Progress MarkLogic

Unit 3

Documents

Load Data

MLCP – Creating ml-Gradle Custom Tasks

Accessing and Updating Data

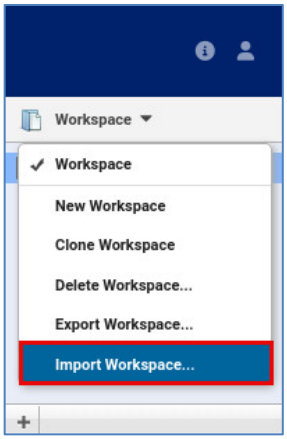
Access Control and CoRB2

Upload Files Using WebDav (optional)

Exercise 1: Load Data

This exercise focuses on creating documents with Server-side JavaScript (SJS) or XQuery. Feel free to adjust the presented workspace to have a better familiarity with the used functions.

1. In Query Console: http://localhost:8000/qconsole
2. Click on `Workspace` >> `Import Workspace` as shown below:



3. Click `Browse` and navigate to, "/home/cent/Desktop/fundamentals/solutions/unit_03/exercise_01/Load Documents - SJS.xml"
    - There may be some key differences with "Load Documents – XQY.xml". This can be attributed to the fact that XQuery is more natural in dealing XML while being just as capable in handling JSON content.
4. The instructor will go through each of the tabs.
    - Take note of the amount of time it took to execute the last tab.
5. Open the Admin UI: http://localhost:8001.

Exercise 2: MLCP – Creating ml-Gradle Custom Tasks

As can be observed on the last tab of the previous unit, loading a massive amount of content using a single transaction may not be possible. MarkLogic Content Pump allows for the ingest of a massive amount of data via batches of inserts.

1. In VSC or a Terminal window, navigate to the star-wars project folder:

```
$ cd /home/cent/Desktop/star-wars
```

2. Copy the prepared project changes from the Solutions folder.

```
$ cp -r /home/cent/Desktop/fundamentals/solutions/unit_03/exercise_02/* .
```

   - The folder structure:

```
exercise_02
│   build.gradle
│
└───src
    └───main
        └───ml-modules
            └───root
                └───mlcp
                        extractMetadata.sjs
```

   - extractMetadata.sjs is similar to your `Extracting binary info` tab in Query Console. We will be using module as part of MLCP import.

3. Before anything else, we publish the module into our module database:

```
$ ./gradlew mlLoadModules
```

4. Review the changes in build.gradle
   - Notice that `dependencies` now include mlcp
   - Notice that tasks `importCharacters` and `importImages` have been created.
   - Notice how `importImages` also include transform related information that matches the new module we just uploaded.

5. Run the import characters task:

```
$ ./gradlew importCharacters
```

   - Check back on the task definition and notice how `output_uri_replace` accepts regular expressions for pattern replacement.

6. Use Query Console to “Explore” `star-wars-content` database.
    - Notice how the URI got constructed.
7. Run the import images task:

```
$ ./gradlew importImages
```

    - Take note of the time it took to upload the images. Compare that against your transaction using Query Console.
    - Check back on the task definition and notice how `transform_module` and `transform_function` is configured. This makes use of our `extractMetadata.sjs` mentioned above
8. Use Query Console to “Explore” `star-wars-content` database.
    - Notice how the extracted information got stored along with the binary data.
9. Run the import triples task:

```
$ ./gradlew importTriples
```

    - Check back on the task definition and notice that `input_file_type` is currently set to `rdf`.
    - At this time, there is no option to have the output in JSON format.
10. Use Query Console to “Explore” `star-wars-content` database. Use the URI filter with the pattern `/triple*` to find the converted file.
    - Notice how the triple file got converted into an XML file.
11. Run the import csv task:

```
$ ./gradlew importCsv
```

    - Check back on the task definition and notice that `input_file_type` is currently set to `delimited_text`.
    - Also notice that we are no longer supplying `port`. MLCP defaults to 8000 and in turn store documents into “Documents” database by default.
12. Use Query Console to “Explore” `Documents` database
    - Notice how each row got converted into a JSON Document.
    - Notice how the `emp_id` column got used to construct the URI along with the configured prefix and suffix.
    - Notice how the document permissions have been set.

