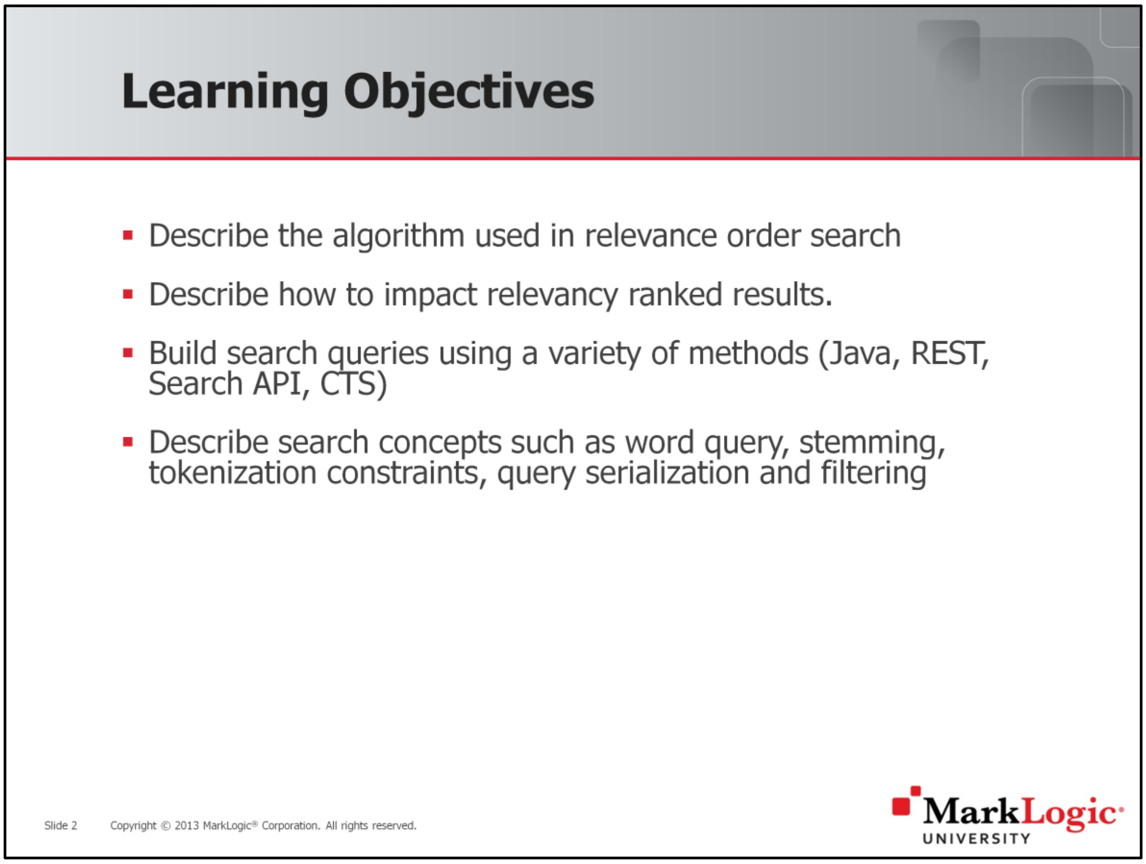
Unit 8: Search Fundamentals

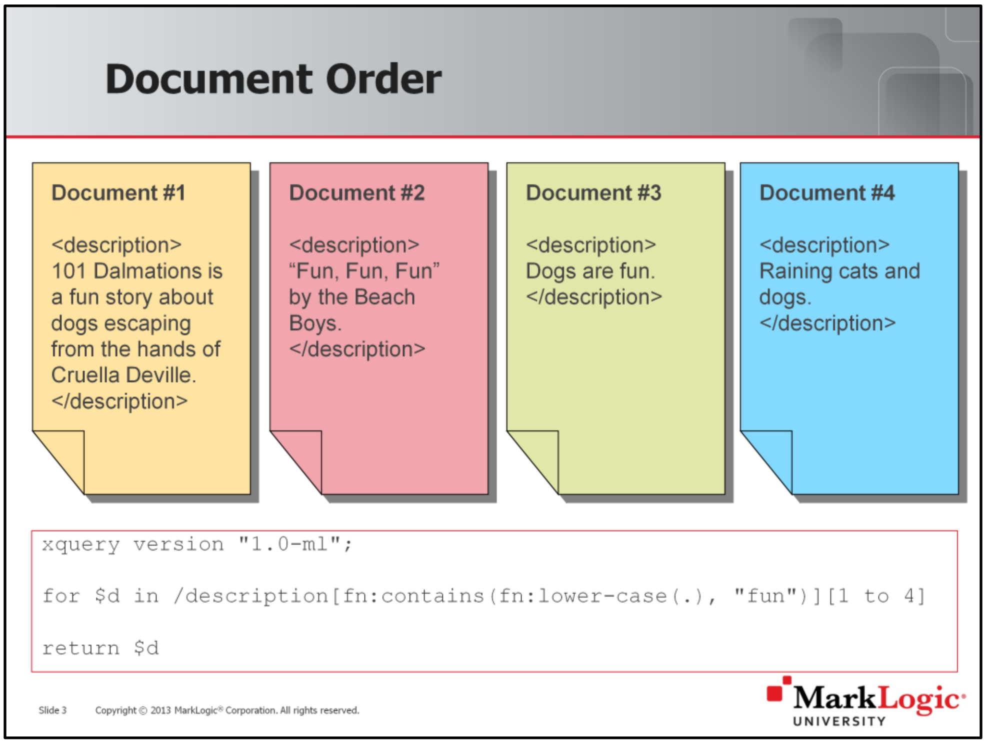
Slide 1 Copyright © 2013 MarkLogic® Corporation. All rights reserved.



Learning Objectives

- Describe the algorithm used in relevance order search
- Describe how to impact relevancy ranked results.
- Build search queries using a variety of methods (Java, REST, Search API, CTS)
- Describe search concepts such as word query, stemming, tokenization constraints, query serialization and filtering

Slide 2 Copyright © 2013 MarkLogic® Corporation. All rights reserved.



Document Order

Document #1

<description>
101 Dalmations is a fun story about dogs escaping from the hands of Cruella Deville.
</description>

Document #2

<description>
“Fun, Fun, Fun” by the Beach Boys.
</description>

Document #3

<description>
Dogs are fun.
</description>

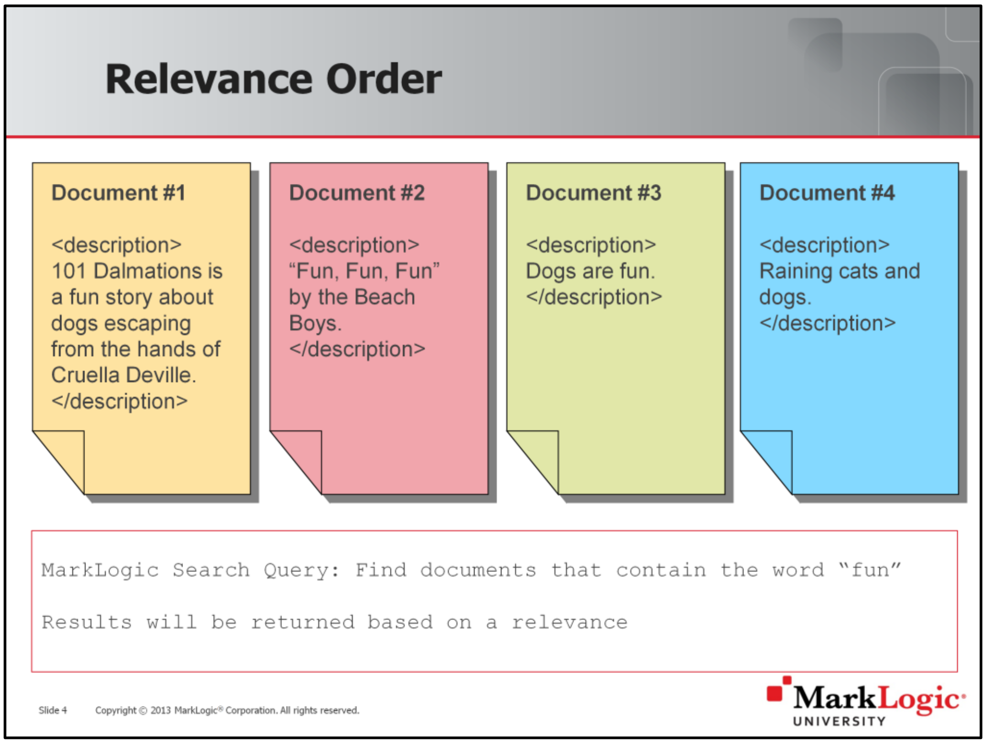
Document #4

<description>
Raining cats and dogs.
</description>

```
xquery version "1.0-ml";

for $d in /description[fn:contains(fn:lower-case(.), "fun")][1 to 4]

return $d
```

Slide 3 Copyright © 2013 MarkLogic® Corporation. All rights reserved.

