Azure Cosmos DB provides language-integrated, transactional execution of JavaScript that lets you write **stored procedures**, **triggers**, and **user-defined functions (UDFs)**. When using the SQL API in Azure Cosmos DB, you can define the stored procedures, triggers, and UDFs in JavaScript language. You can write your logic in JavaScript and execute it inside the database engine.

**Note**

For partitioned containers, when executing a stored procedure, a partition key value must be provided in the request options. Stored procedures are always scoped to a partition key. Items that have a different partition key value will not be visible to the stored procedure. This also applied to triggers as well.

**How to write stored procedures**

Stored procedures are written using JavaScript, they can create, update, read, query, and delete items inside an Azure Cosmos container. Stored procedures are registered per collection, and can operate on any document or an attachment present in that collection.

**Example**

Here is a simple stored procedure that returns a "Hello World" response.

var helloWorldStoredProc = {

id: "helloWorld",

serverScript: function () {

var context = getContext();

var response = context.getResponse();

response.setBody("Hello, World");

}

}

The context object provides access to all operations that can be performed in Azure Cosmos DB, as well as access to the request and response objects. In this case, you use the response object to set the body of the response to be sent back to the client.

### Create an item using stored procedure

When you create an item by using stored procedure, the item is inserted into the Azure Cosmos DB container and an id for the newly created item is returned. Creating an item is an asynchronous operation and depends on the JavaScript callback functions. The callback function has two parameters - one for the error object in case the operation fails and another for a return value; in this case, the created object. Inside the callback, you can either handle the exception or throw an error. In case a callback is not provided and there is an error, the Azure Cosmos DB runtime will throw an error.

function createToDoItem(itemToCreate) {

var context = getContext();

var container = context.getCollection();

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

itemToCreate,

function (err, itemCreated) {

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

context.getResponse().setBody(itemCreated.id)

});

if (!accepted) return;

}

### Arrays as input parameters for stored procedures

When defining a stored procedure in Azure portal, input parameters are always sent as a string to the stored procedure. Even if you pass an array of strings as an input, the array is converted to string and sent to the stored procedure. To work around this, you can define a function within your stored procedure to parse the string as an array. The following code shows how to parse a string input parameter as an array:

function sample(arr) {

if (typeof arr === "string") arr = JSON.parse(arr);

arr.forEach(function(a) {

// do something here

console.log(a);

});

}

**Triggers**

Azure Cosmos DB supports pre-triggers and post-triggers. Pre-triggers are executed before modifying a database item and post-triggers are executed after modifying a database item.

The following example shows how a pre-trigger is used to validate the properties of an Azure Cosmos DB item that is being created.

function validateToDoItemTimestamp() {

var context = getContext();

var request = context.getRequest();

// item to be created in the current operation

var itemToCreate = request.getBody();

// validate properties

if (!("timestamp" in itemToCreate)) {

var ts = new Date();

itemToCreate["timestamp"] = ts.getTime();

}

// update the item that will be created

request.setBody(itemToCreate);

}

Pre-triggers cannot have any input parameters. The request object in the trigger is used to manipulate the request message associated with the operation. In the previous example, the pre-trigger is run when creating an Azure Cosmos DB item, and the request message body contains the item to be created in JSON format.

When triggers are registered, you can specify the operations that it can run with. This trigger should be created with a TriggerOperation value of TriggerOperation.Create, which means using the trigger in a replace operation as shown in the following code is not permitted.

**Execute Trigger**

dynamic newItem = new

{

category = "Personal",

name = "Groceries",

description = "Pick up strawberries",

isComplete = false

};

Uri containerUri = UriFactory.CreateDocumentCollectionUri("myDatabase", "myContainer");

RequestOptions requestOptions = new RequestOptions { PreTriggerInclude = new List<string> { "trgPreValidateToDoItemTimestamp" } };

await client.CreateDocumentAsync(containerUri, newItem, requestOptions);

**Post-triggers**

One thing that is important to note is the transactional execution of triggers in Azure Cosmos DB. The post-trigger runs as part of the same transaction for the underlying item itself. An exception during the post-trigger execution will fail the whole transaction. Anything committed will be rolled back and an exception returned.

