

**Explore Indexing Policies**

Template Version: 2.0

**Introduction**

During this lab, you will learn how to change indexing policies using the Cosmos DB Async Java SDK.

**Estimated Time**

30 minutes.

**Objectives**

At the end of this lab, you will be able to:

         Set indexing policies.

* Document Level
* Collection Level

         Understand indexing policies affect querying.

Lab: Explore Indexing Policies

During this lab, you will learn how to change indexing policies using the Cosmos DB Java Async SDK.

Exercise 1: Start the Index Management Solution

This exercise shows you how to view different index policies using the provided java solution.

1. From your workshop git repository, open the **index-management** solution in your IDE or text editor and edit the **com.microsoft.azure.cosmosdb.sample.AccountSettings.java** file.
2. Change the **ACCOUNT\_HOST** propertyvalue to the **endpoint URI** your Cosmos DB Account.
3. Change the **ACCOUNT\_KEY** propertyvalue to the **Primary Key** your Cosmos DB Account.
4. Open the **com.microsoft.azure.cosmosdb.sample.IndexManager.java** file within the **index-management** project.

Notice there are four method calls, each corresponding to an example read query:

* ExplicitlyExcludeFromIndex
* UseManualIndexing
* UseLazyIndexing
* ExcludePathsFromIndex

ExplicitlyExcludeFromIndex

The default index policy on a DocumentCollection will AUTOMATICALLY index ALL documents added. There may be scenarios where you want to exclude a specific doc from the index even though all other documents are being indexed automatically. This task demonstrates how to use an index directive to control this:

1. This method will first create a collection in a IndexManagement database and display the indexing policy (Consistent).
2. It will also create a document and submit a query to find it based on the **contact** property**.**
3. To view the results, package and run the program by invoking the following Maven commands from the project folder in your CLI:

mvn clean package

mvn exec:java

We will be able to find this document because the indexing policy is set to index all new documents and properties.

1. You will now add code to create a second document this time setting an **IndexingDirective** to **Exclude** the document and attempting to query against it.
2. In the **ExplicityExcludeFromIndex** method, add the following code underneath the creation logic for the first document. You should see a comment which says “Add your code here for exercise 1.4.1”.
3. First specify the IndexingDirective by adding a RequestOptions object. Add this code:

RequestOptions documentRequestOptions = new RequestOptions();

documentRequestOptions.setIndexingDirective(IndexingDirective.Exclude);

1. Next create a second document and pass it and necessary parameters to the helper method:

String doc2JsonString = "{'id':'doc2','order':'002','contact':'email'}";

CreateDocument(collectionUri, doc2JsonString, documentRequestOptions);

1. Rebuild the package and run the program by invoking the following Maven commands from the project folder in your CLI:

mvn clean package

mvn exec:java

1. Notice while two documents are created, only 1 result is returned by the query.

We will not be able to find the second document because it was not added to the index.

1. Open the collection documents list in the **Data Explorer** in the [Azure Portal](https://portal.azure.com) and verify the same. Navigate to Azure Cosmos DB Account -> Data Explorer -> IndexManagement database ->ExplicityExcludeFromIndexcollection -> Documents.
2. Note: While it is not part of the collection’s index, this document is still accessible and can be queried by its selfLink.

**UseManualIndexing**

The default index policy on a DocumentCollection will AUTOMATICALLY index ALL documents added. There may be cases where you can want to turn-off automatic indexing and only selectively add specific documents to the index. This task demonstrates how to control this by setting the value of IndexingPolicy.Automatic to false. (Reverse of Task 1)

1. This method will first create a collection in a IndexManagement database.
2. It will also create a document and submit a query to find it based on the **contact** property**.**
3. In the **com.microsoft.azure.cosmosdb.sample.Main.java** file, invoke the **UseManualIndexing** method after the existing (ExplicitlyExcludeFromIndex) method call:

indexManager.ExplicitlyExcludeFromIndex();

indexManager.UseManualIndexing(); //Add this

1. To view the results, package and run the program by invoking the following Maven commands from the project folder in your CLI:

mvn clean package

mvn exec:java

We will NOT be able to find this document because automatic indexing is turned off.

1. You will now add code to create a second document this time setting an **IndexingDirective** to **Include** and attempting to query against it.
2. In the **com.microsoft.azure.cosmosdb.sample.IndexManager.java** file and **UseManualIndexing** method, add the following code underneath the creation logic for the first document. You should see a comment which says “Add your code here for exercise 1.4.2”.
3. First specify the IndexingDirective by adding a RequestOptions object. Add this code:

RequestOptions documentRequestOptions = new RequestOptions();

documentRequestOptions.setIndexingDirective(IndexingDirective.Include);

1. Next create a second document and pass it and necessary parameters to the helper method:

String doc2JsonString = "{'id':'doc4','order':'004','contact':'phone'}";

CreateDocument(collectionUri, doc2JsonString, documentRequestOptions);

1. Rebuild the package and run the program by invoking the following Maven commands from the project folder in your CLI:

mvn clean package

mvn exec:java

1. Notice while two documents are created, only 1 result is returned by the query. Also notice the RU cost is less than the previous operation due to the manual index policy.