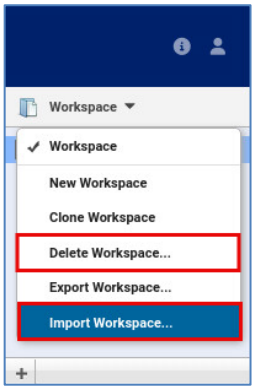
Challenge

To test your familiarity with MLCP, adjust your `top-songs` project to support a gradle task that will load content from `/home/cent/Desktop/fundamentals/data/top-songs/`. Have the content assigned the collection of `song` for the XML files and `album covers` for the images. Extract any metadata available for the images and save them in addition to the image files.

Exercise 3: Accessing and Updating Data

This exercise focuses on accessing and updating documents. This uses the Server-side JavaScript (SJS) as the programming language. An alternative XQuery equivalent is also available. Feel free to adjust the presented workspace to have a better familiarity with the used functions.

1. In Query Console: http://localhost:8000/qconsole
2. Click on `Workspace` >> `Delete Workspace…` to remove the previous Workspace.



3. Click on `Workspace` >> `Import Workspace…` and load the workspace located at:

   "/home/cent/Desktop/fundamentals/solutions/unit_03/exercise_03/Access and Update - SJS.xml"

   - There may be some key differences with "Access and Update – XQY.xml". This can be attributed to the fact that XQuery is more natural in dealing XML while being just as capable in handling JSON content.

4. The instructor will go through each tab.

Partial updates using the Client REST API is also possible. Unfortunately, there is no equivalent for PATCH using the built-in xdmp library. Let us use the terminal window to test that.

5. Using your Terminal window, run the following command to add a new property to our document:

```
$ curl --anyauth -uadmin:admin -X PATCH -H 'Content-Type: application/json' \
-d '{
  "patch": [{
    "insert": {
      "context": "/decimal",
      "position": "after",
      "content": { "rest-child": "added via rest" }
    }
  }]
}' \
'http://localhost:8000/v1/documents?uri=/sample-data-types.json'
```

   - There may be some syntactical difference when trying this out on a different platform, especially with the payload. See Appendix A for Windows OS.

6. Use Query Console and click on "Explore" `Documents` database to view the changes on the specified document.

7. Run the following command to add a new collection to our document:

```
$ curl --anyauth -uadmin:admin -X PATCH -H "Content-Type: application/json" \
-d '{
  "patch": [{
    "insert": {
      "context": "/array-node(\"collections\")",
      "position": "last-child",
      "content": "col5"
    }
  }]
}' \
'http://localhost:8000/v1/documents?uri=/sample-data-types.json&category=collections'
```

   - There may be some syntactical difference when trying this out on a different platform, especially with the payload. See Appendix A for Windows OS.

8. Use Query Console and click on “Explore” `Documents` database to view the changes on the specified document.

   - Notice that the collection “col5” was added to the list of collections of the document instead of replacing the current list of collections.

Exercise 4: Access Control and CORB2

This exercise focuses on access control and efficient update on all content using CORB2. This uses the Server-side JavaScript (SJS) as programming language. An alternative XQuery equivalent is also available. Feel free to adjust the presented workspace to have a better familiarity with the used functions.

1. In Query Console: http://localhost:8000/qconsole, delete the previous Workspace and Import the workspace located at:

"/home/cent/Desktop/fundamentals/solutions/unit_03/exercise_04/Access Control - SJS.xml"

   - There may be some key differences with "Access Control – XQY.xml". This can be attributed to the fact that XQuery is more natural in dealing XML while being just as capable in handling JSON content.