XPath queries return results in document order, meaning the first X amount of matches found will be what is returned. This doesn’t mean that the best matching documents will be higher in the result set.



Relevance Order

Document #1

<description>
101 Dalmations is a fun story about dogs escaping from the hands of Cruella Deville.
</description>

Document #2

<description>
"Fun, Fun, Fun" by the Beach Boys.
</description>

Document #3

<description>
Dogs are fun.
</description>

Document #4

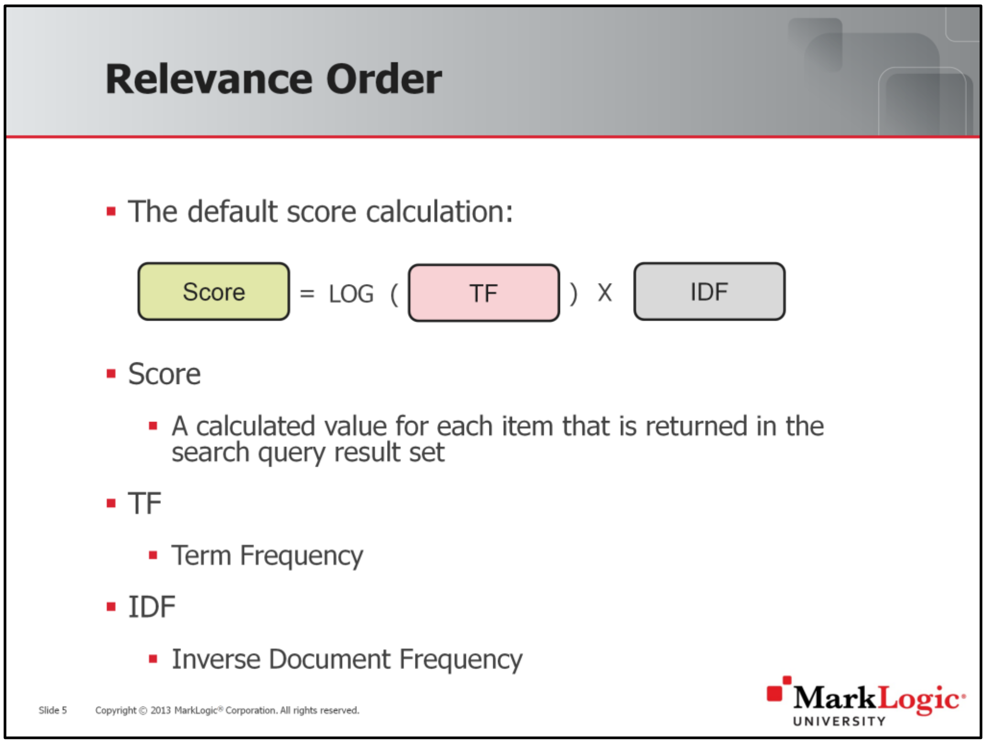
<description>
Raining cats and dogs.
</description>

```
MarkLogic Search Query: Find documents that contain the word "fun"

Results will be returned based on a relevance
```

Slide 4 Copyright © 2013 MarkLogic® Corporation. All rights reserved.

Search in MarkLogic operates like a search engine more so than a standard database matching query.

Search queries in MarkLogic return results in relevance order. Relevance is a statistical measure of how important a word is to a document within the context of an entire collection of documents.



Relevance Order

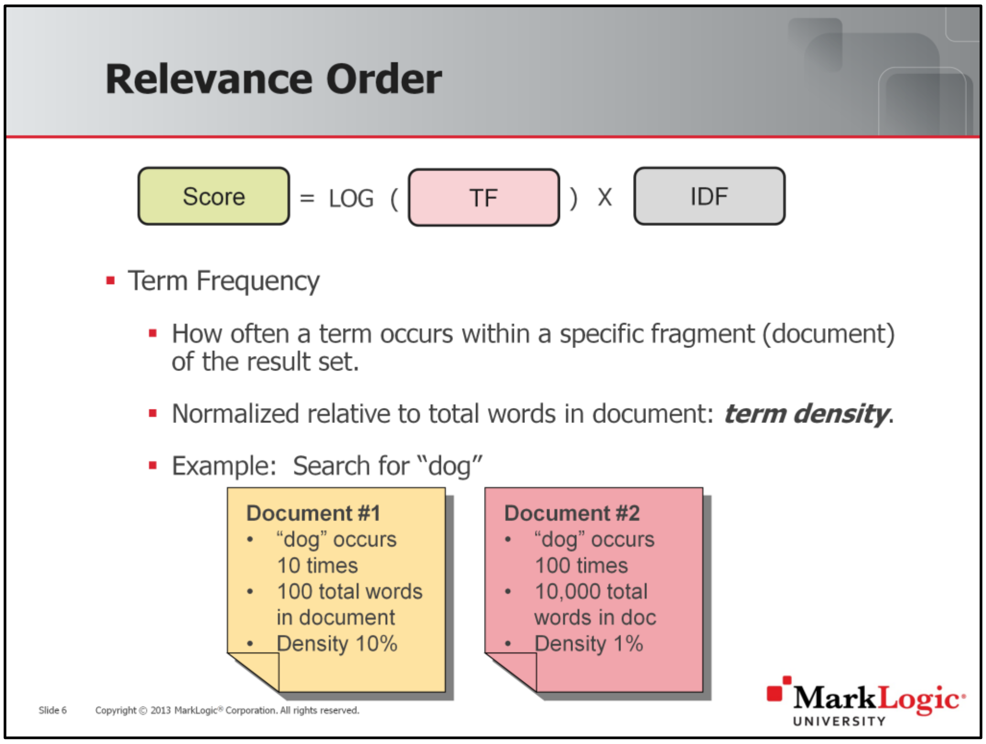
- The default score calculation:

$$\boxed{\text{Score}} = \text{LOG}\left(\boxed{\text{TF}}\right) \times \boxed{\text{IDF}}$$

- Score
  - A calculated value for each item that is returned in the search query result set
- TF
  - Term Frequency
- IDF
  - Inverse Document Frequency

Slide 5 Copyright © 2013 MarkLogic® Corporation. All rights reserved.

When a search is executed, MarkLogic server produces a result set of items that match the search expression. Each of these items contains a score, which is a calculated value which by default uses the TF / IDF formula. Ordering items in the result set by score provides search results based on relevancy.



Relevance Order

Score = LOG (TF) X IDF

- Term Frequency
  - How often a term occurs within a specific fragment (document) of the result set.
  - Normalized relative to total words in document: ***term density***.
  - Example: Search for "dog"

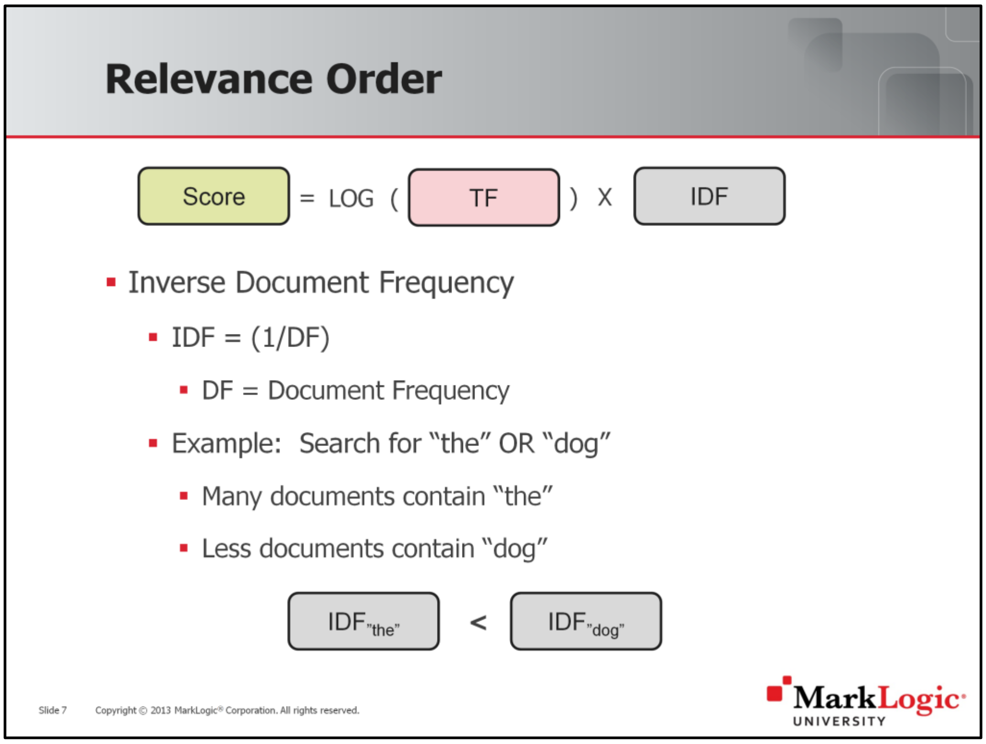
Document #1
- "dog" occurs 10 times
- 100 total words in document
- Density 10%

Document #2
- "dog" occurs 100 times
- 10,000 total words in doc
- Density 1%

Slide 6 Copyright © 2013 MarkLogic® Corporation. All rights reserved.

