Module 05: Execute queries in Azure Cosmos DB SQL API

In [ ]:
```
using Microsoft.Azure.Cosmos;
using System;
using System.Collections.Generic;

CosmosClient client = new (connectionString);
Database database = client.GetDatabase("cosmicworks"
Container container = database.GetContainer("product

public class Product
{
        public string id { get; set; }
        public string categoryId { get; set; }
        public string categoryName { get; set; }
        public string sku { get; set; }
        public string name { get; set; }
        public string description { get; set; }
        public double price { get; set; }
}
```

Understand SQL query language

Here is an example of a JSON object that would be in this container:

```
{
    "id": "86FD9250-4BD5-42D2-B941-
1C1865A6A65E",
    "categoryId": "F3FBB167-11D8-41E4-84B4-
5AAA92B1E737",
    "categoryName": "Components, Touring
Frames",
    "sku": "FR-T67U-58",
    "name": "LL Touring Frame - Blue, 58",
    "description": "The product called \"LL
Touring Frame - Blue, 58\"",
    "price": 333.42,
    "tags": [
        {
            "id": "764C1CC8-2E5F-4EF5-83F6-
8FF7441290B3",
            "name": "Tag-190"
        },
        {
            "id": "765EF7D7-331C-42C0-BF23-
A3022A723BF7",
            "name": "Tag-191"
        }
}
```

Create queries with SQL

In [ ]:
```
var sql = "SELECT TOP 10 * FROM products";
QueryDefinition query = new (sql);

var iterator = container.GetItemQueryIterator<Produc
```

```
while (iterator.HasMoreResults)
{
    var currentResultSet = await iterator.ReadNextAs
    foreach (Product product in currentResultSet)
    {
        Console.WriteLine($"[{product.id}]\t{product
    }
}
```

In [ ]:

```
// Here is another query that returns only a few fie
var sql = @"SELECT TOP 10 products.id,
                          products.name,
                          products.price,
                          products.categoryName
            FROM products";

QueryDefinition query = new (sql);

var iterator = container.GetItemQueryIterator<Produc

while (iterator.HasMoreResults)
{
    var currentResultSet = await iterator.ReadNextAs
    foreach (Product product in currentResultSet)
    {
        Console.WriteLine($"[{product.id}]\t{product
    }
}
```

In [ ]:

```
// It doesn't matter what name is used here for the
var sql = @"SELECT TOP 10 p.name,
                          p.price
            FROM p";

QueryDefinition query = new (sql);

var iterator = container.GetItemQueryIterator<dynami

while (iterator.HasMoreResults)
{
    var currentResultSet = await iterator.ReadNextAs
    foreach (var record in currentResultSet)
    {
        Console.WriteLine($"[{record.name,40}]\t{rec
    }
}
```

In [ ]:

```
// We can also filter our queries using the WHERE ke
// 50 and 100:
var sql = @"SELECT TOP 10   p.name,
                            p.categoryName,
                            p.price
            FROM products p
            WHERE p.price >= 50
                AND p.price <= 100";

QueryDefinition query = new (sql);

var iterator = container.GetItemQueryIterator<dynami

while (iterator.HasMoreResults)
{
    var currentResultSet = await iterator.ReadNextAs
    foreach (var record in currentResultSet)
    {
        Console.WriteLine($"[{record.name,40}]\t{rec
    }
}
```

```
}
```

Project query results

Azure Cosmos DB SQL API extends SQL to manipulate JSON results.

In [ ]:

```
// use an alias
var sql = @"SELECT TOP 10 p.categoryName AS category
QueryDefinition query = new (sql);

var iterator = container.GetItemQueryIterator<dynami

while (iterator.HasMoreResults)
{
    var currentResultSet = await iterator.ReadNextAs
    foreach (var record in currentResultSet)
    {
        Console.WriteLine($"- {record.category}");
    }
}
```

In [ ]:

```
// use DISTINCT
var sql = "SELECT DISTINCT TOP 10 p.categoryName FRO
QueryDefinition query = new (sql);

var iterator = container.GetItemQueryIterator<dynami

while (iterator.HasMoreResults)
{
    var currentResultSet = await iterator.ReadNextAs
    foreach (var record in currentResultSet)
    {
        Console.WriteLine($"- {record.categoryName}"
    }
}
```

Implement type-checking in queries

NoSQL is schema-less, the responsibility for type checking will often fall on your queries.

```
// Let's assume this is a document on the
Product container.
{
    "id": "6374995F-9A78-43CD-AE0D-
5F6041078140",
    "categoryid": "3E4CEACD-D007-46EB-82D7-
31F6141752B2",
    "sku": "FR-R38R-60",
    "name": "LL Road Frame - Red, 60",
    "price": 337.22
}
```

In [ ]:

```
// Note how in the previous document there are no ta
var sql = @"SELECT TOP 10 p.id,
                          IS_DEFINED(p.tags) AS tags
                          IS_ARRAY(p.tags) as tags_a
                          IS_NUMBER(p.price) as pric
                          IS_STRING(p.price) as pric
            FROM products p ";
```

```
QueryDefinition query = new (sql);

var iterator = container.GetItemQueryIterator<dynami

while (iterator.HasMoreResults)
{
    var currentResultSet = await iterator.ReadNextAs
    foreach (var p in currentResultSet)
    {
        Console.WriteLine($"[{p.id,40}]\t{p.tags_exi
    }
}
```

Use built-in functions

SQL for the Azure Cosmos DB SQL API ships with built-in functions for common tasks in a query.