2. In VSC or a Terminal window, navigate to the star-wars project folder:

```
$ cd /home/cent/Desktop/star-wars
```

3. Copy the prepared project changes.

```
$ cp -r /home/cent/Desktop/fundamentals/solutions/unit_03/exercise_04/corb/* .
```

   - The folder structure:

```
exercise_04/corb
│   build.gradle
│
└───src
    └───main
        └───ml-modules
            └───root
                └───corb2
                    ├───employees
                    │       envelope.sjs
                    │       envelope.xqy
                    │       uris.sjs
                    │       uris.xqy
                    │
                    └───permissions
                            transform.sjs
                            transform.xqy
                            uris.sjs
                            uris.xqy
```

4. Notice the additional configurations, and dependencies in the build.gradle file:

```
configurations {
  ...
  corb {
    attributes {
      attribute(Usage.USAGE_ATTRIBUTE, objects.named(Usage, Usage.JAVA_RUNTIME))
    }
  }
}

dependencies {
  ...
  corb "com.marklogic:marklogic-corb:2.5.6"
  corb 'com.marklogic:marklogic-xcc:11.2.0'
}
```

5. Deploy the new modules.

```
$ ./gradlew mlLoadModules
```

6. Run the first new custom Gradle task:

```
$ ./gradlew corbSetPermissions
```

   a. This makes use of modules installed in the corresponding modules database of the target port.

```
task corbSetPermissions(type: com.marklogic.gradle.task.CorbTask) {
  xccConnectionUri = "xcc://${mlUsername}:${mlPassword}@${mlHost}:${mlRestPort}"
  urisModule = "/corb2/permissions/uris.sjs"
  processModule = "/corb2/permissions/transform.sjs"
}
```

7. Use Query Console to “Explore” `star-wars-content` database.
   - Notice how the documents now have their permissions set.
8. Run the second new custom Gradle task:

```
$ ./gradlew corbSetEnvelope
```

   a. This makes use of modules as part of the filesystem (ADHOC). There is no need to deploy the modules remotely.

```
task corbSetEnvelope(type: com.marklogic.gradle.task.CorbTask) {
  xccConnectionUri = "xcc://${mlUsername}:${mlPassword}@${mlHost}:8000"
  urisModule = "src/main/ml-modules/root/corb2/employees/uris.sjs|ADHOC"
  processModule = "src/main/ml-modules/root/corb2/employees/envelope.sjs|ADHOC"
}
```

9. Use Query Console to “Explore” `Documents` database.
   - Notice how the documents now have their structure adjusted to follow the envelope pattern. Additionally, the “latitude” and “longitude” properties are now in decimal instead of string.

Challenge

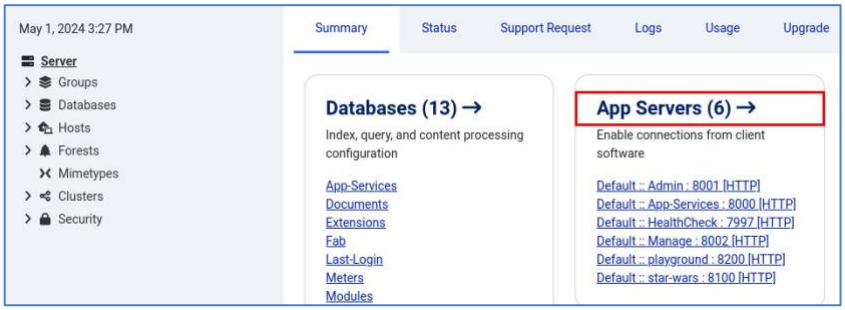
To test your familiarity with CoRB2, adjust your `top-songs` project to support a task that will:

- Check for the presence of the `released` element and add an attribute with the correct date format.
- Check for the presence of `length` element and add an attribute with the correct `xs.duration` format.

Exercise 5: (Optional) Upload Files using WebDav

This is an optional exercise on how to use MarkLogic like an FTP server.

