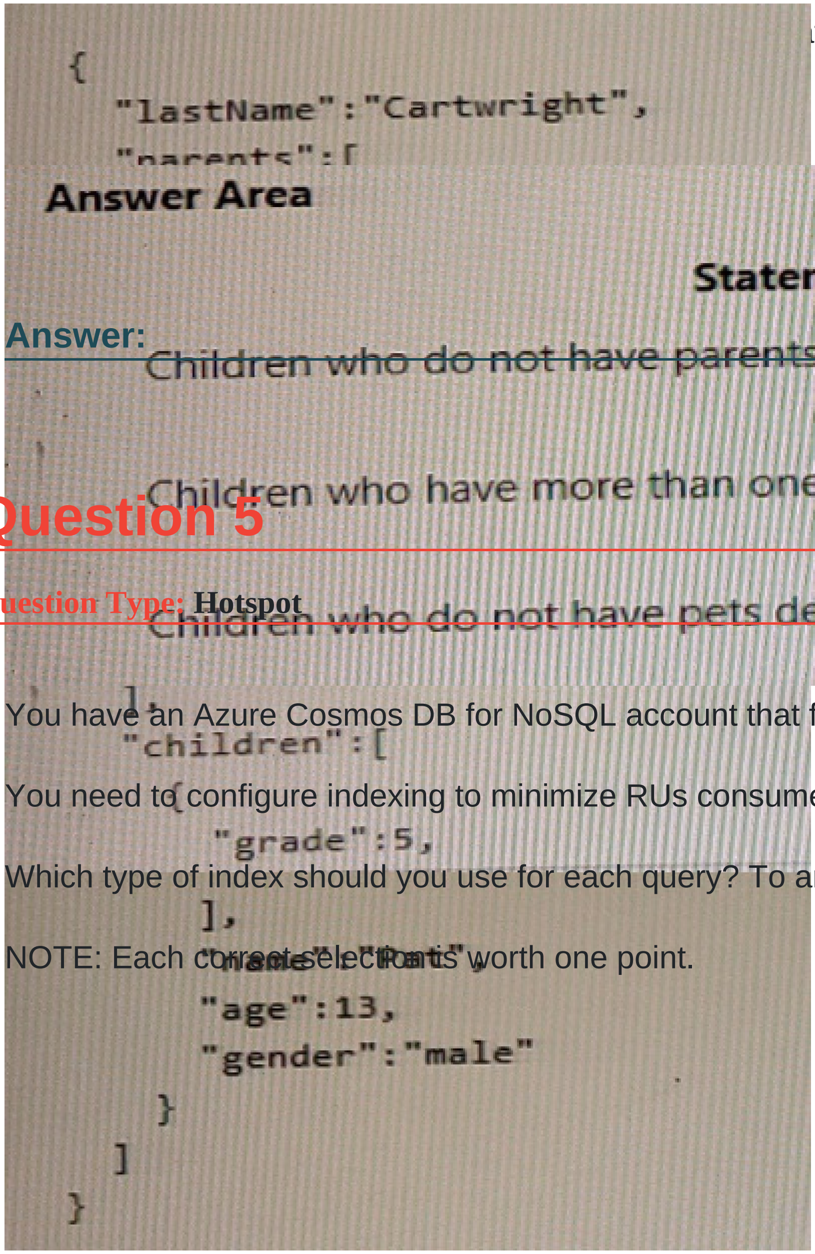
Depending on your specific use case and requirements, you may need to choose a different option.For example, you could use a serverless account if all containers have low or sporadic traffic that does not require predictable performance or geo-distribution1.Alternatively, you could use a provisioned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

