Unit 9

Faceted Search

Return and Process Facet Data

Customize Facet Response Data

Create a Second Facet

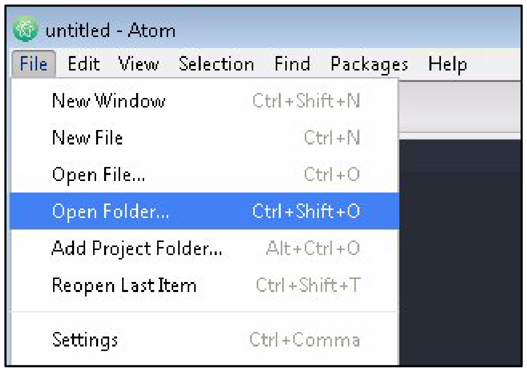
DIY: Create a Bucketed Constraint

Exercise 1: Return and Process Facet Data

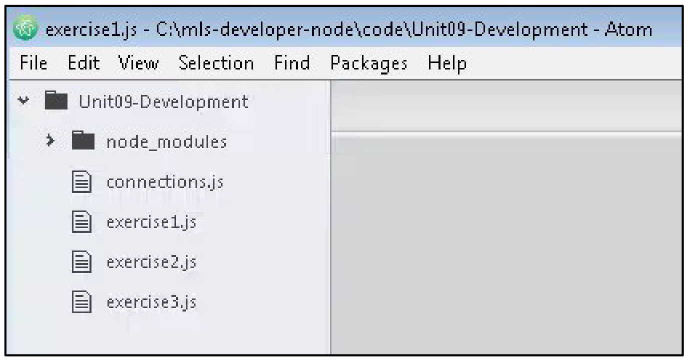
In this exercise you will implement a search that will bring back facet data. You will study the facet data that comes back in the response and work with it to output facet values.

1. From the command line, navigate to the **c:\mls-developer-node\code\Unit09-Development** directory and run **npm install marklogic**.
2. Open the Atom editor from your Desktop:

3. In the Atom editor select File→Open Folder…:

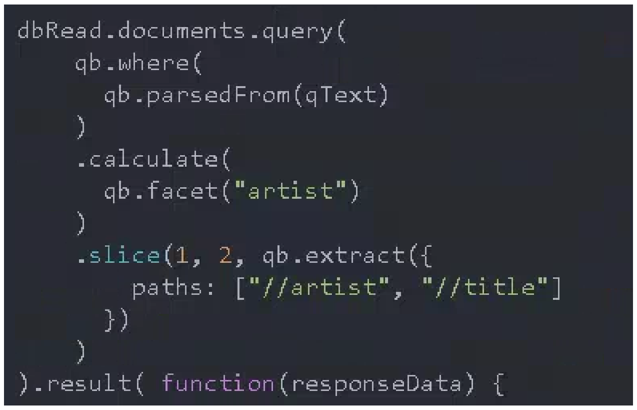

untitled - Atom
File Edit View Selection Find Packages Help
New Window Ctrl+Shift+N
New File Ctrl+N
Open File... Ctrl+O
Open Folder... Ctrl+Shift+O
Add Project Folder... Alt+Ctrl+O
Reopen Last Item Ctrl+Shift+T
Settings Ctrl+Comma

4. Open the **Unit09-Development** folder from c**:\mls-developer-node\code\**
5. You should see the following structure in place in your editor:


exercise1.js - C:\mls-developer-node\code\Unit09-Development - Atom
File Edit View Selection Find Packages Help
Unit09-Development
node_modules
connections.js
exercise1.js
exercise2.js
exercise3.js

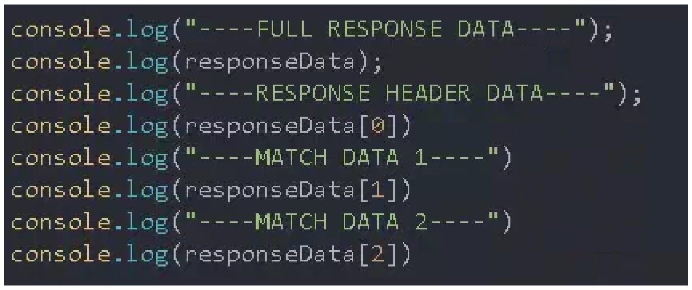
6. Select **exercise1.js**.

