

**Exploring Cosmos DB via the SQL API**

**Introduction**

During this lab, you will learn how to manage your data in Cosmos DB via the SQL API using the Azure Portal.

**Estimated Time**

30 minutes

**Objectives**

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

         Use the SQL query syntax in the DocumentDB API to query various documents in a collection.

         Use JavaScript to create and use a simple stored procedure that generates a new document.

         Create a pre-trigger that can be fired as part of a DocumentDB operation.

Lab: Exploring Cosmos DB via the SQL API

During this lab, you will learn how to use the SQL query syntax in the DocumentDB API to query various documents in a collection, use JavaScript to create and use a simple stored procedure that generates a new document, and create a pre-trigger that can be fired as part of a DocumentDB operation.

**Exercise 1: Create Queries using the SQL Syntax**

In this hands-on exercise, we will use the SQL query syntax in the DocumentDB API to query various documents in a collection.

**Tasks**

1.       **View Supporting Files**

1.       View the documentdb-json-bulk folder in your workshop resources. This folder contains the documents to be used for this exercise.

2.       Extract the contents of the compressed folder to an empty directory.

You should see **50** \*.json files in the directory. These files will be used later in this hands-on exercise. Please take note of the directory where you extracted these files.

**2.      Create Account Resource**

1.       In a new window, sign in to the **Azure Portal** ([http://portal.azure.com](http://portal.azure.com/)).

2.       In the **Jumpbar (left navigation bar)**, click **New**, click **Databases**, and then click **Azure Cosmos DB**).

3.       In the **Azure Cosmos DB** blade, specify the desired configuration for the new Azure Cosmos DB account using the following steps:

                                                                           i.      In the **ID** box, enter a name to uniquely identify the account. When the **ID** is validated, a green check mark appears in the **ID** box. The ID value becomes the host name within the URI. The ID may contain only lowercase letters, numbers, and the '-' character, and must be between 3 and 50 characters.

                                                                         ii.      In the **API** section, select **SQL**.

                                                                       iii.      In the **Subscription** section, select the Azure subscription that you want to use for the account. If your account has only one subscription, that subscription is selected by default.

                                                                       iv.      In the **Resource Group** section, create a new resource group for your account named **CosmosWorkshop.**

                                                                         v.      Use **Location** to specify the geographic location closest to your current location in which to host your account.

4.       Once the new account options are configured, click the **Create** button to begin the deployment. To check the status of the deployment, check the Notifications hub at the top-right corner of your Azure portal.

5.       After the Azure Cosmos DB account is created, you will receive a notification in the **Notifications Hub** indicating that the deployment action is complete.

6.       Click the **Go to Resource** button in the notification to view the **Azure Cosmos DB** account resource. If you cannot find the notification, click the **bell** icon in the Azure portal to view your list of notifications.

**3.      Enable Diagnostic Logging**

Perform this step to be used in a future lab under the Monitoring module.

1.       In the Cosmos DB account that you just created, click **Diagnostic Logs** under the **Monitoring** section on the left bar.

2.       Click on **Turn on diagnostics.**

3.       In the new pop up window, enter a Name for your log, and check **send to log analytics.**

4.       Configure log analytics and create new OMS Workspace.

5.       Enter a name for your workspace and choose the same resource group and Subscription you used earlier.

6.       Select the closest location.

7.       Select **Free Pricing** tier

8.       Check DataPlaneRequests, MongoRequests, and Requests.

9.       Click **Save**.

4.       **Upload Sample Documents**

1.       At the top of the Azure Cosmos DB account blade, locate and click the **Add Collection** button.

2.       In the *Add Collection* blade, specify the following values for your new *collection and database*:

3.       In the **Database Id** box, enter the name **ecommerce**.

4.       In the **Collection Id** box, enter the name **customers**.

5.       In the **Storage Capacity** section, select the **Fixed (10 GB)** option.

6.       In the **Initial Throughput Capacity (RU/s)** section, enter the value **400** in the numeric box.

7.       Click the **OK** button.

8.       Wait for the "**Creating collection**" operation to complete.

9.       Click the **Document Explorer** option on the left-hand side of the blade.

10.   In the **Document Explorer** section, locate the *drop-down list* at the top of the blade. In the list, select the **customers** collection within the **ecommerce** database.

