# Sorting DocumentDB data using Order By

Microsoft Azure DocumentDB supports querying documents using SQL over JSON documents. Query results can be ordered using the ORDER BY clause in SQL query statements.

After reading this article, you’ll be able to answer the following questions:

* How do I query with Order By?
* How do I configure an indexing policy for Order By?
* What’s coming next?

[Samples](#samples) and an [FAQ](#faq) are also provided.

For a complete reference on SQL querying, see the [DocumentDB Query tutorial](/documentation/articles/documentdb-sql-query).

## How to Query with Order By

Like in ANSI-SQL, you can now include an optional Order By clause in SQL statements when querying DocumentDB. The clause can include an optional ASC/DESC argument to specify the order in which results must be retrieved.

### Ordering using SQL

For example here’s a query to retrieve the top 10 books in descending order of their titles.

SELECT TOP 10 \*   
FROM Books   
ORDER BY Books.Title DESC

### Ordering using SQL with Filtering

You can order using any nested property within documents like Books.ShippingDetails.Weight, and you can specify additional filters in the WHERE clause in combination with Order By like in this example:

SELECT \*   
FROM Books   
WHERE Books.SalePrice > 4000  
ORDER BY Books.ShippingDetails.Weight

### Ordering using the LINQ Provider for .NET

Using the .NET SDK version 1.2.0 and higher, you can also use the OrderBy() or OrderByDescending() clause within LINQ queries like in this example:

foreach (Book book in client.CreateDocumentQuery<Book>(UriFactory.CreateDocumentCollectionUri("db", "books"))  
 .OrderBy(b => b.PublishTimestamp)  
 .Take(100))  
{  
 // Iterate through books  
}

DocumentDB supports ordering with a single numeric, string or Boolean property per query, with additional query types coming soon. Please see [What’s coming next](#Whats_coming_next) for more details.

## Configure an indexing policy for Order By

Recall that DocumentDB supports two kinds of indexes (Hash and Range), which can be set for specific paths/properties, data types (strings/numbers) and at different precision values (either maximum precision or a fixed precision value). Since DocumentDB uses Hash indexing as default, you must create a new collection with a custom indexing policy with Range on numbers, strings or both, in order to use Order By.

[AZURE.NOTE] String range indexes were introduced on July 7, 2015 with REST API version 2015-06-03. In order to create policies for Order By against strings, you must use SDK version 1.2.0 of the .NET SDK, or version 1.1.0 of the Python, Node.js or Java SDK.

Prior to REST API version 2015-06-03, the default collection indexing policy was Hash for both strings and numbers. This has been changed to Hash for strings, and Range for numbers.

For more details see [DocumentDB indexing policies](/documentation/articles/documentdb-indexing-policies).

### Indexing for Order By against all properties

Here’s how you can create a collection with “All Range” indexing for Order By against any/all numeric or string properties that appear within JSON documents within it. Here we override the default index type for string values to Range, and at the maximum precision (-1).

DocumentCollection books = new DocumentCollection();  
books.Id = "books";  
books.IndexingPolicy = new IndexingPolicy(new RangeIndex(DataType.String) { Precision = -1 });  
  
await client.CreateDocumentCollectionAsync(UriFactory.CreateDatabaseUri("db"), books);

[AZURE.NOTE] Note that Order By only will return results of the data types (String and Number) that are indexed with a RangeIndex. For example, if you have the default indexing policy which only has RangeIndex on numbers, an Order By against a path with string values will return no documents.

If you have defined a partition key for your collections, note that Order By is supported only within queries that filter against a single partition key.

### Indexing for Order By for a single property

Here’s how you can create a collection with indexing for Order By against just the Title property, which is a string. There are two paths, one for the Title property (“/Title/?”) with Range indexing, and the other for every other property with the default indexing scheme, which is Hash for strings and Range for numbers.

booksCollection.IndexingPolicy.IncludedPaths.Add(  
 new IncludedPath {   
 Path = "/Title/?",   
 Indexes = new Collection<Index> {   
 new RangeIndex(DataType.String) { Precision = -1 } }   
 });  
  
await client.CreateDocumentCollectionAsync(UriFactory.CreateDatabaseUri("db"), booksCollection);

## Samples

Take a look at this [Github samples project](https://github.com/Azure/azure-documentdb-dotnet/tree/master/samples/code-samples/Queries) that demonstrates how to use Order By, including creating indexing policies and paging using Order By. The samples are open source and we encourage you to submit pull requests with contributions that could benefit other DocumentDB developers. Please refer to the [Contribution guidelines](https://github.com/Azure/azure-documentdb-net/blob/master/Contributing.md) for guidance on how to contribute.

## FAQ

**What is the expected Request Unit (RU) consumption of Order By queries?**

Since Order By utilizes the DocumentDB index for lookups, the number of request units consumed by Order By queries will be similar to the equivalent queries without Order By. Like any other operation on DocumentDB, the number of request units depends on the sizes/shapes of documents as well as the complexity of the query.

**What is the expected indexing overhead for Order By?**

The indexing storage overhead will be proportionate to the number of properties. In the worst case scenario, the index overhead will be 100% of the data. There is no difference in throughput (Request Units) overhead between Range/Order By indexing and the default Hash indexing.

**How do I query my existing data in DocumentDB using Order By?**

In order to sort query results using Order By, you must modify the indexing policy of the collection to use a Range index type against the property used to sort. See [Modifying Indexing Policy](documentdb-indexing-policies.md#modifying-the-indexing-policy-of-a-collection).

**What are the current limitations of Order By?**

Order By can be specified only against a property, either numeric or String when it is range indexed with the Maximum Precision (-1).

You cannot perform the following:

* Order By with internal string properties like id, \_rid, and \_self (coming soon).
* Order By with properties derived from the result of an intra-document join (coming soon).
* Order By multiple properties (coming soon).
* Order By with queries on databases, collections, users, permissions or attachments (coming soon).
* Order By with computed properties e.g. the result of an expression or a UDF/built-in function.

## Next steps

Fork the [Github samples project](https://github.com/Azure/azure-documentdb-dotnet/tree/master/samples/code-samples/Queries) and start ordering your data!

## References

* [DocumentDB Query Reference](/documentation/articles/documentdb-sql-query)
* [DocumentDB Indexing Policy Reference](/documentation/articles/documentdb-indexing-policies)
* [DocumentDB SQL Reference](https://msdn.microsoft.com/library/azure/dn782250.aspx)
* [DocumentDB Order By Samples](https://github.com/Azure/azure-documentdb-dotnet/tree/master/samples/code-samples/Queries)