Question 4

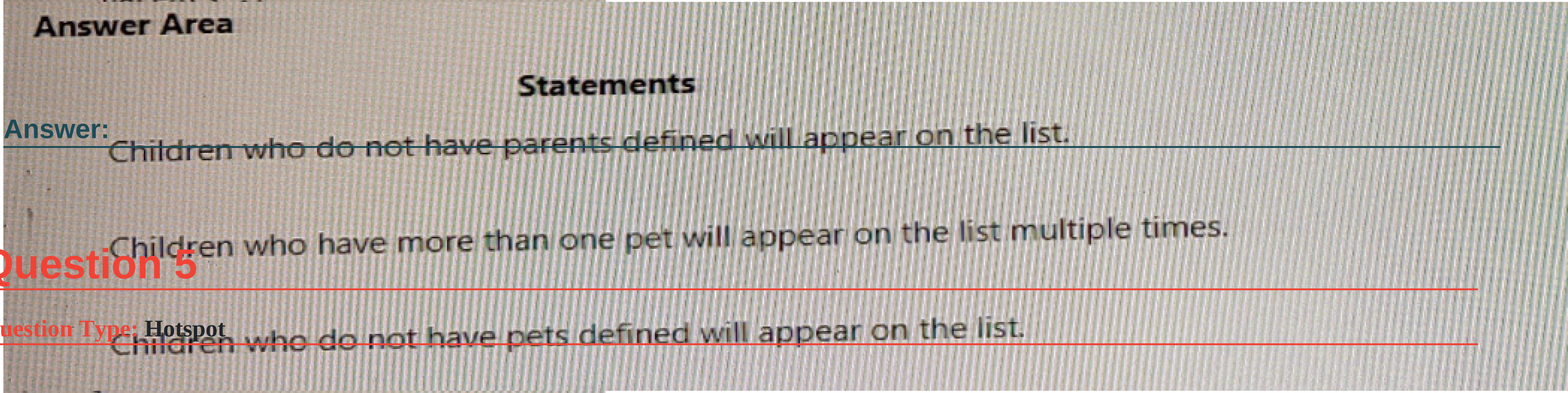
Question Type: Hotspot

You have a container that stores data about families. The following is a sample document.

tement is true. otherwise, select No.



```
{
  "lastName":"Cartwright",
  "parents":[
  ],
  "children":[
    {
      "grade":5,
      ],
      "name":"Pants",
      "age":13,
      "gender":"male"
    }
  ]
}
```



Answer Area

Statements

Children who do not have parents defined will appear on the list.

Children who have more than one pet will appear on the list multiple times.

Children who do not have pets defined will appear on the list.

Answer:

Question 5

Question Type: **Hotspot**

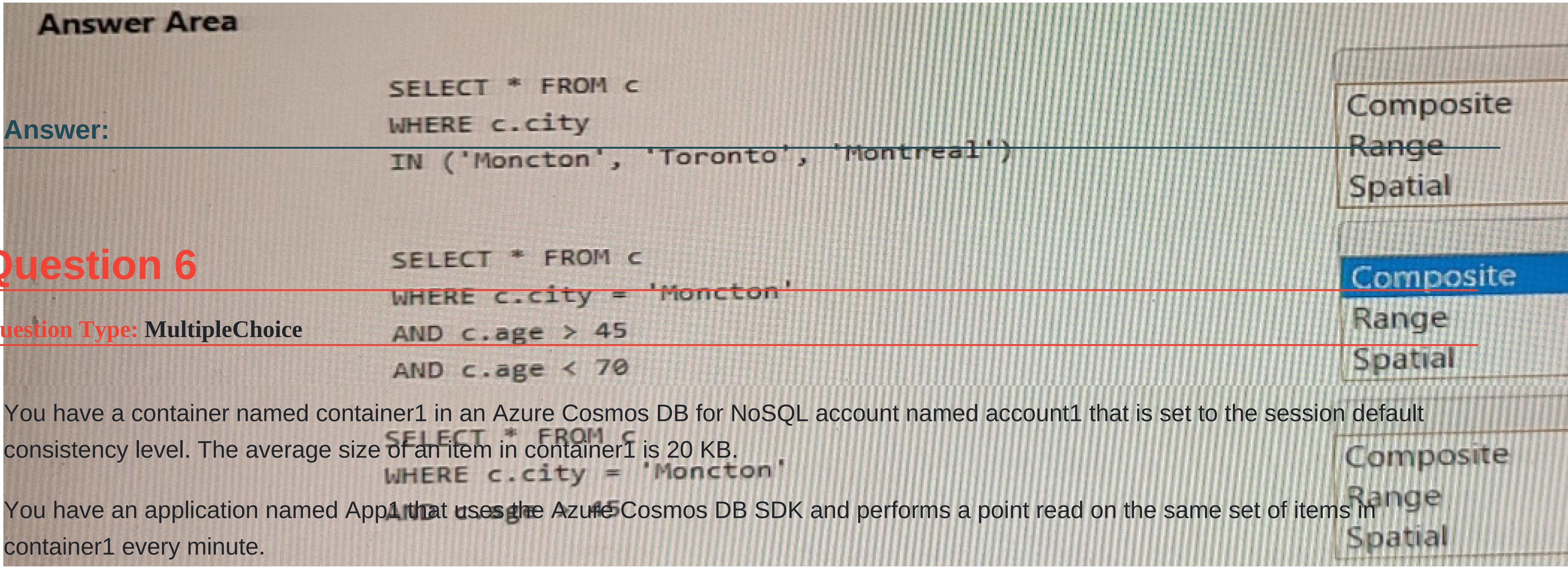
You have an Azure Cosmos DB for NoSQL account that frequently receives the same three queries.

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:



Answer Area

```
SELECT * FROM c
WHERE c.city
IN ('Moncton', 'Toronto', 'Montreal')
```

Composite
Range
Spatial

```
SELECT * FROM c
WHERE c.city = 'Moncton'
AND c.age > 45
AND c.age < 70
```

[✓ Composite]
Range
Spatial

```
SELECT * FROM c
WHERE c.city = 'Moncton'
AND c.age > 45
```

Composite
Range
Spatial

Question 6

Question Type: MultipleChoice

You have a container named container1 in an Azure Cosmos DB for NoSQL account named account1 that is set to the session default consistency level. The average size of an item in container1 is 20 KB.

You have an application named App1 that uses the Azure Cosmos DB SDK and performs a point read on the same set of items in container1 every minute.

You need to minimize the consumption of the request units (RUs) associated to the reads by App1. What should you do?

Options:

A- In account1, change the default consistency level to bounded staleness.

B- In App1, change the consistency level of read requests to consistent prefix.

C- In account1, provision a dedicated gateway and integrated cache

D- In App1, modify the connection policy settings.

Answer:

C

Explanation:

The cost of a point read for a 1 KB item is 1 RU.The cost of other operations depends on factors such as item size, indexing policy, consistency level, and query complexity1. To minimize the consumption of RUs, you can optimize these factors according to your application needs.

For your scenario, one possible way to minimize the consumption of RUs associated to the reads by App1 is tochange the consistency level of read requests to consistent prefix. Consistent prefix is a lower consistency level than session, which is the default consistency level for Azure Cosmos DB.Lower consistency levels consume fewer RUs than higher consistency levels2.Consistent prefix guarantees that reads never see out-of-order writes and that monotonic reads are preserved1. This may be suitable for your application if you can tolerate some eventual consistency.

Question 7

Question Type: MultipleChoice

You have an Azure Cosmos DB for NoSQL account named account1 that supports an application named App1. App1 uses the consistent prefix consistency level.

You configure account1 to use a dedicated gateway and integrated cache.

You need to ensure that App1 can use the integrated cache.

Which two actions should you perform for APP1? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

Options:

A- Change the connection mode to direct

B- Change the account endpoint to https://account1.sqlx.cosmos.azure.com.

C- Change the consistency level of requests to strong.

