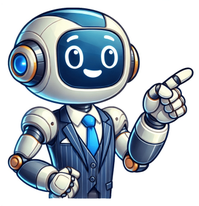
Click to prove you're human



Cosmos DB is a fully managed NoSQL database service that offers high availability, scalability, and performance. It is a popular choice for developers who need to store and query large amounts of data. One of the most common tasks that developers need to perform on Cosmos DB is to delete data. In this article, we will discuss how to delete data from Cosmos DB using a delete query. We will start by providing a brief overview of Cosmos DB and delete queries. Then, we will walk through an example of how to delete data from a Cosmos DB database. Finally, we will discuss some of the best practices for using delete queries in Cosmos DB. Cosmos DB Cosmos DB is a fully managed NoSQL database service that offers high availability, scalability, and performance. It is a popular choice for developers who need to store and query large amounts of data. Cosmos DB supports a variety of data models, including document, key-value, and graph. It also supports multiple consistency models, so developers can choose the level of consistency that best meets their needs. Delete queries A delete query is a query that deletes data from a Cosmos DB database. Delete queries can be used to delete data from a single document, a collection of documents, or an entire database. Example The following is an example of a delete query that deletes a document from a Cosmos DB database: DELETE FROM c WHERE id = ‘my-document-id’ This query will delete the document with the ID `my-document-id` from the collection `c`. Best practices When using delete queries in Cosmos DB, it is important to follow some best practices to ensure that your data is deleted correctly and efficiently. Use the `IF EXISTS` clause to avoid deleting documents that do not exist. Use the `LIMIT` clause to limit the number of documents that are deleted. Use the `OFFSET` clause to skip over documents that you do not want to delete. Use the `PARTITION KEY` and `ROW KEY` clauses to delete documents from specific partitions or rows. By following these best practices, you can ensure that your delete queries are efficient and effective. Cosmos DB Delete Query Description Example DELETE FROM WHERE ; Deletes all documents in the collection that match the specified filter. DELETE FROM myCollection WHERE age > 30; DELETE FROM WHERE = ; Deletes all documents in the collection that have the specified partition key value. DELETE FROM myCollection WHERE partitionKey = ‘my-partition-key’; DELETE FROM WHERE ; Deletes the document with the specified document ID. DELETE FROM myCollection WHERE documentId = ‘my-document-id’; Cosmos DB is a fully managed NoSQL database service that offers high availability, scalability, and performance. Cosmos DB supports a variety of data models, including documents, graphs, and key-value pairs. You can use Cosmos DB to store and query data from a variety of sources, including applications, websites, and IoT devices. One of the most common operations you will need to perform on a Cosmos DB database is deleting documents. Cosmos DB provides a number of ways to delete documents, including using the Azure Portal, the Cosmos DB API, and the Cosmos DB CLI. In this tutorial, you will learn how to delete documents from a Cosmos DB database using the Cosmos DB delete query. What is a Cosmos DB delete query? A Cosmos DB delete query is a SQL statement that deletes one or more documents from a Cosmos DB database. The syntax of a Cosmos DB delete query is similar to the syntax of a SQL delete statement. The following is an example of a Cosmos DB delete query: DELETE FROM mydb.mycollection WHERE id = ‘myid’ This query will delete the document with the id `myid` from the collection `mycollection` in the database `mydb`. How to write a Cosmos DB delete query? To write a Cosmos DB delete query, you must first specify the database and collection that you want to delete documents from. You can do this by using the `FROM` clause. The following is an example of the `FROM` clause: FROM mydb.mycollection Once you have specified the database and collection, you can use the `DELETE` keyword to delete documents from the collection. The following is an example of the `DELETE` keyword: DELETE FROM mydb.mycollection You can use the `WHERE` clause to specify the documents that you want to delete. The `WHERE` clause can be used to filter documents by their id, properties, or tags. The following is an example of the `WHERE` clause: DELETE FROM mydb.mycollection WHERE id = ‘myid’ This query will delete the document with the id `myid` from the collection `mycollection`. You can also use the `LIMIT` clause to limit the number of documents that are deleted. The `LIMIT` clause can be used to delete a specific number of documents, or to delete all documents up to a certain point. The following is an example of the `LIMIT` clause: DELETE FROM mydb.mycollection LIMIT 10 This query will delete the first 10 documents from the collection `mycollection`. In this tutorial, you learned how to delete documents from a Cosmos DB database using the Cosmos DB delete query. You learned how to specify the database and collection, how to use the `DELETE` keyword, and how to use the `WHERE` and `LIMIT` clauses. For more information on Cosmos DB delete queries, please refer to the following resources: [Cosmos DB documentation]([Cosmos DB API reference]([Cosmos DB CLI reference](3. Examples of Cosmos DB delete queries The following are some examples of Cosmos DB delete queries: * **Delete all documents from a collection:** DELETE FROM customers * **Delete all documents from a collection that match a certain criteria:** DELETE FROM customers WHERE age > 21 Delete a specific document from a collection: DELETE FROM customers WHERE id = ‘12345678-90ab-cdef-1234-567890abcdef’ Delete multiple documents from a collection: DELETE FROM customers WHERE id IN (‘12345678-90ab-cdef-1234-567890abcdef’, ‘23456789-0123-4567-8901-234567890abcdef’) ** 4. Tips for writing Cosmos DB delete queries When writing Cosmos DB delete queries, there are a few things to keep in mind: * **Use the `WHERE` clause to specify the documents that you want to delete.** The `WHERE` clause allows you to filter the documents that are deleted. For example, the following query deletes all documents from the `customers` collection that have the `age` property greater than 21: DELETE FROM customers WHERE age > 21 Use the `LIMIT` clause to limit the number of documents that you delete. The `LIMIT` clause allows you to specify the maximum number of documents that are deleted. For example, the following query deletes the first 10 documents from the `customers` collection: DELETE FROM customers LIMIT 10 Use the `OFFSET` clause to skip the first n documents. The `OFFSET` clause allows you to skip the first n documents from the collection. For example, the following query deletes the documents from the `customers` collection starting with the 11th document: DELETE FROM customers OFFSET 10 Use the `ORDER BY` clause to order the documents before deleting them. The `ORDER BY` clause allows you to order the documents before they are deleted. For example, the following query deletes the documents from the `customers` collection in descending order by the `age` property: DELETE FROM customers ORDER BY age DESC ** Cosmos DB delete queries can be used to delete documents from a collection. By using the `WHERE`, `LIMIT`, `OFFSET`, and `ORDER BY` clauses, you can control the documents that are deleted. Q: What is a Cosmos DB delete query? A Cosmos DB delete query is a SQL query that deletes one or more documents from a Cosmos DB database. The delete query uses the `DELETE` keyword followed by the document’s partition key and the document’s unique identifier (_rid). For example, the following delete query deletes the document with the partition key `”my-partition-key”` and the _rid `”my-rid”`: DELETE FROM my-container WHERE partitionKey = ‘my-partition-key’ AND _rid = ‘my-rid’ Q: How do I use a Cosmos DB delete query to delete all documents in a container? To delete all documents in a container, you can use the following delete query: DELETE FROM my-container This query will delete all documents in the `my-container` container, regardless of their partition key or _rid. Q: What are the limitations of Cosmos DB delete queries? There are a few limitations to Cosmos DB delete queries. First, you can only delete one document at a time. Second, you cannot use a delete query to delete a document that is being updated or deleted by another client. Third, you cannot use a delete query to delete a document that is locked by another client. Q: How can I use Cosmos DB delete queries to improve performance? You can use Cosmos DB delete queries to improve performance by batching together multiple delete queries. This can reduce the number of round trips to the database and improve the overall performance of your application. You can also use Cosmos DB indexes to improve the performance of your delete queries. By indexing the fields that you use in your delete queries, you can reduce the number of documents that need to be scanned to find the documents that you want to delete. Q: What are the best practices for using Cosmos DB delete queries? Here are a few best practices for using Cosmos DB delete queries: Use batching to improve performance. Use indexes to improve performance. Avoid deleting documents that are being updated or deleted by another client. Avoid deleting documents that are locked by another client. In this blog post, we discussed how to delete documents from a Cosmos DB database using the Cosmos DB delete query. We covered the syntax of the delete query, as well as some best practices for using it. We also provided an example of how to use the delete query to delete a document from a Cosmos DB database. We hope that this blog post has been helpful in understanding how to delete documents from a Cosmos DB database