Getting Started with PouchDB - Part 6

In the last several blog posts, you worked with a very flat document structure. However, in a more real-world scenario you may have a more complicated JSON object with several nested objects. Working with those types of objects requires you to query and index data slightly differently. This blog posts shows you how to create a complex document structure and query that data.

Document Structures

In this blog post you are going to add a series of Job and Invoice documents to your PouchDB database. Each one of these contains properties to represent the data required to describe the data. Some of these properties are themselves objects, and some are arrays of objects. Let's look at each of these documents.

In this series of documents, add a prefix to the *_id* property of each document that describes the type of the document. This helps you eliminate the need for a *doctype* property and allows you to search on specific types of documents using the built-in index on the *_id* property.

Job Document

A job document has some standard properties; *_id*, *serviceDate* and *customer*. Notice that the *_id* property has a prefix of "job_" on it. This helps us to search for just job documents using the built-in *_id* index. It has a *technician* property that is another object that contains two other properties to describe an employee. The final property, *workDone*, is an array of other objects to describe a one or more tasks that were completed on this job.

```
{
  "_id": "job_2020",
  "serviceDate": "2018-02-03",
  "customer": "452",
  "technician": {
    "EmployeeID": "1034",
    "name": "Sheriff, Madison"
  },
  "workDone": [{
    "description": "Drywall installation",
    "price": 500
  },
  {
    "description": "Painting",
    "price": 100
  }]
}
```

Invoice Document

An invoice document contains a few standard properties; *_id*, *invoiceDate* and *invoiceTotal* which are self-explanatory. Notice that the *_id* property has a prefix of "invoice_" on it. This helps us to search for just invoice documents using the built-in *_id* index. The *customer* property is another object that describes the customer for whom the job was done. The *lineItems* property contains an array of items with a link to a job documents' *_id* property, the description of the work done and the price for that work.

```
{
  "_id": "invoice_2534",
  "invoiceDate": "2018-02-04",
  "invoiceTotal": 375,
  "customer": {
    "_id": "834",
    "name": "John Smith",
    "address": "123 Main Street",
    "city": "Nashville",
    "state": "TN",
    "postalCode": "37211"
  },
  "lineItems": [
    {
      "jobId": "job_2010",
      "description": "Driveway repair",
      "price": 225
    },
    {
      "jobId": "job_2011",
      "description": "Carport repair",
      "price": 150
    }
  ]
}
```