Term Frequency represents how often the query term appears in a document relative to the total words in the document. Because by default TF is understood in terms of the overall document, we say TF is normalized. In other words, the density of the search term based on the total terms in the document is considered in the calculation. Greater density = greater TF factor = greater relevance.

Note: For the scope of this example we assume that a 1:1 relationship exists between document:fragment. Depending on your implementation of fragmentation settings in MarkLogic, this may or may not be the case. Understand that relevancy calculations are based on the fragment.



Relevance Order

Score = LOG (TF) X IDF

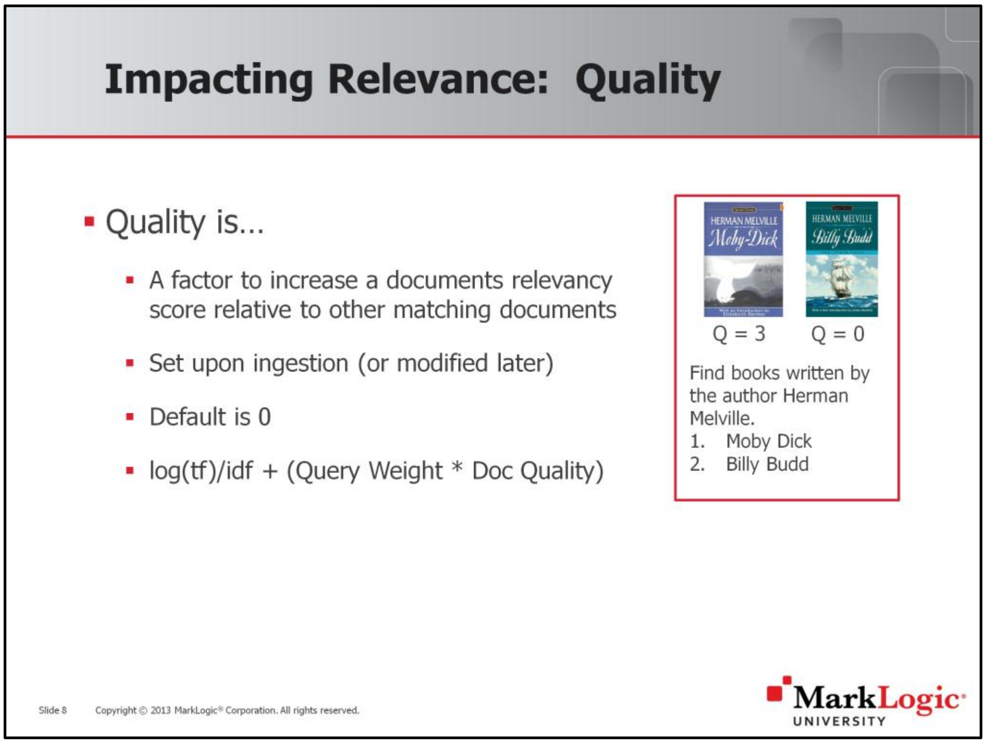
- Inverse Document Frequency
  - IDF = (1/DF)
    - DF = Document Frequency
  - Example: Search for "the" OR "dog"
    - Many documents contain "the"
    - Less documents contain "dog"

$IDF_{\text{"the"}} < IDF_{\text{"dog"}}$

Slide 7 Copyright © 2013 MarkLogic® Corporation. All rights reserved.

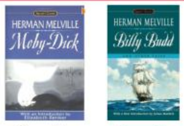
MarkLogic UNIVERSITY

Inverse Document Frequency provides a factor based on the total number of documents in which a term occurs. If the term occurs in a large number of documents, that term will have a reduced value. With this approach, rare words get a boost.



Impacting Relevance: Quality

- Quality is…
  - A factor to increase a documents relevancy score relative to other matching documents
  - Set upon ingestion (or modified later)
  - Default is 0
  - log(tf)/idf + (Query Weight * Doc Quality)



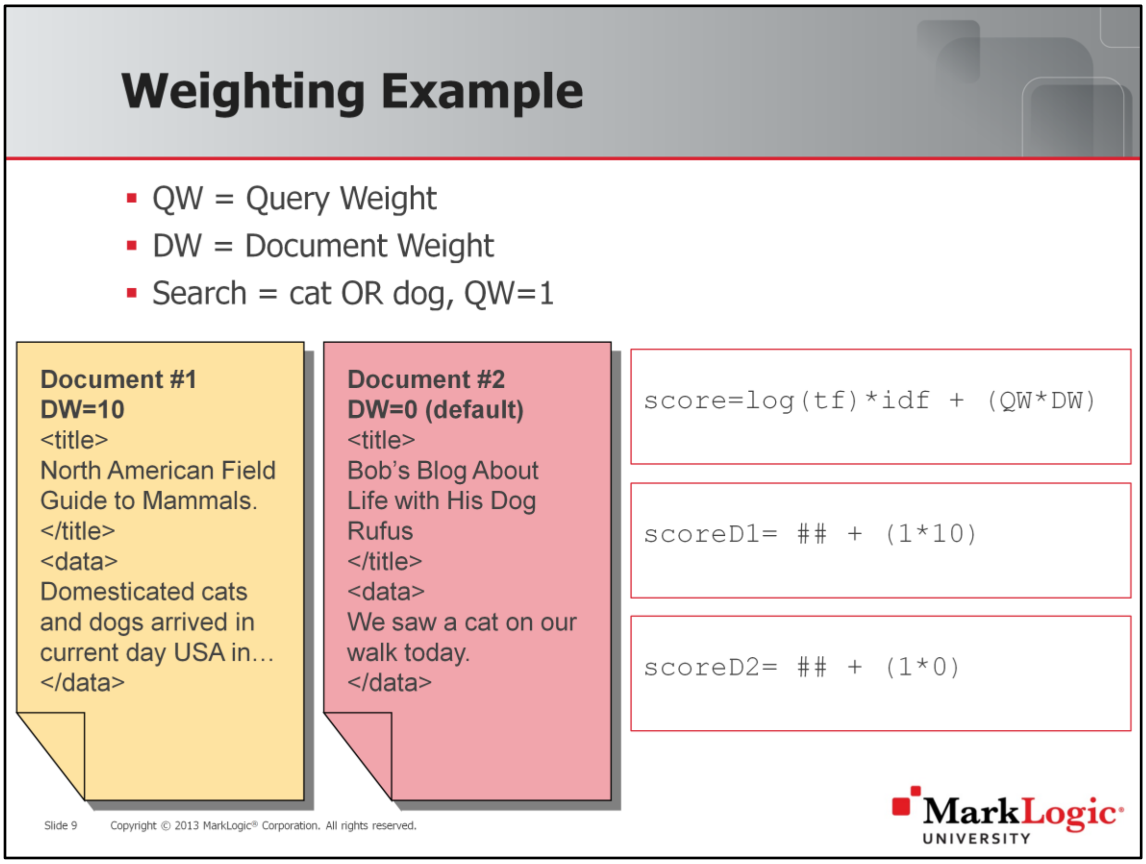
Q = 3 Q = 0

Find books written by the author Herman Melville.
1. Moby Dick
2. Billy Budd

Slide 8 Copyright © 2013 MarkLogic® Corporation. All rights reserved.

Upon ingestion to MarkLogic, a document can be assigned a weight which will impact its relevance in search queries. Increasing the weight of a document will make it more relevant. For example, lets say a new version of a book comes out. The old version still exists, but you assume customers looking to buy the book would prefer the latest version. Loading the new book with the greater weight would help accomplish this goal.

Queries can also contain weights, allowing you to place emphasis on a particular portion of a query.



Weighting Example

- QW = Query Weight
- DW = Document Weight
- Search = cat OR dog, QW=1

Document #1
DW=10
<title>
North American Field Guide to Mammals.
</title>
<data>
Domesticated cats and dogs arrived in current day USA in…
</data>

Document #2
DW=0 (default)
<title>
Bob's Blog About Life with His Dog Rufus
</title>
<data>
We saw a cat on our walk today.
</data>

```
score=log(tf)*idf + (QW*DW)
```

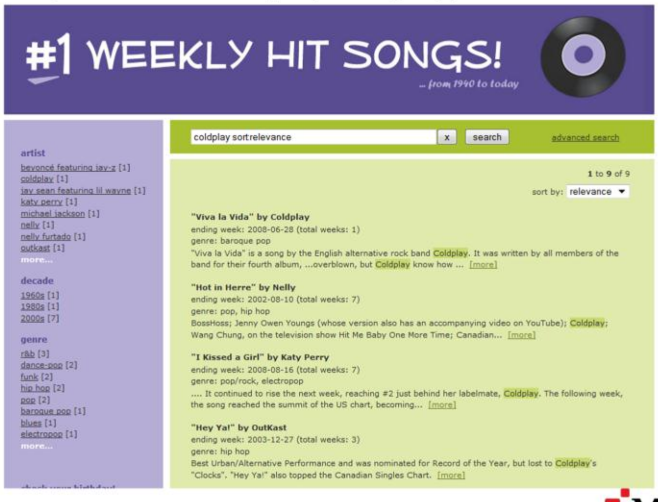
```
scoreD1= ## + (1*10)
```

```
scoreD2= ## + (1*0)
```

Slide 9 Copyright © 2013 MarkLogic® Corporation. All rights reserved.