Here are some examples of these functions:

In [ ]:
```
// CONCAT
var sql = "SELECT TOP 10 VALUE CONCAT(p.name, ' | ',

QueryDefinition query = new (sql);

var iterator = container.GetItemQueryIterator<string

while (iterator.HasMoreResults)
{
    var currentResultSet = await iterator.ReadNextAs
    foreach (var record in currentResultSet)
    {
        Console.WriteLine($"- {record}");
    }
}
```

In [ ]:
```
// LOWER
var sql = "SELECT TOP 10 VALUE LOWER(p.sku) FROM pro

QueryDefinition query = new (sql);

var iterator = container.GetItemQueryIterator<string

while (iterator.HasMoreResults)
{
    var currentResultSet = await iterator.ReadNextAs
    foreach (var record in currentResultSet)
    {
        Console.WriteLine($"- {record}");
    }
}
```

In [ ]:
```
// RTRIM, LTRIM, LEFT, RIGHT

var sql = @"SELECT TOP 10
                        LTRIM(RTRIM(p.sku)) as sku,
                        LEFT(p.name, 10) name
            FROM products p";

QueryDefinition query = new (sql);

var iterator = container.GetItemQueryIterator<dynami

while (iterator.HasMoreResults)
{
    var currentResultSet = await iterator.ReadNextAs
```

```
    foreach (var p in currentResultSet)
    {
        Console.WriteLine($"[{p.name,40}]\t{p.sku}")
    }
}
```

In [ ]:

```
// CONTAINS

var sql = @"SELECT TOP 10 p.name
            FROM products p
            WHERE CONTAINS(p.name, ""classic"", true

QueryDefinition query = new (sql);

var iterator = container.GetItemQueryIterator<dynami

while (iterator.HasMoreResults)
{
    var currentResultSet = await iterator.ReadNextAs
    foreach (var p in currentResultSet)
    {
        Console.WriteLine($"[{p.name,40}]");
    }
}
```

ARRAY_CONTAINS

Following example returns all documents (top 10) that have a tagname value Tag-81 in it It makes use of ARRAY() to create a new array

```
"tags": [
    {
        "id": "3A3A99B6-E3BF-46D0-BAD9-
F5F4DBB720F4",
        "name": "Tag-70"
    },
    {
        "id": "51CD93BF-098C-4C25-9829-
4AD42046D038",
        "name": "Tag-81" // looking for this
tag!
    },
    {
        "id": "6C2F05C8-1E61-4912-BE1A-
C67A378429BB",
        "name": "Tag-5"
    },
    {
        "id": "B48D6572-67EB-4630-A1DB-
AFD4AD7041C9",
        "name": "Tag-100"
    }
]
```

is translated into an array:

```
[
  "Tag-70",
  "Tag-81",
  "Tag-5",
  "Tag-100"
]
```

In [ ]:

```
// ARRAY_CONTAINS

var sql = @"SELECT TOP 10
                p.id,
                ARRAY(SELECT DISTINCT VALUE t.name F
            FROM products p
            WHERE ARRAY_CONTAINS(ARRAY(SELECT DISTIN

QueryDefinition query = new (sql);

var iterator = container.GetItemQueryIterator<dynami

while (iterator.HasMoreResults)
{
    var currentResultSet = await iterator.ReadNextAs
    foreach (var p in currentResultSet)
    {
        Console.WriteLine($"[{p.id,40}]\t{p.tagNames
    }
}
```

Author complex queries with the Azure Cosmos DB SQL API

Create cross-product queries

Unlike a JOIN in a relational database, a JOIN in Azure Cosmos DB SQL API scope is a single item only. A JOIN creates a cross-product between different sections of a single item.

```
// Suppose this is a document in our product
container.
{
    "id": "80D3630F-B661-4FD6-A296-
CD03BB7A4A0C",
    "categoryId": "629A8F3C-CFB0-4347-8DCC-
505A4789876B",
    "categoryName": "Clothing, Vests",
    "sku": "VE-C304-L",
    "name": "Classic Vest, L",
    "description": "A worn brown classic",
    "price": 32.4,
    "tags": [
              {
                "id": "2CE9DADE-DCAC-436C-
9D69-B7C886A01B77",
                "name": "apparel", "class":
"group"
              },
              {
                "id": "CA170AAD-A5F6-42FF-
B115-146FADD87298",
                "name": "worn", "class":
"trade-in"
              },
              {
                "id": "CA170AAD-A5F6-42FF-
B115-146FADD87298",
                "name": "no-damaged",
"class": "trade-in"
              }
            ]
```

```
}
```

In [ ]:

```
// Return the tag name for all tags embedded in the

var sql = @"SELECT TOP 20 p.id, p.name, t.name AS ta
            FROM products p
            JOIN t IN p.tags";

QueryDefinition query = new (sql);

var iterator = container.GetItemQueryIterator<dynami

while (iterator.HasMoreResults)
{
    var currentResultSet = await iterator.ReadNextAs
    foreach (var p in currentResultSet)
    {
        Console.WriteLine($"[{p.id}]\t{p.name,40}\t{
    }
}
```

Implement correlated subqueries

We can optimize JOIN expressions further by writing subqueries to filter the number of array items we want to include in the cross-product set.


```
{
    "id": "4DA12D36-495E-4DCA-95B0-
F18CAA099779",
    "categoryId": "56400CF3-446D-4C3F-B9B2-
68286DA3BB99",
    "categoryName": "Bikes, Mountain Bikes",
    "sku": "BK-M82S-42",
    "name": "Mountain-100 Silver, 42",
    "description": "The product called
\"Mountain-100 Silver, 42\"",
    "price": 3399.99,
    "tags": [
        {
            "id": "3C26DF5C-CE21-4EF6-AEE2-
E8E1066D06B1",
            "name": "Tag-81" // we want to
return this item (no other tags needed)
        },
        {
            "id": "BB35DF88-8BCE-4267-838B-
```