11.   Click the **…** option at the top of the blade and click **Upload.**

12.   In the **Upload Document** blade, locate and click the *folder* button to select files to upload.

13.   In your operating system's file dialog, locate and select all 50 \***.json** files that you extracted earlier in this hands-on exercise. Click the appropriate OS-specific button to upload these files.

14.   After returning to the browser and the **Upload Document** blade, locate and click the **Upload** button.

15.   Wait for the **Upload** operation to complete. In the **File Upload Status section**, you should see a *Result* indicating a status of **Succeeded** for all **50** documents.

16.   Close the **Upload Document** blade.

**5.      Performing SQL Query**

1.       In the Azure Cosmos DB account blade, click the **Data Explorer** option under the **Collections** section on the left-side of the blade.

2.       In the **Data Explorer** blade, select the **ecommerce** database and the **customers** collection.

3.       Locate the section in the current blade where you can edit the query text.

For the remainder of this hands-on exercise, we will refer to this section as the *query editor*.

4.       In the *query editor*, replace the current query with the following query:

SELECT \* FROM customers

5.       Click the **Run Query** button.

6.       In the **Results** blade, observe the results of your query.

This query should have returned all records in your collection.

7.       Close the **Results** blade.

8.       In the **Data Explorer** blade, locate the query editor and replace the current query with the following query:

SELECT \* FROM customers

WHERE customers.source = "direct-mail"

9.       Click the *Run Query* button.

10.   In the **Results** blade, observe the results of your query.

This query should have returned only records with the specified *source* value.

11.   Close the **Results** blade.

12.   In the **Data Explorer** blade, locate the *query editor* and replace the current query with the following query:

SELECT {

  "name": customers.name,

  "employer": customers.company

} AS entity

FROM customers

WHERE customers.source = "word-of-mouth"

13.   Click the **Run Query** button.

14.   In the **Results** blade, the results of your query should look similar (in format) to this sample. The query here returns the **customer names** and the **employers** *filtered* by **word of mouth** customer sources.

[

    {

        "entity": {

            "name": "Helen Mccarty",

            "employer": "FIBEROX"

        }

    },

    {

        "entity": {

            "name": "Dorothy Romero",

            "employer": "EXOSTREAM"

        }

    }

]

15.   Close the **Results** blade.

16.   In the **Data Explorer** blade, locate the query editor and replace the current query with the following query:

SELECT {

     "full-name": customers.name,

     "contact-details": {

        "phone": customers.phone,

        "address": customers.address

    },

     "employment": {

        "employer": customers.company,

        "work-email": customers.email

     }

} AS person

FROM customers

WHERE customers.source = "retail-location"

17.   Click the **Run Query** button.

18.   In the **Results** blade, the results of your query should look similar (in format) to this sample. This query returns the **full name, contact details, employment** as one entity *filtered* by customers who have a retail-location source.

[

    {

        "person":{

            "full-name":"Doreen Pope",

            "contact-details":{

                "phone":"+1 (963) 523-3749",

                "address":"798 Irwin Street, Avalon, District Of Columbia, 9258"

            },

            "employment":{

                "employer":"RENOVIZE",

                "work-email":"doreenpope@renovize.com"

            }

        }

    },

    {

        "person":{

            "full-name":"Rice Cunningham",

            "contact-details":{

                "phone":"+1 (877) 577-2261",

                "address":"566 Kossuth Place, Lemoyne, Arizona, 6294"

            },

            "employment":{

                "employer":"QABOOS",

                "work-email":"ricecunningham@qaboos.com"

            }

        }

    }

]

19.   Close the **Results** blade.

20.   In the **Data Explorer** blade, locate the query editor and replace the current query with the following query:

SELECT

    c.name,

    c.email,

    c.balance AS wealth

FROM customers c

WHERE c.source = "other"

ORDER BY c.balance DESC

21.   Click the **Run Query** button.

22.   In the **Results** blade, the results of your query should look similar (in format) to this sample. The Results here return the **name, email, and wealth** of the customers table *filtered* by the “other” source column and ordered by their wealth in a descending order.

