Azure Cosmos DB Query Cheat Sheet

SQL + JavaScript

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
}
```

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

Get account information

Head to https://portal.azure.com to find your account's connection information.

Example:

Account : your-account
Port : 10250
Username: your-account
Password : a1b2c3d4e5==
SSL : true

Connect with Mongo Shell

In your terminal:

```
mongo.exe
your-account.documents.azure.com:10250
-u your-account -p a1b2c3d4e5==
-ssl -sslAllowInvalidCertificates
```

Import to API for MongoDB

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 API for MongoDB

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 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"
            }
         ]
      }
   ],
   "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's id

```
> db.collection.findOne( {},
  {id: true, _id: false} )

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

Find a document by property

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

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

Find *all* documents' ids

```
> db.collection.find ( {},
  {id: true, _id: false} )

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

Insert a document

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

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

Update a document

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

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

Update a document property

```
> db.collection.update(
  {id: "SmithFamily"},
  {$set: { city: "Seattle"}} )
)
WriteResult(
{"nMatched" : 1},
{"nUpserted": 0},
{"nModified": 1} )
```

Comparison query

```
> db.collection.find(
  {familySize: { $gt: 3}},
  {id: true, _id: false})

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

Remove a document

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

WriteResult( {"nRemoved" : 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

```
CloudStorageAccount storageAccount = CloudStorage-
Account.Parse(connectionString);
CloudTableClient tableClient = storageAccount.Cre-
ateCloudTableClient();
```

Create a table

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

Create table container

```
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);

items.Add(item);
```

Retrieve operation

```
TableOperation retrieveOperation = TableOpera-
tion.Retrieve<CustomerEntity>(items[i].Partition-
Key, items[i].RowKey);
table.Execute(retrieveOperation);
```

Query using index

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

TableQuery.GenerateFilterCondition("xx",
QueryComparisons.Equal, items[i].xx));
```

Instantiating the CloudTableClient via AppSetting

`TableConnectionMode (Gateway,Direct)` -
We recommend Direct, the default, for best latency/throughput

`TableConnectionProtocol(Https,Tcp)` -
We recommend Tcp, the default, for best latency/throughput

`TablePreferredLocations` -
Array of Azure regions. You can configure Azure CosmosDB accounts with 1-30+ regions and configure the SDK for multi-homing

`TableConsistencyLevel (Strong, BoundedStaleness, ConsistentPrefix, Session, Eventual)` -
This allows you to tradeoff latency/availability/and consistency in multi-region configuration

Table service REST API

Set table service properties

Method PUT

https://<account-name>.table.core.windows.net/?restype=service&comp=properties

Request Headers

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

Table service REST API (cont...)

Get table service properties

Method GET

https://<account-name>.table.core.windows.net/?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://<account-name>.table.core.windows.net/<table-resource>

Request Headers

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

Get table service stats

Method GET

https://myaccount-secondary.table.core.windows.net/?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.core.windows.net/Tables

Request Headers

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

Create table

Method POST

https://myaccount.table.core.windows.net/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.core.windows.net/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.core.windows.net/mytable?comp=acl

Request Headers

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

Set table ACL

Method PUT

https://myaccount.table.core.windows.net/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.core.windows.net/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.core.windows.net/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.core.windows.net/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.core.windows.net/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.core.windows.net/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.core.windows.net/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

Example GraphSON