**user-defined functions**

function tax(income) {

if(income == undefined)

throw 'no input';

if (income < 1000)

return income \* 0.1;

else if (income < 10000)

return income \* 0.2;

else

return income \* 0.4;

}

**private async Task CreateUDF()**

{

string udfId = "Tax";

var udfTax = new UserDefinedFunction

{

Id = udfId,

Body = File.ReadAllText(@"D:\Gaurav\Project\CosmosDbWithDotNetCore\CosmosDbWithDotNetCore\udf\Tax.js"),

};

Uri containerUri = UriFactory.CreateDocumentCollectionUri("myDatabase", "myContainer");

await client.CreateUserDefinedFunctionAsync(containerUri, udfTax);

}

**private void ExecuteUDF()**

{

Uri containerUri = UriFactory.CreateDocumentCollectionUri("myDatabase", "myContainer");

var results = client.CreateDocumentQuery<dynamic>(containerUri, "SELECT \* FROM Incomes t WHERE udf.Tax(t.income) > 20000");

}

using Microsoft.Azure.Documents;

using Microsoft.Azure.Documents.Client;

using System;

using System.Linq;

using System.Threading.Tasks;

namespace CosmosDbWithDotNetCore

{

class Program

{

**private const string databaseId = "traildb";**

**private const string connectionPolicy = "";**

**private const string authorizationKey = "eZCc0Gq9guq6ws5HUz1l5OZ7LymVFhsYL0ur8M6TJXFUIF7fhggT114X9WBaR1Ac6LOW38yNx1AJQ89iAd8kqg==";**

**private const string endpointUrl = "https://trialdocdb.documents.azure.com:443/";**

**private static DocumentClient client;**

public Program()

{

**client = new DocumentClient(new Uri(endpointUrl), authorizationKey);**

}

static void Main(string[] args)

{

try

{

Program p = new Program();

p.CreateDataBase().Wait();

p.CreateCollection().Wait();

p.CreateDocument().Wait();

p.ReadDocument();

p.UpdateDocument();

p.DeleteCollection();

p.DeleteDatabase().Wait();

}

catch (DocumentClientException de)

{

Exception baseException = de.GetBaseException();

Console.WriteLine("{0} error occurred: {1}, Message: {2}", de.StatusCode, de.Message, baseException.Message);

}

catch (Exception e)

{

Exception baseException = e.GetBaseException();

Console.WriteLine("Error: {0}, Message: {1}", e.Message, baseException.Message);

}

finally

{

Console.WriteLine("End of demo, press any key to exit.");

Console.ReadKey();

}

}

#region Database

/// <summary>

/// Create Database

/// </summary>

/// <returns></returns>

private async Task CreateDataBase()

{

**// 1 - Query for a Database**

Database database1 = client.CreateDatabaseQuery().Where(db => db.Id == databaseId).AsEnumerable().FirstOrDefault();

Console.WriteLine("1. Query for a database returned: {0}", database1 == null ? "no results" : database1.Id.ToString());

**// 2 - Create a Database**

var database = client.CreateDatabaseIfNotExistsAsync(new Database { Id = databaseId });

Console.WriteLine("2. Query for a database returned: {0}", database == null ? "no results" : database.Id.ToString());

//var database2 = client.CreateDatabaseAsync(new Database { Id = databaseId }, new RequestOptions { OfferThroughput = 1000 });

//Console.WriteLine("3. Query for a database returned: {0}", database2 == null ? "no results" : database2.Id.ToString());

**//// 3 - Get a single database**

//var database4 = client.ReadDatabaseAsync(UriFactory.CreateDatabaseUri(databaseId));

//Console.WriteLine("\n3. Read a database resource: {0}", database4);

**// 4 - List all databases for an account**

var databases = await client.ReadDatabaseFeedAsync();

Console.WriteLine("\n4. Reading all databases resources for an account");

foreach (var db in databases)

{

Console.WriteLine(db);

}

}

/// <summary>

**/// Delete Database**

/// </summary>

/// <returns></returns>

private async Task DeleteDatabase()

{

await client.DeleteDatabaseAsync(UriFactory.CreateDatabaseUri(databaseId));

Console.WriteLine("\n Database deleted");

}

#endregion

#region Collection

/// <summary>

**/// Create Collection**

/// </summary>

/// <returns></returns>

private async Task CreateCollection()

{

// client.CreateDocumentCollectionQuery(UriFactory.CreateDatabaseUri(databaseId),new SqlQuerySpec { QueryText="",Parameters = new SqlParameterCollection { } });

await client.CreateDocumentCollectionIfNotExistsAsync(UriFactory.CreateDatabaseUri(databaseId), new DocumentCollection { Id = "FamilyCollection\_oa" });

//await client.CreateDocumentCollectionAsync(UriFactory.CreateDatabaseUri(databaseId), new DocumentCollection { Id = "FamilyCollection\_oa" });

