Azure Cosmos DB Query Cheat Sheet

SQL + JavaScript

Example: families collection

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
}
```

Query interfaces

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

Aggregates functions

Mathematical	COUNT, MIN, MAX, SUM, and AVG

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 + 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
  }
]
```

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

Operators

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

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`
Aggregate functions	`SELECT COUNT(1) FROM Families WHERE c.grade >= 5`
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`
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`

MongoDB API

Import to MongoDB API

Example:

Database : your-db
Collection : your-coll
File : C:\users\you\Documents*.json

In your terminal:

```
mongoimport.exe
-host your-account.documents.azure.com -port 10250
-u your-account
-p a1b2c3d4e5== --ssl
--sslAllowInvalidCertificates
--db your-db --collection your-coll --file
C:\Users\you\Documents\*.json
```

Restore to MongoDB API

Example:

Mongo dump: ./dumps/dump-2016-08-31

In your terminal:

```
mongorestore.exe
-host your-account.documents.azure.com -port 10250
-u your-account
-p a1b2c3d4e5== --ssl
--sslAllowInvalidCertificates
./dumps/dump-2016-08-31
```

Find request charge

First, run the operation you want to find the request charge for in the Mongo Shell.

Then run:

```
> db.runCommand(
{getLastRequestStatistics: 1} )
```

```
{"_t" : "GetRequestStatisticsResponse",
 "ok": 1,
 "CommandName": "OP_QUERY",
 "RequestCharge": 2.48,
 "RequestDurationInMilliSeconds":
  4.0048}
```

Example: cars collection

```
{
   "make":"Honda",
   "owner":"WakefieldFamily",
},
{
   "make":"Toyata",
   "owner":"AndersenFamily",
},
```

Example: families collection

1

```
{
   "id":"AndersenFamily",
   "lastName":"Andersen",
   "parents":[
      {
         "firstName":"Thomas"
      },
      {
         "firstName":"Mary Kay"
      }
   ],
   "children":[
      {
         "firstName":"Henriette Thaulow",
         "gender":"female",
         "grade":5,
         "pets":[
            {
               "givenName":"Fluffy"
            }
         ]
      }
   ],
  "familySize":3,
   "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
      }
   ],
   "familySize":4,
   "address":{
      "state":"NY",
      "county":"Manhattan",
      "city":"NY"
   },
   "creationDate":"2015-07-20T12:00Z",
   "isRegistered":false
}
```

Find a document by property

```
> db.families.findOne(
  {lastName: "Wakefield"},
  {id: true, _id: false} )
```

```
{"id" : "AndersenFamily"}
```

Insert a document

```
> db.families.insert(
  {id: "SmithFamily"} )
```

```
WriteResult( {"nInserted" : 1} )
```

Update a document

```
> db.families.update(
  {id: "SmithFamily"},
  {id: "SmithFamily", city: "New York"} )
```

```
WriteResult(
{"nMatched" : 1},
{"nUpserted": 0},
{"nModified": 1} )
```

Update a document property

```
> db.families.update(
  {id: "SmithFamily"},
  {$set: { city: "Seattle"}} )
)
```

```
WriteResult(
{"nMatched" : 1},
{"nUpserted": 0},
{"nModified": 1} )
```

Remove a document

```
> db.families.remove(
  {id: "SmithFamily"} )
```

```
WriteResult( {"nRemoved" : 1} )
```

Aggregate queries

Join family and cars collections to families with the cars they own.

```
> db.families.aggregate([
  {
   $lookup:
     {
       from: "cars",
       localField: "id",
       foreignField: "owner",
       as: "familycars"
     }
   }
])
```

Get the number of families with children

```
> db.families.aggregate([
  { $lookup: { "children": {$exists: true}} },
  { $group: { _id: null, totoal: {$sum: 1}} },
] );
```

 Need help or have questions? Email us at **askcosmosdb@microsoft.com** to talk to a Cosmos database engineer! www.cosmosdb.com | Tweet: @AzureCosmosDB, #cosmosdb

Microsoft

Azure Cosmos DB Query Cheat Sheet

Table API

Initialize a CloudTableClient

```
TableconnectionPolicy connectionPolicy =
new TableConnectionPolicy()
{
          UsedirectMode = true,
          UseTcpProtocol = true,
EnableEndpointdiscovery = true
};
CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(connectionString);
CloudTableClient = storageAccount.CreateCloudTable
Client(connectionPolicy, Microsoft.Azure.Cosmos-
DB.ConsistencyLevel.BoundedStaleness);
```

Create a table

```
CloudTable table = tableClient.GetTableRefer-
ence(" ");
table.CreateIfNotExists();
```

