

# Unit 14

## スニペット、強調表示、ソート、ページネーション

スニペットを使用する

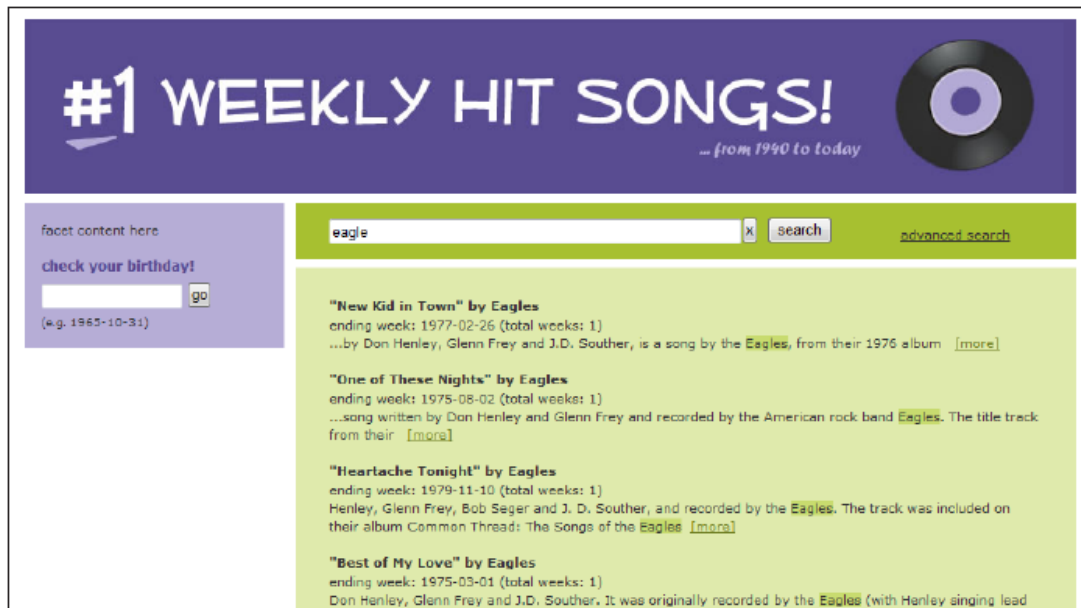
強調表示を使用する

ソートオプションを作成する

ページネーションを設定する

## スニペットを使用する

通常、検索結果にはマッチしたドキュメントの部分が表示されます。この際、検索にマッチした部分は強調表示され、また周りのテキストを表示してコンテキストを示します。このような検索結果を、スニペットと呼びます。



search API では search:response の出力としてスニペットを返します。これにより、ドキュメントのどこでマッチしていたのかを示す検索結果ページを簡単に作成できます。

```
<search:response total="53" start="1" page-length="10">
  <search:result index="1" uri="/songs/The-O-Jays+Love-Train.xml" path="fn:doc("/songs/The-O-Jays+Love-Train.xml")" score="462" confidence="0.614061" fitness="0.881754">
    <search:snippet>
      <search:match path="fn:doc("/songs/The-O-Jays+Love-Train.xml")/*:top-song/*:title">
        Love
        <search:highlight>Train</search:highlight>
      </search:match>
    </search:snippet>
  </search:result>
</search:response>
```

しかし、どのようなスニペットを表示したらよいのかは、アプリケーションごとに大きく異なります。このため、search API ではスニペットをカスタマイズできます。これにはビルトインのスニペットアルゴリズムを使うか、あるいは自分でスニペット用のコードを記述します。例えば、以下のコードでは、スニペットを返す際に descr 要素を優先させます。

```
<options xmlns="http://marklogic.com/appservices/search">
  <transform-results apply="snippet">
    <preferred-elements>
      <element ns="http://marklogic.com/MLU/top-songs" name="descr"/>
    </preferred-elements>
  </transform-results>
</options>
```

## Exercise 1: スニペットを使用する

この演習では、Top Songs 検索結果用のスニペットを設定します。

1. クエリコンソールに以下のコードを入力し（ex14-1.txt からコピーできます）、[Run]ボタンをクリックします。これで、search API がデフォルトで返すものを確認できます。

```
import module namespace search = "http://marklogic.com/appservices/search" at
"/MarkLogic/appservices/search/search.xqy";

search:search("train")
```