Impacting Relevance: Word Query

- Why does the Coldplay song appear first?



coldplay sortrelevance

artist
beyoncé featuring jay-z [1]
coldplay [1]
jay sean featuring lil wayne [1]
katy perry [1]
michael jackson [1]
nelly [1]
nelly furtado [1]
outkast [1]

decade
1960s [1]
1980s [1]
2000s [7]

genre
r&b [3]
dance-pop [2]
funk [2]
hip hop [2]
pop [2]
baroque pop [1]
blues [1]
electropop [1]

1 to **9** of 9

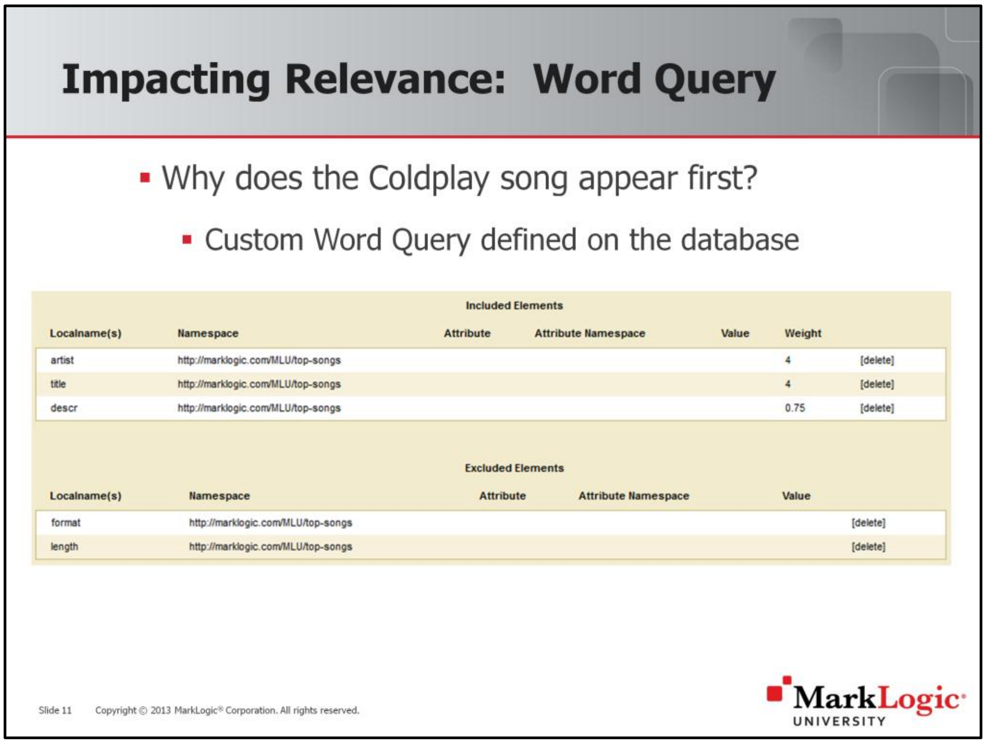
"Viva la Vida" by Coldplay
ending week: 2008-06-28 (total weeks: 1)
genre: baroque pop
"Viva la Vida" is a song by the English alternative rock band Coldplay. It was written by all members of the band for their fourth album, ...overblown, but Coldplay know how ... [more]

"Hot in Herre" by Nelly
ending week: 2002-08-10 (total weeks: 7)
genre: pop, hip hop
BossHoss; Jenny Owen Youngs (whose version also has an accompanying video on YouTube); Coldplay; Wang Chung, on the television show Hit Me Baby One More Time; Canadian... [more]

"I Kissed a Girl" by Katy Perry
ending week: 2008-08-16 (total weeks: 7)
genre: pop/rock, electropop
.... It continued to rise the next week, reaching #2 just behind her labelmate, Coldplay. The following week, the song reached the summit of the US chart, becoming... [more]

"Hey Ya!" by OutKast
ending week: 2003-12-27 (total weeks: 3)
genre: hip hop
Best Urban/Alternative Performance and was nominated for Record of the Year, but lost to Coldplay's "Clocks". "Hey Ya!" also topped the Canadian Singles Chart. [more]

Slide 10



Impacting Relevance: Word Query

- Why does the Coldplay song appear first?
  - Custom Word Query defined on the database

Included Elements

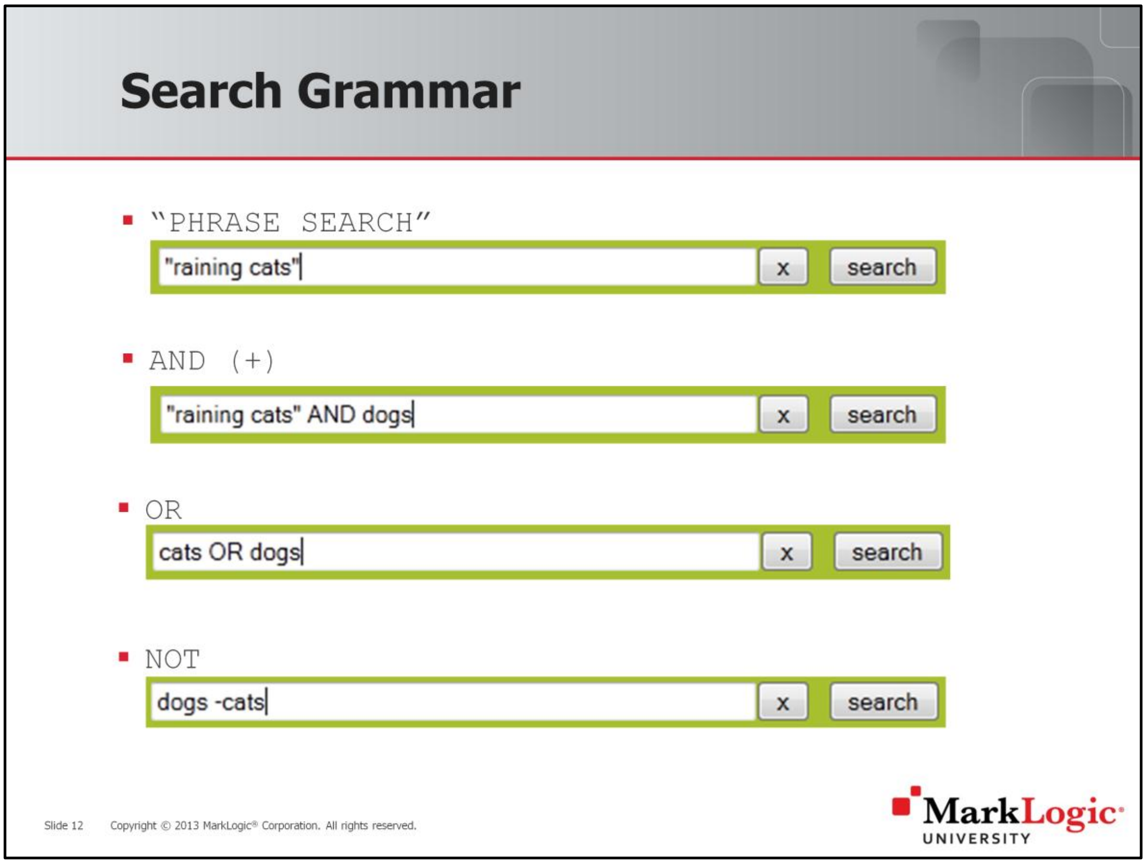
Localname(s)	Namespace	Attribute	Attribute Namespace	Value	Weight	
artist	http://marklogic.com/MLU/top-songs				4	[delete]
title	http://marklogic.com/MLU/top-songs				4	[delete]
descr	http://marklogic.com/MLU/top-songs				0.75	[delete]

Excluded Elements

Localname(s)	Namespace	Attribute	Attribute Namespace	Value	
format	http://marklogic.com/MLU/top-songs				[delete]
length	http://marklogic.com/MLU/top-songs				[delete]

Slide 11

MarkLogic UNIVERSITY



Search Grammar

- "PHRASE SEARCH"

"raining cats"

- AND (+)