1. Click on “App Servers” (any as shown below):

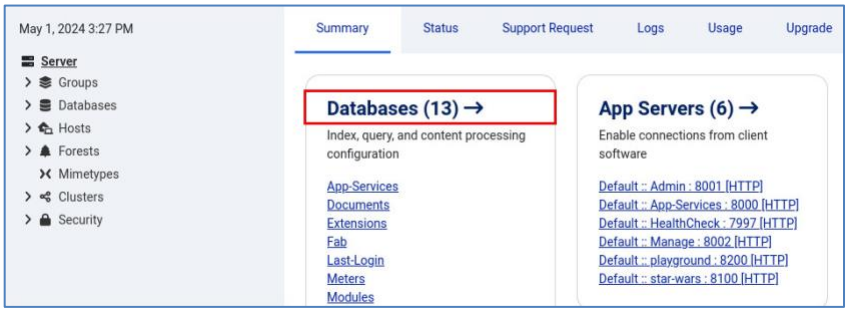


2. Click on the “Create WebDav” tab:

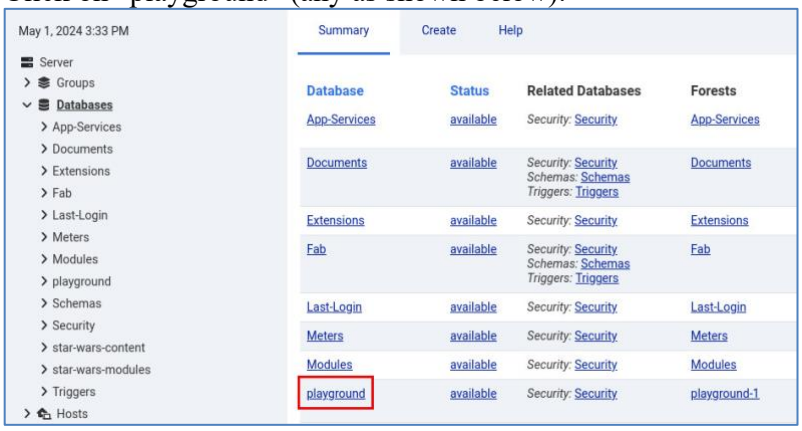
3. Use the following information and click on “OK”:

Field	Value
Server Name	playground-webdav
Root	/
Port	8201
Database	playground

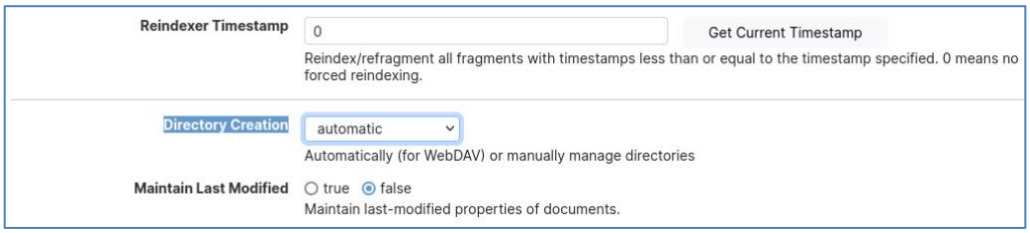
4. Return to the Server and click on “Databases” (any as shown below):



5. Click on “playground” (any as shown below):



6. Look for the “Directory Creation” option and set the value to “automatic” and click the “Ok” button to save the changes.



7. If not available, open a Terminal window (Applications→System Tools→Mate Terminal).
8. Go to your “Documents” directory:

```
$ cd /home/cent/Documents
```

9. Create a directory for our webdav mapping:

```
$ mkdir playground-webdav-mnt
```

10. Map our app server into the created directory:

```
$ sudo mount -t davfs "http://localhost:8201" playground-webdav-mnt
```

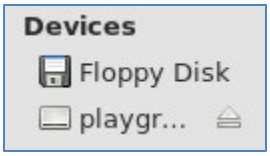
    a. When prompted, enter password for *cent* user is *centos*.

    b. When prompted, provide our MarkLogic Server credentials: admin / admin.

