

**View Request Charges**

Template Version: 2.0

**Introduction**

During this lab, you will learn how to view Request Charge headers with the Cosmos DB Java (Async) SDK. You will learn how to view Request Units consumed per request in Cosmos DB.

**Estimated Time**

35 minutes.

**Objectives**

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

* Create and provision Account resources
* Navigate the Cosmos DB service blades.
* Create Cosmos DB databases and collections
* Upload documents to a collection
* View Request Charge headers.
* Understand how RU consumption relates to query complexity.

Lab: View Request Charges

During this lab, you will learn how to view Request Charge headers with the Cosmos DB Java (Async) SDK to see how you can view Request Units consumed per request in Cosmos DB.

**Exercise 1: Create a Cosmos DB Account**

This exercise shows you how to create a Cosmos DB account.

**Tasks**

**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 **Jump bar (left navigation bar)**, click **Create a resource**, click **Databases**, and then click **Azure Cosmos DB**).
   1. In the **Azure Cosmos DB** blade, specify the desired configuration for the new Azure Cosmos DB account using the following steps:
   2. 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.
   3. 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.
   4. In the **Resource Group** section, create a new resource group for your account named **CosmosWorkshop.**
   5. In the **API** section, select **SQL**.
   6. Use **Location** to specify the geographic location closest to your current location in which to host your account.
   7. **Disable** Geo-Redundancy.
   8. **Disable** Multi-region Writes
3. 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. **Note: This may take several minutes.**
4. After the Azure Cosmos DB account is created, you will receive a notification in the **Notifications Hub** indicating that the deployment action is complete.
5. 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.

**Explore Cosmos DB Service in the Portal**

1. Spend a few minutes locating the following features:
   1. Monitoring Metrics
   2. Data Explorer
   3. Global Replication

**Record your Cosmos DB URI**

1. In the **Overview** page for your Cosmos DB Resource, locate the **URI** for your Cosmos DB account.
2. Record this value in a new notepad text document.

**Record your account key**

1. In the left-hand navigation pane for your Cosmos DB account, click on the **Keys** item in the **Settings** section.
2. Locate the **Primary Key** for your Cosmos DB account.
3. Copy and paste the **Primary** **Key** into the same text document created in task 2.

**Add a Collection**

1. From the Overview or Data Explorer blades, select the **New Collection** button.
2. Create a collection with the following values:
   1. Database id: **ViewRequestCharges**
   2. Collection id: **customers**
   3. Storage capacity: **Fixed (10 GB)**
   4. Partition Key: (leave empty)
   5. Throughput: **400 RU/s**
   6. Unique Keys: (leave empty)
3. Add documents to the collection:
   1. Navigate to **Data Explorer** from the left navigation menu.
   2. Expand the **ViewRequestCharges** database. Then, expand the **customers** collection and select **Documents**.
   3. Select **Upload** of the **Data Explorer** blade top menu.
   4. Navigate to the workshop materials folder and upload json documents contained in the **Sample Data/customer\_data** folder.
   5. Close upload complete dialog and confirm documents exist in the collection by refreshing the collection (option located underneath blue Edit Filter button).

*Exercise 1 has been completed.*

**Exercise 2: View Request Charges**

This exercise shows you how to view Request Charges using the provided Java solution and assumes you have installed the following prerequisites:

1. [Git SCM](https://git-scm.com/downloads) for accessing sample project.
2. [Java 8+](https://www.java.com/en/download/) and [Maven](https://maven.apache.org/download.cgi) for packaging and running examples.
3. Your favorite text editor or IDE.

**Tasks**

**1.      Open and configure the view-request-charges project**

1.       Clone the git repository located at:

https://github.com/johnabele/java-cosmosdb-exercises.git

1. Navigate to the desired folder location.
2. Open terminal/cmd and use command: git clone https://github.com/johnabele/java-cosmosdb-exercises.git
3. Remote repo contents should be downloaded to the folder.

2. Open the **view-request-charges** solution in your IDE or text editor and edit the com.microsoft.azure.cosmosdb.sample.AccountSettings.java file.

3. Change the **ACCOUNT\_HOST** propertyvalue to the **endpoint URI** you recorded in Exercise 1.

4.    Change the **ACCOUNT\_KEY** propertyvalue to the **Primary Key** you recorded in Exercise 1.

**2.      Examine the project.**

1.       Open the com.microsoft.azure.cosmosdb.sample.QueryManager.java file within the **view-request-charges** project.

2.       Notice there are five method calls, each corresponding to an example read query:

a.       QueryWithOneFilters

b.       QueryWithTwoFilters

c.        QueryWithRangeOperators

d.       QueryWithSingleJoin

e.       QueryWithDoubleJoin

Given the brief descriptions of each query, which queries do you think will require more computational effort and therefore consume more Request Units?

**4.      Complete and Execute the queries.**

1. Inside the QueryManager.java file, set the database id to **ViewRequestCharges** and collection id to **customers**.
2. Next, add the following query text to the QueryWithOneFilter method:

SELECT \* FROM c WHERE c.name = 'Dorothy Romero'

1. Open the Main.java file and add the following code to the main method:

QueryManager queryManager = new QueryManager();

try{

queryManager.QueryWithOneFilter();

}catch(Exception ex){

System.out.println("An error occurred.");

}

1. From the root folder of the project, open a terminal/bash/cmd interface and run the following Maven CLI commands:

mvn clean package

mvn exec:java

1. Observe the results and Request Charge for the query.
2. Repeat for the following queries:
   1. **Filter on two properties**

SELECT \* FROM c WHERE c.name = 'Dorothy Romero' OR c.balance = '$1,444.96'

* 1. **Range operator on datetime**

SELECT \* FROM c WHERE c.registered >= '2017-02-18T05:49:33 +4:00'

* 1. **Single join**

SELECT f.id FROM customers f JOIN c IN f.companies

1. **Double join**

SELECT f.name AS customer, c.companyName AS company, p.location AS location FROM customers f JOIN c IN f.companies JOIN p IN c.locations

**5.      Interpreting the console outputs**

1.       For each of the 5 example queries the console will output the following:

a.       SQL query name

b.        Results or summarized results

c.       Request Charges (RUs consumed)

How did query complexity affect Request Charge? Were you surprised by any of the results?

*Exercise 2 has been completed.*

**Results**

The queries and their expected RU consumption:

**1.      Simple query equality**

RequestCharge: 2.98

**2.      Filter on two properties**

RequestCharge: 3.99

**3.      Range operator**

RequestCharge: 6.62

**4.      Single join**

RequestCharge: 9.26

**5.      Double join**

RequestCharge: 9.46

As query complexity and payload size increase, the Request Units consumed for each query will increase as well.

Compare query 1 (single filter) and query 2 (double filter). Both perform similar filters on un-partitioned columns. Query 2 returns double the size of the payload as query 1, but the Request Charges are not doubled, Request Units do not necessarily scale linearly with response size.

Request Units consumed are dependent on both payload size and computational work. Therefore, for throughput calculation estimations, it is critical to not only know the number of queries but the types of queries for a specified workload.