Big Data Orientation

Lab 3 – Working with NoSQL Data in Microsoft Azure

Overview

Azure supports many services for working with NoSQL data. In this lab, you will explore NoSQL data using the Cosmos DB service – a highly scalable data store for key-value table, JSON document, and graph data.

Before You Start

To complete this lab, you will need the following:

- A web browser
- A Windows, Linux, or Mac OS X computer

Important: Before you can perform the lab exercises, you must complete the previous labs in this course.

Exercise 1: Working with a Key-Value Table

In this exercise, you will use Cosmos DB to work with key-value data in Azure Table storage.

Provision Azure Cosmos DB for Table Storage

To get started, you must provision an instance of Azure Cosmos DB.

1. In the Microsoft Azure portal, in the menu, click **New**. Then in the **Databases** menu, click **Azure Cosmos DB**.
2. In the **Azure Cosmos DB** blade, enter the following settings, and then click **Create**:
    - **ID**: *Enter a unique ID for your Cosmos DB service*
    - **API**: Table (key-value)
    - **Subscription**: *Select your Azure subscription*
    - **Resource Group**: *Select the resource group you created in the previous labs*
    - **Location:** *Select any available region*
    - **Pin to dashboard:** Unselected
3. In the Azure portal, view **Notifications** to verify that deployment has started. Then wait for the Cosmos DB service to be deployed (this can take a few minutes.)

Create a Table

Cosmos DB supports Azure Table Storage for key-value pairs.

1. View the blade for the Cosmos DB instance you created for table storage, and click the **Data Explorer** tab.
2. In the Data Explorer, click **New Table**.

3. In the **Add Table** pane, enter the following details and click **OK**:
   - **Table Id**: Employees
   - **Storage capacity**: Fixed (10 GB)
   - **Throughput**: 2000
4. In the Data Explorer, expand the new **Employees** table and select **Entities**.
5. In the **Entities** tab, click **Add Entity**.
6. In the **Add Table Entity** pane, Enter the following property values (clicking **+ Add Property** as required) and then click **Add Entity**.

Property Name	Type	Value
PartitionKey	String	Marketing
RowKey	String	dan@contoso.com
FirstName	String	Dan
LastName	String	Drayton

7. Add a second entity with the following properties:

Property Name	Type	Value
PartitionKey	String	Marketing
RowKey	String	joann@contoso.com
FirstName	String	Joann
LastName	String	Chambers
PhoneNumber	String	555-123-4567

8. Add a third entity with the following properties:

Property Name	Type	Value
PartitionKey	String	Human Resources
RowKey	String	rosie@contoso.com
FirstName	String	Rosie
LastName	String	Reeves
PhoneNumber	String	555-321-7654

Query a Table

Now that you have some data in your table, you can query it.

1. In the **Entities** pane, click **Add new clause**.
2. Set the following properties for the clause:

And/Or	Field	Type	Operator	Value
	PartitionKey	String	=	Marketing

3. Click ▷ **Run** to run the query and apply the filter. Note that only the rows where the **PartitionKey** value is "Marketing" are returned.
4. Click **Add new clause** to add a second clause to the query.
5. Set the following properties for the second clause:

And/Or	Field	Type	Operator	Value
And	LastName	String	=	Chambers

6. Click ▷ **Run** to run the query and apply the filter. Note that only the row for Joann Chambers is returned.

Exercise 2: Working with Documents

In this exercise, you will use Azure Cosmos DB to work with a collection of JSON documents.

Provision Azure Cosmos DB for Document Storage

To get started, you must provision an instance of Azure Cosmos DB.

1. In the Microsoft Azure portal, in the menu, click **New**. Then in the **Databases** menu, click **Azure Cosmos DB**.
2. In the **Azure Cosmos DB** blade, enter the following settings, and then click **Create**:
    - **ID**: *Enter a unique ID for your Cosmos DB service*
    - **API**: SQL (DocumentDB))
    - **Subscription**: *Select your Azure subscription*
    - **Resource Group**: *Select the resource group you created in the previous labs*
    - **Location:** *Select any available region*
    - **Pin to dashboard:** Unselected
3. In the Azure portal, view **Notifications** to verify that deployment has started. Then wait for the Cosmos DB service to be deployed (this can take a few minutes.)

Create a Document Collection

Cosmos DB supports Document DB databases for collections of JSON documents.

1. View the blade for the Cosmos DB instance you created for document storage, and click the **Data Explorer** tab.
2. In the Data Explorer, click **New Collection**.
3. In the **Add Collection** pane, enter the following details and click **OK**:
    - **Database id**: DocDB
    - **Collection Id**: EmployeeDocs
    - **Storage capacity**: Fixed (10 GB)
    - **Throughput**: 2000
    - **Partition key**: /dept
4. In the Data Explorer, expand the new **EmployeeDocs** collection and select **Documents**.
5. In the **Documents** tab, click **New Document**.
6. In the document editor, create the following JSON document and then click **Save**:

```
{
    "id": "1",
    "dept": "Marketing",
    "email": "dan@contoso.com",
    "firstName": "Dan",
    "lastName": "Drayton"
}
```