Create table entity

```
CustomerEntity item = new CustomerEntity()
{
PartitionKey = Guid.NewGuid().ToString(),
RowKey = Guid.NewGuid().ToString(),
Email = $" ",
PhoneNumber = "",
Bio = GetRandomString(1000)
};
```

Insert operation

```
TableOperation = TableOperation.Insert(item);
table.Execute(insertOperation);
```

Retrieve operation

```
TableOperation retrieveOperation = TableOpera-
tion.Retrieve<CustomerEntity>(item.PartitionKey,
item.RowKey);
table.Execute(retrieveOperation);
```

Query

```
TableQuery<CustomerEntity> rangeQuery = new
TableQuery<CustomerEntity>().Where(

TableQuery.GenerateFilterCondition("PartitionKey",
QueryComparisons.Equal, item.PartitionKey);
```

Table service REST API

Set table service properties

Method PUT
https://myaccount.table.cosmosdb.azure.com/?res-type=service&comp=properties

Request Headers
- Authorization
- Date or x-ms-date
- x-ms-version
- x-ms-client-request-id

Get table service properties

Method GET
https://myaccount.table.cosmosdb.azure.com/?restype=service&comp=properties

Request Headers
- Authorization
- Date or x-ms-date
- x-ms-version
- x-ms-client-request-id

Preflight table request

MHTTP Verb OPTIONS
http://myaccount.table.cosmosdb.azure.com/<table-resource>

Request Headers
- Origin
- Access-Control-Request-Method
- Access-Control-Request-Headers

Get table service stats

Method GET
https://myaccount.table.cosmosdb.azure.com/?restype=service&comp=stats

Request Headers
- Authorization
- Date or x-ms-date
- x-ms-version
- x-ms-client-request-id

Query table

Method GET
https://myaccount.table.cosmosdb.azure.com/Tables

Request Headers
- Authorization
- Date or x-ms-date
- x-ms-version
- x-ms-client-request-id

Create table

Method POST
https://myaccount.table.cosmosdb.azure.com/Tables

Request Headers
- Authorization
- Date or x-ms-date
- x-ms-version
- Content-Type
- Accept
- Prefer
- Content-Length
- x-ms-client-request-id

Delete table

Method DELETE
https://myaccount.table.cosmosdb.azure.com/Tables('mytable')

Request Headers
- Authorization
- Date or x-ms-date
- x-ms-version
- x-ms-client-request-id

Get table ACL

Method GET/HEAD
https://myaccount.table.cosmosdb.azure.com/mytable?com-

Request Headers
- Authorization
- Date or x-ms-date
- x-ms-version
- x-ms-client-request-id

Set table ACL

Method PUT
https://myaccount.table.cosmosdb.azure.com/mytable?comp=acl

Request Headers
- Authorization
- Date or x-ms-date
- x-ms-version
- x-ms-client-request-id

Query entities

Method GET
https://myaccount.table.cosmosdb.azure.com/mytable(PartitionKey='<partition-key>',RowKey='<row-key>')?$select=<comma-separated-property-names>
https://myaccount.table.core.windows.net/mytable()?$filter=<query-expression>&$select=<comma-separated-property-names>

Request Headers
- Authorization
- Date or x-ms-date
- x-ms-version
- Accept
- x-ms-client-request-id

Insert entity

Method POST
https://myaccount.table.cosmosdb.azure.com/mytable

Request Headers
- Authorization
- Date or x-ms-date
- x-ms-version
- Content-Type
- Content-Length
- Accept
- Prefer
- x-ms-client-request-id

Insert or Merge entity / Insert or Replace entity

Method MERGE / Method PUT
https://myaccount.table.cosmosdb.azure.com/mytable(PartitionKey='myPartitionKey',RowKey='myRowKey')

Request Headers
- Authorization
- Date or x-ms-date
- x-ms-version
- Content-Type
- Content-Length
- x-ms-client-request-id

Update entity

Method PUT
https://myaccount.table.cosmosdb.azure.com/mytable(PartitionKey='myPartitionKey',RowKey='myRowKey')

Request Headers
- Authorization
- Date or x-ms-date
- x-ms-version
- Content-Type
- Content-Length
- If-Match
- x-ms-client-request-id

Merge entity

Method MERGE
https://myaccount.table.cosmosdb.azure.com/mytable(PartitionKey='myPartitionKey',RowKey='myRowKey')

Request Headers
- Authorization
- Date or x-ms-date
- x-ms-version
- Content-Type
- Content-Length
- If-Match
- x-ms-client-request-id