D- Change the consistency level of requests to session.

E- Change the account endpoint to https://account1.documents.azure.com

Answer:

B, D

Explanation:

the Azure Cosmos DB integrated cache is an in-memory cache that is built-in to the Azure Cosmos DB dedicated gateway. The dedicated gateway is a front-end compute that stores cached data and routes requests to the backend database.You can choose from a variety of dedicated gateway sizes based on the number of cores and memory needed for your workload1.The integrated cache can reduce the RU consumption and latency of read operations by serving them from the cache instead of the backend containers2.

For your scenario, to ensure that App1 can use the integrated cache, you should perform these two actions:

Change the account endpoint to https://account1.sqlx.cosmos.azure.com. This is the dedicated gateway endpoint that you need to use to connect to your Azure Cosmos DB account and leverage the integrated cache.The standard gateway endpoint (https://account1.documents.azure.com) will not use the integrated cache2.

Change the consistency level of requests to session. This is the highest consistency level that is supported by the integrated cache.If you use a higher consistency level (such as strong or bounded staleness), your requests will bypass the integrated cache and go directly to the backend containers

Question 8

Question Type: **MultipleChoice**

You have a database named db1 in an Azure Cosmos DB f You have a third-party application that is exposed thro You need to migrate data from the application to a What should you use?

Options:

A- Database Migration Assistant

B- Azure Data Factory

C- Azure Migrate

Answer:

B

Explanation:

you can migrate data from various data sources to Azure Cosmos DB using different tools and methods.The choice of the migration tool depends on factors such as the data source, the Azure Cosmos DB API, the size of data, and the expected migration duration1. Some of the common migration tools are:

Azure Cosmos DB Data Migration tool: This is an open source tool that can import data to Azure Cosmos DB from sources such as JSON files, MongoDB, SQL Server, CSV files, and Azure Cosmos DB collections.This tool supports the SQL API and the Table API of Azure Cosmos DB2.

Azure Data Factory: This is a cloud-based data integration service that can copy data from various sources to Azure Cosmos DB using connectors.This tool supports the SQL API, MongoDB API, Cassandra API, Gremlin API, and Table API of Azure Cosmos DB3.

Azure Cosmos DB live data migrator: This is a command-line tool that can migrate data from one Azure Cosmos DB container to another container within the same or different account.This tool supports live migration with minimal downtime and works with any Azure Cosmos DB API4.

