Getting Started with PouchDB - Part 3

In the last two blog posts, you have been introduced to the PouchDB NoSQL database. You learned to create a new database, modify documents within that database, and retrieve documents using the allDocs(). Now that you have inserted several documents into your PouchDB database, you might wish to retrieve documents based on data in fields other than the *_id* property. In this third part of our on-going blog posts on PouchDB, you learn to use the find() plug-in to perform queries on any property in your documents.

Mango Queries

The pouchdb-find plug-in can be downloaded at https://bit.ly/2s1AoYp. The find() method, also known as Mango, is a structured query mechanism allowing you to create secondary indexes on which you perform searches. This method is useful for answering questions like; find all documents where the *lastName* is equal to 'sheriff', or *cost* is greater than 75. Download the pouchdb.find plug-in and add a link to the pouchdb-find.js file. This should be listed after the link to the pouchdb-xxxxx.js file.

```
<script src="../Scripts/pouchdb-6.4.3.min.js"></script>
<script src="../Scripts/pouchdb.find.js"></script>
```

Create Indexes

You don't have to create indexes on the field(s) you wish to query upon, but if you don't, a complete scan of all documents is done. Depending on how many documents are in your database, this can be quite an expensive (and slow) operation. If you know the most common fields you are going to be querying on, create an index(es) on those fields. The code sample shown below, illustrates how to create an index on the *lastName* property and another index on the *cost* property.

```
function createIndexes() {
  // Create index on last Name
  db.createIndex({
    index: {
      fields: ['lastName']
    }
  }).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });

  // Create index on cost
  db.createIndex({
    index: {
      fields: ['cost']
    }
  }).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

Notice that the *fields* property is an array of property names. You may specify a set of fields to index on. For example, you might wish to index first on the *lastName* field, then on the *firstName* field.

Error on Sort if No Index

If you are attempting to sort on a field that has not been indexed, you receive an error from PouchDB. If the field you are using in the *selector* does not have an index, then the default index (the *_id* field) is used, and a full document scan is performed. PouchDB does not have the ability to sort the data unless there is an index on that field, thus, you receive an error.

```
function showError() {
  // NOTE: Create an index on the sort field or you get an error
  db.find({
    selector: { firstName: 'Paul' },
    sort: ['firstName']
  }).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

The error message you receive by running the function above is.

```
Error: Cannot sort on field(s) "firstName" when using the default
index
```

Warning if no Index

If you use a field(s) in the selector and no index is available, a complete document scan is performed in the database. The data is returned with the appropriate documents selected, however, you receive a warning that no matching index is found. You can then decide if you wish to create one.

```
function showWarning() {
// NOTE: You get an error if you use a property name in the
'selector' that is not indexed
  db.find({
    selector: { firstName: 'Paul' }
  }).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

The response object you receive looks like the following.

```
{
  "docs": [
    {
      "firstName": "Paul",
      "lastName": "Sheriff",
      "docType": "technician",
      "_id": "psheriff",
      "_rev": "1-36a6815e79f54819bc0b4ee3bd435aa1"
    }
  ],
  "warning": "no matching index found, create an index to optimize
query time"
}
```

Find Last Name

You created an index on the lastName field earlier. You can now use that index by specifying *lastName* as one of the fields in your *selector* property. You may further qualify what is returned by including the *fields* property. This property is an array of property names from your document you wish to return from this query. You may include the *sort* property if you want, but it really is unnecessary as when an index is used, the documents are returned by the index order anyway.

```
function findLastName() {
  db.find({
    selector: { lastName: 'Sheriff' },
    fields: ['_id', 'firstName', 'lastName'],
    sort: ['lastName']
  }).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

The result from the query above looks like the following.

```
{
  "docs": [
    {
      "_id": "msheriff",
      "firstName": "Madison",
      "lastName": "Sheriff"
    },
    {
      "_id": "psheriff",
      "firstName": "Paul",
      "lastName": "Sheriff"
    }
  ]
}
```

Find Multiple Last Names

There are many selector operators you may use instead of just searching for an exact match. Selector operators are prefixed with a dollar sign, and include the following: $eq, $gt, $gte, $lt, $lte, $in. For a complete list of selector operators, visit https://pouchdb.com/api.html#create_index.

Create a function that uses the $in operator to locate more than one last name within your documents. The code below shows searching for last names that match either 'Sheriff' or 'Jones'.

```
function findLastNames() {
  pouchDBSamplesCommon.hideMessageAreas();
  db.find({
    selector: {
      lastName: { $in: ['Sheriff', 'Jones'] }
    },
    fields: ['_id', 'firstName', 'lastName']
  }).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

The return result from the above query looks like the following. Notice that this selector operator does not use the index based on the last name field. Certain types of operators are not always able to use indexes; $in, $or and $regex are a few of them that can't.

```
{
  "docs": [
    {
      "_id": "bjones",
      "firstName": "Bruce",
      "lastName": "Jones"
    },
    {
      "_id": "msheriff",
      "firstName": "Madison",
      "lastName": "Sheriff"
    },
    {
      "_id": "psheriff",
      "firstName": "Paul",
      "lastName": "Sheriff"
    }
  ],
  "warning": "no matching index found, create an index to optimize
query time"
}
```

Find Cost Greater than 75

Another useful selector operator is greater than ($gt). You created an index earlier on the *cost* property of the documents that have *docType* of 'service'. Use the following code to find all services that have a cost greater than 75.

```
function greaterThan() {
  pouchDBSamplesCommon.hideMessageAreas();
  db.find({
    selector: { cost: { $gt: 75 } },
    fields: ['_id', 'cost']
  }).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

Search on Two Fields

You are not restricted to just indexing on one field. You may include two or more fields in the index *fields* property. In the *selector* property, you may now specify two fields to match upon. In the sample code below, you are querying the documents to find where the doctype is equal to 'service' and the cost is greater than 75.

```
function searchTwoFields() {
  // Create index on two fields
  db.createIndex({
    index: {
      fields: ['docType', 'cost']
    }
  }).then(function (response) {
    // Search on two fields
    return db.find({
      selector: {
        docType: 'service',
        cost: { $gt: 75 }
      },
      fields: ['_id', 'cost']
    });
  }).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

Summary

In this third part of our series on PouchDB, you learned to use the find() method. This method is a plug-in to PouchDB, so you must download it separately. The find() method allows you to search on fields other than the *_id* field in your documents. Be sure to add an index for the field(s) you with to search upon, as doing so, increases the performance of your search. In the next blog post, you learn to use the query() method and map functions.

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 3" from the drop-down.