2. 返された要素を確認してください。search:response、search:result、search:snippet、search:match、search:highlight があります。
3. クエリコンソールで、transform-results ならびに descr を優先要素とすることを含む options ノードを作成します。

```
import module namespace search = "http://marklogic.com/appservices/search" at
"/MarkLogic/appservices/search/search.xqy";

declare variable $options :=
  <options xmlns="http://marklogic.com/appservices/search">
    <transform-results apply="snippet">
      <preferred-elements>
        <element ns="http://marklogic.com/MLU/top-songs" name="descr"/>
      </preferred-elements>
    </transform-results>
  </options>;

search:search("train", $options)
```

4. [Run]ボタンをクリックして結果を見ます。検索は依然としてドキュメント内の要素に対して行われていますが、スニペットは descr 要素だけから返されています。
5. search:result 要素に uri 属性があることを確認します。これを使って、曲のコンテンツの残りすべてにアクセスしてみます。
6. テキストエディタで index.xqy ファイルを開きます。7 行めあたりにある<transform-results apply="raw" />要素を以下のように置き換えます（クエリコンソールからコピーできます）。

```
declare variable $options :=
  <options xmlns="http://marklogic.com/appservices/search">
    <transform-results apply="snippet">
```

```

    <preferred-elements>
      <element ns="http://marklogic.com/MLU/top-songs" name="descr"/>
    </preferred-elements>
  </transform-results>
</options>;

```

7. 37 行めあたりにある `search-results` 関数を編集し、`description` 関数を作成します (ex14-1c.txt からコピーできます)。これによりオプションが変更された search API の結果を受け取ることができます (つまりこれまでの `<transform-results apply="raw">` の代わりにスニペットの結果を受け取れるようになります)。

```

declare function local:search-results()
{
  let $q := xdmp:get-request-field("q")
  let $results :=
    for $song in search:search($q, $options)/search:result
    let $uri := fn:data($song/@uri)
    let $song-doc := fn:doc($uri)
    return
      <div>
        <div class="songname">{"$song-doc//ts:title/text()}"
          by {"$song-doc//ts:artist/text()"}</div>
        <div class="week"> ending week:
          {fn:data($song-doc//ts:weeks/@last)}
          (total weeks: {fn:count($song-doc//ts:weeks/ts:week)})</div>
        {if ($song-doc//ts:genres/ts:genre)
          then <div class="genre">genre:
            {fn:lower-case(fn:string-join(($song-doc//ts:genres/ts:genre), ", ")}
          </div>
          else ()}
        <div class="description">{local:description($song)}
          &#160;<a href="index.xqy?uri={xdmp:url-encode($uri)}">[more]</a>
        </div>
      </div>
  return
    if($results)
    then $results
    else <div>Sorry, no results for your search.</div>
};

declare function local:description($song)
{
  for $text in $song/search:snippet/search:match/node()
  return $text
};

```

8. このファイルを保存し、ブラウザに再読み込みします。
9. ブラウザでテストします。「significant」という語を検索します。
10. 説明部分に「significant」という語を含むスニペットが表示されていることがわかります。

## 強調表示を使用する

結果のスニペット内に強調表示を使用するには、search API の search:highlight 要素を使用します。これにより自動的にスニペットが強調表示されます。

```
<search:result index="1" uri="/songs/The-O-Jays+Love-Train.xml"
path="fn:doc("/songs/The-O-Jays+Love-Train.xml")" score="480" confidence="0.631514"
fitness="0.91987">
  <search:snippet>
    <search:match path="fn:doc("/songs/The-O-Jays+Love-Train.xml")/*:top-
song/*:descr/*:p[1]">"Love <search:highlight>Train</search:highlight>" is a song by
Kenny Gamble and Leon Huff that was a hit record...
    </search:match>
    <search:match path="fn:doc("/songs/The-O-Jays+Love-Train.xml")/*:top-
song/*:descr/*:p[2]">
      Besides its release as a single, "Love
      <search:highlight>Train</search:highlight>
      " was the last
      song on The O'Jays' album
    </search:match>
  </search:snippet>
</search:result>
```

search:highlight の値は、CSS コードで容易に参照でき、強調表示されます。

```
for $text in $song/search:snippet/search:match/node()
return
  if(fn:node-name($text) eq xs:QName("search:highlight"))
  then <span class="highlight">{$text/text()}</span>
  else $text
```

