Getting Started with PouchDB - Part 4

In the last three blog posts, you created a PouchDB database and modified documents within it. You learned to search for documents within the database using allDocs() and find(). In this fourth part of our ongoing series on PouchDB, you learn to use map queries using the query() method.

Map Queries

Most of the queries you need to perform can be accomplished using allDocs() or the find() plug-in. However, if you need to do something fairly complex, you can take advantage of the query() method. This method allows you to pass in a map function as the first parameter. In the map function you write any logic you want, to determine which documents to return.

Think of the allDocs() and find() functions as very simple WHERE clauses in a traditional relational database. Think of the query() function as a method which allows you to do very fancy WHERE clauses. As you know, simple WHERE clauses probably make up 95% of your SQL statements. The same is the case in NoSQL databases. Always try to use allDocs() and find() before you turn to using the query() method.

The query() Method Definition

The query() method takes up to 3 parameters:

```
db.query(function | "design doc name", [options], [callback])
```

The first parameter is either an inline map function, the name of a map function defined elsewhere in your JavaScript, or a string with the name of a design document stored in your database. Design documents are explained later in this blog post. The *options* object has many of the same properties as what you used when calling the allDocs() method. The last parameter you probably won't use as you should be using promises and not callbacks. However, this last parameter can

come in handy if you needed to further refine the data returned from the query as that data is passed to the callback function.

The map function you pass in calls the emit() function to return a key/value pair. The map function is called once for each document in your database. Pass to the first parameter of the emit() function, a valid property name in a document you wish to emit as the "key". Pass to the second parameter, a valid property name(s) in a document you wish to emit at the "value". An example of an inline function you might supply to the query() function looks like the following.

```
function (doc, emit) {
  emit(key field, value field);
}
```

Temporary Views

When you pass in a function, or a function name to the query() method, you are creating a temporary view. This means a full document scan is performed and a temporary index is created with the resulting data. The data is returned by using this index, and then this index is thrown away. It is not recommended that you use temporary views in a production application. They are useful, however, for you to try things out during development. All the techniques you are going to learn now about temporary views are the same as for when you create persistent views in a design document.

Using the query() Method

In the code below, pass to the query() function, a function with two parameters; *doc* and *emit*. The *doc* parameter is filled in with the document each time it is called, the emit is a callback function passed in from the query() method. To ensure you only get those documents that have people in them, add an if statement to check for a *lastName* property. If you find a document that has a lastName property, emit the *doc._id* value as the "key", and concatenate the *firstName*, a space, and the *lastName* values as the "value".

```
function simpleQuery() {
  db.query(function (doc, emit) {
    // Only get documents with lastName property
    if (doc.lastName) {
      // Emit _id as key, first name + last name as the value
      emit(doc._id, doc.firstName + ' ' + doc.lastName);
    }
  }).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

The output from the above query looks like the following:

```
{
  "total_rows": 5,
  "offset": 0,
  "rows": [
    {
      "key": "bjones",
      "id": "bjones",
      "value": "Bruce Jones"
    },
    {
      "key": "jkuhn",
      "id": "jkuhn",
      "value": "John Kuhn"
    },
    {
      "key": "mshane",
      "id": "mshane",
      "value": "Molly Shane"
    },
    {
      "key": "msheriff",
      "id": "msheriff",
      "value": "Madison Sheriff"
    },
    {
      "key": "psheriff",
      "id": "psheriff",
      "value": "Paul Sheriff"
    }
  ]
}
```

NOTE: If you eliminate the if statement to check for the *lastName* property, you get back all the 'service' documents, in addition to the 'technician' documents, with their *value* property set to 'undefined undefined'.

A Key, But No Value

If you just want to get a key and not a value, pass one parameter to the emit() function. The *value* property will be null when the result is output.

```
function simpleQueryNoValue() {
  db.query(function (doc, emit) {
    if (doc.lastName) {
      // Emit first name + last name as the key, no value
      emit(doc.firstName + ' ' + doc.lastName);
    }
  }).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

The output from the above query looks like the following:

```
{
  "total_rows": 5,
  "offset": 0,
  "rows": [
    {
      "key": "Bruce Jones",
      "id": "bjones",
      "value": null
    },
    {
      "key": "John Kuhn",
      "id": "jkuhn",
      "value": null
    },
    {
      "key": "Madison Sheriff",
      "id": "msheriff",
      "value": null
    },
    {
      "key": "Molly Shane",
      "id": "mshane",
      "value": null
    },
    {
      "key": "Paul Sheriff",
      "id": "psheriff",
      "value": null
    }
  ]
}
```

NOTE: You may also pass a null as the first parameter to have a null key. But, doing so is of limited value, so you probably won't do this.