7. Take a moment to study the comments and code.
8. Note the query that specified to **calculate** facets for the artist property. Remember this must have a range index backing it up in order to work:



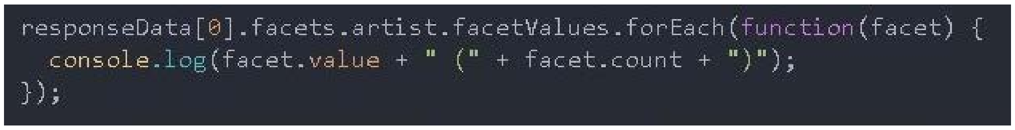
```
dbRead.documents.query(
    qb.where(
      qb.parsedFrom(qText)
    )
    .calculate(
      qb.facet("artist")
    )
    .slice(1, 2, qb.extract({
        paths: ["//artist", "//title"]
      })
    )
).result( function(responseData) {
```

9. Note that we are returning the full response data, as well as each item in the array of response data. This is simply to help you get more familiar with the response data that results from these queries:



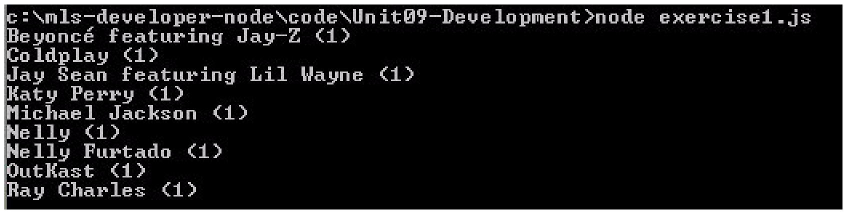
```
console.log("----FULL RESPONSE DATA----");
console.log(responseData);
console.log("----RESPONSE HEADER DATA----");
console.log(responseData[0])
console.log("----MATCH DATA 1----")
console.log(responseData[1])
console.log("----MATCH DATA 2----")
console.log(responseData[2])
```

10. Now let’s test the code.
11. At the command line, go to the **c:\mls-developer-node\Unit09-Development** directory.
12. Enter **node exercise1.js** and press enter.
13. Take some time to studythe response data.
14. Next, let’s output the facet counts.
15. Comment out all the existing **console.log** statements. Uncomment out the following code:



```
responseData[0].facets.artist.facetValues.forEach(function(facet) {
  console.log(facet.value + " (" + facet.count + ")");
});
```

16. This code will work with the response data to output the facet value and the count of documents with that value. Remember that this information is coming directly out of range index that you configured on the artist property.
17. Save your code.
18. Now let's test the code.
19. At the command line, go to the **c:\mls-developer-node\Unit09-Development** directory.
20. Enter **node exercise1.js** and press enter.
21. View the response data:

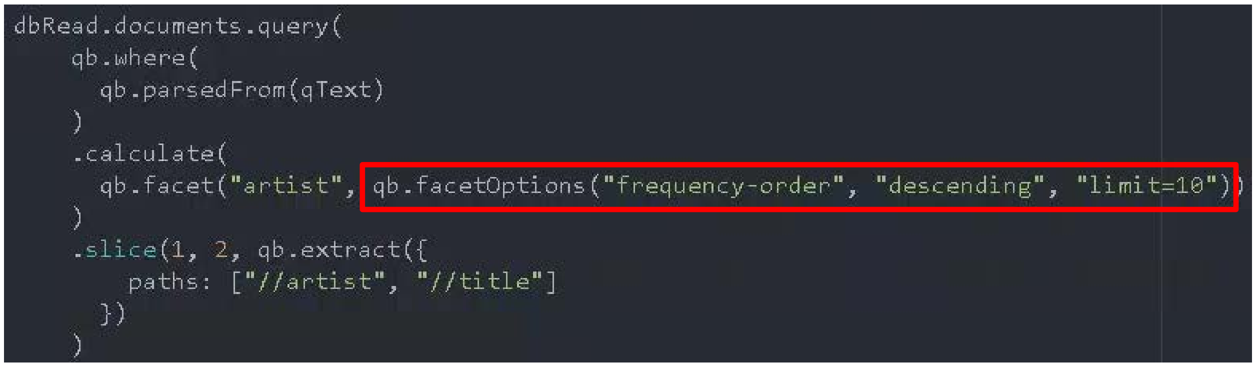


```
c:\mls-developer-node\code\Unit09-Development>node exercise1.js
Beyoncé featuring Jay-Z (1)
Coldplay (1)
Jay Sean featuring Lil Wayne (1)
Katy Perry (1)
Michael Jackson (1)
Nelly (1)
Nelly Furtado (1)
OutKast (1)
Ray Charles (1)
```

