DocumentDB SQL

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

DocumentDB is Microsoft's newest NoSQL document database platform that runs on Azure. DocumentDB is designed keeping in mind the requirements of managing data for latest applications. This tutorial talks about querying documents using the special version of SQL supported by DocumentDB with illustrative examples.

Audience

This tutorial is designed for developers who want to get acquainted with how to query DocumentDB using a familiar Structured Query Language (SQL).

Prerequisites

It is an elementary tutorial that explains the basics of DocumentDB and there are no prerequisites as such. However, it will certainly help if you have some prior exposure to NoSQL technologies.

Disclaimer & Copyright

Table of Contents

1. DOCUMENTDB SQL – OVERVIEW

DocumentDB is Microsoft's newest NoSQL document database platform that runs on Azure. In this tutorial, we will learn all about querying documents using the special version of SQL supported by DocumentDB.

NoSQL Document Database

DocumentDB is Microsoft's newest NoSQL document database, however, when we say NoSQL document database, what precisely do we mean by NoSQL, and document database?

- SQL means Structured Query Language which is a traditional query language of relational databases. SQL is often equated with relational databases.

- It is really more helpful to think of a NoSQL database as a non-relational database, so NoSQL really means non-relational.

There are different types of NoSQL databases which include key value stores such as:

- Azure Table Storage
- Column-based stores, like Cassandra
- Graph databases, like NEO4
- Document databases, like MongoDB and Azure DocumentDB

Why SQL Syntax?

This can sound strange at first, but in DocumentDB which is a NoSQL database, we query using SQL. As mentioned above, this is a special version of SQL rooted in JSON and JavaScript semantics.

- SQL is just a language, but it's also a very popular language that's rich and expressive. Thus, it definitely seems like a good idea to use some dialect of SQL rather than come up with a whole new way of expressing queries that we would need to learn if you wanted to get documents out of your database.

- SQL is designed for relational databases, and DocumentDB is a non-relational document database. DocumentDB team has actually adapted the SQL syntax for the non-relational world of document databases, and this is what is meant by rooting SQL in JSON and JavaScript.

- The language still reads as familiar SQL, but the semantics are all based on schema-free JSON documents rather than relational tables. In DocumentDB, we will be working with JavaScript data types rather than SQL data types. We will be familiar with SELECT,

FROM, WHERE, and so on, but with JavaScript types, which are limited to numbers and strings, objects, arrays, Boolean, and null are far fewer than the wide range of SQL data types.

- Similarly, expressions are evaluated as JavaScript expressions rather than some form of T-SQL. For example, in a world of denormalized data, we're not dealing with the rows and columns, but schema-free documents with hierarchal structures that contain nested arrays and objects.

How does SQL Work?

The DocumentDB team has answered this question in several innovative ways. Few of them are listed as follows:

- First, assuming you've not changed the default behavior to automatically index every property in a document, you can use dotted notation in your queries to navigate a path to any property no matter how deeply nested it may be within the document.

- You can also perform an intra-document join in which nested array elements are joined with their parent element within a document in a manner very similar to the way a join is performed between two tables in the relational world.

- Your queries can return documents from the database as it is, or you can project any custom JSON shape you want based on as much or as little of the document data that you want.

