QUERYING WITH N1QL

This cheatsheet uses the dataset found in the online N1QL tutorial at http://query.couchbase.com

Basics
SELECT Statement

```
SELECT count(*), state
FROM customer
WHERE customer.ccInfo.cardType="discover"
GROUP BY customer.state
ORDER BY customer.state
LIMIT 5 OFFSET 5
```

This query counts the number of customer per state who have a Discover credit card. The result set is grouped by state and ordered by state. Output is limited to 5 documents, after skipping the first 5.

SIMPLE ARITHMETIC
Normalized, rounded, and truncated ratings

```
SELECT
AVG(reviews.rating) / 5 as normalizedRating,
ROUND((avg(reviews.rating) / 5), 2) as
roundedRating,
TRUNC((avg(reviews.rating) / 5), 3) as truncRating
FROM reviews AS reviews
WHERE reviews.customerId = "customer62"
```

With N1QL, you can use the +, -, * and / operators. N1QL has functions for rounding (ROUND) and truncation (TRUNC). See the result set for this query below -

```
"results": [
  {
    "normalizedRating": 0.65,
    "roundedRating": 0.65,
    "truncRating": 0.65
  }
]
```

Try using other aggregation functions like SUM, MIN, and MAX.

STRING CONCATENATION AND MATCHING

The II operator concatenates the first and last names to form the full name. The LIKE operator filters customers who have email addresses ending with .biz.

```
SELECT firstName || " " || lastName as fullName
FROM customer
WHERE emailAddress LIKE "%.biz"
 "resultset": [
 {

     "fullName": "Joyce Murazik"

  }, ….
```

N1QL also provides string functions such as **lower**, **upper**, **substr**, and **length**.

DISTINCT

Sometimes you may want to remove duplicate results. For this, we have the DISTINCT keyword.

To count the number of unique customers who have purchased something –

```
SELECT COUNT( DISTINCT customerId )
FROM purchases
```

NULL AND MISSING VALUES

JSON documents can contain NULL values or omit fields entirely. The NULL/MISSING operators let you test for these conditions.

```
SELECT fname, children
FROM tutorial
WHERE children IS NULL
```

Now, try changing IS NULL to IS MISSING.

EXPLAIN

By using EXPLAIN, you can understand how the statement would have operated if the EXPLAIN keyword was omitted.

EXPLAIN <Query Statement>

Try explaining the SELECT statement on page 1.

Data Structures
ARRAYS AND OBJECTS

N1QL supports nested JSON objects where you can use the dot "." operator to access fields nested inside other objects as well as the bracket [index] to access elements inside an array.

For example, consider the following object:

{ "address" : { "city": "Toronto"}, "revision": [2014] }

address.city will return "Toronto" and revision[0] will return 2014.

In addition, these are some of the functions on arrays –

```
array_length(<array>)
array_prepend(<value>, <array>)
array_append(<array>, <value>)
array_concat(<array1>, <array2>)
```

COLLECTION EXPRESSIONS
A collection in N1QL is an array-valued subpath or expression. Collection predicates allow you to test a boolean condition over the elements of a collection.

The ANY operator allows you to search through an array, returning TRUE when at least one match is found. With the EVERY operator, every single elements needs to match.

To search for purchase orders with a particular item purchased 5 times or more –

SELECT *
FROM purchases
WHERE ANY item **IN** purchases.lineItems **SATISFIES**
item.count >= 5 **END**

Try changing ANY to EVERY.

ARRAY and FIRST
To map and filter elements of a collection, you can use the ARRAY and FIRST operators.

To get an array of products for each purchase order -

SELECT
ARRAY item.product **FOR** item **IN**
purchases.lineItems **END** AS product_ids
FROM purchases

Changing ARRAY to FIRST will produce the first product in each purchase order.

Joins
JOIN, NEST, and UNNEST

A JOIN in N1QL is similar to SQL; a single result is produced for each matching left and right-hand input.

NEST produces a single result for each left-hand input, while the right-hand input is collected and nested into a single array-valued field in the result.

To assemble a complete list of products purchased by a customer –

SELECT c, pr
FROM purchases pu
JOIN customer c ON KEYS pu.customerId
NEST product pr
ON KEYS ARRAY li.product FOR li IN pu.lineItems
END
WHERE pu.customerId = "customer1"

The UNNEST clause allows you to take contents of a nested array and join them with the parent object.

To list products belonging to a particular category -

SELECT p
FROM product p
UNNEST p.categories as category
WHERE category = "Appliances"