7. Click **New Document** and create a second document with the following JSON:

```
{
    "id": "2",
    "dept": "Marketing",
    "email": "joann@contoso.com",
    "firstName": "Joann",
    "lastName": "Chambers"
}
```

8. Click **New Document** and create a third document with the following JSON:

```
{
    "id": "3",
    "dept": "Human Resources",
    "email": "rosie@contoso.com",
    "firstName": "Rosie",
    "lastName": "Reeves"
}
```

Query a Document Collection

Now that you have a collection of documents, you can query it:

1. In the Data Explorer, click **New SQL Query**.
2. In the **Query 1** tab, modify the existing query to use the following SQL statement:

```
SELECT c.firstName, c.lastName
FROM c
WHERE c.dept = 'Marketing'
```

3. Click ▷ **Execute Query** to run the query. Note that only the documents for Dan Drayton and Joann Chambers are returned.

Exercise 3: Working with a Graph

In this exercise, you will use Azure Cosmos DB to work with a Graph.

Provision Azure Cosmos DB for Graph Storage

To get started, you must provision an instance of Azure Cosmos DB.

1. In the Microsoft Azure portal, in the menu, click **New**. Then in the **Databases** menu, click **Azure Cosmos DB**.
2. In the **Azure Cosmos DB** blade, enter the following settings, and then click **Create**:
    - **ID**: *Enter a unique ID for your Cosmos DB service*
    - **API**: Gremlin (graph))
    - **Subscription**: *Select your Azure subscription*
    - **Resource Group**: *Select the resource group you created in the previous labs*
    - **Location:** *Select any available region*
    - **Pin to dashboard:** Unselected
3. In the Azure portal, view **Notifications** to verify that deployment has started. Then wait for the Cosmos DB service to be deployed (this can take a few minutes.)

Create a Graph Collection

Cosmos DB supports Graph collections that you can manipulate and query using the Gremlin language.

1. View the blade for the Cosmos DB instance you created for document storage, and click **+ Add Graph**.
2. In the **Add Graph** pane, enter the following details and click **OK**:
    - **Graph Id**: EmployeeGraph
    - **Storage capacity**: Fixed (10 GB)
    - **Throughput**: 2000
    - **Partition key**: *Leave Blank*
    - **Database**: GraphDB
3. In the **Overview** page for your new graph, note the **Gremlin URI** (which should be in the form *cosmosdbname*.graphs.azure.com)– you will need this later to connect to your graph from a client app.
4. On the **Keys** tab for your graph, note the **Primary Key** – you will also need this to connect to your graph from a client app.

Install Node.JS and Run a Script to Populate the Graph

Although you can create vertices in a graph using the Data Explorer interface in the Azure portal, it lacks the ability to edit vertices and create edges between them. You will therefore use a Node.JS script to populate your graph.

1. Open a new tab on your browser and navigate to https://nodejs.org/en/download/.
2. Download and install the appropriate version of Node.JS for your platform (Windows, Mac OS X, or Linux).
3. In the folder where you extracted the lab files for this course, in the **graph** folder, open **config.js** in a code or text editor.
4. In config.js, replace the values for the **config.endpoint** and **config.primaryKey** variables with the **Gremlin URI** and **Primary Key** values you noted in the Azure portal. Then save and close config.js.
5. Open a command window in the **graph** folder, and run the following command to install the **gremlin-secure** Node package (which is required to connect securely to a Gremlin data source):
   ```
   npm install gremlin-secure
   ```
6. After the package has been installed, enter the following command to Node app, which will populate your graph:
   ```
   node app
   ```
7. Wait for the app to finish, and then press CTRL + C to exit it and close the command window.

Explore the Graph

Now that your graph contains some vertices and edges, you can explore it in the Data Explorer.

1. Return to the browser tab containing the Azure portal, and view the Data Explorer tab for the Cosmos DB instance you created for graph data.
2. In Data Explorer, expand **EmployeeGraph** and click **Graph**.

3. In the **Graph** pane, click **Load graph**.
4. Note that the graph contains employees that are related to one another based on which employees follow one another on social media.
5. Select the **dan@contoso.com** vertex, and note its properties, which include a **label** indicating that this vertex represents a person, and an **age** property
6. Select **rosie@contoso.com** and note that this person does not have an **age** property.
7. In the filter text box, replace the default `g.V()` filter with the following Gremlin expression, and then click **Apply Filter**.

   ```
   g.V().hasLabel('person').has('age')
   ```
8. View the vertices that are returned, which include all vertices that have a label of **person** and an **age** property.
9. Apply a new filter based on the following expression:

   ```
   g.V().hasLabel('person').has('age').values('firstName')
   ```
10. Note that this returns a JSON string containing the **firstName** values for each person with an **age** property.
11. Apply a new filter based on the following expression:

    ```
    g.V('dan@contoso.com').outE('follows').inV()
    ```
12. Note that this returns the people that are followed by **dan@contoso.com**.
13. Apply a new filter based on the following expression:

    ```
    g.V('dan@contoso.com').outE('follows').inV().outE('follows').inV()
    ```
14. Note that this returns the people that are followed by the people who are followed by **dan@contoso.com** (which includes Dan himself, because he follows Joann, who in turn follows him!)