"raining cats" AND dogs

- OR

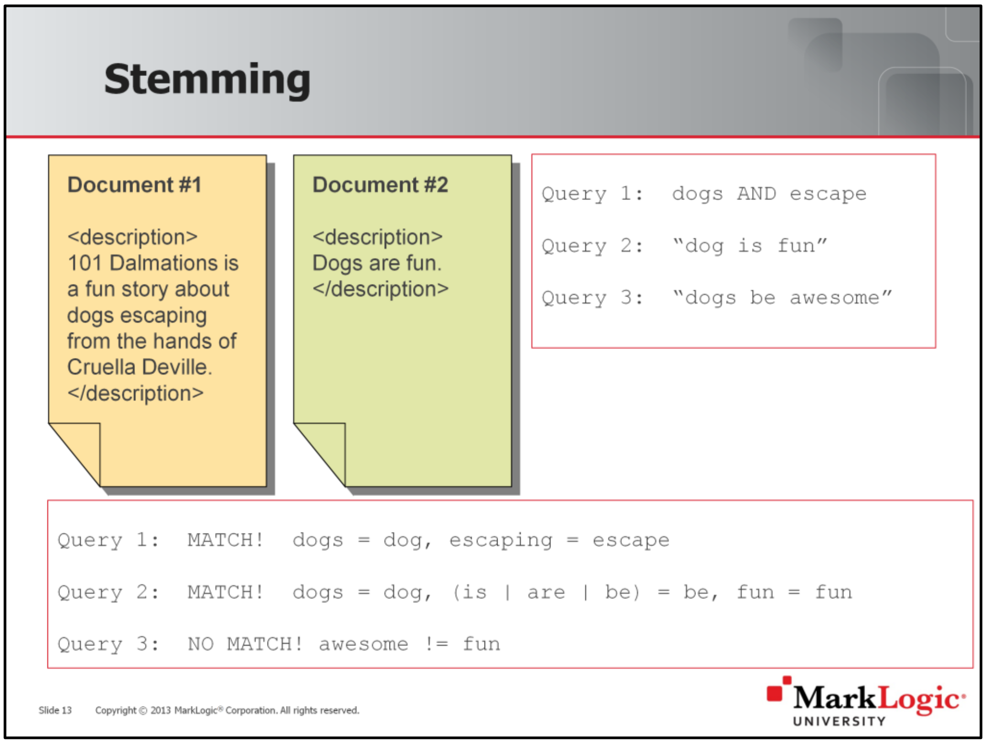
cats OR dogs

- NOT

dogs -cats

Slide 12 Copyright © 2013 MarkLogic® Corporation. All rights reserved.

When executing a string search with the Java API, we don't have to do anything special to understand search grammar. Because the Java API utilizes the REST API (and therefore the Search API) under the covers, search grammar is natively understood.



Stemming

Document #1

<description>
101 Dalmations is a fun story about dogs escaping from the hands of Cruella Deville.
</description>

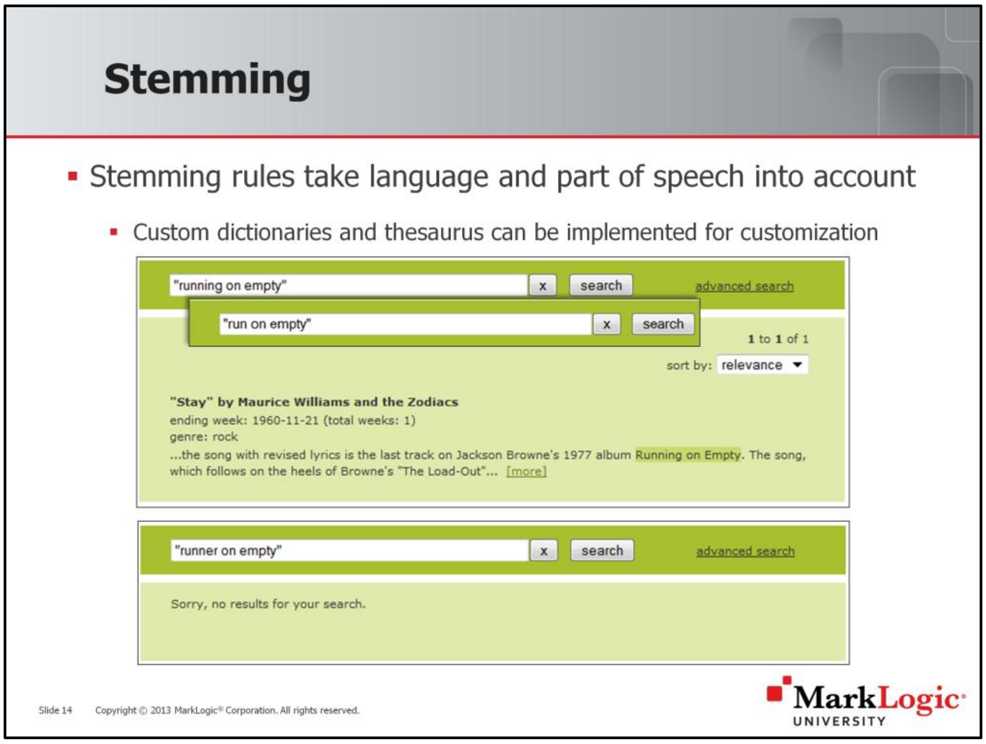
Document #2

<description>
Dogs are fun.
</description>

```
Query 1:  dogs AND escape

Query 2:  "dog is fun"

Query 3:  "dogs be awesome"
```

```
Query 1:  MATCH!  dogs = dog, escaping = escape

Query 2:  MATCH!  dogs = dog, (is | are | be) = be, fun = fun

Query 3:  NO MATCH! awesome != fun
```

Slide 13 Copyright © 2013 MarkLogic® Corporation. All rights reserved.



Stemming

- Stemming rules take language and part of speech into account
  - Custom dictionaries and thesaurus can be implemented for customization

"running on empty"

"run on empty"

1 to 1 of 1

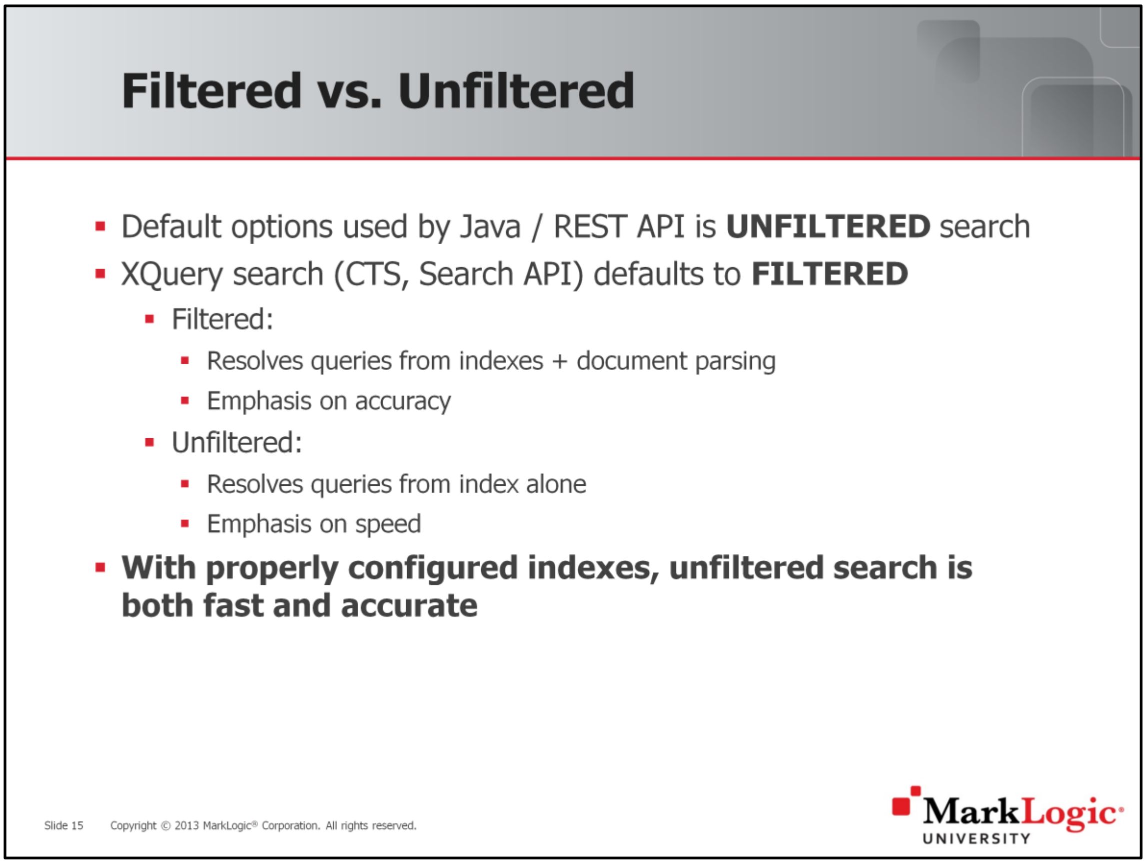
sort by: relevance

"Stay" by Maurice Williams and the Zodiacs
ending week: 1960-11-21 (total weeks: 1)
genre: rock
...the song with revised lyrics is the last track on Jackson Browne's 1977 album Running on Empty. The song, which follows on the heels of Browne's "The Load-Out"... [more]

"runner on empty"

Sorry, no results for your search.

