Module 06: Define and implement an indexing strategy for 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("products");

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; }
}
```

Define indexes in Azure Cosmos DB SQL API

Understand indexes

- Every Azure Cosmos DB SQL API container has a built-in index policy.
- By default, the index includes all properties of every item in the container.
- By default, all create, update, or delete operations update the index.

Example of the default policy in action:

Item 1 in the product container

```
{
   "name": "Touring-1000 Blue",
   "tags": [
              { "name": "bike" },
              { "name": "touring" },
              { "name": "blue" }
           ]
}
```

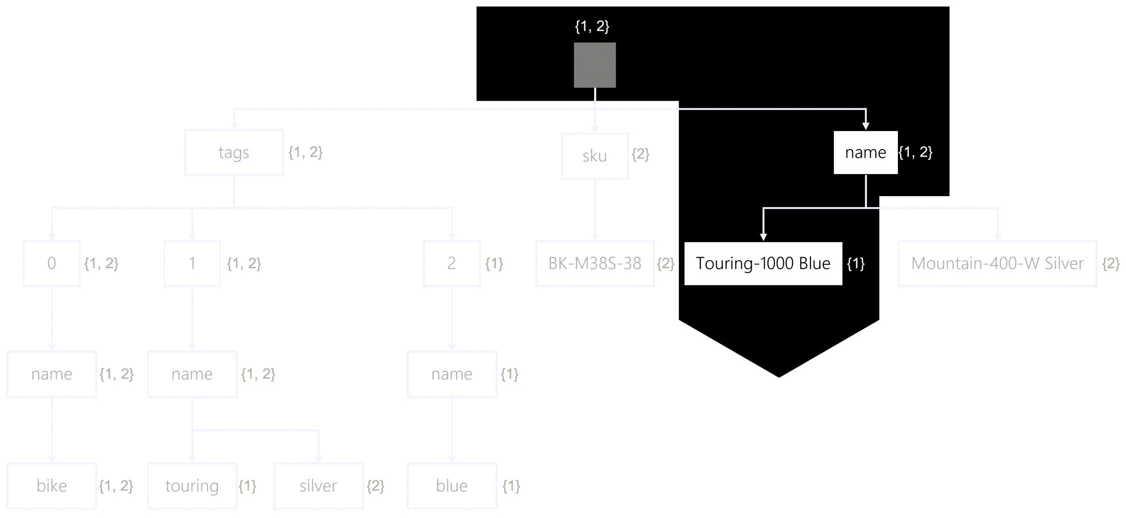
Item 2 in the product container

```
{
    "name": "Mountain-400-W Silver",
    "sku": "BK-M38S-38",
    "tags": [
               { "name": "bike" },
               { "name": "silver" }
            ]
}
```

How is the index used when we run this query?

```
SELECT p.id
FROM products p
WHERE p.name = 'Touring-1000 Blue'
```

Index created for these product container items



Understand indexing policies

The default indexing policy consists of the following settings:

- The inverted index is updated for all create, update, or delete operations on an item
- All properties for every item is automatically indexed
- Range indexes are used for all strings or numbers

Indexing policies are defined and managed in JSON. This is the default:

```
{
  "indexingMode": "consistent",
  "automatic": true,
  "includedPaths": [
    {
      "path": "/*"
    }
  ],
  "excludedPaths": [
    {
      "path": "/\"_etag\"/?"
    }
  ]
}
```

Indexing modes and Include/Exclude paths

Index policies can be updated to better meet your container's usage patterns.

Configure indexing mode:

- **Consistent**: Updates index synchronously with all item modifications. Default mode.
- **None**: Disables indexing on a container. Useful for bulk operations.

Including and excluding paths:

Three primary operators are used when defining a property path:

- The ? operator indicates that a path terminates with a string or number (scalar) value
- The [] operator indicates that this path includes an array and avoids having to specify an array index value
- The * operator is a wildcard and matches any element beyond the current path

Consider this JSON object that represents a product item in our Azure Cosmos DB SQL API container:

```
{
  "id": "8B363B8B-378E-402A-9E68-A935302000B8",
  "name": "HL Touring Frame - Yellow, 46",
  "category": {
    "id": "F3FBB167-11D8-41E4-84B4-5AAA92B1E737",
    "name": "Components, Touring Frames"
  },
  "metadata": {
    "sku": "FR-T98Y-46"
  },
  "price": 1003.91,
  "tags": [
    {
      "name": "accessory"
    },
    {
      "name": "yellow"
    },
    {
      "name": "frame"
    }
  ]
}
```

Path examples:

Path expression	Description
/*	All properties
/name/?	The scalar value of the **name** property
/category/*	All properties under the **category** property
/metadata/sku/?	The scalar value of the **metadata.sku** property
/tags/[]/name/?	Within the **tags** array, the scalar values of all possible **name** properties

Review indexing policy strategies

An indexing policy:

- Is two sets of include/exclude expressions that evaluates which actual properties are indexed.
- Must include the root path and all possible values (/*) as either an included or excluded path.

Example indexing policy that includes all properties except category.id:

```
{
    "indexingMode": "consistent",
    "automatic": true,
    "includedPaths": [
                        {
                            "path": "/*"
                        }
                    ],
    "excludedPaths": [
                        {
                            "path": "/category/id/?“
                        }
                        ]
}
```

Example indexing policy excluding all properties and selectively indexes only the name and tags[].name properties:

```
{
    "indexingMode": "consistent",
    "automatic": true,
    "includedPaths": [
                        {
                            "path": "/name/?"
                        },
                        {
                            "path": "/tags/[]/name/?"
                        }
                    ],
    "excludedPaths": [
                        {
                            "path": "/*"
                        }
                    ]
}
```

In [ ]:

```
using Newtonsoft.Json;

var query = new QueryDefinition("SELECT TOP 10 * FROM products");
var requestOptions = new QueryRequestOptions() { PopulateIndexMetrics = false };
var iterator = container.GetItemQueryIterator<dynamic>(query, requestOptions: reques
var resultSet = await iterator.ReadNextAsync();
var diagnosticsJsonString = resultSet.Diagnostics.ToString();
dynamic jsonResponse = JsonConvert.DeserializeObject(diagnosticsJsonString);


Console.WriteLine(jsonResponse.children[1].children[1].children[0].children[0].data
Console.WriteLine($"RequestCharge: {resultSet.RequestCharge}");
Console.WriteLine($"IndexMetrics: {resultSet.IndexMetrics}");

//Console.WriteLine($"Diagnostics: {resultSet.Diagnostics}");
```

Customize indexes in Azure Cosmos DB SQL API

Customize the indexing policy

The .NET SDK ships with a Microsoft.Azure.Cosmos.IndexingPolicy class that is a representation of a JSON policy object.

Assume we would like to use the following index policy when we create a container

```
{
    "indexingMode": "consistent",
    "automatic": true,
    "includedPaths": [
                        {
                            "path": "/name/?"
                        },
                        {
                            "path": "/categoryName/?"
                        }
                    ],
    "excludedPaths": [
                        {
                             "path": "/*"
```

```
                                    }
                                ]
    }
```

Let’s use the SDK to define the policy, and create the container with that index policy

In [ ]:

```
// first cleanup the existing container
//await database.DeleteAsync();
Database database = await client.CreateDatabaseIfNotExistsAsync("cosmicworks");

IndexingPolicy policy = new ()
{
    IndexingMode = IndexingMode.Consistent,
    Automatic = true
};

policy.IncludedPaths.Add( new IncludedPath{ Path = "/name/?" } );
policy.IncludedPaths.Add( new IncludedPath{ Path = "/categoryName/?" } );

policy.ExcludedPaths.Add( new ExcludedPath{ Path = "/*" } );

ContainerProperties options = new () {
    Id = "products",
    PartitionKeyPath = "/categoryId",
    IndexingPolicy = policy };

Container container = await database.CreateContainerIfNotExistsAsync(options, throu

// check the azure portal for index
```

Evaluate composite indexes