FREE eBook

LEARNING marklogic

#marklogic

Table of Contents

About

Chapter 1: Getting started with marklogic

Remarks

This section provides an overview of what marklogic is, and why a developer might want to use it.

It should also mention any large subjects within marklogic, and link out to the related topics. Since the Documentation for marklogic is new, you may need to create initial versions of those related topics.

Versions

Version	URL to Download	URL to Release Notes	Release Date
8.0-5.5	Download	Rel Notes	2016-07-15
7.0-6.4	Download	Rel Notes	2015-07-15
9.0-1.1	Download	Rel Notes	2017-05-10

Examples

Installation or Setup

Detailed instructions on getting MarkLogic set up or installed can be found in the Installation Guide . The full offering of documentation is available via docs.marklogic.com

The overall setup process involves

1. Installing the binary/rpm
2. Starting the services
3. Configuring first and subsequent hosts

A "Hello World" example is not really necessary to verify setup. The setup process includes verification of success, in that once installed, the user will navigate a browser to the administrative interface. If something has been missed, the services are not started, or something went wrong with the installation process, the interface will not come up. Full step-by-step procedures are included in the document referenced.

Once installed and basic configuration is in place, a great tool to get started with interacting with the system is called "Query Console" and is available via browser, usually at port 8000. The typical URL for an instance installed on a developer's local machine is http://localhost:8000/qconsole

This powerful tool has a separate document available here.

Read Getting started with marklogic online: https://riptutorial.com/marklogic/topic/5308/getting-started-with-marklogic

Chapter 2: Marklogic 8 Features

Remarks

This section provides an overview of what's new in MarkLogic 8 and its other updated versions. Use case for each feature also needs to be added.

Examples

New features in MarkLogic 8

- Server-Side JavaScript
- Native JSON
- Samplestack Sample Application
- Temporal Documents
- REST Management
- API Improvements
- More Semantics Features, Including SPARQL 1.1, Inferencing, and SPARQL UPDATE
- Node.js Client API
- REST and Java Client API Improvements
- Enhanced HTTP Server Features
- Flexible Replication Enhancements
- Incremental Backup
- Document Library Services (DLS) Improvements
- MLCP Enhancements

Read Marklogic 8 Features online: https://riptutorial.com/marklogic/topic/5386/marklogic-8-features

Chapter 3: Search in MarkLogic

Remarks

This section provides an overview of search in MarkLogic. Intent is to cover cts:search, search:search and qbe with use cases and examples

Examples

Fetching all the documents with word "marklogic"

```
cts:search(
    fn:doc(),
    cts:word-query("marklogic"))
```

Fetching all the documents with word "marklogic", document in collection "first-collection"

This can be done in the following two ways -

```
cts:search(
    fn:collection("first-collection"),
    cts:word-query("marklogic"))
```

In this, the scope is changed from all the documents to documents in collection "first-collection" only.

In the second approach, use of cts:collection-query has been made. This should give better performance than the first approach.

```
cts:search(
    fn:doc(),
    cts:and-query((
        cts:collection-query("first-collection"),
        cts:word-query("marklogic")))
```

Fetching all the documents with a particular value of an element

This query returns all the documents with element "company" and its value as "marklogic"

```
cts:element-value-query(xs:QName('company'), 'marklogic'))
```

Checking presence of elements and attributes in documents

The following query returns the documents which have an element named "company" -

```
cts:element-value-query(
xs:QName('company'), '*', ("wildcarded")))
```

The following query returns the documents which have an element named "company" with an attribute named "name" -

```
cts:element-attribute-value-query(
xs:QName('company'), xs:QName('name'), '*', ("wildcarded")))
```

Read Search in MarkLogic online: https://riptutorial.com/marklogic/topic/5401/search-in-marklogic