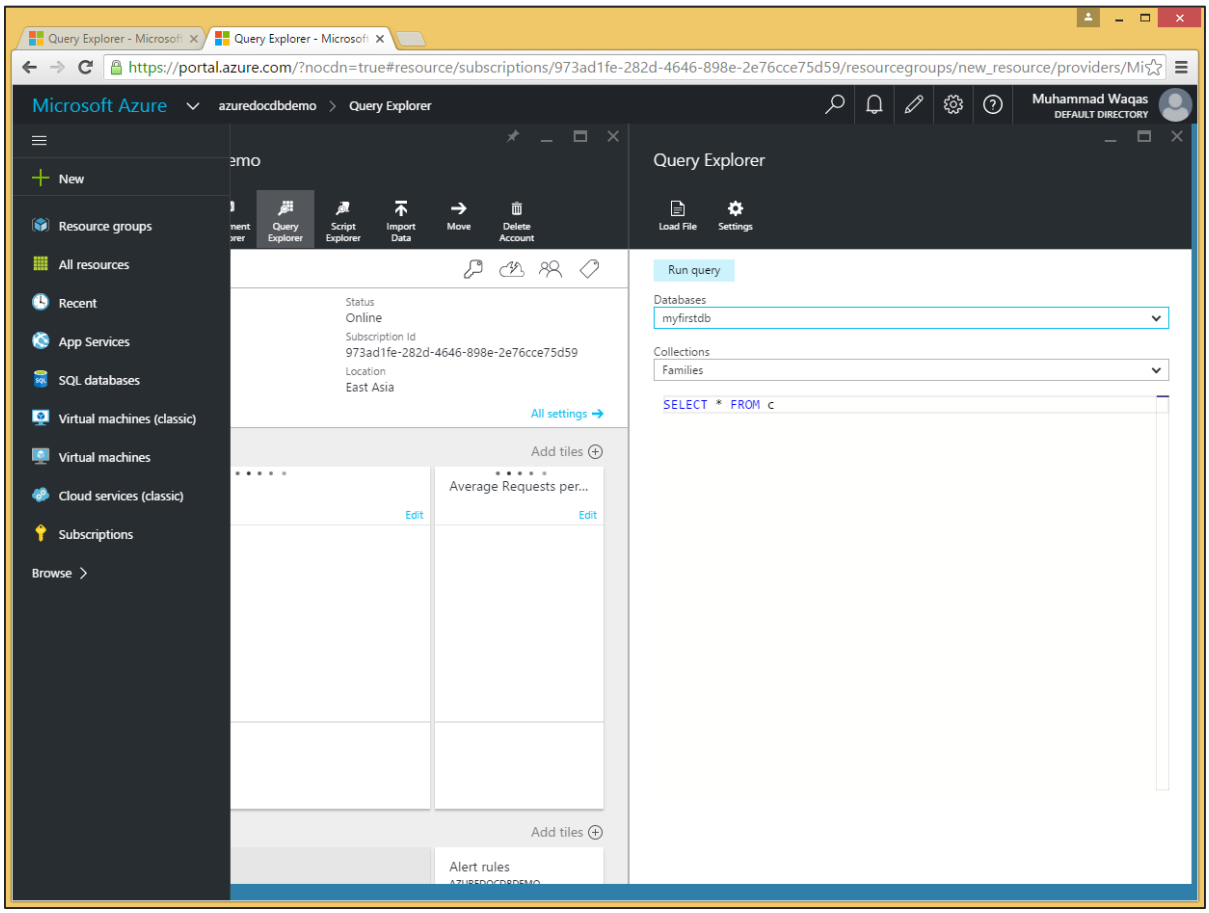
- SQL in DocumentDB supports many of the common operators including:
    - Arithmetic and bitwise operations
    - AND and OR logic
    - Equality and range comparisons
    - String concatenation

- The query language also supports a host of built-in functions.

2. DOCUMENTDB SQL – SELECT CLAUSE

The Azure portal has a Query Explorer that lets us run any SQL query against our DocumentDB database. We will use the Query Explorer to demonstrate the many different capabilities and features of the query language starting with the simplest possible query.

Step 1: Open the Azure Portal, and in the database blade, click the Query Explorer blade.



Remember that queries run within the scope of a collection, and so the Query Explorer lets us choose the collection in this dropdown. We will leave it set to our Families collection that contains the three documents. Let's consider these three documents in this example.