We can now locate the second document in the exercise, because we have manually indexed the document.

UseLazyIndexing

CosmosDB offers synchronous (consistent) and asynchronous (lazy) index updates. By default, the index is updated synchronously on each insert, replace or delete of a document to the collection. There are times when you might want to configure certain collections to update their index asynchronously. Lazy indexing boosts write performance and is ideal for bulk ingestion scenarios for primarily read-heavy collections. It is important to note that you might get inconsistent reads whilst the writes are in progress, However once the write volume tapers off and the index catches up, then reads continue as normal. This task demonstrates how to switch IndexMode to Lazy.

1. This method will first create a collection in a IndexManagement database.
2. It will also create a document and submit a query to find it based on the **contact** property**.**
3. In the com.microsoft.azure.cosmosdb.sample.Main.java file, invoke the **UseLazyIndexing** method after the existing method calls:

indexManager.ExplicitlyExcludeFromIndex();

indexManager.UseManualIndexing();

**indexManager.UseLazyIndexing();** //Add this line

1. To view the results, package and run the program by invoking the following Maven commands from the project folder in your CLI:

mvn clean package

mvn exec:java

We will find this document because consistent indexing is turned on. IndexMode should be **Consistent**.

1. You will now add code to create a change the **IndexingPolicy.IndexMode** to **Lazy**.
2. In the **com.microsoft.azure.cosmosdb.sample.IndexManager.java** file and **UseLazyIndexing** method, add the following code underneath the creation logic for the first document. You should see a comment which says “Add your code here for exercise 1.4.3”.
3. Add the following code to set the IndexingPolicy:

IndexingPolicy indexingPolicy = new IndexingPolicy();

indexingPolicy.setIndexingMode(IndexingMode.Lazy); //Add this line

String collectionUri = CreateCollection("UseLazyIndexing", indexingPolicy);

Lazy Indexing is inherently difficult to demonstrate because it is only noticed during sustained access loads.

1. (Optional) To view the results, package and run the program by invoking the following Maven commands from the project folder in your CLI:

mvn clean package

mvn exec:java

We will find this document because there is no ingest load on the collection. IndexMode should be **Lazy**.

ExcludePathsFromIndex

The default behavior is for CosmosDB to index every attribute in every document automatically. There are times when a document contains large amounts of information, in deeply nested structures that you know you will never search on. In extreme cases like this, you can exclude paths from the index to save on storage cost, improve write performance and also improve read performance because the index is smaller. This task demonstrates how to set IndexingPolicy.ExcludedPaths

1. This method will first create a collection in a IndexManagement database.
2. It will also create a document and submit a query to find it based on the **contact** property**.**
3. In the com.microsoft.azure.cosmosdb.sample.Main.java file, invoke the **ExcludePathsFromIndex** method after the existing method calls:

indexManager.ExplicitlyExcludeFromIndex();

indexManager.UseManualIndexing();

indexManager.UseLazyIndexing();

**indexManager.ExcludePathsFromIndex();** //Add this line

1. To view the results, package and run the program by invoking the following Maven commands from the project folder in your CLI:

mvn clean package

mvn exec:java

1. You will now add code to create an excluded Path on the IndexPolicy.
2. In the **com.microsoft.azure.cosmosdb.sample.IndexManager.java** file and **ExcludePathsFromIndex** method, add the following code underneath the creation logic for the first document. You should see a comment which says “Add your code here for exercise 1.4.4”.
3. Add the following code to set the IndexingPolicy:

List<IncludedPath> includedPaths = new ArrayList<>();

IncludedPath includedPath = new IncludedPath();

includedPath.setPath("/\*");

includedPaths.add(includedPath);

List<ExcludedPath> excludedPaths = new ArrayList<>();

ExcludedPath excludedPath = new ExcludedPath();

excludedPath.setPath("/nonsearchable/\*");

excludedPaths.add(excludedPath);

indexingPolicy.setIncludedPaths(includedPaths);

indexingPolicy.setExcludedPaths(excludedPaths);

1. To view the results, package and run the program by invoking the following Maven commands from the project folder in your CLI:

mvn clean package

mvn exec:java

1. Verify your Indexing Policy in the Scale and Settings option under the ExcludePathFromIndex collection inside the IndexManagement database.

We will not be able to search on any of /nonsearchable/ properties or their sub-properties because of the “/\*” wildcard

Results

Index policies default to automatic and will index all properties synchronously as they are written. As you have seen, it is possible to explicitly exclude or include paths from the index. We have done this on a per-document basis using indexing directives or by writing indexing policies on the collection level.

Building and storing indices requires computational and storage overhead. Depending on use cases, it may be economical and viable to exclude paths that will not be filtered against. In other use cases, e.g. a simple key-value store, it may be appropriate to completely disable indexing. Indexing policies must be considered against access patterns.