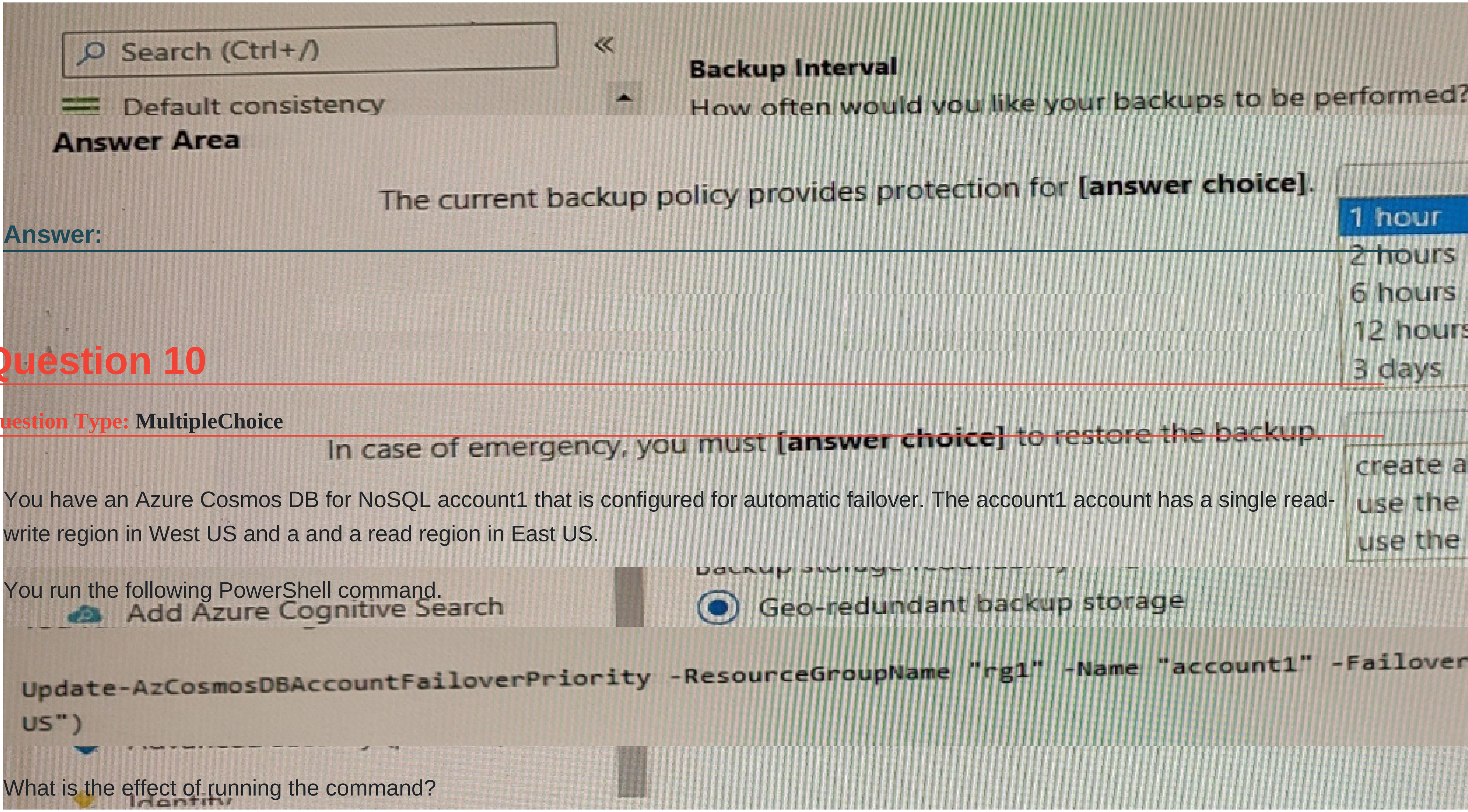
For your scenario, if you want to migrate data from a third-party application that is exposed through an OData endpoint to a container in Azure Cosmos DB for NoSQL, you should useAzure Data Factory.Azure Data Factory has an OData connector that can read data from an OData source and write it to an Azure Cosmos DB sink using the SQL API5. You can create a copy activity in Azure Data Factory that specifies the OData source and the Azure Cosmos DB sink, and run it on demand or on a schedule.

Question 9

Question Type: Hotspot

You configure a backup for an Azure Cosmos DB for NoSQL account as shown in the following exhibit.

Answer:



Backup Interval

How often would you like your backups to be performed?

Answer Area

The current backup policy provides protection for **[answer choice]**.

- 1 hour
- 2 hours
- 6 hours
- 12 hours
- 3 days

In case of emergency, you must **[answer choice]** to restore the backup.