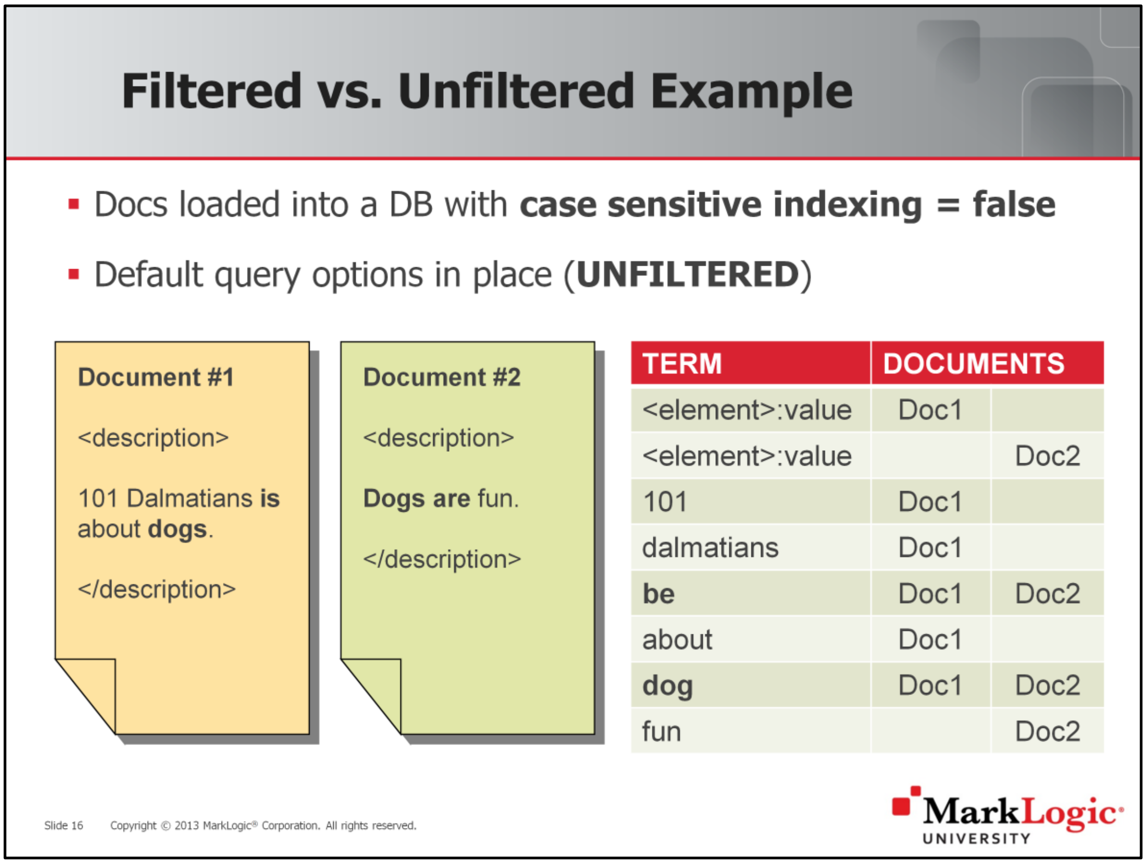
Slide 14 Copyright © 2013 MarkLogic® Corporation. All rights reserved.



Filtered vs. Unfiltered

- Default options used by Java / REST API is **UNFILTERED** search
- XQuery search (CTS, Search API) defaults to **FILTERED**
  - Filtered:
    - Resolves queries from indexes + document parsing
    - Emphasis on accuracy
  - Unfiltered:
    - Resolves queries from index alone
    - Emphasis on speed
- **With properly configured indexes, unfiltered search is both fast and accurate**

Slide 15 Copyright © 2013 MarkLogic® Corporation. All rights reserved.



Filtered vs. Unfiltered Example

- Docs loaded into a DB with **case sensitive indexing = false**
- Default query options in place (**UNFILTERED**)

Document #1

<description>

101 Dalmatians **is** about **dogs**.

</description>

Document #2

<description>

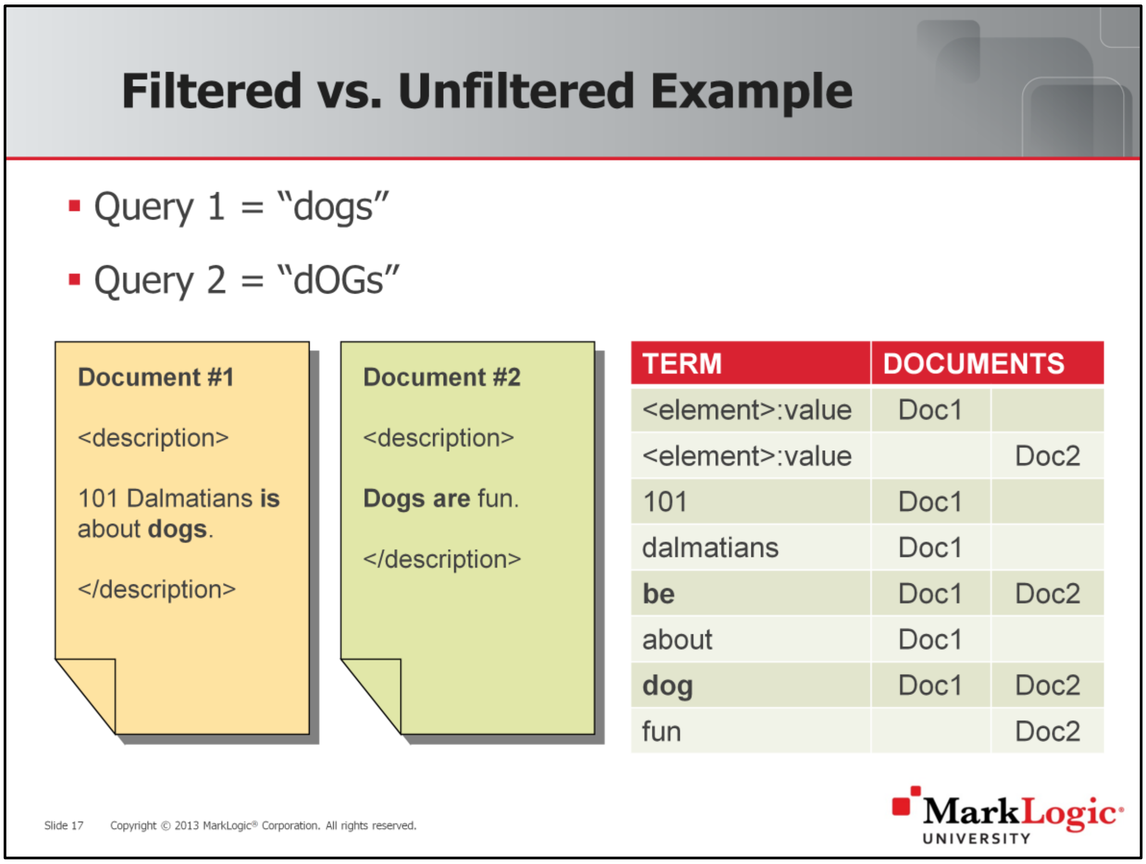
Dogs are fun.

</description>

TERM	DOCUMENTS	
<element>:value	Doc1	
<element>:value		Doc2
101	Doc1	
dalmatians	Doc1	
be	Doc1	Doc2
about	Doc1	
dog	Doc1	Doc2
fun		Doc2

Slide 16 Copyright © 2013 MarkLogic® Corporation. All rights reserved.

Notice the index graphic on the right of the slide. Envision our indexes as looking like this behind the scenes, where each word is stemmed and added to the index, making it easy to identify which documents contain a word. However, the term stored in the index does not care about case. Doc 1 and Doc 2 both show as containing “dog” regardless of the actual case difference in the document.



Filtered vs. Unfiltered Example

- Query 1 = "dogs"
- Query 2 = "dOGs"

Document #1

<description>

101 Dalmatians **is** about **dogs**.

</description>

Document #2

<description>

Dogs are fun.

</description>

TERM	DOCUMENTS	
<element>:value	Doc1	
<element>:value		Doc2
101	Doc1	
dalmatians	Doc1	
be	Doc1	Doc2
about	Doc1	
dog	Doc1	Doc2
fun		Doc2

Slide 17 Copyright © 2013 MarkLogic® Corporation. All rights reserved.