```
{
   "id":"a7111ba7-oea1-43c9-b6b2-efc5e3ae4c0",
   "label":"person",
   "type": "vertex",
   "outE": {
     "knows": [
       {
         "id":"3ee53a60-c561-4c5e-9a9f-9c7924bc9aef",
         "inv":"04779300-1c8e-489d-9493-50fd1325a658"
       },
       {
         "id":"21984248-ee9e-43a8-a7f6-30642bc14609",
         "inv":"a8e3e741-2ef7-4c01-b7c8-199f8e43e3bc"
       }
     ]
   },
  "properties": {
     "firstName": [
       {
         "value":"Thomas",
       }
      ],
   "lastName": [
       {
         "value":"Anderson",
       }
      ],
    "age": [
       {
         "value":45,
       }
      ]
   }
}
```

Gremlin features

Graph Features: Provides Persistence and ConcurrentAccess in preview

Variable Features: Boolean / Integer / Byte / Double / Float / Long / String

Vertex Features: Supports RemoveVertices, MetaProperties, AddVertices, MultiProperties, StringIds, UserSuppliedIds, AddProperty, RemoveProperty

Vertex Property Features: StringIds, UserSuppliedIds, AddProperty, RemoveProperty, BooleanValues, ByteValues, DoubleValues, FloatValues, IntegerValues, LongValues, StringValues

Edge Features: AddEges, RemoveEdges, StringIds, UserSuppliedIds, AddProperty, RemoveProperty

Properties use by GraphSON

Id: ID for Vertex, Must be unique

Label: Label of the Vertex

Type: Distinguish Vertices from Non Graphs

Properties: Bag of User Defined Properties associated with vertex

Partition(Configurable): The partition key of the vertex, used for scaling out graphs to multiple servers

outE: List of out edges from a vertex. Fast execution of traversal if adjacency info is stored.

Connect to App Service

Modify your remote-secure.yaml

Example:

```
Hosts       : ***.graphs.azure.com
Port        : 443
Username    : your-account
Password    : a1b2c3d4e5 ==
SSL         : true
Serializer  : { className:org.apache.tinkerpop.gremlin.
driver.ser.GraphSONMessageSerializerV1d0, config: {
serializeResultToString: true }}
```

Connect with Gremlin console

In your terminal

`bin/gremlin.bat` or `bin/gremlin.sh`

```
:remote connect tinkerpop.server conf/remote-se-
```

Create vertices

Create the first vertext

Example:

```
<- Input
:> g.addV('person').property('firstName',
'Thomas').property('lastName', 'Andersen').proper-
ty('age', 44).property('userid', 1)
-> Output
==>[id:796c,label:person,type:vertex,properties:[-
firstNam-
e:[[id:f02a73,value:Thomas]],lastName:[[id:f5fa,val
ue:Andersen]],age:[[id:f639,value:44]],userid:[[id:
796|userid,value:1]]]]
```

Create the second vertex

Example:

```
<- Input
:> g.addV('person').property('firstName', 'Mary
Kay').property('lastName', 'Andersen').proper-
ty('age', 39).property('userid', 2)
-> Output
==>[id:0ac9b,label:person,type:vertex,properties:[-
firstName:[[id:ea060,value:Mary Kay]],lastNam-
e:[[id:86d3,value:Andersen]],age:[[id:bc81,val-
ue:39]],userid:[[id:0ac9|userid,value:2]]]]
```

Add edges

Add relationships between people

Example:

```
<- Input
:> g.V().hasLabel('person').has('firstName',
'Thomas').addE('knows').to(g.V().hasLabel('per-
son').has('firstName', 'Mary Kay'))
-> Output
==>[id:c12b,label:knows,type:edge,inVLabel:per-
son,outVLabel:person,inV:0d1fa428,outV:1ce821c6]
```

Update a vertex

Example:

```
<- Input
:> g.V().hasLabel('person').has('firstName',
'Thomas').addE('knows').to(g.V().hasLabel('per-
son').has('firstName', 'Mary Kay'))
-> Output
==>[id:c12b,label:knows,type:edge,inVLabel:per-
son,outVLabel:person,inV:0d1fa428,outV:1ce821c6]
```

Query your graph

Example:

```
<- Input (filter query)
:> g.V().hasLabel('person').has('age',
gt(40)).value()
-> Output
==>[label:person,type:vertex,properties:[-
firstName:[[value:Th-
omas]],lastName:[[value:Andersen]],age:[[val
ue:45]]]]
```

Traverse your graph

Example:

```
<- Input(friends of Thomas)
:> :> g.V().hasLabel('person').has('first-
Name', 'Thomas').outE('knows').inV().hasLa-
bel('person').value()
-> Output
==>[label:person,type:vertex,properties:[-
firstName:[[value:Mary Kay]],lastName:[[val-
ue:Andersen]],age:[[value:39]]]]
```

Drop a vertex

Delete a vertex from graph

Example:

```
<- Input (drop Jack vertex)
:> g.V().hasLabel('person').has('firstName',
'Jack').drop()
```

Clear your graph

Clear database of all vertices and edges

Example:

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