using the Cosmos DB delete query. If you have any questions or comments, please feel free to leave them below. Marcus Greenwood Hatch, established in 2011 by Marcus Greenwood, has evolved significantly over the years. Marcus, a seasoned developer, brought a rich background in developing both B2B and consumer software for a diverse range of organizations, including hedge funds and web agencies. Originally, Hatch was designed to seamlessly merge content management with social networking. We observed that social functionalities were often an afterthought in CMS-driven websites and set out to change that. Hatch was built to be inherently social, ensuring a fully integrated experience for users. Now, Hatch embarks on a new chapter. While our past was rooted in bridging technical gaps and fostering open-source collaboration, our present and future are focused on unraveling mysteries and answering a myriad of questions. We have expanded our horizons to cover an extensive array of topics and inquiries, delving into the unknown and the unexplored. Azure Cosmos DB is a database service that is globally distributed. It allows you to manage your data even if you keep them in data centers that are scattered throughout the world. It provides the tools you need to scale both global distribution pattern and computational resources, and these tools are provided by Microsoft Azure.Azure Storage offers a NoSQL key-value store for semi-structured data. Unlike a traditional relational database, each entity (such as a row — in relational database terminology) can have a different structure, allowing your application to evolve without downtime to migrate between schema.Food For ThoughtIf your application runs for multiple years. It accumulates lot of data documents on cosmos which are no longer useful. Therefore, we need a way to clean up the obsolete data documents. In this post, let us look at the way to write a stored procedure on cosmos DB for bulk delete and call it using the ASP.NET Core.Citation Note: The bulk delete stored proc is being used from the from Azure documentation. Thanks to Andrew LiuSteps to create stored procRight click on the stored procedure folder to create and give it a proper nameI named the proc as BulkDelete. This is how it looks after creating a proc.NOTE : Stored Procedures on Cosmos are written in JavaScriptStored procedure to bulk delete.Here is the Gist of codeBulk delete proc expects the query which gives it the bulk documents to delete from. Just pass in the query to proc. It will delete the documents.Note: If your collection is partitioned. Then it expects the partition key value to be passed to it. Again, it expects the value not the key name that it is partitioned with.Now we have the proc ready to delete the documents. Lets see how to use it on .NET .If you follow the repository design pattern in your application. Create a method inside the repository and later call it in service.Link for the gistFirst, we have to open the connection by using statement. The using statement is actually a syntactic convenience. At compile time, the language compiler implements the intermediate language (IL) for a try/catch block. To connect, we have to use DocumentClient Class from Microsoft.Azure.Documents. This Client has all the methods to create document, query for documents etc. More documentation for document client class can be found here.Now you connected to the database on the cosmos. Next step is to know where the proc is in that database. So, we can use the CreateStoredProcedureQuery to get proc. This method takes in URI of the database and collection ID, stored proc name and other feed options(we need these during cross partition querying). And, it returns the Query result of type IQueryable.After we have the proc. We can call ExecuteStoredProcedureAsync on document client class to execute the proc. This method takes in the GUID of the proc and NOT the proc name. We will have access to the GUID from the createstoredprocquery method we wrote previously, which is stored in sprocLink variable. How to use the GUID from it? there is property on it ‘_self’ we can pass in sprocLink._self and what ever inputs the proc takes.NOTE: If you have the partition keys on your collection, you should query for the partition keys and pass in the partition key as the request options for the proc. (Let me know if there is any other optimal solution, I’m open to learn.)ConclusionIn this post, we learnt how to write the stored procedures on Azure Cosmos and also how to integrate and use the procedures on .NET.Thank you for reading the post,Something somewhere incredible is waiting to be known! Share — copy and redistribute the material in any medium or format for any purpose, even commercially. Adapt — remix, transform, and build upon the material for any purpose, even commercially. The licensor cannot revoke these freedoms as long as you follow the license terms. Attribution — You must give appropriate credit , provide