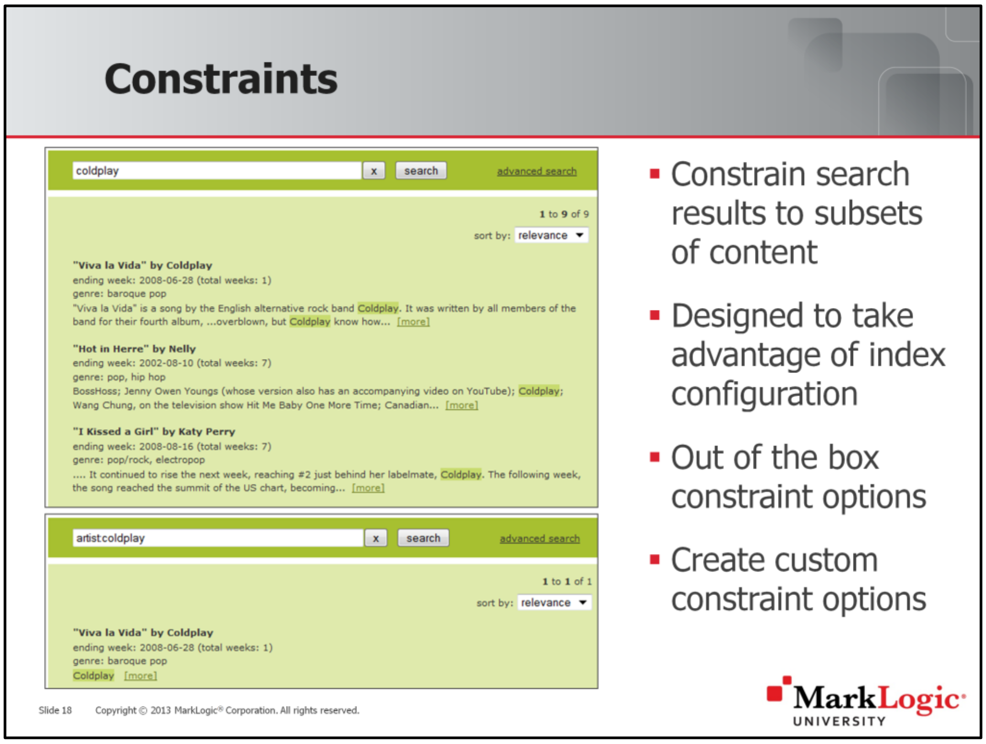
Query 1 "dogs" matches Doc1 and Doc2.
Query 2 "dOGs" matches Doc1 and Doc2.

Because we are doing UNFILTERED search, matching documents are determined based on the index alone. Since these documents were loaded into a database with the "fast case sensitive searches" index set to FALSE, all that gets indexed is the word – "dog".

So we can tell that a document contains the word, but not specifically HOW that word is reflected exactly in the document.

To contrast, if we did this search FILTERED, it would first use the index to determine which documents were candidates, then the candidate documents would be parsed to confirm the exact match, meaning the query for "dOGs" would not return any results. Filtering has a cost as it requires additional disk I/O.

We could also fix the root of the issue by enabling appropriate indexes. This would allow us to get ACCURATE results AND use UNFILTERED search for optimum performance. Indexes will be covered in more detail in an upcoming unit.



Constraints

1 to **9** of 9

sort by: relevance

"Viva la Vida" by Coldplay
ending week: 2008-06-28 (total weeks: 1)
genre: baroque pop
"Viva la Vida" is a song by the English alternative rock band Coldplay. It was written by all members of the band for their fourth album, ...overblown, but Coldplay know how... [more]

"Hot in Herre" by Nelly
ending week: 2002-08-10 (total weeks: 7)
genre: pop, hip hop
BossHoss; Jenny Owen Youngs (whose version also has an accompanying video on YouTube); Coldplay; Wang Chung, on the television show Hit Me Baby One More Time; Canadian... [more]

"I Kissed a Girl" by Katy Perry
ending week: 2008-08-16 (total weeks: 7)
genre: pop/rock, electropop
.... It continued to rise the next week, reaching #2 just behind her labelmate, Coldplay. The following week, the song reached the summit of the US chart, becoming... [more]

1 to **1** of 1

sort by: relevance

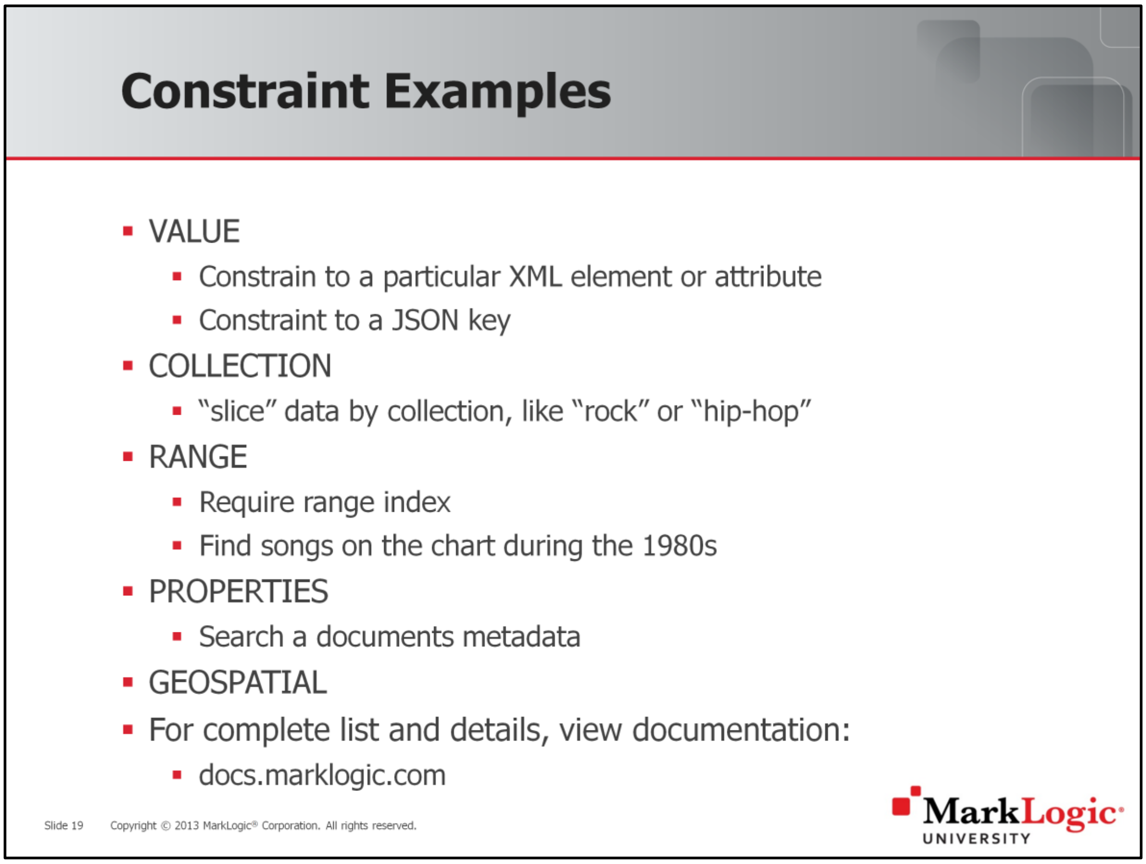
"Viva la Vida" by Coldplay
ending week: 2008-06-28 (total weeks: 1)
genre: baroque pop
Coldplay [more]

- Constrain search results to subsets of content
- Designed to take advantage of index configuration
- Out of the box constraint options
- Create custom constraint options

Slide 18

What enables the ability to use search grammar such as: artist:coldplay?

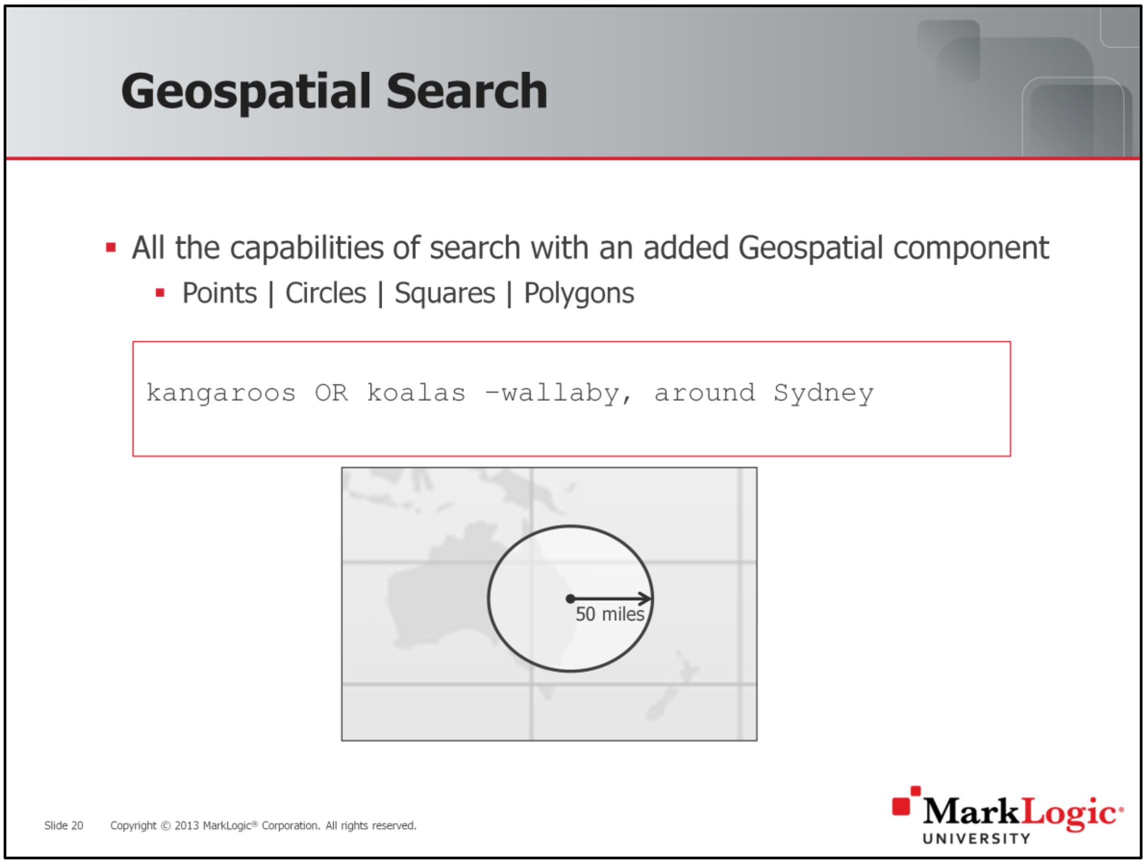
Constraints defined in a query options document enable many customizations to search inside of MarkLogic.



