Getting Started with PouchDB - Part 2

In the last blog post, you learned to insert, update, delete and read single documents in a PouchDB database. Let's now look at how to perform multi-document inserts and reads.

Bulk Operations

There are a few different methods you can invoke to work with multiple records. The table below summarizes each of the methods available through the PouchDB API.

Method	Description
bulkDocs()	Create, update or delete multiple documents.
allDocs	Retrieve multiple documents sorted by _id field. You may also set a range of _id values to retrieve.
bulkGet()	Supply a set of *_id* and *_rev* values, and this method returns the documents associated with each.

Bulk Insert

To insert a set of documents into the database at one time, use the bulkDocs() method. Create an array of documents you wish to insert, and pass this array to the bulkDocs() method. The code below shows inserting a set of user documents and a set of service documents. The result from this insert of multiple documents is an array of JSON documents with three properties, "ok", "id" and "rev". The *ok* property has a 'true' value, the *id* has the original *_id* value you specified, and the *rev* property has the *_rev* field generated by PouchDB.

```
function addMultipleDocs() {
  db.bulkDocs([
    {
      _id: 'psheriff',
      firstName: 'Paul',
      lastName: 'Sheriff',
      docType: 'technician'
    },
    {
      _id: 'bjones',
      firstName: 'Bruce',
      lastName: 'Jones',
      docType: 'technician'
    },
    {
      _id: 'jkuhn',
      firstName: 'John',
      lastName: 'Kuhn',
      docType: 'technician'
    },
    {
      _id: 'msheriff',
      firstName: 'Madison',
      lastName: 'Sheriff',
      docType: 'technician'
    },
    {
      _id: 'mshane',
      firstName: 'Molly',
      lastName: 'Shane',
      docType: 'technician'
    },
    {
      _id: 'Carpentry',
      cost: 100,
      docType: 'service'
    },
    {
      _id: 'Concrete',
      cost: 75,
      docType: 'service'
    },
    {
      _id: 'Yard work',
      cost: 25,
      docType: 'service'
    },
    {
      _id: 'Plumbing',
      cost: 75,
      docType: 'service'
    },
    {
      _id: 'Electrical',
      cost: 85,
      docType: 'service'
```

```
    }
  ]).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
    pouchDBSamplesCommon.displayMessage("Multiple documents
added.");
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

After you run the above code, a response is returned that looks like the following:

```
[
  {
    "ok": true,
    "id": "psheriff",
    "rev": "1-3889989f75da4924a14e6551b8c8b4f0"
  },
  {
    "ok": true,
    "id": "bjones",
    "rev": "1-6959354560114b3aab4506af1f6ff89b"
  },
  {
    "ok": true,
    "id": "jkuhn",
    "rev": "1-a90aa8c09b514d4ba122643348e83f92"
  },
  ... // MORE DOCS HERE
]
```

If, while inserting new documents, any of the *_id* values are duplicated, an error response document is returned as shown below.

```
[
  {
    "status": 409,
    "name": "conflict",
    "message": "Document update conflict",
    "error": true,
    "id": "psheriff"
  },
  ... // MORE DOCS HERE
]
```

The error response document contains different properties from the success response. The *status* property is set to a HTTP status code, which in this case a 409. The *name* property is set to a short description of the HTTP status code, which in this case is 'conflict'. A *message* property reports a description of what PouchDB says what went wrong. The *error* property is always set to true". The *id* property is set to the document *_id* property that was in error.

Bulk Update

If you wish to delete or update a set of documents, pass an array of JSON objects to the bulkDocs() method. Each document in the array needs the *_id* and *_rev* properties set to valid values. Be sure to include the complete document to update or it will only store the specific properties you include.

```
db.bulkDocs([
  {
    _id: 'psheriff',
    _rev: '1-bfe5495126ec488c8a707b50afb49bfe',
    firstName: 'Paul',
    lastName: 'Sheriff-CHANGED',
    docType: 'technician'
  },
  {
    _id: 'bjones',
    _rev: '1-65bfb049dbdc440bba314626cd17cbf5',
    firstName: 'Bruce',
    lastName: 'Jones-CHANGED ',
    docType: 'technician'
  }
]).then(function (response) {
  pouchDBSamplesCommon.displayJSON(response);
  pouchDBSamplesCommon.displayMessage("Multiple documents
updated.");
}).catch(function (err) {
  pouchDBSamplesCommon.displayMessage(err);
});
```

Bulk Delete

To delete a set of documents, pass in an array of JSON objects with the *_id* and *_rev* properties set, and include a property named *_deleted* and set its value to true.