Return Multiple Values as Array

Instead of just returning a single field value as the *value* property, you may return an array. Just create an array as the second parameter to the emit() function and pass in a comma-separated list of property names.

```
function simpleQueryMultipleFields() {
  pouchDBSamplesCommon.hideMessageAreas();
  db.query(function (doc, emit) {
    if (doc.lastName) {
      // Emit _id as key, and two values; last name and first name
      emit(doc._id, [doc.lastName, doc.firstName]);
    }
  }).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

The output from the above query looks like the following:

```
{
  "total_rows": 5,
  "offset": 0,
  "rows": [
    {
      "key": "bjones",
      "id": "bjones",
      "value": [
        "Jones",
        "Bruce"
      ]
    },
    {
      "key": "jkuhn",
      "id": "jkuhn",
      "value": [
        "Kuhn",
        "John"
      ]
    },
    // MORE DOCS HERE
  ]
}
```

Return Multiple Values as Object

Instead of just returning a single field value as the *value* property, you may return an object. Create a JSON object as the second parameter to the emit() function as shown in the code below.

```
function simpleQueryWithObject() {
  pouchDBSamplesCommon.hideMessageAreas();
  db.query(function (doc, emit) {
    if (doc.lastName) {
      // Emit _id as key, object as value
      emit(doc._id, {
        "lname": doc.lastName,
        "fname": doc.firstName
      });
    }
  }).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

The output from the above query looks like the following:

```
{
  "total_rows": 5,
  "offset": 0,
  "rows": [
    {
      "key": "bjones",
      "id": "bjones",
      "value": {
        "lname": "Jones",
        "fname": "Bruce"
      }
    },
    {
      "key": "jkuhn",
      "id": "jkuhn",
      "value": {
        "lname": "Kuhn",
        "fname": "John"
      }
    },
    // MORE DOCS HERE
  ]
}
```

NOTE: You may also specify an array or object for the first parameter of the emit() function to return an array or object for the *key* property too.

Create and Use an External Function

You don't have to pass an inline function to the query() method. You can create a function that accepts a document and the emit callback as shown below. **NOTE**: You do not have to accept the *emit* parameter in this function.

```
function getServices(doc, emit) {
  if (doc.docType === 'service') {
    emit(doc._id, doc.cost);
  }
}
```

When you call the query() method, you now just pass the name of the function, **getServices**, as the first parameter to the query() method. In the function below, you are also including the second parameter to the query() method which is the *options* object. Specifying *include_docs: true* returns the complete document information just as you learned when you used the allDocs() method.

```
function mapFunction() {
  db.query(getServices,
    {
      include_docs: true
    }).then(function (response) {
      pouchDBSamplesCommon.displayJSON(response);
    }).catch(function (err) {
      pouchDBSamplesCommon.displayMessage(err);
    });
}
```

The output from the above query looks like the following:

```
{
  "total_rows": 5,
  "offset": 0,
  "rows": [
    {
      "key": "Carpentry",
      "id": "Carpentry",
      "value": 100,
      "doc": {
        "cost": 100,
        "docType": "service",
        "_id": "Carpentry",
        "_rev": "1-2166646e5efd4a00b7eca9ae09b06839"
      }
    },
    {
      "key": "Concrete",
      "id": "Concrete",
      "value": 75,
      "doc": {
        "cost": 75,
        "docType": "service",
        "_id": "Concrete",
        "_rev": "1-02b5f6046b3143f2b7b62b6e7c7cf7f2"
      }
    },
    {
      "key": "Electrical",
      "id": "Electrical",
      "value": 85,
      "doc": {
        "cost": 85,
        "docType": "service",
        "_id": "Electrical",
        "_rev": "1-34469ab9046140d1bd34e0563d31a8ab"
      }
    },
    {
      "key": "Plumbing",
      "id": "Plumbing",
      "value": 75,
      "doc": {
        "cost": 75,
        "docType": "service",
        "_id": "Plumbing",
        "_rev": "1-86290166aeeb4b309d47c0aa06851cb3"
      }
    },
    {
      "key": "Yard work",
      "id": "Yard work",
      "value": 25,
      "doc": {
        "cost": 25,
        "docType": "service",
        "_id": "Yard work",
```

```
        "_rev": "1-c4e99b9d157a4d508d8c8cfbb162f087"
      }
    }
  ]
}
```

Apply Filter to Query

Just like you did with the allDocs() method you may add the *startkey* and *endkey* properties in the *options* object. The filter is applied to the *key* property returned from the emit().

```
function mapAndFilter() {
  pouchDBSamplesCommon.hideMessageAreas();
  db.query(getServices,
    {
      startkey: 'C',
      endkey: 'C\ufff0',
      include_docs: true
    }).then(function (response) {
      pouchDBSamplesCommon.displayJSON(response);
    }).catch(function (err) {
      pouchDBSamplesCommon.displayMessage(err);
    });
}
```

The output from the above query looks like the following:

```
{
  "total_rows": 5,
  "offset": 0,
  "rows": [
    {
      "key": "Carpentry",
      "id": "Carpentry",
      "value": 100,
      "doc": {
        "cost": 100,
        "docType": "service",
        "_id": "Carpentry",
        "_rev": "1-2166646e5efd4a00b7eca9ae09b06839"
      }
    },
    {
      "key": "Concrete",
      "id": "Concrete",
      "value": 75,
      "doc": {
        "cost": 75,
        "docType": "service",
        "_id": "Concrete",
        "_rev": "1-02b5f6046b3143f2b7b62b6e7c7cf7f2"
      }
    }
  ]
}
```

Persistent Views

As mentioned earlier, temporary views are fine for development, but should not be used in production applications. Instead, use persistent views by creating design documents with the map function pre-defined and stored in the database. The first time you run the query, the data returned from the query is stored as a secondary index in the database. This means all subsequent calls to the query() method are performed much quicker.

