Microsoft

Access and manage data with the Azure Cosmos DB for NoSQL SDKs

Agenda

- Implement Azure Cosmos DB for NoSQL point operations
- Perform cross-document transactional operations with the Azure Cosmos DB for NoSQL
- Process bulk data in Azure Cosmos DB for NoSQL

Implement Azure Cosmos DB for NoSQL point operations

Understand point operations

The *Microsoft.Azure.Cosmos* library includes first-class support for generics in the C# language.

Container class at the most foundational level

```
public class item
{
    public string id { get; set; }
    public string partitionKey { get; set; }
}
```

Example of a *Product* class for a Product container

```
// Assume categoryId is the partition key
public class Product
{
    public string id { get; set; }
    public string name { get; set; }
    public string categoryId { get; set; }
    public double price { get; set; }
    public string[] tags { get; set; }
}
```

Create documents

Let's add a new item of the previously defined Product class type.

```
Product saddle = new()
{
    id = "027D0B9A-F9D9-4C96-8213-C8546C4AAE71",
    categoryId = "26C74104-40BC-4541-8EF5-9892F7F03D72",
    name = "LL Road Seat/Saddle",
    price = 27.12d,
    tags = new string[] { "brown", "weathered" }
};

try
{
    await container.CreateItemAsync<Product>(saddle);
}
catch(CosmosException ex) when (ex.StatusCode == HttpStatusCode.Conflict)
{
    // Add logic to handle conflicting ids
}
catch(CosmosException ex)
{
    // Add general exception handling logic
}
```

Read a document

Read and existing document.

```
// We first need the Unique Id of the document we are searching for.
string id = "027D0B9A-F9D9-4C96-8213-C8546C4AAE71";

// We then need the partition key of the document we are searching for.
string categoryId = "26C74104-40BC-4541-8EF5-9892F7F03D72";
PartitionKey partitionKey = new (categoryId);

// With both the id and the partition key we can now search for the document.
Product saddle = await container.ReadItemAsync<Product>(id, partitionKey);

// If we find the document, we can now do something with its data like display it to the console.
string formattedName = $"New Product [${saddle}]";
Console.WriteLine(formattedName);
```

Update documents

Let's modify the *saddle* item data a couple of times.

```
// You can modify the saddle variable we defined earlier.
saddle.price = 35.00d;

// We can persist the change invoking the asynchronous UpsertItemAsync<> method passing in only the update item.
await container.UpsertItemAsync<Product>(saddle);

// We can modify other properties of the saddle variable.
saddle.tags = new string[] { "brown", "new", "crisp" };

// Even though we upserted the document already, we don't have to read a new item before upserting the item again.
await container.UpsertItemAsync<Product>(saddle);
```

Delete documents

Deleting documents is similar to reading documents.

```
// We first need the Unique Id of the document we want to delete.
string id = "027D0B9A-F9D9-4C96-8213-C8546C4AAE71";

// We then need the partition key of the document we want to delete.
string categoryId = "26C74104-40BC-4541-8EF5-9892F7F03D72";
PartitionKey partitionKey = new (categoryId);

// With the id and partition key, you invoke the asynchronous DeleteItemAsync<> method in a manner like
// the ReadItemAsync<> method.
await container.DeleteItemAsync<Product>(id, partitionKey);
```

Configure time-to-live (TTL) value for a specific document

To implement TTL, you will upsert an item.

```
// We first need to define a TimeToLive property on the Product class.
public class Product
{
    [JsonProperty(PropertyName = "ttl", NullValueHandling = NullValueHandling.Ignore)]
    public int? ttl { get; set; }
    // ...
}

// We then need to assign a value in seconds to the saddle ttl.
saddle.ttl = 1000;

// Update the item using the UpsertItemAsync<> method.
await container.UpsertItemAsync<Product>(saddle);
```

Lab – Create and update documents with the Azure Cosmos DB for NoSQL SDK

- Prepare your development environment
- Create an Azure Cosmos DB for NoSQL account
- Connect to the Azure Cosmos DB for NoSQL account from the SDK
- Perform create and read point operations on items with the SDK
- Perform update and delete point operations with the SDK

Perform cross-document transactional operations with the Azure Cosmos DB for NoSQL

Create a transactional batch with the SDK

```
// The transactional batch supports operations with the same logical partition key.
public record Product(string id, string name, string categoryId);

Product saddle = new("0120", "Worn Saddle", "accessories-used");
Product handlebar = new("012A", "Rusty Handlebar", "accessories-used");

PartitionKey partitionKey = new ("accessories-used");

TransactionalBatch batch = container.CreateTransactionalBatch(partitionKey)
    .CreateItem<Product>(saddle)
    .CreateItem<Product>(handlebar);

using TransactionalBatchResponse response = await batch.ExecuteAsync();
```

```
// Operations with different logical partition keys will fail batch operation.
public record Product(string id, string name, string categoryId);

Product saddle = new("0120", "Worn Saddle", "accessories-used");
Product handlebar = new("012A", "Rusty Handlebar", "accessories-new");

PartitionKey partitionKey = new ("accessories-used");

TransactionalBatch batch = container.CreateTransactionalBatch(partitionKey)
    .CreateItem<Product>(saddle)
    .CreateItem<Product>(handlebar);

using TransactionalBatchResponse response = await batch.ExecuteAsync();
```