- create a
- use the
- use the

● Geo-redundant backup storage

Question 10

Question Type: MultipleChoice

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" -Failover
US")
```

What is the effect of running the command?

Options:

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:

B

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 thatthe account will be configured for multi-region writes.

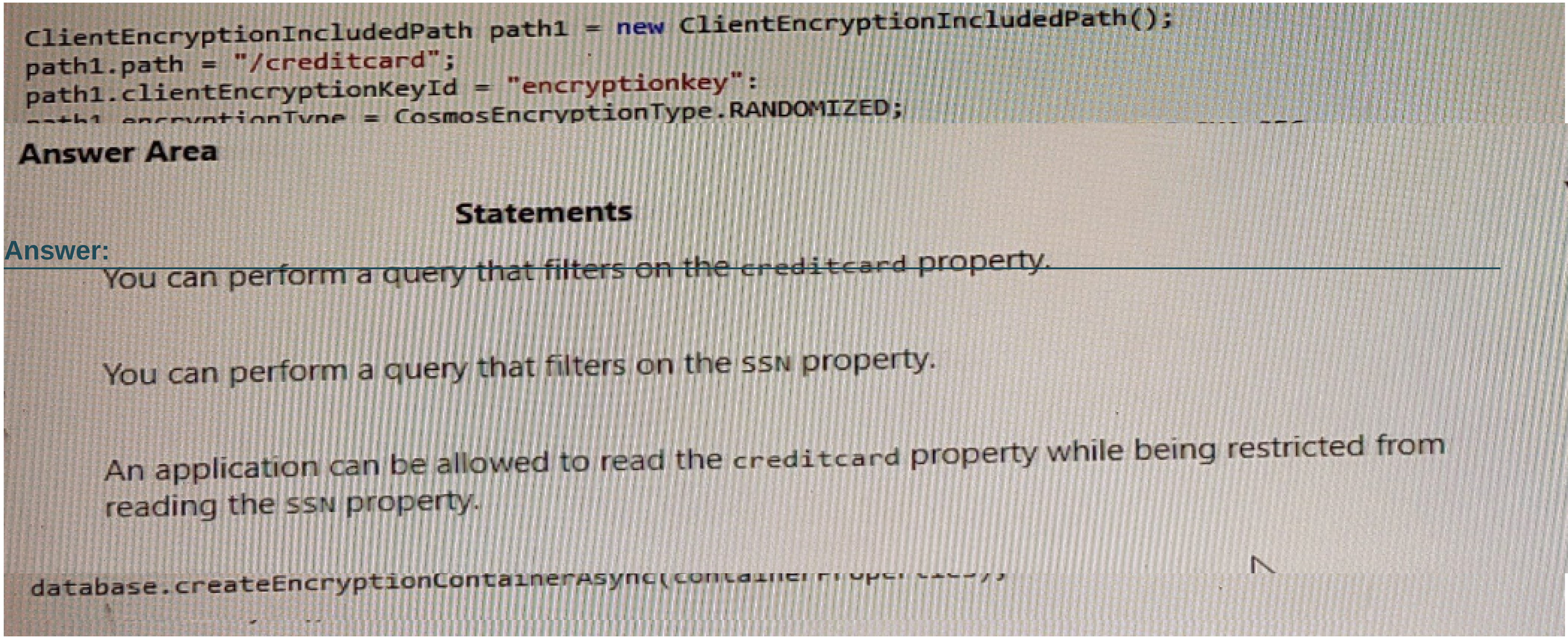
Multi-region writes is a feature of Azure Cosmos DB that allows you to write data to any region in your account and have it automatically replicated to all other regions. This feature provides high availability and low latency for write operations across multiple regions.To enable multi-region writes, 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 11

Question Type: Hotspot

You have a container named container1 in an Azure Cosmos DB for NoSQL account named account1.

You configure container1 to use Always Encrypted by using an encryption policy as shown in the C# and the Java exhibits. (Click the C# tab to view the encryption policy in C#. Click the Java tab to see the encryption policy in Java.)



```
ClientEncryptionIncludedPath path1 = new ClientEncryptionIncludedPath();
path1.path = "/creditcard";
path1.clientEncryptionKeyId = "encryptionkey";
path1.encryptionType = CosmosEncryptionType.RANDOMIZED;
```

Answer Area

Statements

Answer:

You can perform a query that filters on the `creditcard` property.

You can perform a query that filters on the `SSN` property.

An application can be allowed to read the `creditcard` property while being restricted from reading the `SSN` property.

```
database.createEncryptionContainerAsync(containerProperties);
```