MarkLogic UNIVERSITY

Unit 11

Database Transactions

Perform a Document Update

Perform a Patch Update

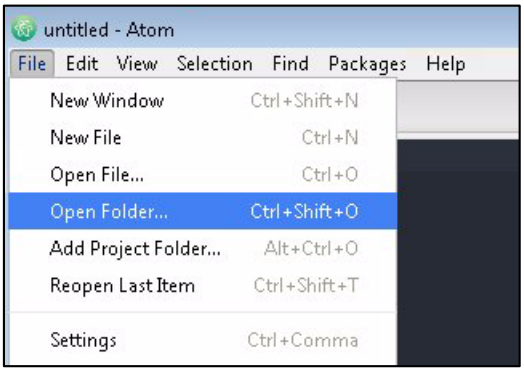
Perform a Multi-Statement Transaction

Exercise 1: Perform an Update

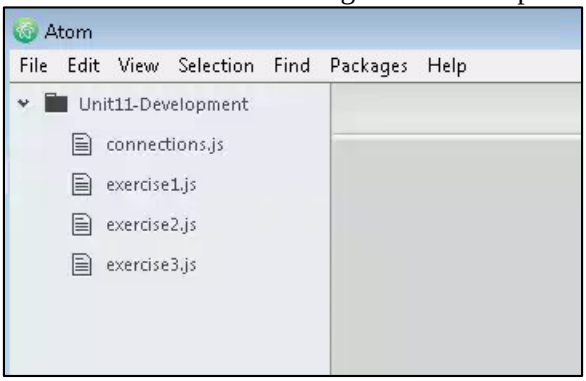
In this exercise you will perform a document update by writing a document at a URI in the database that already exists.

1. From the command line, navigate to the **c:\mls-developer-node\code\Unit11-Development** directory and run **npm install marklogic**.
2. Open the Atom editor from your Desktop:

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



4. Open the **Unit11-Development** folder from c**:\mls-developer-node\code**
5. You should see the following structure in place in your editor:



6. Select **exercise1.js**.

7. Take a moment to study the comments and code.
8. Note that you are building a new document structure in memory:

```
var uri = "/songs/Kesha+Tik-Tok.json";

var newDoc =
  {
    "top-song":
    {
      "artist": "Kesha",
      "title": "Tik Tok",
      "descr": "Some information about this song."
    }
  };

var newDocDescriptor = [
  { "uri": uri,
    "contentType": "application/json",
    "content": newDoc
  }
];
```

9. Note that you this example will probe the database to see if this URI already exists. Because the URI does already exist, when you write the new document to the database at this URI you are performing a document update:

```
dbRead.documents.probe(uri).result(
  function(response) {
    if (response.exists) {
      console.log(response.uri + " exists.  Let's update it.");
      dbWrite.documents.write(newDocDescriptor).result().then(
        function(response){
          console.log("Finished with write.");
          dbRead.documents.read(uri).result(
            function(documents){
              documents.forEach(function(document){
                console.log("Updated Doc = " + JSON.stringify(document.content));
              });
            });
        });
    } else {
      console.log("URI " + response.uri + " does not exist, nothing to update.");
    }
  }
);
```

10. Now let's test the code.

11. At the command line, go to the **c:\mls-developer-node\Unit11-Development** directory.
12. Enter **node exercise1.js** and press enter.
13. You should see the following response:

```
c:\mls-developer-node\code\Unit11-Development>node exercise1.js
/songs/Kesha+Tik-Tok.json exists.  Let's update it.
Finished with write.
Updated Doc = {"top-song":{"artist":"Kesha","title":"Tik Tok","descr":"Some info
rmation about this song."}}
```

Exercise 2: Perform a Patch Update

In this exercise you will perform a patch update in order to update a specific portion of a document.