Review batch operation results with the SDK

The *TransactionalBatchResponse* class contains multiple members to interrogate the results of the batch operation.

```
using TransactionalBatchResponse response = await batch.ExecuteAsync();

if (! response.IsSuccessStatusCode)
{
    // Do something.
}
```

Implement optimistic concurrency control

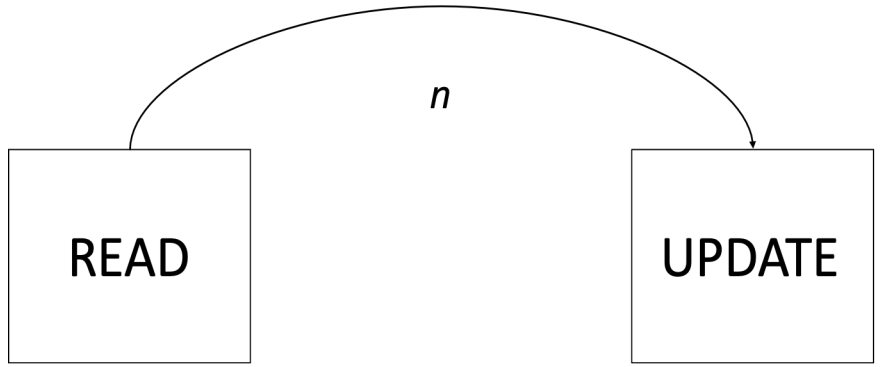
```
// Since read and write in this example are distinct operations,
// there is a latency between these operations, represented as n
// in the diagram shown to the right.
string categoryId = "9603ca6c-9e28-4a02-9194-51cdb7fea816";

PartitionKey partitionKey = new (categoryId);

Product product = await container.ReadItemAsync
   <Product>("01AC0", partitionKey);

product.price = 50d;

await container.UpsertItemAsync<Product>(product, partitionKey);
```




```
// The C# code only required minor changes to implement optimistic concurrency control.
string categoryId = "9603ca6c-9e28-4a02-9194-51cdb7fea816";

PartitionKey partitionKey = new (categoryId);
ItemResponse<Product> response = await container.ReadItemAsync<Product>("01AC0", partitionKey);

Product product = response.Resource;
string eTag = response.ETag;
product.price = 50d;

ItemRequestOptions options = new ItemRequestOptions { IfMatchEtag = eTag };

await container.UpsertItemAsync<Product>(product, partitionKey, requestOptions: options);
```

Lab – Batch multiple point operations together with the Azure Cosmos DB for NoSQL SDK

- Prepare your development environment
- Create an Azure Cosmos DB for NoSQL account and configure the SDK project
- Creating a transactional batch
- Creating an errant transactional batch

Process bulk data in Azure Cosmos DB for NoSQL

Create bulk operations with the SDK

```
// Enable bulk operations by setting the CosmosClientOption AllowBulkExecution option to true.
CosmosClientOptions options = new ()
{
    AllowBulkExecution = true
}

// Alternatively, you could send the Azure Cosmos DB endpoint/key pair instead of the connection string.
CosmosClient client = new (connectionString, options);

// Let's assume the GetOurProductsFromSomeWhere generates 25,000 products and add them to the bulk tasks in a loop.
List<Product> productsToInsert = GetOurProductsFromSomeWhere();

List<Task> concurrentTasks = new List<Task>();

foreach(Product product in productsToInsert)
{
    concurrentTasks.Add(
        container.CreateItemAsync<Product>(
            product,
            new PartitionKey(product.partitionKeyValue)
        )
    );
}

// The Task.WhenAll will create batches to group our operations by physical partition,
// then distribute the requests to run concurrently.
Task.WhenAll(concurrentTasks);
```

Bulk operation caveats and best practices

Review bulk operation caveats 주의사항

- The provisioned throughput (RU/s) is higher than if the operations were executed individually.
- While waiting to fill the batch, if it doesn't have enough items, it will wait 100 ms for more items.
- Batches are created for optimization with a maximum of 2 Mb (or 100 operations).

Implement bulk best practices

- While not requered for bulk operations, it's a good practice to provide the partition key.
- Use stream API in serialize-deserialize scenarios.
- Configure worker task per partition key.

Lab – Move multiple documents in bulk with the Azure Cosmos DB for NoSQL SDK

- Prepare your development environment
- Create an Azure Cosmos DB for NoSQL account and configure the SDK project
- Bulk inserting a twenty-five thousand documents
- Observe the results

Review

1 **Which method should you use to update an item in Azure Cosmos DB for NoSQL using the .NET SDK?**

- ❑ PatchItemAsync<>
- ☑ UpdateItemAsync<>
- ❑ UpsertItemAsync<>

2 **Which class contains the methods for Create, Read, Update, and Delete point operations on items in Azure Cosmos DB for NoSQL?**

- ☑ Container
- ❑ Database
- ❑ CosmosClient

3 **Which property of the TransactionalBatchResponse returns the HTTP status code indicating success or failure of the transaction?**

- ☑ StatusCode
- ❑ ErrorMessage
- ❑ IsSuccessStatusCode