```
db.bulkDocs([
  {
    _id: 'psheriff',
    _rev: '1-bfe5495126ec488c8a707b50afb49bfe',
    _deleted: true
  },
  {
    _id: 'bjones',
    _rev: '1-65bfb049dbdc440bba314626cd17cbf5',
    _deleted: true
  }
]).then(function (response) {
  pouchDBSamplesCommon.displayJSON(response);
  pouchDBSamplesCommon.displayMessage("Multiple documents
deleted.");
}).catch(function (err) {
  pouchDBSamplesCommon.displayMessage(err);
});
```

allDocs() Method

One of the best ways to retrieve documents from your PouchDB database is to use the allDocs() method. The allDocs() method uses the automatic index that is created based on the values in the *_id* property of your documents. The allDocs() method allows you to retrieve all, or a subset of documents from the database. Documents retrieved using the allDocs() method are returned in *_id* order. Let's look at calling the allDocs() method with no parameters.

```
function getAllDocIdsAndRevs() {
  db.allDocs().then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

The response returned from allDocs() contains three properties. The "total_rows" property reports how many documents in this database. The "offset" property reports if you had skipped any documents before providing the rows in the "rows" property. The "rows" property is an array of JSON objects which contains "id", "key" and "value" properties. The *id* and the *key* properties contain the original value of the *_id* property. The *value* property is an object with a single property named "rev". This value is the *_rev* property generated by PouchDB.

```
{
  "total_rows": 10,
  "offset": 0,
  "rows": [
    {
      "id": "bjones",
      "key": "bjones",
      "value": {
          "rev": "1-65bfb049dbdc440bba314626cd17cbf5"
      }
    },
    {
      "id": "jkuhn",
      "key": "jkuhn",
      "value": {
          "rev": "1-30b056415cdd4ec69843ec3979c83702"
      }
    },
    ... // MORE DOCS HERE
  ]
}
```

While the above data provides you with what is necessary to then retrieve any document using the get() method, you might want to get all the document data using allDocs(). Pass an *options* object to the allDocs() method to control what this method returns. The *options* object has a multitude of properties you can set. Check out more about the options at https://pouchdb.com/api.html#batch_fetch. For this sample, just set the *include_docs* property to true to tell allDocs() to return the full document data.

```
function getAllDocuments() {
  db.allDocs({ include_docs: true }).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

Count Number of Documents

As you saw in the above response objects, you always get the *total_rows* property returned. If you wish to just get a total count of documents, but not all the document data, set two properties in the options object to the following values; *limit:0* and *include_docs: false*.

```
function countDocuments() {
  db.allDocs({
    limit: 0,
    include_docs: false
  }).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

The response from this query results in a response object that looks like the following:

```
{
    "total_rows": 10,
    "offset": 0,
    "rows": []
}
```

NOTE: The *total_rows* property is always the total number of documents in the database.

Get by Range

Since an index is created automatically each time you insert a document into the PouchDB database, this means you may filter the data using the *_id* property. Include two properties; *startkey* and *endkey* within the *options* object and specify a starting value and ending value. Consider the following code below.

```
function getByRange() {
  let options = {
    include_docs: true,
    startkey: 'bjones',
    endkey: 'jkuhn'
  }
  // Get the data
  db.allDocs(options).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

The *startkey* property is set to one id 'bjones', and the *endkey* property is set to one that comes later in the sort order of the ids. All documents between, and including, these two keys are returned.

Get by Partial Keys

The startkey and endkey do not need contain full id value, you may search on partial data too. For example, consider the following code.

```
function getByPrefix() {
  pouchDBSamplesCommon.hideMessageAreas();
  let options = {
    include_docs: true,
    startkey: 'msh',
    endkey: 'msh\ufff0'
  }
  // Get the data
  db.allDocs(options).then(function (response) {
    pouchDBSamplesCommon.displayJSON(response);
  }).catch(function (err) {
    pouchDBSamplesCommon.displayMessage(err);
  });
}
```

In the options object, the *startkey* property is set to 'msh', so it would match documents such as 'msheriff', 'mshane', etc. The *endkey* property contains the string 'msh\ufff0'. The value '\ufff0' is special high Unicode character that represents the last values in the sort order. For this example, it is like specifying 'mshzzzzzzzzzz' for the *endkey* property. This sample allows you to retrieve any records that only start with 'msh'.

Summary

In this second part of this series of blog posts on PouchDB, you learned to bulk insert, update and delete documents from the database. In addition, you learned to use the allDocs() method to retrieve and count documents. In the next part of this blog post series you learn to run Mango queries using the find() plug-in to PouchDB.

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 2" from the drop-down.