## Exercise 2: 強調表示を使用する

この演習では、Top Songs の index.xqy クエリページの結果において、検索語を強調表示します。

1. クエリコンソールで、前回の演習で使ったコードがあるタブを開きます (ex14-2.txt からコピーできます)。[Run]ボタンをクリックします。

```
import module namespace search = "http://marklogic.com/ appservices/search" at
"/MarkLogic/appservices/search/search.xqy";

declare variable $options :=
<options xmlns="http://marklogic.com/appservices/search">
  <transform-results apply="snippet">
    <preferred-elements>
      <element ns="http://marklogic.com/MLU/top-songs" name="descr"/>
    </preferred-elements>
  </transform-results>
</options>;

search:search("train", $options)
```

2. search:highlight 要素を確認してください。

```
"Love
<search:highlight>Train</search:highlight>
" is a song by Kenny Gamble and Leon Huff that was a hit record...
```

3. エディタ内で description 関数にコードを追加し、説明 (description) 部分で検索語が強調表示されるようにします。

```
declare function local:description($song)
{
  for $text in $song/search:snippet/search:match/node()
  return
    if (fn:node-name($text) eq xs:QName("search:highlight"))
    then <span class="highlight">{$text/text()}</span>
    else $text
};
```

4. index.xqy を保存し、ブラウザで再読み込みします。
5. 「train」という語で検索します。
6. 説明部分で、「train」がすべて強調表示されていることがわかります。

## ソートオプションを作成する

search:search()のような検索式では、デフォルトでは結果は関連度順に返されます。XPath 式では、デフォルトではデータベース内での出現順に返されます。XQuery 式で「order by」ステートメント (Unit 4 で紹介しています) を使用したり、あるいは search API のオプションで search:search() リクエストを使用することで、結果の表示順を変更できます。

search API で結果の順序を指定する場合には、options ノードを使います。例えば、以下の options ノードでは、結果を title または artist の順番で表示します。

```
<options xmlns="http://marklogic.com/appservices/search">
  <search:operator name="sort">
    <search:state name="title">
      <search:sort-order direction="ascending" type="xs:string">
        <search:element ns="http://marklogic.com/MLU/top- songs" name="title"/>
      </search:sort-order>
      <search:sort-order>
        <search:score/>
      </search:sort-order>
    </search:state>
    <search:state name="artist">
      <search:sort-order direction="ascending" type="xs:string">
        <search:element ns="http://marklogic.com/MLU/top-songs" name="artist"/>
      </search:sort-order>
      <search:sort-order>
        <search:score/>
      </search:sort-order>
    </search:state>
  </search:operator>
</options>
```