a link to the license, and indicate if changes were made . You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original. No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits. You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation . No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material. I have been working with couple of applications built with CosmosDB and one of the things that surprised me was one cannot clear all documents in a collection from the Azure web portal or using the Storage Explorer. As I was struggling to do this while doing some tests on the application I decided to write a blog on the solution I used. There are two ways to achieve the same Using a stored procedureUsing CosmosDb SDK I came up with a script in Node which can be done with any of the programming languages such as C#,Python supported by the SDK Let’s go through the steps: Step 1: Open VScode and Create a file named cosmosdb_helper.js Step 2: Let’s install the necessary packages needed. Install documentdb javascript sdk with the following command, npm i documentdb and you will see the output as follows Let’s install require to handle the dependencies with the following command, npm i require and you will see the output as follows, Step 3: Let’s do some coding. You will be able to understand the following code with the comments added on each line, Suppose if you have partitionKey created with your collection, you need to pass queryoptions with the partitionKey in selectAll as well as deletDocument as follows, Step 4: Let’s run the script and see the output, You can run the helper script as follows, if you want to list all documents in the collection, node cosmosdb_helper.js selectAll which will list the output of all documents in the collection. If you want to delete all documents within a collection, you can run the script as, node cosmosdb_helper.js deletAll which will remove all documents in the collection. As mentioned above, 2nd way is to use the stored procecure given by Microsoft employee as mentioned here. Hope the helper script will help someout out there in order to delete all documents in a collection. You can get the whole code from Cosmosd_Helper Cosmos is Microsoft’s NoSQL database offering on the Azure Cloud platform. If your application data needs are relatively simple, then Cosmos is an excellent choice.Scenario and motivation for this articleYou have a Customers collection. You want to delete all records using a simple DELETE statement. Sorry, thats not possible. Cosmos does not support DELETE statements.Wish list!! This does not workLack of support for DELETE and UPDATEUnfortunately Cosmos does not support the DELETE statement. In a traditional SQL world you would have done the following:DELETE FROM Customers c WHERE c.Country="USA"Why is this a problem?You now need to write some custom code! Yes! This would have been a one-liner in RDBMS world. But for Cosmos, this entails significant code writing.Option 1 — Javascript stored proceduresCosmos supports stored procedures stored proc for bulk deletionHow to execute a stored procedure?Drawbacks with stored procedures?I find them hard to develop and debugDifficult to automate the execution of Cosmos stored proces via CI/CDOption 2 — PowerShell to the rescueFortunately, the community has come to the rescue. Here is a very cute PowerShell module that interacts with Cosmos via the REST interfaceWiki documentation on various cmdletsHow to install the module?I always recommend using the CurrentUser scope. This is the path of least resistance.Install-Module -Scope CurrentUser -Name CosmosDBExample JSON{"id": "1002","firstName": "John1002","lastName": "Doe1002"}This will delete all records in the specified Cosmos collection. Useful in a dev-test scenario. You want to clean up the dev-test system to be free of any noisy data that might have crept in from previous test runs.Set-StrictMode -Version "latest"Clear-Host. $PSScriptRoot\common.ps1Write-Host "Getting Cosmos context"$CosmosContext=New-CosmosDbContext -Database $Global:CustomersManagementDatabase -ResourceGroupName $Global:CosmosResourceGroup -Account $Global:CosmosAccountNameWrite-Host ("Got Cosmos context for Cosmos account: '{0}' and database: '{1}'" -f $CosmosContext.Account, $CosmosContext.Database)function GetAllDocuments {#Remember to include Partition key path in field list$allDocs = Get-CosmosDbDocument -CollectionId $Global:CustomersMasterContainer -QueryEnableCrossPartition $true -Query "SELECT c.id FROM c" -Context $CosmosContextif ($null -eq $allDocs){return ,@()}return $allDocs}function DeleteDocuments {param ($documents)foreach ($doc in $documents) {#Remember to specify Partition keyRemove-CosmosDbDocument -Context $CosmosContext -CollectionId $Global:CustomersMasterContainer -Database $Global:CustomersManagementDatabase -Id $doc.id -PartitionKey $doc.id}}$documents=GetAllDocumentsWrite-Host ("Found {0} documents in the Container {1}" -f $documents.length,$Global:CustomersMasterContainer)DeleteDocuments -documents $documentsWrite-Host "Deletion complete"Scenario — Seed a Collection with documentsThis will pre-populate a Collection with JSON documents. This is helpful when you want to prepopulate your dev-test database with good quality baseline dataSet-StrictMode -Version "latest"Clear-Host. $PSScriptRoot\common.ps1Write-Host "Getting Cosmos context"$CosmosContext=New-CosmosDbContext -Database $Global:CustomersManagementDatabase -ResourceGroupName $Global:CosmosResourceGroup -Account $Global:CosmosAccountNameWrite-Host ("Got Cosmos context for Cosmos account: '{0}' and database: '{1}'" -f $CosmosContext.Account, $CosmosContext.Database)$jsonFiles=Get-ChildItem -Path $PSScriptRoot\json\ -Filter *.json -RecurseWrite-Host ("Found {0} json files" -f $jsonFiles.Length)foreach ($jsonFile in $jsonFiles) {$docContents=[system.io.file]::ReadAllText($jsonFile)$jsonObject=$docContents | ConvertFrom-JsonNew-CosmosDbDocument -Context $CosmosContext -CollectionId $Global:CustomersMasterContainer -DocumentBody $docContents -PartitionKey $jsonObject.idWrite-Output "Created document from file $jsonFile"}Advantages of PowerShell approach over Stored procesEasy to debug. All you need is Visual Studio Code and PowerShell CorePowerShell core is cross OSVery easy to automate via CI/CD.You will find accompanying PowerShell scripts in this repo: to run the accompanying PowerShell scripts?Clone this repositoryNavigate to the folder which contains this README.MD fileLaunch PowerShell CoreEnsure you have the core modules installedChange the variables in Common.ps1 to point to your Cosmos accountCreate a Cosmos account by runnning the script CreateCosmos.ps1Create a database and a collection (refer variables in Commons.ps1)Run PopulateCollection.ps1 You should now see 2 documents createdPurge all documents in the containerRun DeleteAllRecords.ps1PowerShell modules neccessaryAz.AccountsAz.ResourcesAz.CosmosDbCosmosDbChange the following variables in the file Common.ps1 to suit your environment.Set-StrictMode -Version "2.0"$ErrorActionPreference="Stop"$Global:CosmosResourceGroup="rg-demo-cosmos-serverless"$Global:CosmosAccountName="saudemocosmosserverless"$Global:CustomersManagementDatabase="CustomerManagement"$Global:CustomersMasterContainer="customers"$Global:Location="uksouth" While working on a project, I came across a use case to bulk delete documents in Azure Cosmos DB Container using Azure Data Factory (ADF). There are few approaches, however it mainly involves either implementing a stored procedure or a function. This article provides the steps to implement bulk delete of documents in Cosmos DB entirely in a ADF flow. Azure Cosmos DB SQL API has a large quantity of documents. The Azure Data Factory pipeline requires to bulk delete documents before loading new set of documents Access to Azure CloudA data source, either a csv file or excel file with the dataA data sink, Cosmos DB SQL API instanceADF pipeline that extracts the source, transforms the data from the source and loads into Cosmos DB sink i. Add source pointing to the Cosmos DB container that has records to be deleted ii. Go to the ‘Source options’ for the source and select ‘Query’ so you can enter the SQL query that will retrieve the documents to delete. In the example here, I am selecting all 3 bedroom home listing so those can be deleted from Cosmos DB iii. Add ‘Alter Row’ transformation to the source and go to ‘Alter row settings’. Select ‘Delete if’ under ‘Alter row conditions’ and enter ‘true()’ in the expression iv. Add sink pointing to the same Cosmos DB container v. Go to sink settings and select ‘Allow delete’ under ‘Update method’. If Cosmos DB has a large set of documents to delete, provide ‘Batch size’ to split the deletion To bulk delete documents in Azure Cosmos DB, Azure Data Factory provides easier approach because the flow is entirely in the data factory and it does not call external stored procedures or functions. This also allows better control over the data flow and provides better troubleshooting mechanism in case of any errors If you need to delete all documents from a container in Cosmos DB from the portal you can set the Time to Live value to 1 sec for all items in the container.Setting TTL to 1 will cause Cosmos to automatically begin deleting all items in a container without you needing to issue an explicit delete from a client application (or deleting items one by one). How you can configure TTL for Cosmos is below …As noted by Microsoft …Deletion of expired items is a background task that consumes left-over Request Units.Even after the TTL has expired, if the container is overloaded with requests and if there aren’t enough RU’s available, the data deletion is delayed. Data is deleted once there are enough RUs available to perform the delete operation. Data is not returned by any queries (by any API) after the TTL has expired even if the delete is delayed. … so after setting the TIL to 1 give it a few minutes to finish.