Following is the **AndersenFamily** document.

```
{
    "id": "AndersenFamily",
    "lastName": "Andersen",
```

```
    "parents": [
            { "firstName": "Thomas", "relationship":  "father" },
            { "firstName": "Mary Kay", "relationship":  "mother" }
    ],
    "children": [
            {
                    "firstName": "Henriette Thaulow",
                    "gender": "female",
                    "grade": 5,
                    "pets": [ { "givenName": "Fluffy", "type":  "Rabbit" } ]
            }
    ],
    "location": { "state": "WA", "county": "King", "city": "Seattle" },
    "isRegistered": true
}
```

Following is the **SmithFamily** document.

```
{
    "id": "SmithFamily",
    "parents": [
            { "familyName": "Smith", "givenName": "James" },
            { "familyName": "Curtis", "givenName": "Helen" }
    ],
    "children": [
            {
                    "givenName": "Michelle",
                    "gender": "female",
                    "grade": 1
            },
            {
                    "givenName": "John",
                    "gender": "male",
                    "grade": 7,
```

```
                "pets": [
                        { "givenName": "Tweetie", "type": "Bird" }
                ]
        }
    ],
    "location": {
            "state": "NY",
            "county": "Queens",
            "city": "Forest Hills"
    },
    "isRegistered": true
}
```

Following is the **WakefieldFamily** document.

```
{
    "id": "WakefieldFamily",
    "parents": [
            { "familyName": "Wakefield", "givenName": "Robin" },
            { "familyName": "Miller", "givenName": "Ben" }
    ],
    "children": [
            {
                "familyName": "Merriam",
                "givenName": "Jesse",
                "gender": "female",
                "grade": 6,
                "pets": [
                        { "givenName": "Charlie Brown", "type": "Dog" },
              { "givenName": "Tiger", "type": "Cat" },
              { "givenName": "Princess", "type": "Cat" }
                ]
            },
            {
```

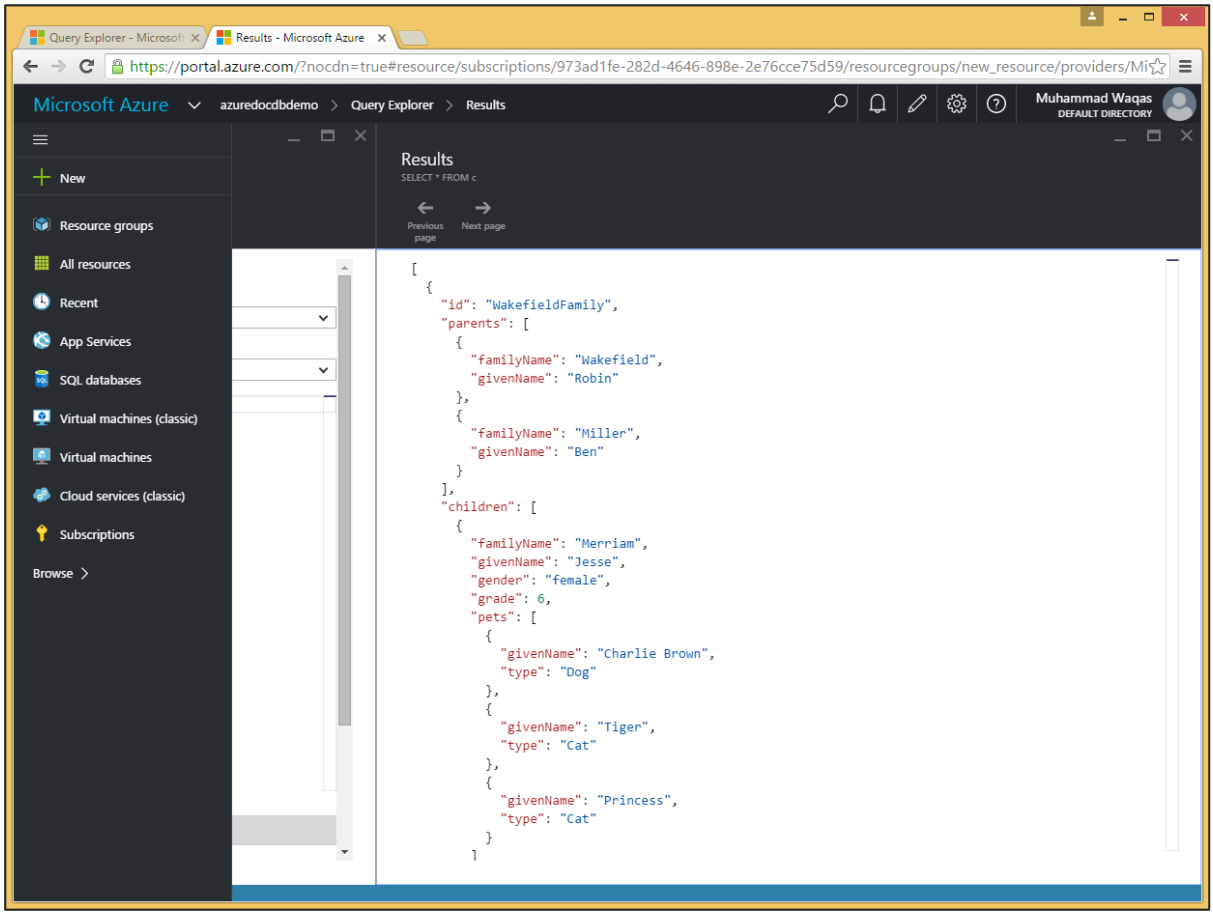
```
                "familyName": "Miller",
                "givenName": "Lisa",
                "gender": "female",
                "grade": 3,
                "pets": [
                      { "givenName": "Jake", "type": "Snake" }
                ]
          }
    ],
    "location": { "state": "NY", "county": "Manhattan", "city": "NY" },
    "isRegistered": false}
```