この後、以下の式を使って「train」の検索結果を title 順で並べることができます。

```
train sort:title
```

一方、以下の式では結果を artist 順で並べます。

```
train sort:artist
```

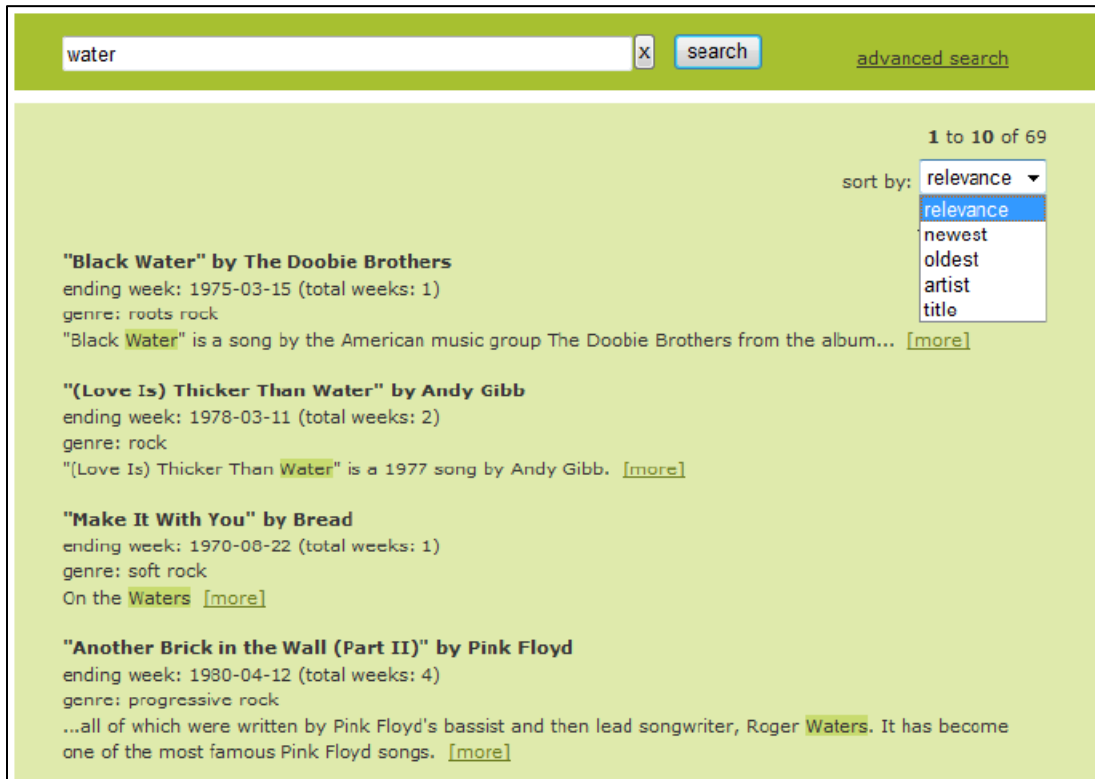
## レンジインデックス

search API の options ノードを使用する際には、参照されるすべての要素や属性にレンジインデックスが必要です。インデックスのユニットで説明したように、レンジインデックスは管理画面の Configure - Databases - top-songs - Element (あるいは Attribute) Range Indexes から設定できます。

search API のソートで使用するレンジインデックスでは、「scalar type」が「string」に設定されている場合、デフォルトのコレーション (<http://marklogic.com/collation/> など) を持つレンジインデックスが必要です。

## 検索結果のソートを理解する

ソート順を変更しても、検索で返される結果の数は変わりません。これは単に表示順が変わるだけです。例えば「water」を検索した場合、「water」という語を含むすべてのドキュメントが返されます。ドキュメントの表示順は変更できます (関連度順の代わりに、artist 順にするなど) が、「water」を含むドキュメントの総数は変わりません。



The screenshot shows a search interface with a search bar containing the text "water". To the right of the search bar is a button labeled "search" and a link labeled "advanced search". Below the search bar, the results are displayed on a light green background. At the top right of the results area, it says "1 to 10 of 69". Below this, there is a "sort by:" dropdown menu with the following options: "relevance" (selected), "relevance", "newest", "oldest", "artist", and "title". The search results are listed below the dropdown menu. Each result includes the song title, the artist, the ending week, the total weeks, the genre, and a brief description of the song. The results are as follows:

- "Black Water" by The Doobie Brothers**  
ending week: 1975-03-15 (total weeks: 1)  
genre: roots rock  
"Black Water" is a song by the American music group The Doobie Brothers from the album... [\[more\]](#)
- "(Love Is) Thicker Than Water" by Andy Gibb**  
ending week: 1978-03-11 (total weeks: 2)  
genre: rock  
"(Love Is) Thicker Than Water" is a 1977 song by Andy Gibb. [\[more\]](#)
- "Make It With You" by Bread**  
ending week: 1970-08-22 (total weeks: 1)  
genre: soft rock  
On the Waters [\[more\]](#)
- "Another Brick in the Wall (Part II)" by Pink Floyd**  
ending week: 1980-04-12 (total weeks: 4)  
genre: progressive rock  
...all of which were written by Pink Floyd's bassist and then lead songwriter, Roger Waters. It has become one of the most famous Pink Floyd songs. [\[more\]](#)