Exercise 2: Create a Second Facet

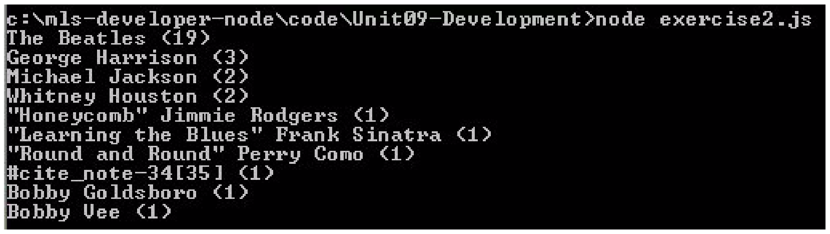
In this exercise you will implement a search and control how the facet data is brought back to you in the response. You will study the facet data that comes back in the response and work with it to output facet values.

1. In your editor, within the **Unit09-Development** project, select **exercise2.js**.
2. Study the comments and code.
3. Note the use of the query builder to define facet options:



```
dbRead.documents.query(
    qb.where(
      qb.parsedFrom(qText)
    )
    .calculate(
      qb.facet("artist", qb.facetOptions("frequency-order", "descending", "limit=10"))
    )
    .slice(1, 2, qb.extract({
        paths: ["//artist", "//title"]
      })
    )
```

1. Now let's test the code.
2. At the command line, go to the **c:\mls-developer-node\Unit09-Development** directory.
3. Enter **node exercise2.js** and press enter.
4. View the response data:

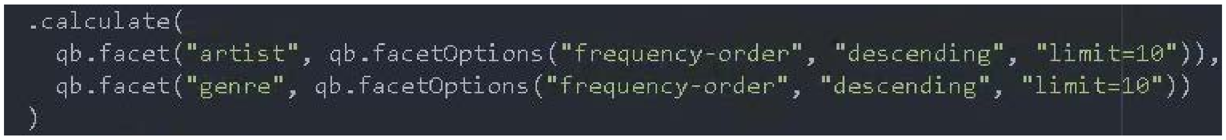


```
c:\mls-developer-node\code\Unit09-Development>node exercise2.js
The Beatles (19)
George Harrison (3)
Michael Jackson (2)
Whitney Houston (2)
"Honeycomb" Jimmie Rodgers (1)
"Learning the Blues" Frank Sinatra (1)
"Round and Round" Perry Como (1)
#cite_note-34[35] (1)
Bobby Goldsboro (1)
Bobby Vee (1)
```

Exercise 3: Customize Facet Response Data

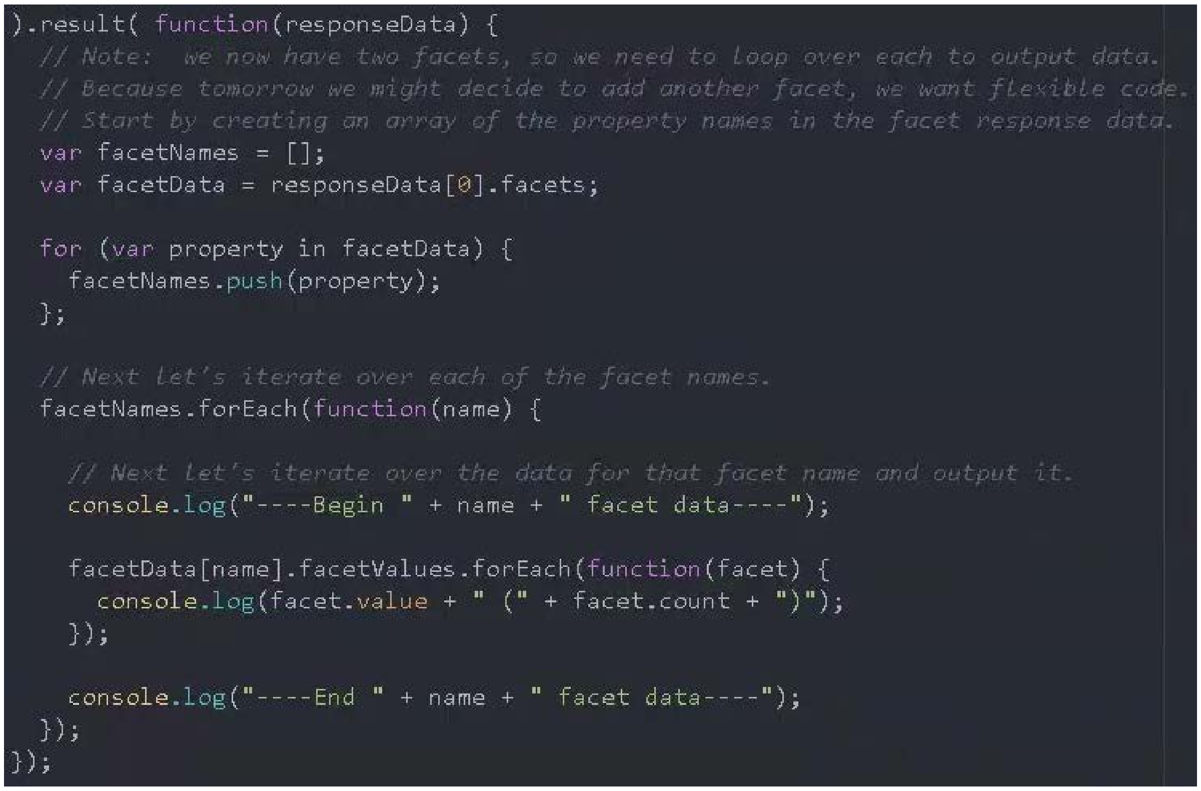
In this exercise you will implement a second facet and output facet values for both.