[

    {

        "name":"Ruby Floyd",

        "email":"rubyfloyd@corecom.com",

        "wealth":"$3,985.15"

    },

    {

        "name":"Trina Parrish",

        "email":"trinaparrish@maximind.com",

        "wealth":"$3,865.36"

    },

    {

        "name":"Jeanine Higgins",

        "email":"jeaninehiggins@techmania.com",

        "wealth":"$3,796.23"

    }

]

*Exercise 1 has been completed.*

**Exercise 2: Create Stored Procedures**

This exercise shows how to create and use a simple stored procedure that generates a new document.

**Tasks**

**1.      Create Stored Procedure**

1.       In the Azure Cosmos DB account blade, click the **Data Explorer** option.

2.       In the **Data Explorer** section, expand the **ecommerce** database node and then expand the **customers** collection node. Click the **Documents** link.

3.       Click the **New Stored Procedure** button at the top of the **Data Explorer** section.

4.       In the new stored procedure tab, enter the name **CreateSampleCustomer** in the **Stored Procedure Id** box.

5.       Within the stored procedure's editor, enter the following JavaScript function. The stored procedure here creates a sample document when it is executed. This sample document represents a sample customer with a firstName, lastName, and a createdOn values. If an error occurs during document creation, then the function fails, and the error is displayed:

function createSampleCustomer() {

    var context = getContext();

    var collection = context.getCollection();

    var accepted = collection.createDocument(collection.getSelfLink(),

        {

            "firstName": "Sample",

            "lastName": "Person",

            "createdOn": new Date().toISOString()

        },

        function (err, documentCreated) {

            if (err) throw new Error('Error' + err.message);

            context.getResponse().setBody(documentCreated.id);

        }

    );

    if (!accepted) return;

}

6.       Click the **Save** button at the top of the stored procedure tab to persist your stored procedure.

**2.      Execute and Validate Stored Procedure**

1.       Click the **Data Explorer** option on the left-hand side of the blade.

2.       In the **Data Explorer** section, expand the **ecommerce** database node and then expand the **customers** collection node. Click the **Stored Procedures** to expand.

3.       If your list of scripts does not appear, click the **Refresh** button at the top of the section.

4.       Once the list of scripts has appeared, select the **CreateSampleCustomer** stored procedure.

5.       In the **CreateSampleCustomer** blade, click the **Execute** button at the top of the blade.

6.       Do not enter any additional parameters and click **Execute.**

7.       Wait for the stored procedure to finish execution and display a GUID result in the **Results** section of the blade.

8.       In the **Data Explorer** section, expand the **ecommerce** database node and then expand the **customers** collection node. Click the **Documents** link to view the various documents in the collection.

9.       Observe the newly created document in your collection and the number of documents that have become 51.

*Exercise 2 has been completed.*

**Exercise 3: Create Pre-Triggers**

**Tasks**

**1.      Create Stored Procedure**

1.       In the Azure Cosmos DB account blade, click the **Data Explorer** option.

2.       In the **Data Explorer** section, expand the **ecommerce** database node and then expand the **customers** collection node. Click the **Documents** link to view the empty collection.

3.       Click the dropdown arrow button at the top of the **Data Explorer** section.

4.       Click **New Trigger.**

5.       In the new trigger tab, enter the following values in each form field:

                                                                           i.      **Trigger Id**: ApplyOrgLabel

                                                                         ii.      **Trigger Type**: Pre

                                                                       iii.      **Trigger Operation**: Create

6.       Within the trigger's editor, enter the following JavaScript function:

function applyOrgLabel() {

    var context = getContext();

    var request = context.getRequest();

    var newDocument = request.getBody();

    if (!newDocument.organization) {

        newDocument["organization"] = "Contoso"

    }

    request.setBody(newDocument);

}

7.       Click the **Save** button at the top of the trigger tab to persist your trigger.

**2.      Triggers**

Your new trigger can now be used as part of a DocumentDB API request. It must be explicitly specified as part of a request to create a document regardless of the SDK, API or tool you use to access the database using the DocumentDB API.

Here is an example of a JavaScript request that will create a new document and use the new pre-trigger:

var document = {

    "alias": "aosmith",

    "department": "IT"

};

var options = {

    preTriggerInclude: "ApplyOrgLabel"

};

client.CreateDocumentAsync(

    collection.getSelfLink(),

    document,

    options

);

*Exercise 3 has been completed.*