Create a Design Document

To add a design document to your database, create a JSON object with an *_id* and *views* properties. The *_id* property needs to be unique and should be prefixed with "_design" followed by a slash, then a general name of the queries you are defining within this design document. You may create one or more design document views, which you learn how to do later. After defining your design document JSON object, store it into your database using the put() method as shown in the following code.

```
function createDesignDoc() {
  let ddoc = {
    _id: '_design/generalQueries',
    views: {
      allTechnicians: {
        map: function (doc) {
          if (doc.docType === 'technician') {
            emit(doc._id, doc.firstName + ' ' + doc.lastName);
          }
        }.toString()
      }
    }
  };

  // Save the design document
  db.put(ddoc).then(function () {
    // Successfully added
    pouchDBSamplesCommon.displayMessage("Design document created
successfully.");
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

Let's take a closer look at the *views* property as shown in the code below. In line 1 define the *views* property as an object. Create a property with a unique name for each map function you wish to create. In line 2 you see a property name of *allTechnicians*. This property is another JSON object with a single property named *map* (line 3). This property is defined as a function that accepts a document object. Do not pass the emit parameter like you did with the temporary views. The body of the function (lines 4-6) can be any code you want, just like the ones you used in the temporary views in this blog post. Finally, you apply the toString() method (line 7) to this complete property so it can be stored into the database.

```
1. views: {
2.   allTechnicians: {
3.     map: function (doc) {
4.       if (doc.docType === 'technician') {
5.         emit(doc._id, doc.firstName + ' ' + doc.lastName);
6.       }
7.     }.toString()
8.   }
9. }
```

Create Two Design Documents

If you remember from part 2 of this blog post series, you inserted two different kinds of documents into the handyman database; technician and service documents. You may wish to retrieve only one or the other, so these queries are ideal for persistent views. Create these two views using the code shown below:

```
function createDesignDocs() {
  // First check to see if the design document exists
  db.get("_design/generalQueries")
    .then(function (doc) {
      pouchDBSamplesCommon.displayMessage("Design document: '" +
doc._id + "' already exists.");
    }).catch(function (err) {
      if (err.status == 404) {
        let ddoc = {
          _id: '_design/generalQueries',
          views: {
            allTechnicians: {
              map: function (doc) {
                if (doc.docType === 'technician') {
                  emit(doc._id, doc.firstName + ' ' + doc.lastName);
                }
              }.toString()
            },
            allServices: {
              map: function (doc) {
                if (doc.docType === 'service') {
                  emit(doc._id, doc.cost);
                }
              }.toString()
            }
          }
        };
        // Save the design document
        db.put(ddoc).then(function () {
          // Successfully added
          pouchDBSamplesCommon.displayMessage("Design document
created successfully.");
        }).catch(function (err) {
          pouchDBSamplesCommon.displayMessage(err);
        });
      }
      else {
        pouchDBSamplesCommon.displayMessage(err);
      }
    });
}
```

Retrieve all Technician Documents

Now that you created the two design documents, pass the name of the "allTechnicians" view to the query() method as the first parameter. You don't specify the word "_design" in the first parameter, but you do use the name that comes after that, followed by a slash, then the property name you used to create the view. The second parameter can still be an *options* object with any properties set that you want to use.

```
function getTechnicians() {
  pouchDBSamplesCommon.hideMessageAreas();
  db.query("generalQueries/allTechnicians",
    {
      include_docs: true
    }).then(function (response) {
      pouchDBSamplesCommon.displayJSON(response);
    }).catch(function (err) {
      pouchDBSamplesCommon.displayMessage(err);
    });
}
```

Retrieve all Service Documents

Let's look at one more example of calling the allServices view. The code is the same as what you used for retrieving all technicians, but the name of the view is different.

```
function getServices() {
  db.query("generalQueries/allServices",
    {
      include_docs: true
    }).then(function (response) {
      pouchDBSamplesCommon.displayJSON(response);
    }).catch(function (err) {
      pouchDBSamplesCommon.displayMessage(err);
    });
}
```

Summary

In this fourth part of our series on PouchDB, you worked with the query() method to filter documents on fields other than *_id*. The query() method is part of what is called map/reduce queries. In this blog post you learned how to map data into an index that can be used for searching. You also learned to create temporary views and persistent views. Persistent views are the ones you should be using in your production applications, while temporary views should only be used in development. In the next blog post you learn to create reduce functions to provide summary data out of your document data.

Sample Code

You can download the complete sample code at my website. http://www.pdsa.com/downloads. Choose "PDSA/Fairway Blog", then "Getting Started with PouchDB - Part 4" from the drop-down.