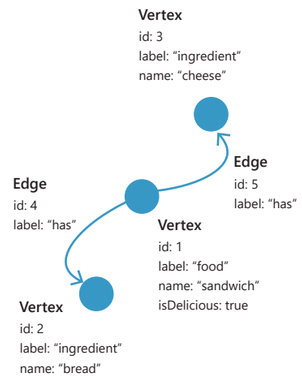
Delete entity

Method DELETE
https://myaccount.table.cosmosdb.azure.com/mytable(PartitionKey='myPartitionKey',RowKey='myRowKey')

Request Headers
- Authorization
- Date or x-ms-date
- x-ms-version
- If-Match
- x-ms-client-request-id

Gremlin / Graph API

Anatomy of Tinkerpop Graph



Graph object types

Vertices (also known as nodes): Represent objects. They have unique ID, a unique label and one ore more properties.
Edges: Represent relationships between objects (vertices). They have unique Id, a unique label and one or more properties.

Graph object attributes

ID: A property that uniquely identifies each element (vertex or edge) of a graph.
Labels: A unique string element that can be used to identify types of objects (vertices or edges). This is a required field for any grapph object.
Property: A key-value pair that can contain strings, boolean or number values.

Gremlin language primer

Try this queries with the Gremlin console or the Data Explorer on Azure Portal.

Traversal variable

Every query starts with *g*, the variable that represents the graph traversal object. For example, this query will get all the vertices in a graph:

```
:> g.V()
```

Add the vertices

Use the .addV("label") step, followed by any number of calls to the .property("key", "value") step.

Example:

```
:> g.addV("food")
        .property("name", "sandwich")
        .property("isDelicious", true)

:> g.addV("ingredient")
        .property("name", "cheese")

:> g.addV("ingredient")
        .property("name", “bread")
```

Select a vertex by property

Start with the `g.v() selector`, and filter using the `.has ("key", "value")` with any exisiting property as the key.

Example:

```
:> g.V().has("name", "sandwich")
```

Add an edge between two vertices

After selecting a vertex, use the `.addE("label")` step, followed by any number of properties, then use the `.to(g.V() …)` to select another vertex to point it to.

Example:

```
:> g.V().has("name", "sandwich")
        .addE("has")
        .property("amount", 2)
        .to(g.V().has("name", "bread"))

:> g.V().has("name", "sandwich")
        .addE("has")
        .property("amount", 1)
        .to(g.V().has("name", “cheese"))
```

Traverse a graph

After selecting a vertex, use the `.out("label")` step to select all adjacent vertices that are connected with edges that have a specific label.

Example:

```
<- Input
:> g.V().has("name", "sandwich")
        .out("has")
        .properties("name")

Output ->
:> [cheese, bread]
```

Loop a traversal

Use the `.repeat(out("label"))` step, followed by .until() with a condition.

Example:

```
<- Input
:> g.V().has("name", "sandwich")
        .repeat(out("has"))
        .until(hasLabel("ingredient"))
        .path().by("name")
Output ->
:> [sandwich, bread], [sandwich, cheese],
```

You can also use the `.times()` step and provide a number of times it should be repeated:

Example:

```
<- Input
:> g.V().has("name", "sandwich")
        .repeat(out("has"))
        .times(1)
        .path().by("name")
Output ->
:> [sandwich, bread], [sandwich, cheese]
```

Count number of vertices

Use the `.count()` step at the end of a traversal.

Example:

```
<- Input
:> g.V().out().hasLabel(“ingredient")
        .count()

Output ->
:> [2]
```

Return the path of a traversal

Use the .path() step at the end of a traversal, use the `.by()` step to return values for a specific field.

Example:

```
<- Input
:> g.V().out()
        .path()
        .by("name")

Output ->
:> ["sandwich", "bread"], ["sandwich", “cheese"]
```

Select a specific element from a traversal

Use the `.as()` step to name an object, and the `.select()` step to return it at the end of a traversal. After that, you can use the `.by()` step to get their values.

Example:

```
<- Input
:> g.V().outE().as("e").inV()
        .select("e")
        .by("amount")

Output ->
:> [2, 1]
```

Drop a vertex or edge

After selecting a vertex or edge, append the `.drop()` step at the end.

Example:

```
:> g.V().has("name", "cheese").drop()
```

Drop an entire graph

This command will drop all vertices and edges:

```
:> g.V().drop()
```

 Need help or have questions? Email us at **askcosmosdb@microsoft.com** to talk to a Cosmos database engineer! www.cosmosdb.com | Tweet: @AzureCosmosDB, #cosmosdb

Microsoft