Microsoft Azure DocumentDB Query Cheat Sheet

This cheat sheet helps you quickly write DocumentDB queries by showing some common SQL queries used to retrieve data from two simple JSON documents.

Example Family JSON Documents

1

```
{
    "id":"AndersenFamily",
    "lastName":"Andersen",
    "parents":[
        {
            "firstName":"Thomas"
        },
        {
            "firstName":"Mary Kay"
        }
    ],
    "children":[
        {
            "firstName":"Henriette Thaulow",
            "gender":"female",
            "grade":5,
            "pets":[
                {
                    "givenName":"Fluffy"
                }
            ]
        }
    ],
    "address":{
        "state":"WA",
        "county":"King",
        "city":"seattle"
    },
    "creationDate":"2015-01-03T12:00Z",
    "isRegistered":true,
    "location":{
        "type":"Point",
        "coordinates":[
            31.9,
            -4.8
        ]
    }
}
```

2

```
{
    "id":"WakefieldFamily",
    "parents":[
        {
            "familyName":"Wakefield",
            "givenName":"Robin"
        },
        {
            "familyName":"Miller",
            "givenName":"Ben"
        }
    ],
    "children":[
        {
            "familyName":"Merriam",
            "givenName":"Jesse",
            "gender":"female",
            "grade":1,
            "pets":[
                {
                    "givenName":"Goofy"
                },
                {
                    "givenName":"Shadow"
                }
            ]
        },
        {
            "familyName":"Miller",
            "givenName":"Lisa",
            "gender":"female",
            "grade":8
        }
    ],
    "address":{
        "state":"NY",
        "county":"Manhattan",
        "city":"NY"
    },
    "creationDate":"2015-07-20T12:00Z",
    "isRegistered":false
}
```

Built-in functions

Mathematical	ABS, CEILING, EXP, FLOOR, LOG, LOG10, POWER, ROUND, SIGN, SQRT, SQUARE, TRUNC, ACOS, ASIN, ATAN, ATN2, COS, COT, DEGREES, PI, RADIANS, SIN, and TAN
Type checking	IS_ARRAY, IS_BOOL, IS_NULL, IS_NUMBER, IS_OBJECT, IS_STRING, IS_DEFINED, and IS_PRIMITIVE
String	CONCAT, CONTAINS, ENDSWITH, INDEX_OF, LEFT, LENGTH, LOWER, LTRIM, REPLACE, REPLICATE, REVERSE, RIGHT, RTRIM, STARTSWITH, SUBSTRING, and UPPER
Array	ARRAY_CONCAT, ARRAY_CONTAINS, ARRAY_LENGTH, and ARRAY_SLICE
Geospatial	ST_WITHIN, ST_DISTANCE, ST_INTERSECTS, ST_ISVALID, and ST_ISVALIDDETAILED

SQL Query

```
-- Find families by ID
SELECT *
FROM Families f
WHERE f.id = "AndersenFamily"
```

```
[{
    "id": "AndersenFamily",
    "lastName": "Andersen",
    …
}]
```

SQL + JSON

```
-- Find families where City equals State and return
Name and City

SELECT {"Name":f.id, "City":f.address.city} AS Family
FROM Families f
WHERE f.address.city = f.address.state
```

```
[{
    "Family": {
        "Name": "WakefieldFamily",
        "City": "NY"
    }
}]
```

SQL + intra JOIN

```
-- Get the child names using an intra-document JOIN

SELECT c.givenName
FROM Families f
JOIN c IN f.children
WHERE f.id = 'WakefieldFamily'
ORDER BY f.address.city ASC
```

```
[
  { "givenName": "Jesse" },
  { "givenName": "Lisa"}
]
```

SQL + JavaScript UDF

```
-- Register UDF for REGEX_MATCH with this code

function (input, pattern) {
    return input.match(pattern) !== null;
}

-- Use JavaScript
SELECT udf.REGEX_MATCH(Families.address.city, ".*eattle")
```

```
[
  {
    "$1": true
  },
  {
    "$1": false
  }
]
```

Operators

Arithmetic	+, -, *, /, %
Bitwise	|, &, ^, <,>>, >>> (zero-fill right shift)
Logical	AND, OR, NOT
Comparison	=, !=, >, >=, <, <=, <>, ??
String	|| (concatenate)
Ternary	?

Query Interfaces

Server-side	SQL, JavaScript integrated query
Client-side	.NET (LINQ), Java, JavaScript, Node.js, Python

Sample Queries

Comparison (range) operators	`SELECT * FROM Families.children[0] c WHERE c.grade >= 5`
Logical operators	`SELECT * FROM Families.children[0] c WHERE c.grade >= 5 AND c.isRegistered = true`
ORDER BY keyword	`SELECT f.id, f.address.city FROM Families f ORDER BY f.address.city`
IN keyword	`SELECT * FROM Families WHERE Families.address.state IN ("NY", "WA", "CA", "PA", "OH", "OR", "MI", "WI")`
Ternary (?) and Coalesce (??) operators	`SELECT (c.grade < 5)? "elementary": ((c.grade < 9)? "junior": "high") AS gradeLevel FROM Families.children[0] c`
Escape/quoted accessor	`SELECT f["lastName"] FROM Families f WHERE f["id"] = "AndersenFamily"`
Object/Array Creation	`SELECT [f.address.city, f.address.state] AS CityState FROM Families f`
Value keyword	`SELECT VALUE "Hello World"`
Intra-document JOINS	`SELECT f.id AS familyName, c.givenName AS childGivenName, c.firstName AS childFirstName, p.givenName AS petName FROM Families f JOIN c IN f.children JOIN p IN c.pets`
Parameterized SQL	`SELECT * FROM Families f WHERE f.lastName = @lastName AND f.address.state = @addressState`
String Built-in functions	`SELECT Families.id, Families.address.city FROM Families WHERE STARTSWITH(Families.id, "Wakefield")`
Array Built-in functions	`SELECT Families.id FROM Families WHERE ARRAY_CONTAINS(Families.parents, { givenName: "Robin", familyName: "Wakefield" })`
Math Built-in functions	`SELECT VALUE ABS(-4)`
Type Built-in functions	`SELECT IS_DEFINED(f.lastName), IS_NUMBER(4) FROM Families f`
BETWEEN keyword	`SELECT * FROM Families.children[0] c WHERE c.grade BETWEEN 1 AND 5`
TOP keyword	`SELECT TOP 100 * FROM Families f`
Geospatial functions	`SELECT * FROM Families f WHERE ST_Distance(f.location, {"type":"Point", "coordinates":[31.9, -4.8]}) < 30000`