Chapter 4: Very simple CRUD examples for XML documents

Examples

Create a simple document

This very simple snippet of XQuery can be executed in QueryConsole using the built-in "Documents" database as a sandbox. Each piece of the snippet has a comment to explain what the following line of code means.

```
xquery version "1.0-ml";
(: Let's first insert a simple document to get started :)
(: You need a URI- the location where the document is found in the database :)
let $uri := "/stuff/mysimpledocument.xml"
(: Documents need content. This is a simple XML node :)
let $doc :=
  <my-document>
    <body>Very simple example</body>
  </my-document>
(: Permissions are a big topic. For now, we'll use the default permissions. :)
let $permissions := xdmp:default-permissions()
(: Document collections are optional. One or more can be specified :)
(: Adding a collection for further examples that will use it. :)
let $collections := "simple-example"
(: Now we're just going to insert this document in the database :)
let $insert := xdmp:document-insert($uri,$doc,$permissions,$collections)
return <message>Document saved to {$uri}</message>
```

When this is executed, the console returns

```
<message>Document saved to /stuff/mysimpledocument.xml</message>
```

Now that the document exists, we can test additional operations...

Read/access our sample document

If we know the desired URI of the document we are looking for:

```
fn:doc("/stuff/mysimpledocument.xml")
```

Returns the full document from the database, using the URI to locate it.

Since this is XQuery, we can use XPath to find the document when we know about the structure, but not the URI:

```
/my-document
```

Returns the element "my-document" and its contents...

Update the simple document

We will now add some additional XML nodes to the "my-document" element and update the document. The snippet again contains comments to explain what is happening.

```
xquery version "1.0-ml";
(: We are preserving the same URI as we used originally :)
let $uri := "/stuff/mysimpledocument.xml"
(: Need to get the existing contents so we can append to those :)
let $orig-content := fn:doc($uri)/my-document
(: Documents need content. This is a simple XML node :)
let $new-content :=
  <notes>
    <note>Anything can be changed</note>
    <note>New content is added in this example, but we could replace it all too</note>
  </notes>
(: Now to build the new xml. There's lots of ways to do this :)
(: line 17 inserts the original contents into this new node construct :)
(: line 18 inserts the $new-content after the original contents :)
let $new-doc-content :=
  <my-document>
    {$orig-content/node()}
    {$new-content}
  </my-document>
(: Leave permissions untouched... :)
let $permissions := xdmp:document-get-permissions($uri)
(: Leave collections untouched... :)
let $collections := xdmp:document-get-collections($uri)
(: Now we're just going to insert this document in the database :)
let $insert := xdmp:document-insert($uri,$new-doc-content,$permissions,$collections)
return <message>Document {$uri} updated</message>
```

When executed, this returns the message

```
<message>Document /stuff/mysimpledocument.xml updated</message>
```

Now run the different read commands above to see the updated content. It should look like this:

```
<my-document>
    <body>Very simple example</body>
    <notes>
        <note>Anything can be changed</note>
        <note>New content is added in this example, but we could replace it all too</note>
    </notes>
</my-document>
```

Bonus: Simple Search example (Another way to read)

MarkLogic is first and foremost a search engine, so let's use two different methods to search for this document.

Using search:search()

This gives a peek into using search:search() to develop search applications. This library provides Google-like search results and will likely speed up your development of simple search tools. More information and a deeper dive can be found here.

```
xquery version "1.0-ml";
import module namespace search = "http://marklogic.com/appservices/search"
     at "/MarkLogic/appservices/search/search.xqy";
(: What is search without a keyword? :)
let $term := "very simple"
return search:search($term)
```

The result looks a bit confusing, but you can see that it returns one result, our example document.