### Exercise 3: ソートオプションを作成する

この演習では、search API の options を使って、Top Songs アプリケーションの検索結果用に「ソート」のドロップダウンメニューを作ります。

1. **index.xqy** の冒頭の方で、options ノード内に「sort」という名前の演算子を作成します (**ex14-3.txt** からコピーできます)。

```
declare variable $options :=
<options xmlns="http://marklogic.com/appservices/search">
  <transform-results apply="snippet">
    <preferred-elements>
      <element ns="http://marklogic.com/MLU/top-songs" name="descr"/>
    </preferred-elements>
  </transform-results>
  <search:operator name="sort">
    <search:state name="relevance">
      <search:sort-order direction="descending">
        <search:score/>
      </search:sort-order>
    </search:state>
    <search:state name="newest">
      <search:sort-order direction="descending" type="xs:date">
        <search:attribute ns="" name="last"/>
        <search:element ns="http://marklogic.com/MLU/top-songs" name="weeks"/>
      </search:sort-order>
      <search:sort-order>
        <search:score/>
      </search:sort-order>
    </search:state>
    <search:state name="oldest">
      <search:sort-order direction="ascending" type="xs:date">
        <search:attribute ns="" name="last"/>
        <search:element ns="http://marklogic.com/MLU/top-songs" name="weeks"/>
      </search:sort-order>
      <search:sort-order>
        <search:score/>
      </search:sort-order>
    </search:state>
    <search:state name="title">
      <search:sort-order direction="ascending" type="xs:string">
        <search:element ns="http://marklogic.com/MLU/top-songs" name="title"/>
      </search:sort-order>
      <search:sort-order>
        <search:score/>
      </search:sort-order>
    </search:state>
    <search:state name="artist">
      <search:sort-order direction="ascending" type="xs:string">
```

```

        <search:element ns="http://marklogic.com/MLU/top- songs" name="artist"/>
      </search:sort-order>
    <search:sort-order>
      <search:score/>
    </search:sort-order>
  </search:state>
</search:operator>
</options>;

```

2. 演算子の名前は「sort」になっています。また state として「relevance」、「newest」、「oldest」、「title」、「artist」があります。各 state には、楽曲ドキュメントに対応する名前の search:element があります（「artist」という state には、「artist」という名前の search:element があります）。
3. 「newest」という state にある、search:attribute 要素と search:element 要素の name 属性を確認してください。
4. 「newest」の search:sort-order には、direction (descending) と type (xs:date) があることを確認してください。
5. weeks 要素の last 属性用に作成したインデックス (scalar type は date) が、「newest」ならびに「oldest」という検索 state に対応していることがわかります。
6. **artist** 用の要素レンジインデックスを設定します。管理画面で **Configure - Databases - top-songs - Element Range Indexes** に移動し、[Add]タブを選択します。scalar type として **string** を選択します。artist 要素の名前空間を入力し、コレーションはデフォルトのままにします。

**range element index** -- An index for fast element inequality comparisons.
 delete

**scalar type**

string ▼

An atomic type specification.

**namespace uri**

http://marklogic.com/MLU/top-songs

A namespace URI.

**localname**

artist

One or more localnames.

**collation**

http://marklogic.com/collation/

Root Collation ▼

collation builder

A collation URI for string comparisons.

**range value positions**

☐ true
 ☒ false

Index range value positions for faster near searches involving range queries (slower document loads and larger database files).

**invalid values**

reject ▼

Allow ingestion of documents that do not have matching type of data.