Insert Sample Documents

Create a set of job and invoice documents into the database so you can test working with nested documents. Add the following function and call it to insert the test documents.

```
function addMultipleDocs() {
  db.bulkDocs([
    {
      "_id": "job_2010",
      "serviceDate": "2018-02-03",
      "customer": "834",
      "technician": {
        "EmployeeID": "1023",
        "name": "Sheriff, Paul"
      },
      "workDone": [{
        "description": "Driveway repair",
        "price": 225
      }]
    },
    {
      "_id": "job_2011",
      "serviceDate": "2018-02-03",
      "customer": "834",
      "technician": {
        "EmployeeID": "1023",
        "name": "Sheriff, Paul"
      },
      "workDone": [{
        "description": "Carport repair",
        "price": 150
      }]
    },
    {
      "_id": "job_2020",
      "serviceDate": "2018-02-03",
      "customer": "452",
      "technician": {
        "EmployeeID": "1034",
        "name": "Sheriff, Madison"
      },
      "workDone": [{
        "description": "Drywall installation",
        "price": 500
      },
      {
        "description": "Painting",
        "price": 100
      }]
    },
    {
      "_id": "job_2030",
      "serviceDate": "2018-02-04",
      "customer": "651",
      "technician": {
        "EmployeeID": "1051",
        "name": "Jones, Bruce"
      },
      "workDone": [{
        "description": "Lawn mowing",
        "price": 100
        }]
      },
      {
        "_id": "job_2040",
        "serviceDate": "2018-03-05",
        "customer": "834",
        "technician": {
          "EmployeeID": "1189",
          "name": "Kuhn, John"
        },
        "workDone": [{
          "description": "Install doorbell",
          "price": 75
        }]
      },
      {
        "_id": "job_2050",
        "serviceDate": "2018-03-06",
        "customer": "983",
        "technician": {
          "EmployeeID": "1023",
          "name": "Sheriff, Paul"
        },
        "workDone": [{
          "description": "Drywall repair",
          "price": 95
        },
        {
          "description": "Painting",
          "price": 50
        }]
      },
      {
        "_id": "job_2060",
        "serviceDate": "2018-04-08",
        "customer": "389",
        "technician": {
          "EmployeeID": "1189",
          "name": "Kuhn, John"
        },
        "workDone": [{
          "description": "Ceiling fan install",
          "price": 150
        }]
      },
      {
        "_id": "invoice_2534",
        "invoiceDate": "2018-02-04",
        "invoiceTotal": 375,
        "customer": {
          "_id": "834",
          "name": "John Smith",
          "address": "123 Main Street",
          "city": "Nashville",
          "state": "TN",
          "postalCode": "37211"
        },
      "lineItems": [
        {
          "jobId": "job_2010",
          "description": "Driveway repair",
          "price": 225
        },
        {
          "jobId": "job_2011",
          "description": "Carport repair",
          "price": 150
        }
      ]
    },
    {
      "_id": "invoice_2536",
      "invoiceDate": "2018-02-04",
      "invoiceTotal": 600,
      "customer": {
        "_id": "452",
        "name": "Henry James",
        "address": "98 5th Ave",
        "city": "Brentwood",
        "state": "TN",
        "postalCode": "37027"
      },
      "lineItems": [
        {
          "jobId": "job_2020",
          "description": "Drywall installation",
          "price": 500
        },
        {
          "jobId": "job_2020",
          "description": "Painting",
          "price": 100
        }
      ]
    },
    {
      "_id": "invoice_2537",
      "invoiceDate": "2018-02-05",
      "invoiceTotal": 100,
      "customer": {
        "_id": "651",
        "name": "Grant Able",
        "address": "113 Woods Lane",
        "city": "Brentwood",
        "state": "TN",
        "postalCode": "37027"
      },
      "lineItems": [
        {
          "jobId": "job_2030",
          "description": "Lawn mowing",
          "price": 100
        }
      ]
    },
    {
      "_id": "invoice_2538",
      "invoiceDate": "2018-03-06",
      "invoiceTotal": 75,
      "customer": {
        "_id": "843",
        "name": "John Smith",
        "address": "123 Main Street",
        "city": "Nashville",
        "state": "TN",
        "postalCode": "37211"
      },
      "lineItems": [
        {
          "jobId": "job_2040",
          "description": "Install doorbell",
          "price": 75
        }
      ]
    },
    {
      "_id": "invoice_2539",
      "invoiceDate": "2018-03-07",
      "invoiceTotal": 145,
      "customer": {
        "_id": "983",
        "name": "Mike Tinder",
        "address": "8733 Mockingbird Street",
        "city": "Franklin",
        "state": "TN",
        "postalCode": "37064"
      },
      "lineItems": [
        {
          "jobId": "job_2050",
          "description": "Drywall Repair",
          "price": 95
        },
        {
          "jobId": "job_2050",
          "description": "Painting",
          "price": 50
        }
      ]
    },
    {
      "_id": "invoice_2540",
      "invoiceDate": "2018-02-06",
      "invoiceTotal": 150,
      "customer": {
        "_id": "389",
        "name": "Sally Sherland",
        "address": "11 14th Avenue",
        "city": "Nashville",
        "state": "TN",
        "postalCode": "37211"
      },
      "lineItems": [
        {
          "jobId": "job_2060",
          "description": "Ceiling fan install",
          "price": 150
        }
      ]
    }
  ]).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
    pouchDBSamplesCommon.displayMessage("Multiple documents
added.");
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

Working with Intelligent ID's

As you can see from the sample data you just entered, each *_id* property has a prefix that describes the type of document. After the prefix is a unique job id or unique invoice id. Adding a prefix to the *_id* property helps us with searching as shown in the next two samples.

Retrieve Job Documents

An advantage of creating an "intelligent _id" property is it allows you to retrieve documents using the built-in index on the *_id* property. For example, to retrieve all job documents, simply specify the *startkey* and *endkey* properties as follows.

```
function getAllJobs() {
  db.allDocs({
    startkey: 'job_',
    endkey: 'job_\ufff0',
    include_docs: true
  }).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

Retrieve Invoice Documents

To retrieve just the invoice documents, you can use the startkey and endkey properties as shown below.

```
function getAllInvoices() {
  pouchDBSamplesCommon.hideMessageAreas();
  db.allDocs({
    startkey: 'invoice_',
    endkey: 'invoice_\ufff0',
    include_docs: true
  }).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

Other Examples of Intelligent ID's

You can do quite a bit with the _*id*_ property. For example, if you frequently look up jobs by date, add the service date to the _*id*_ property in each job document. Here are some examples:

```
"id": "Job_2018_02_03_2010"
"id": "Job_2018_02_03_2011"
"id": "Job_2018_03_05_2040"
"id": "Job_2018_04_08_2060"
```

You may now use the startkey and endkey to search for all jobs within a year.

```
{
  startkey: 'job_2018',
  endkey: 'job_2018\ufff0'
}
```

Or, you can search for a specific month using the following values.