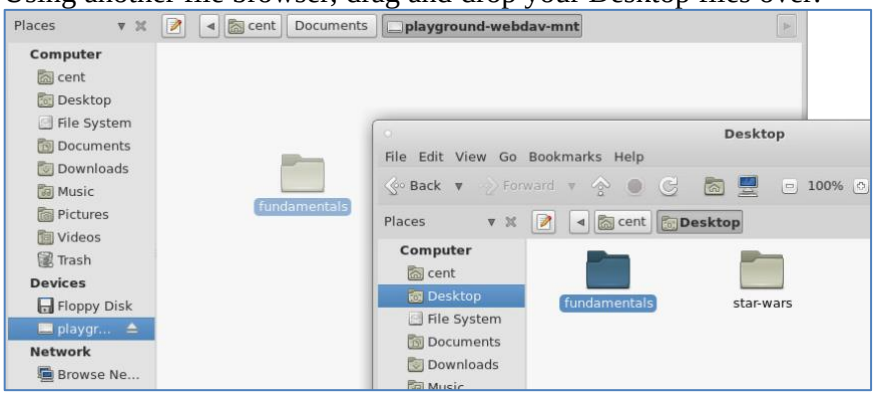
11. Change the permissions to the directory to allow user “cent” to own the folder:

```
$ sudo chown cent playground-webdav-mnt
```

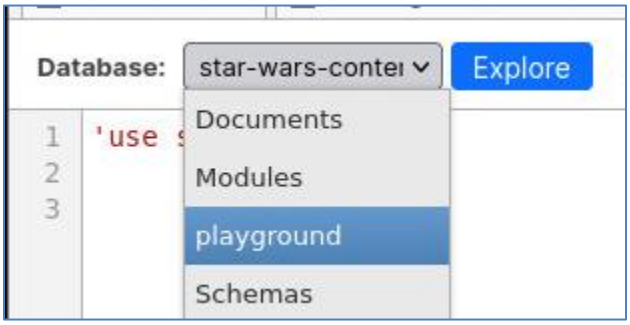
12. Using the file browser:  , under “Devices” click on the directory as shown below:



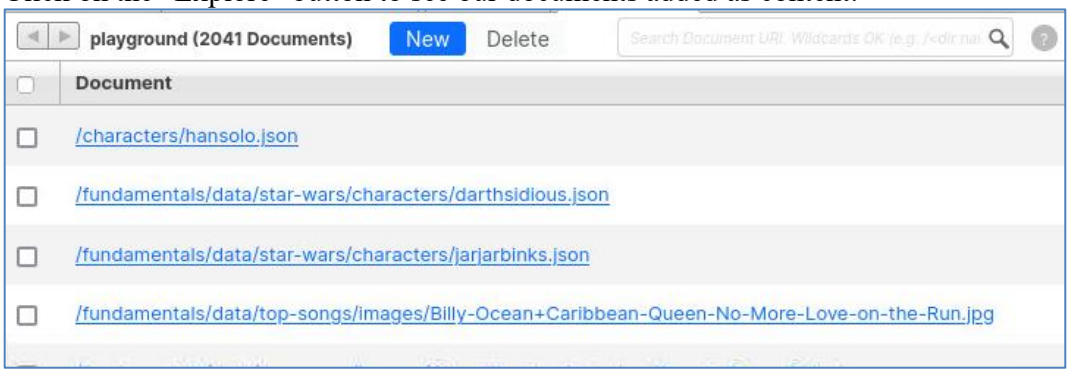
13. Using another file browser, drag and drop your Desktop files over:



14. Using your web browser, go back to Query Console: http://localhost:8000/
15. Create a new tab and change the database to playground:



16. Click on the "Explore" button to see our documents added as content:



Appendix A: Curl command for PATCH in Windows

1. This will update content of the document:

```
$ curl --anyauth -uadmin:admin -X PATCH -H "Content-Type: application/json" ^
-d ^"{^
  \"patch\": [{^
    \"insert\": {^
      \"context\": \"/decimal\",^
      \"position\": \"after\",^
      \"content\": { \"rest-child\": \"added via rest\" }^
    }^
  }]^
}^" ^
"http://localhost:8000/v1/documents?uri=/sample-data-types.json"
```

2. This will update the collection of the document:

```
$ curl --anyauth -uadmin:admin -X PATCH -H "Content-Type: application/json" ^
-d ^"{^
  \"patch\": [{^
    \"insert\": {^
      \"context\": \"/array-node('collections')\",^
      \"position\": \"last-child\",^
      \"content\": \"col5\"^
    }^
  }]^
}^" ^
"http://localhost:8000/v1/documents?uri=/sample-data-types.json^&category=collections"
```

   a. Notice that the collection was “added” to the existing collections of the document instead of replacing all of them.