7. ステップ 6 を繰り返し、**title** 要素用のレンジインデックスを作成します。コレーションはデフォルトのままにします。
8. このデータベースの[Status]タブで、再インデックス付けが完了したことを確認します。
9. index.xqy ファイルを**保存**し、これまでの作業を確認します。検索フィールドに、「train」の後に sort 演算子ならびに state の名前を 1 つ（「oldest」など）を付けて入力します。[search]ボタンをクリックします。



10. 「train」を含むドキュメントが古い順（「oldest」から）で表示されます。
11. このクエリを以下のように変更します。

```
train sort:artist
```

12. [search]ボタンをクリックします。今度はドキュメントが **artist** のアルファベット順で並んでいることがわかります。
13. エンドユーザーにソートのドロップダウンメニューを提供します。ex14-3b.txt からソート関数 3 つをコピーし、**search-results** 関数の前に貼り付けます。

```
(: gets the current sort argument from the query string :)
declare function local:get-sort($q){
  fn:replace(fn:tokenize($q," ") [fn:contains(., "sort")], "[()]" , "")
};

(: adds sort to the search query string :)
declare function local:add-sort($q){
  let $sortby := local:sort-controller()
  return
    if($sortby)
    then
      let $old-sort := local:get-sort($q)
      let $q :=
        if($old-sort)
        then search:remove-constraint($q,$old-sort,$options)
        else $q
      return fn:concat($q," sort:",$sortby)
    else $q
};

(: determines if the end-user set the sort through the drop-down or through editing
the search text field :)
declare function local:sort-controller(){
  if(xdmp:get-request-field("submitbtn") or not(xdmp:get-request-field("sortby")))
```

```

    then
      let $order := fn:replace(fn:substring-after(fn:tokenize(xdmp:get-request-
field("q","sort:newest"), " ") [fn:contains(., "sort")], "sort:"), "[()]", "")
      return
        if(fn:string-length($order) lt 1)
        then "relevance"
        else $order
    else xdmp:get-request-field("sortby")
  };

(: builds the sort drop-down with appropriate option selected :)
declare function local:sort-options(){
  let $sortby := local:sort-controller()
  let $sort-options :=
    <options>
      <option value="relevance">relevance</option>
      <option value="newest">newest</option>
      <option value="oldest">oldest</option>
      <option value="artist">artist</option>
      <option value="title">title</option>
    </options>
  let $newsortoptions :=
    for $option in $sort-options/*
    return
      element {fn:node-name($option)}
      {
        $option/@*,
        if($sortby eq $option/@value)
        then attribute selected {"true"} else (),
        $option/node()
      }
  return
    <div id="sortbydiv">
      sort by:
      <select name="sortby" id="sortby" onchange='this.form.submit()'>
        {$newsortoptions}
      </select>
    </div>
};

```

14. `search-results` 関数に、`add-sort` 関数と `sort-options` 関数を呼び出すコードを貼り付けます (14-3c.txt からコピーできます)。

```
declare function local:search-results()
{
  let $q := local:add-sort(xdmp:get-request-field("q"))
  let $results :=
    for $song in search:search($q, $options)/search:result
    let $uri := fn:data($song/@uri)
    let $song-doc := fn:doc($uri)
    return <div>
      <div class="songname">{"$song-doc//ts:title/text()}" by {"$song-
doc//ts:artist/text()"}</div>
      <div class="week"> ending week:
        {fn:data($song-doc//ts:weeks/@last)} (total weeks:
        {fn:count($song-doc//ts:weeks/ts:week)})</div>
        {if ($song//ts:genres/ts:genre) then <div class="genre">genre: {fn:lower-
case(fn:string-join(($song-doc//ts:genres/ts:genre), ", " ))}
        </div> else ()}
      <div class="description">{local:description($song)}
      &#160;<a href="index.xqy?uri={xdmp:url-encode($uri)}">[more]</a>
    </div>
  </div>
  return
    if($results)
    then (local:sort-options(), $results)
    else <div>Sorry, no results for your search.</div>
};
```

15. `index.xqy` を保存し、ブラウザでテストします。「midnight」 (あるいは他の語) で検索します。
16. 今度はソートのドロップダウンが表示されています (検索結果とともに)。
17. `artist` (あるいは他のソートオプション) を選択します。
18. 結果が `artist` 順になったことを確認します。
19. `index.xqy` の最後の方で、検索フィールドの値を変更し、検索フィールド内にソート演算子が表示されるようにします。

```
<input type="text" name="q" id="q" size="50" value="{local:add-sort(xdmp:get-request-
field("q"))}"/>
```