The Query Explorer opens up with this simple query SELECT * FROM c, which simply retrieves all documents from the collection. Although it is simple, it's still quite different than the equivalent query in a relational database.

Step 2: In relational databases, SELECT * means return all columns while in DocumentDB. It means that you want each document in your result to be returned exactly as it's stored in the database.

But when you select specific properties and expressions instead of simply issuing a SELECT *, then you are projecting a new shape that you want for each document in the result.

Step 3: Click 'Run' to execute query and open the Results blade.



As can be seen the WakefieldFamily, the SmithFamily, and the AndersonFamily are retrieved.

Following are the three documents which are retrieved as a result of the **SELECT * FROM c** query.

```
[
  {
    "id": "WakefieldFamily",
    "parents": [
      {
        "familyName": "Wakefield",
        "givenName": "Robin"
      },
```

```
    {
      "familyName": "Miller",
      "givenName": "Ben"
    }
  ],
  "children": [
    {
      "familyName": "Merriam",
      "givenName": "Jesse",
      "gender": "female",
      "grade": 6,
      "pets": [
        {
          "givenName": "Charlie Brown",
          "type": "Dog"
        },
        {
          "givenName": "Tiger",
          "type": "Cat"
        },
        {
          "givenName": "Princess",
          "type": "Cat"
        }
      ]
    },
    {
      "familyName": "Miller",
      "givenName": "Lisa",
      "gender": "female",
      "grade": 3,
      "pets": [
        {
          "givenName": "Jake",
          "type": "Snake"
```

```
        }
      ]
    }
  ],
  "location": {
    "state": "NY",
    "county": "Manhattan",
    "city": "NY"
  },
  "isRegistered": false,
  "_rid": "Ic8LAJFujgECAAAAAAAAAA==",
  "_ts": 1450541623,
  "_self": "dbs/Ic8LAA==/colls/Ic8LAJFujgE=/docs/Ic8LAJFujgECAAAAAAAAAA==/",
  "_etag": "\"00000500-0000-0000-0000-567582370000\"",
  "_attachments": "attachments/"
},
{
  "id": "SmithFamily",
  "parents": [
    {
      "familyName": "Smith",
      "givenName": "James"
    },
    {
      "familyName": "Curtis",
      "givenName": "Helen"
    }
  ],
  "children": [
    {
      "givenName": "Michelle",
      "gender": "female",
      "grade": 1
    },
    {
```

```
        "givenName": "John",
        "gender": "male",
        "grade": 7,
```

End of ebook preview
If you liked what you saw…
Buy it from our store @ **https://store.tutorialspoint**