


Define and implement an indexing strategy for Azure Cosmos DB for NoSQL



Agenda



- Define indexes in Azure Cosmos DB for NoSQL
- Customize indexes in Azure Cosmos DB for NoSQL

Define indexes in Azure Cosmos DB for NoSQL



Understand indexes



Every Azure Cosmos DB for NoSQL 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.

learn

<https://learn.microsoft.com/ko-kr/training/paths/define-implement-indexing-strategy-cosmos-db-sql-api/>

Example of the default policy in action

Item 1 in the product container

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

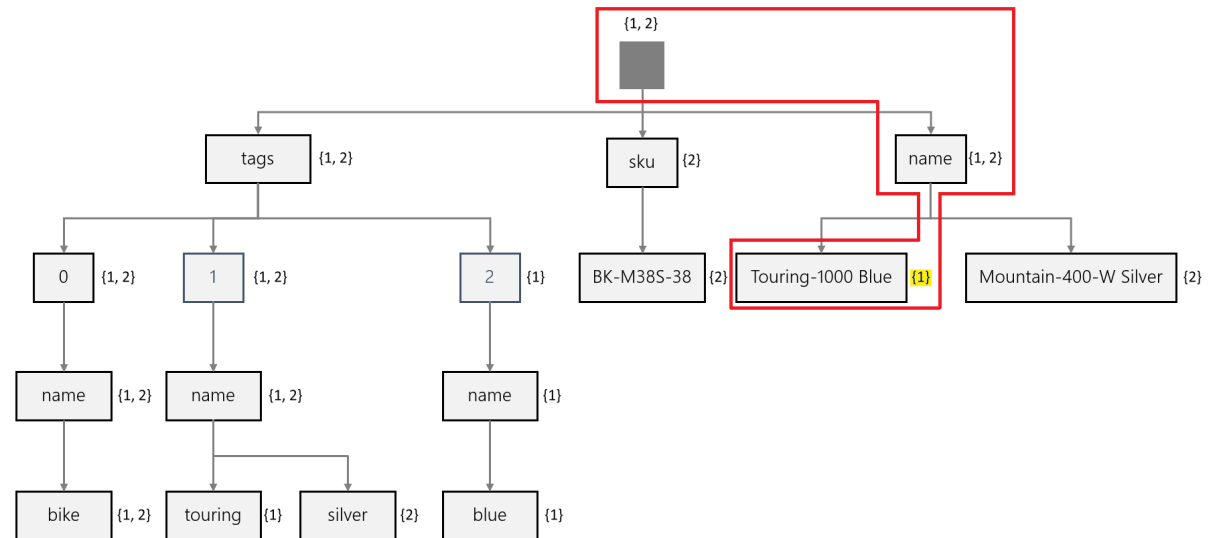
Item 2 in the product container

```
{  
  "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.

The default indexing policy, in JSON

```
{
  "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

Primary operators used to define a property path:

- **?** - Path terminates with a string or number (scalar) value.
- **[]** - Path includes an array, avoids specifying an array index value.
- ***** - Is a wildcard and matches any element beyond the current path.

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

Indexing policy strategies, what property to include/exclude

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.

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

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

Lab – Review the default index policy for an Azure Cosmos DB for NoSQL container with the portal



- Create an Azure Cosmos DB for NoSQL account
- Seed the Azure Cosmos DB for NoSQL account with data
- View and manipulate the default indexing policy

Customize indexes in Azure Cosmos DB for NoSQL



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

```
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, throughput: 400);
```

Evaluate composite indexes

For queries that sort or filter on multiple properties, create one or more composite indexes.

Let's assume we will run the following queries

```
-- This query has a filter on two properties.  
-- Note that the direction (ASC/DESC) and order of the  
-- composite index properties is not used by the filter.
```

```
SELECT *  
FROM products p  
WHERE p.price > 50 AND p.category = "Saddles"
```

```
-- This query will sort the results on two properties.  
-- Note that this composite index must match the order  
-- of the properties.
```

```
SELECT *  
FROM products p  
ORDER BY p.price ASC, p.name ASC
```

We can define a composite index for each query type →

```
{  
  "indexingMode": "consistent",  
  "automatic": true,  
  "includedPaths": [  
    {  
      "path": "/*"  
    }  
  ],  
  "excludedPaths": [  
    {  
      "path": "/_etag/?"  
    }  
  ],  
  "compositeIndexes": [  
    [  
      {  
        "path": "/category",  
        "order": "ascending"  
      },  
      {  
        "path": "/price",  
        "order": "descending"  
      }  
    ],  
    [  
      {  
        "path": "/price",  
        "order": "ascending"  
      },  
      {  
        "path": "/name",  
        "order": "ascending"  
      }  
    ]  
  ]  
}
```

Lab – Configure an Azure Cosmos DB for NoSQL container's index policy with the portal



- Prepare your development environment
- Create an Azure Cosmos DB for NoSQL account
- Create a new indexing policy using the .NET SDK
- Observe an indexing policy created by the .NET SDK using the Data Explorer

Review



1 You have created a bulk operation in Azure Cosmos DB for NoSQL that will insert hundreds of thousands of items into a container. You need to temporarily disable indexing and re-enable it after the bulk operation is complete. Prior to starting the bulk operation, which indexing mode should you configure in the indexing policy?

- ☒ None
- ☐ Consistent
- ☐ Lazy

2 You are authoring a new indexing policy for a container in Azure Cosmos DB for NoSQL. You would like to define an included path that includes all possible properties from the root of any JSON document. Which path expression should you use?

- ☒ /*
- ☐ /[]
- ☐ /?.

3 The most used query in your Azure Cosmos DB for NoSQL application is *SELECT * FROM products p ORDER BY p.name ASC, p.price ASC*. You would like to define a composite index to make the query more efficient and consume fewer RU/s. Which composite index should you use to support this query?

- ☐ (price ASC, name ASC)
- ☐ (price DESC, name ASC)
- ☒ (name ASC, price ASC)