}

/// <summary>

**/// Delete Collection**

/// </summary>

private void DeleteCollection()

{

client.DeleteDocumentCollectionAsync(UriFactory.CreateDocumentCollectionUri(databaseId, "FamilyCollection\_oa"));

Console.WriteLine("\n Collection deleted");

}

#endregion

#region Document

/// <summary>

**/// Create Document**

/// </summary>

/// <returns></returns>

private async Task CreateDocument()

{

Family andersenFamily = new Family

{

Id = "Andersen.1",

LastName = "Andersen",

Parents = new Parent[]

{

new Parent { FirstName = "Thomas" },

new Parent { FirstName = "Mary Kay" }

},

Children = new Child[]

{

new Child

{

FirstName = "Henriette Thaulow",

Gender = "female",

Grade = 5,

Pets = new Pet[]

{

new Pet { GivenName = "Fluffy" }

}

}

},

Address = new Address { State = "WA", County = "King", City = "Seattle" },

IsRegistered = true

};

await client.CreateDocumentAsync(UriFactory.CreateDocumentCollectionUri(databaseId, "FamilyCollection\_oa"), andersenFamily);

Family wakefieldFamily = new Family

{

Id = "Wakefield.7",

LastName = "Wakefield",

Parents = new Parent[]

{

new Parent { FamilyName = "Wakefield", FirstName = "Robin" },

new Parent { FamilyName = "Miller", FirstName = "Ben" }

},

Children = new Child[]

{

new Child

{

FamilyName = "Merriam",

FirstName = "Jesse",

Gender = "female",

Grade = 8,

Pets = new Pet[]

{

new Pet { GivenName = "Goofy" },

new Pet { GivenName = "Shadow" }

}

},

new Child

{

FamilyName = "Miller",

FirstName = "Lisa",

Gender = "female",

Grade = 1

}

},

Address = new Address { State = "NY", County = "Manhattan", City = "NY" },

IsRegistered = false

};

await client.CreateDocumentAsync(UriFactory.CreateDocumentCollectionUri(databaseId, "FamilyCollection\_oa"), wakefieldFamily);

}

/// <summary>

**/// Read document**

/// </summary>

private void ReadDocument()

{

var data = client.CreateDocumentQuery<Family>(UriFactory.CreateDocumentCollectionUri(databaseId, "FamilyCollection\_oa"), new FeedOptions { MaxItemCount = -1 });

foreach (Family family in data)

{

Console.WriteLine("\tRead {0}", family);

}

var WakefieldDataQuery = client.CreateDocumentQuery<Family>(UriFactory.CreateDocumentCollectionUri(databaseId, "FamilyCollection\_oa"), "SELECT \* FROM Family WHERE Family.LastName = 'Wakefield'", new FeedOptions { MaxItemCount = -1 });//.Where(f => f.LastName == "Wakefield");

foreach (Family family in WakefieldDataQuery)

{

Console.WriteLine(("\n {0}", family));

}

}

/// <summary>

**/// Update Document**

/// </summary>

private void UpdateDocument()

{

Family updateFamily = new Family

{

Id = "Andersen.1",

LastName = "Andersen",

Parents = new Parent[]

{

new Parent { FirstName = "Thomas1" },

new Parent { FirstName = "Mary Kay1" }

},

Children = new Child[]

{

new Child

{

FirstName = "Henriette Thaulow1",

Gender = "female",

Grade = 5,

Pets = new Pet[]

{

new Pet { GivenName = "Fluff1y" }

}

}

},

Address = new Address { State = "WA1", County = "Kin1g", City = "Seattle1" },

IsRegistered = false

};

client.ReplaceDocumentAsync(UriFactory.CreateDocumentUri(databaseId, "FamilyCollection\_oa", "Andersen.1"), updateFamily);

var UpdatedAndersenQuery = client.CreateDocumentQuery<Family>(UriFactory.CreateDocumentCollectionUri(databaseId, "FamilyCollection\_oa"), new FeedOptions { MaxItemCount = -1 }).Where(f => f.LastName == "Andersen");

foreach (Family family in UpdatedAndersenQuery)

{

Console.WriteLine(("\n Update Details \n {0}", family));

}

}

/// <summary>

**/// Delete Document**

/// </summary>

private void DeleteDocument()

{

client.DeleteDocumentAsync(UriFactory.CreateDocumentUri(databaseId, "FamilyCollection\_oa", "Andersen.1"));

Console.WriteLine(("\n Document Deleted."));

}

#endregion

}

}