1. In your editor, within the **Unit11-Development** project, select **exercise2.js**.
2. Study the comments and code.
3. Note that we are creating a variable for the MarkLogic patchBuilder interface:

```
var marklogic = require("marklogic");
var dbConn = require("./connections.js");
var pb = marklogic.patchBuilder;
```

4. Note that we are going to update the value of the artist property for this document. So first we go and read the original value so we can see the effect of our update. Then we use the patch builder to do a **replaceInsert**, replacing the old artist property with the new artist property and value:

```
dbRead.documents.read(uri)
.result()
.then(function(response) {
  oldArtist = response[0].content["top-song"].artist;
  var newArtist = "Ke$ha";
  if (oldArtist) {
    return dbWrite.documents.patch(
      response[0].uri,
      pb.replaceInsert(
        "/top-song/artist",
        "/top-song/node('artist')",
        "last-child",
        newArtist)
    ).result();
  }
}).then(function(response) {
  return dbRead.documents.read(response.uri).result();
}).then(function(response) {
  console.log("Old artist: " + oldArtist);
  console.log("New artist: " + "response[0].content["top-song"].artist);
});
```

Note:
replaceInsert is just one of the patch builder methods available. See the documentation for more:

http://docs.marklogic.com/guide/node-dev/partial-update#id_35482

5. Now let's test the code.
6. At the command line, go to the **c:\mls-developer-node\Unit11-Development** directory.
7. Enter **node exercise2.js** and press enter.
8. You should see the following response:

```
c:\mls-developer-node\code\Unit11-Development>node exercise2.js
Old artist: Kesha
New artist: Ke$ha
```

Exercise 3: Perform a Multi-Statement Transaction

In this exercise you will perform a multi-statement transaction that reads a document, inserts that document at a new URI in the database, and deletes the old document.

Because it is a multi-statement transaction, each statement can see the effect of the prior statement in the transaction, but you must explicitly commit the transaction in your code.

1. In your editor, within the **Unit11-Development** project, select **exercise3.js**.
2. Study the comments and code.
3. Note the **transactionalMove** function, which is opening the transaction and performing the multi-statement transaction using a chain of promises to complete each of the statements in the transaction before deciding to commit:

```
function transactionalMove(oldUri, newUri) {
  var transactionId = null;
  dbWrite.transactions.open().result().
  then(function(txid) {
    transactionId = txid;
    return dbWrite.documents.read({uris: oldUri, txid: transactionId.txid}).result
  }).
  then(function(document) {
    document[0].uri = newUri;
    return dbWrite.documents.write(
      {
        documents: document,
        txid: transactionId.txid
      }).result()
  })
  .then(function(response) {
    return dbWrite.documents.remove({uri: oldUri, txid: transactionId.txid}).resul
  })
  .then(function(response) {
    return dbWrite.transactions.commit(transactionId.txid).result();
  })
  .then(function(response){
    var uriArray = [];
    uriArray.push(oldUri, newUri);
    return resultSummary(uriArray);
  })
```

4. Note the use of the **catch** method to rollback the transaction should an error occur:

```
.catch(function(error) {
  dbWrite.transactions.rollback(transactionId.txid);
  console.log("Error: ", error);
});
```

5. Finally, note the **resultSummary** function. This is so we can validate the result of the multi-statement transaction after it has commited:

```
function resultSummary(docUris){
  docUris.forEach(function(docUri){
    dbRead.documents.probe(docUri).result()
    .then(function(response) {
      if (response.exists) {
        console.log(response.uri + " exists.");
      } else {
        console.log(response.uri + " does not exist.");
      };
    });
  });
};
```

6. Now let's test the code.
7. At the command line, go to the **c:\mls-developer-node\Unit11-Development** directory.
8. Enter **node exercise3.js** and press enter.
9. You should see the following response:

```
c:\mls-developer-node\code\Unit11-Development>node exercise3.js
/songs/Kesha+Tik-Tok2.json exists.
/songs/Kesha+Tik-Tok.json does not exist.
```