20. これを保存し、ブラウザでテストします。

## ページネーションを設定する

検索で数百万件がヒットするような場合、すべての結果を web ページに表示させるのは現実的ではありません。最も関連性が高い結果は、リストの冒頭に表示されるべきです。通常ユーザーが必要な結果は、最初の 10 から 30 個にあるでしょう。

The screenshot shows a search interface with a search bar containing 'beatles', a search button, and an 'advanced search' link. Below the search bar, there are pagination controls: '1 to 10 of 91', a 'relevance' dropdown menu, and a set of numbered links (1, 2, 3, 4, 5) with a right arrow. The search results are displayed in a list format, with each item showing the song title, ending week, and a brief description. The first result is 'She Loves You' by The Beatles, and the second is 'Can't Buy Me Love' by The Beatles. Each result has a '[more]' link at the end of the description.

クエリ側のページネーションでは、ユーザーがサブミットする検索リクエストに、返されるページに関する情報も含まれています。その後、サーバーはこの結果の長さ、結果の開始位置、この検索基準にマッチした項目の想定数などを返します。

## search API : 合計、開始位置、ページ長

search API が返す結果には、合計、開始位置、ページ長などが含まれています。これらの値はエンドユーザーへのインターフェイスで活用できます。

```
<search:response total="91" start="1" page-length="10">
  <search:result index="1" uri="/songs/The-Beatles+She-Loves-You.xml" path="fn:doc('/songs/The-Beatles+She-Loves-You.xml')
    score="459" confidence="0.617545" fitness="0.967909">
    <search:snippet>
      <search:match path="fn:doc('/songs/The-Beatles+She-Loves-You.xml')/*top-song/*artist">
        The
        <search:highlight>Beatles</search:highlight>
      </search:match>
```

開始位置を指定するには、search:search()関数の 3 つめのパラメータ（オプション）を使用します。例えば以下のコードは、11 番めから 20 番めのドキュメントを返します。

```
search:search("beatles", (), 11)
```

## Exercise 4: ページネーションを設定する

この演習では、Top Songs アプリケーション用のページネーションを設定します。

1. クエリコンソールに以下のコードを入力し、[Run]ボタンをクリックします。

```
import module namespace search = "http://marklogic.com/appservices/search" at
"/MarkLogic/appservices/search/search.xqy";

search:search("beatles")
```

2. search:response 要素に、total、start、page-length 属性があることを確認してください。
3. コードを修正し、開始パラメータに値を渡します。

```
search:search("beatles", (), 41)
```

4. [Run]ボタンをクリックし、search:response 属性を確認します。
5. ex14-4.txt から、ページネーション関数をコピーします。
6. index.xqy で、search-results 関数の前 (148 行めあたり) にページネーション関数を貼り付けます。

```
declare function local:pagination($resultspag)
{
  let $start := xs:unsignedLong($resultspag/@start)
  let $length := xs:unsignedLong($resultspag/@page-length)
  let $total := xs:unsignedLong($resultspag/@total)
  let $last := xs:unsignedLong($start + $length - 1)
  let $end := if ($total > $last) then $last else $total
  let $qtext := $resultspag/search:qtext[1]/text()
  let $next := if ($total > $last) then $last + 1 else ()
  let $previous := if (($start > 1) and ($start - $length > 0)) then fn:max(($start - $length), 1) else ()
  let $next-href :=
    if ($next)
    then fn:concat("/index.xqy?q=", if ($qtext) then fn:encode-for-uri($qtext)
else (), "&start=", $next, "&submitbtn=page")
    else ()
  let $previous-href :=
    if ($previous)
    then fn:concat("/index.xqy?q=", if ($qtext) then fn:encode-for-uri($qtext)
else (), "&start=", $previous, "&submitbtn=page")
    else ()
  let $total-pages := fn:ceiling($total div $length)
  let $currpage := fn:ceiling($start div $length)
  let $pagemin :=
    fn:min(for $i in (1 to 4)
    where ($currpage - $i) > 0
    return $currpage - $i)
```

```

let $rangestart := fn:max(($pagemin, 1))
let $rangeend := fn:min(($total-pages,$rangestart + 4))

return (
  <div id="countdiv"><b>{$start}</b> to <b>{$end}</b> of {$total}</div>,
  local:sort-options(),
  if($rangestart eq $rangeend)
  then ()
  else
    <div id="pagenumdiv">
      { if ($previous) then <a href="{ $previous-href}" title="View previous
{$length} results"></a> else () }
      {
        for $i in ($rangestart to $rangeend)
        let $page-start := (($length * $i) + 1) - $length
        let $page-href := concat("/index.xqy?q=",if ($qtext) then encode-for-
uri($qtext) else (), "&start=", $page-start, "&submitbtn=page")
        return
          if ($i eq $currpage)
          then <b>#{160}<u>{$i}</u>#{160}</b>
          else <span class="hspace">#{160}<a href="{ $page-
href}">{$i}</a>#{160}</span>
      }
      { if ($next) then <a href="{ $next-href}" title="View next {$length}
results"></a> else
() }
    }
  </div>
)
};