1. In your editor, within the **Unit09-Development** project, select **exercise3.js**.
2. Study the comments and code.
3. Note that to output multiple facets, we simply tell the API to calculate multiple facets and then we will have the data to iterate over in the response. Remember that this second facet on the genre property is also backed by a range index that you setup earlier:



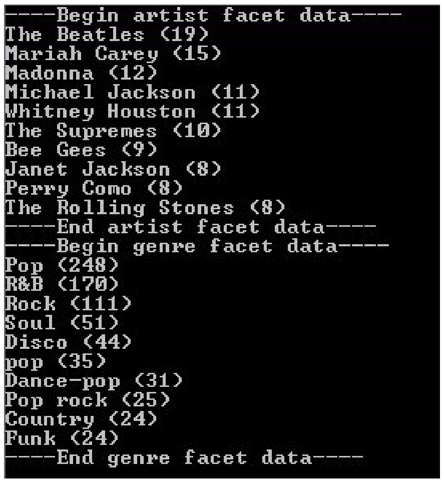
```
.calculate(
  qb.facet("artist", qb.facetOptions("frequency-order", "descending", "limit=10")),
  qb.facet("genre", qb.facetOptions("frequency-order", "descending", "limit=10"))
)
```

4. Note that now that there will be two facets returned, you will need to do another iteration in processing the response data. And because tomorrow you may wish to add another facet, we want the code to be flexible and deal with the fact there may be 3, 4, 5, etc. facets returned in the result:



```
).result( function(responseData) {
  // Note:  we now have two facets, so we need to loop over each to output data.
  // Because tomorrow we might decide to add another facet, we want flexible code.
  // Start by creating an array of the property names in the facet response data.
  var facetNames = [];
  var facetData = responseData[0].facets;

  for (var property in facetData) {
    facetNames.push(property);
  };

  // Next let's iterate over each of the facet names.
  facetNames.forEach(function(name) {

    // Next let's iterate over the data for that facet name and output it.
    console.log("----Begin " + name + " facet data----");

    facetData[name].facetValues.forEach(function(facet) {
      console.log(facet.value + " (" + facet.count + ")");
    });

    console.log("----End " + name + " facet data----");
  });
});
```

5. Now let’s test the code.
6. At the command line, go to the **c:\mls-developer-node\Unit09-Development** directory.
7. Enter **node exercise3.js** and press enter.
8. View the response data:



```
----Begin artist facet data----
The Beatles (19)
Mariah Carey (15)
Madonna (12)
Michael Jackson (11)
Whitney Houston (11)
The Supremes (10)
Bee Gees (9)
Janet Jackson (8)
Perry Como (8)
The Rolling Stones (8)
----End artist facet data----
----Begin genre facet data----
Pop (248)
R&B (170)
Rock (111)
Soul (51)
Disco (44)
pop (35)
Dance-pop (31)
Pop rock (25)
Country (24)
Funk (24)
----End genre facet data----
```

DIY: Create a Bucketed Constraint

In this exercise you will work to implement a bucketed constraint on the height range index that you created earlier.

The objective is to determine the count of characters that fall between various height ranges. Specifically:

1. Characters greater than 0 meters and less than or equal to 1 meter.
2. Characters greater than 1 meters and less than or equal to 2 meters.
3. Characters greater than 2 meters and less than or equal to 3 meters.
4. Characters greater than 3 meters.
5. Use some of the examples from the discussion and exercises as a pattern if desired.

Note:
For more information on bucketed constraints, please reference:

http://docs.marklogic.com/guide/node-dev/search#id_94740