Using cts:search()

More advanced search situations might call for more granular search capabilities. This is just to whet your appetite for what is available in search. More detailed information is found here.

```
xquery version "1.0-ml";
(: What is search without a keyword? :)
let $term := "very simple"
(: Complex queries can be made from individual cts queries. Here, we just have one simple
query :)
let $query := cts:word-query($term,"case-insensitive")
(: Return the documents that match the query :)
return cts:search(fn:doc(),$query)
```

This is an incredibly simple example. BTW, if we want to get back the URI for the matching documents, instead of the documents themselves, we can change the last line of this snippet to-

```
return
  for $result in cts:search(fn:doc(),$query)
    return fn:base-uri($result)
```

Delete - Last, but not least

To round out simple examples of CRUD operations, we present the following examples. Always use great care in deleting documents.

```
(: When we know the URI, we can delete it very easily :)
let $uri := "/stuff/mysimpledocument.xml"
return xdmp:document-delete($uri)
```

or simplified:

```
xdmp:document-delete("/stuff/mysimpledocument.xml")
```

You can certainly use XPath to find the document, get the URI of it, and then delete it with something like this, but the danger is that any documents that are returned by the XPath expressions will be removed. Not always a good thing.

```
(: Use caution when using XPath to select target docs to delete :)
for $doc in /my-document
return xdmp:document-delete(fn:base-uri($doc))
```

Want to delete all documents? This will do it, but be very careful you know what database your code will execute against.

```
for $doc in fn:doc()
return xdmp:document-delete(fn:base-uri($doc))
```

Read Very simple CRUD examples for XML documents online:
https://riptutorial.com/marklogic/topic/5498/very-simple-crud-examples-for-xml-documents

Chapter 5: Working with fn:count, xdmp:estimate and cts:frequency

Remarks

This section provides an overview of fn:count, xdmp:estimate and cts:frequency along with examples and use cases

Examples

Using fn:count() to get the number of matching documents

The XML document, I will be using throughout the examples is -

```
<a>
    <b>test-value</b>
    <d>fragment-d</d>
    <c-root>
        <d>fragment-d</d>
        <e>fragment-e</e>
    </c-root>
</a>
```

The following queries returns the number of documents with value `fragment-d` for element `d` -

- Using a cts:search

  ```
  fn:count(cts:search(fn:doc(), cts:element-value-query(xs:QName("d"), "fragment-d")))
  ```

- Using XPath

  ```
  fn:count(fn:doc()[//d="fragment-d"]))
  ```

Using xdmp:estimate() to get the number of matching documents

```
xdmp:estimate(cts:search(fn:doc(), cts:element-value-query(xs:QName("d"), "fragment-d")))
```

> xdmp:estimate can not be used on XPaths unlike fn:count is used in previous example
>
> xdmp:estimate actually gives the number of matching fragments

Counting documents when fragments are defined

The XML document to consider in this example -

```
<a>
    <b>test-value</b>
```

```
    <d>fragment-d</d>
    <c-root>
        <d>fragment-d</d>
        <e>fragment-e</e>
    </c-root>
</a>
```

A fragment root is declared on `<c-root>`

If this is the only document in the database, xdmp:estimate and fn:count are going to behave differently -

```
xdmp:estimate(cts:search(fn:doc(), cts:element-value-query(xs:QName("d"), "fragment-d")))
```

Result of the above query will be `2` (Number of fragments)

```
fn:count(cts:search(fn:doc(), cts:element-value-query(xs:QName("d"), "fragment-d")))
```

Result of the above query will be `1` (Number of documents)

> In terms of performance xdmp:estimate is much better than fn:count as it takes the advantages of indexes while resolving the search results

Read Working with fn:count, xdmp:estimate and cts:frequency online: https://riptutorial.com/marklogic/topic/6300/working-with-fn-count--xdmp-estimate-and-cts-frequency

Credits

S. No	Chapters	Contributors
1	Getting started with marklogic	Community, Harry Bakken, Tamas
2	Marklogic 8 Features	Ankit Bhardwaj
3	Search in MarkLogic	Ankit Bhardwaj
4	Very simple CRUD examples for XML documents	Harry Bakken
5	Working with fn:count, xdmp:estimate and cts:frequency	Ankit Bhardwaj