```

7. search-results 関数を編集し、ページネーション関数を呼び出すようにします (ex14-4b.txt からコピーできます)。

```

declare function local:search-results()
{
  let $start := xs:unsignedLong(xdmp:get-request-field("start"))
  let $q := local:add-sort(xdmp:get-request-field("q"))
  let $results := search:search($q, $options, $start)
  let $items :=
    for $song in $results/search:result
    let $uri := fn:data($song/@uri)
    let $song-doc := fn:doc($uri)
    return
      <div>
        <div class="songname">{"$song-doc//ts:title/text()}" by {"$song-
doc//ts:artist/text()"}</div>
        <div class="week"> ending week: {fn:data($song-doc//ts:weeks/@last)}
        (total weeks: {fn:count($song-doc//ts:weeks/ts:week)})</div>
        {if ($song-doc//ts:genres/ts:genre) then <div class="genre">genre:
{fn:lower-case(fn:string-join(($song-doc//ts:genres/ts:genre)," ")}</div> else ()}
        <div class="description">{local:description($song)}#{160};
        <a href="index.xqy?uri={xdmp:url-encode($song/@uri)}">[more]</a>
      </div>

```



```
        </div>
    return
    if($items)
    then (local:pagination($results), $items)
    else <div>Sorry, no results for your search.<br/><br/><br/></div>
};
```

8. `index.xqy` を保存し、ブラウザでテストします。「beatles」（あるいは他の語）で検索します。
9. ページングのコントロールが、検索結果に表示されるようになったことがわかります。
10. URL に、「start」という名前の変数があります。
11. `index.xqy` の `pagination` 関数（160 行めあたり）にある、`[next]` ボタンがクリックされると `start` 変数の値を変更するコードを確認します。

```
then concat("/index.xqy?q=",if ($qtext) then encode-for-uri($qtext) else
(), "&start=", $next, "&submitbtn=page")
```

12. 4 行下にある、`previous`（戻る）矢印がクリックされると `start` 変数を設定するコードを確認します。

```
then concat("/index.xqy?q=",if ($qtext) then encode-for-uri($qtext) else
(), "&start=", $previous, "&submitbtn=page")
```

13. 177 行めあたりにある、`local:sort-options()` への呼び出しを確認します。

`index.xqy` ファイルをリファクタリングされたバージョンで置き換えます。ここでは、2 つの関数をライブラリモジュールに移動しています。このライブラリモジュールはトップページにソートとページネーションのオプションを表示し、また誕生日の検索結果において楽曲の詳細を表示させる（「more」）のものです。

14. `mls-projects/top-songs` 内の `index.xqy` の名前を、`index_backup_1.xqy` に変更します。
15. エクスプローラーで、`/home/cent/Desktop/mls-developer/unit14/refactored` 内の `index.xqy` ファイルと `modules` フォルダを、`mls-projects/top-songs` フォルダにコピーします。
16. ブラウザで `http://localhost:8040` を更新します。依然としてこのアプリケーションがちゃんと動くことを確認します。
17. トップページに、ページネーションとソートのオプションが表示されていることを確認します。

18. テキストエディタで **index.xqy** と **modules/display-lib.xqy** を開き、description 関数と display-song-detail 関数がライブラリモジュールに移動していることを確認します。