```
{
  startkey: 'job_2018_03',
  endkey: 'job_2018_03\ufff0'
}
```

The point here is to use your _*id*_ property intelligently so you can use the built-in index on the _*id*_ property and avoid creating additional indexes.

Locate Customer by Name

When you have a property that contains another object, you need to use the dot notation when creating the fields for your index. The code below shows creating an index based on the *customer.name* property. This tells PouchDB to locate a *customer* property, and to further locate within that property a property called *name*.

```
db.createIndex({
  index: {
    fields: ['customer.name']
  }
}).then(function (response) {
  pouchDBSamplesCommon.displayJSON(response);
}).catch(function (err) {
  pouchDBSamplesCommon.displayMessage(err);
});
```

To locate a specific customer's name, you use the dot notation in the *selector* property of the options object you pass to the find() method as shown below.

```
function getCustomer() {
  let search = "Mike Tinder";

  db.find({
    selector: { "customer.name": search }
  }).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

Price Greater than 150

If a property of your document contains an array of other objects, you can't create an index on a property of the objects within the array. You may still use the find() method to perform this search, but it is going to perform a complete scan of all documents. The following code using the $elemMatch selector operator to find an element with a property called *price*. It then searches for where any price value is greater than 150.

```
function getWorkDonePrice() {
  let search = 150;

  pouchDBSamplesCommon.hideMessageAreas();
  db.find({
    selector: {
      "workDone": {
        "$elemMatch": { "price": { $gt: search } }
      }
    }
  }).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

Create Design Document

The find() method works for properties that contain objects, but does not work so well for properties that are arrays of objects. As you have seen in a previous blog post, you may create a design document using a map() function to create a secondary index. This is the most efficient way to search for data within an array of objects.

The code highlighted in the following listing shows the declaration of two views, The first view looks at each document for a *customer* property that is an object and has a *name* property. If it finds a document, it calls the emit() function to emit the customer.name value into the index.

The second view looks at each document for a *lineItems* property. The *lineItems* array is then looped through and emits the *price* of each object. In this way an index of all price values is created and is ready to be searched.

```
function createDesignDoc() {
  pouchDBSamplesCommon.hideMessageAreas();
  // First check to see if the design document exists
  db.get("_design/allQueries")
    .then(function (doc) {
      pouchDBSamplesCommon.displayMessage("Design document: '" +
doc._id + "' already exists.");
    }).catch(function (err) {
      if (err.status == 404) {
        var ddoc = {
          _id: '_design/allQueries',
          views: {
            byCustomerName: {
              map: function (doc) {
                if (doc.customer.name) {
                  emit(doc.customer.name);
                }
              }.toString()
            },
            byPrice: {
              map: function (doc) {
                if (doc.lineItems) {
                  for (var i = 0; i < doc.lineItems.length; i++) {
                    emit(doc.lineItems[i].price);
                  }
                }
              }.toString()
            }
          }
        };
        // Save the design document
        db.put(ddoc).then(function (response) {
          // Successfully added
          pouchDBSamplesCommon.displayMessage("Design document
created successfully.");
          pouchDBSamplesCommon.displayJSON(response);

        }).catch(function (err) {
          pouchDBSamplesCommon.displayMessage(err);
        });
      }
      else {
        pouchDBSamplesCommon.displayMessage(err);
      }
    });
}
```

Find Customer by Name

Create a function that calls the query() method and passes in the name of the design document followed by the view name. The second parameter to the query() method is an options object on which you add the *startkey* and *endkey* properties of the value(s) you wish to locate.

```
function getCustomer() {
  pouchDBSamplesCommon.hideMessageAreas();
  db.query("allQueries/byCustomerName",
    {
      startkey: 'John Smith',
      endkey: 'John Smith\ufff0',
      include_docs: true
    }).then(function (response) {
      pouchDBSamplesCommon.displayJSON(response);
    }).catch(function (err) {
      pouchDBSamplesCommon.displayMessage(err);
    });
}
```

Find Price

To search for a range of prices, you take advantage of the *byPrice* view you created in the design document. Call the query() method using the *byPrice* view and pass in the startkey and endkey with the specific values you wish to search for. In the code below, you want to return all documents that have a price between 50 and 100.

```
function getByPrice() {
  pouchDBSamplesCommon.hideMessageAreas();
  db.query("allQueries/byPrice",
    {
      startkey: 50,
      endkey: 100,
      include_docs: true
    }).then(function (response) {
      pouchDBSamplesCommon.displayJSON(response);
    }).catch(function (err) {
      pouchDBSamplesCommon.displayMessage(err);
    });
}
```

Summary

In this blog post you learned to work with documents that contain nested objects and arrays. There are just a couple of things you need to do differently to search for data within these nested objects or arrays. You also learned how to use your *_id* property intelligently to take advantage of the built-in index on the *_id* property.

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 6" from the drop-down.