Constraint Examples

- VALUE
  - Constrain to a particular XML element or attribute
  - Constraint to a JSON key
- COLLECTION
  - "slice" data by collection, like "rock" or "hip-hop"
- RANGE
  - Require range index
  - Find songs on the chart during the 1980s
- PROPERTIES
  - Search a documents metadata
- GEOSPATIAL
- For complete list and details, view documentation:
  - docs.marklogic.com

Slide 19 Copyright © 2013 MarkLogic® Corporation. All rights reserved.

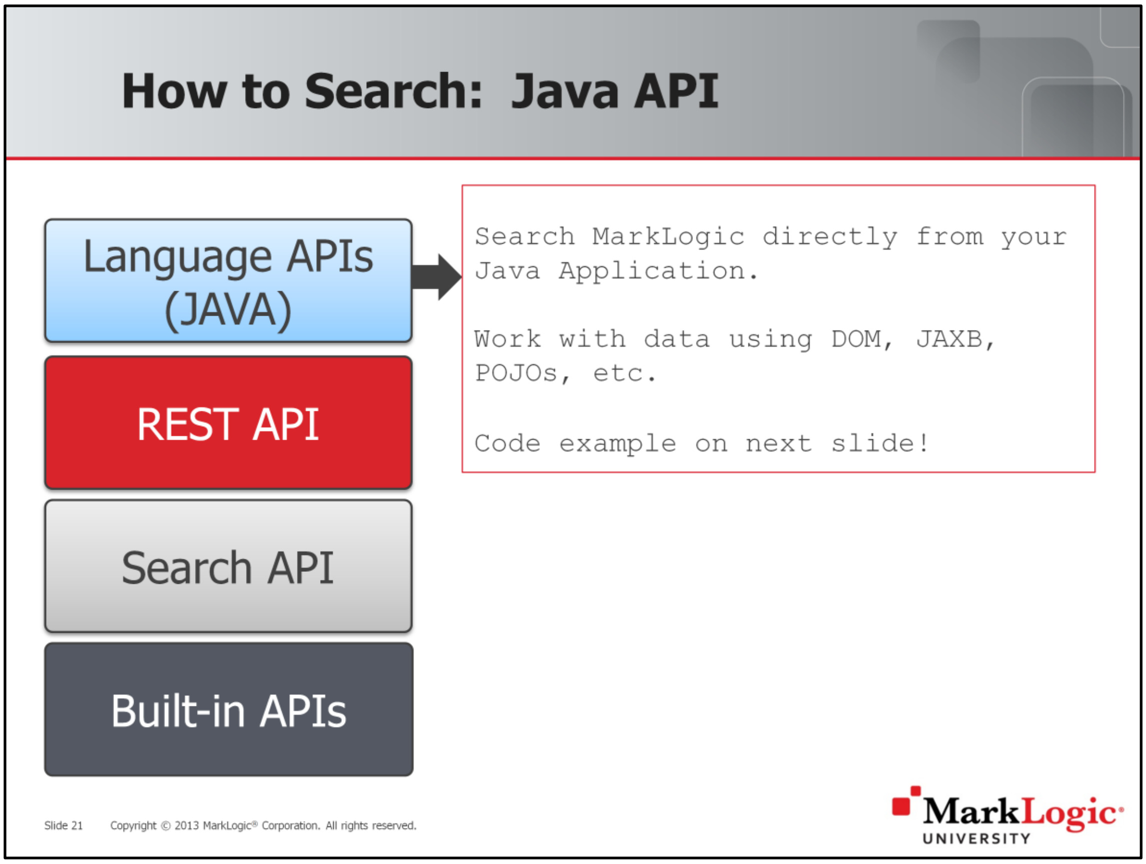


Geospatial Search

- All the capabilities of search with an added Geospatial component
  - Points | Circles | Squares | Polygons

```
kangaroos OR koalas -wallaby, around Sydney
```

Slide 20 Copyright © 2013 MarkLogic® Corporation. All rights reserved.



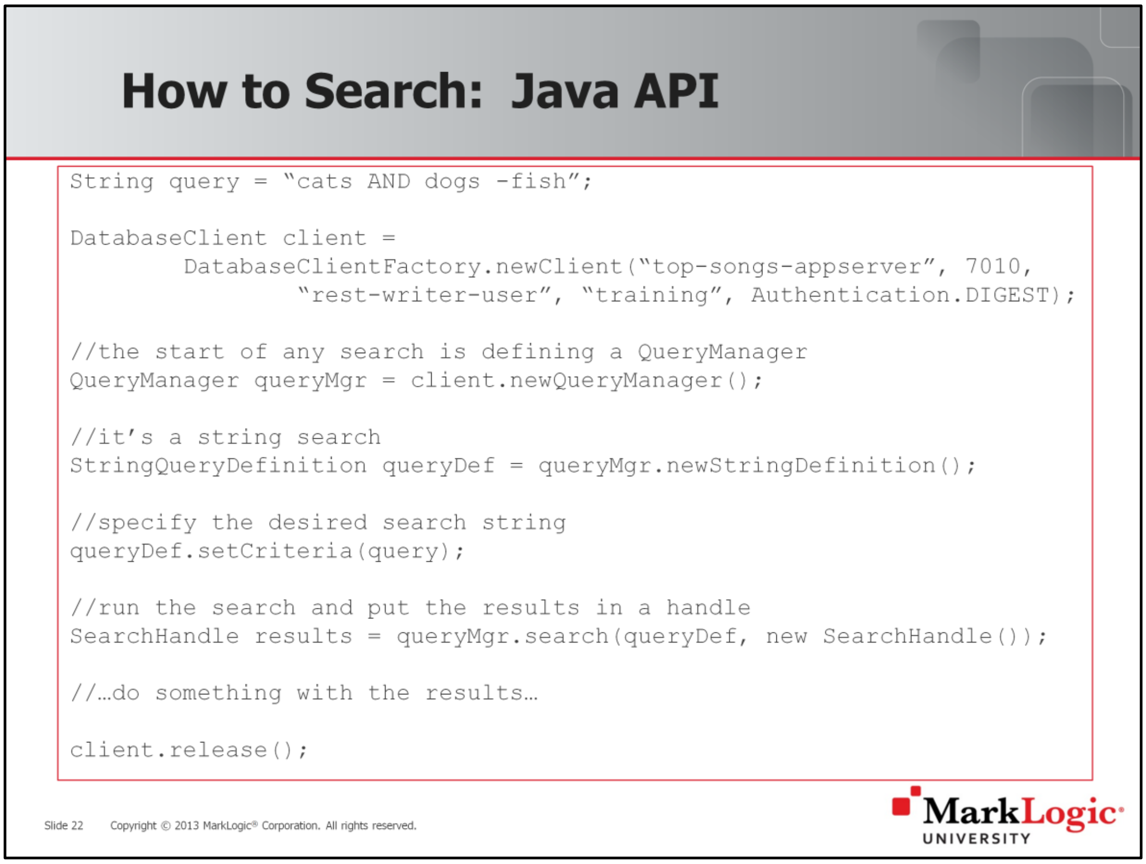
How to Search: Java API

Language APIs (JAVA)
REST API
Search API
Built-in APIs

```
Search MarkLogic directly from your
Java Application.

Work with data using DOM, JAXB,
POJOs, etc.

Code example on next slide!
```

Slide 21 Copyright © 2013 MarkLogic® Corporation. All rights reserved.

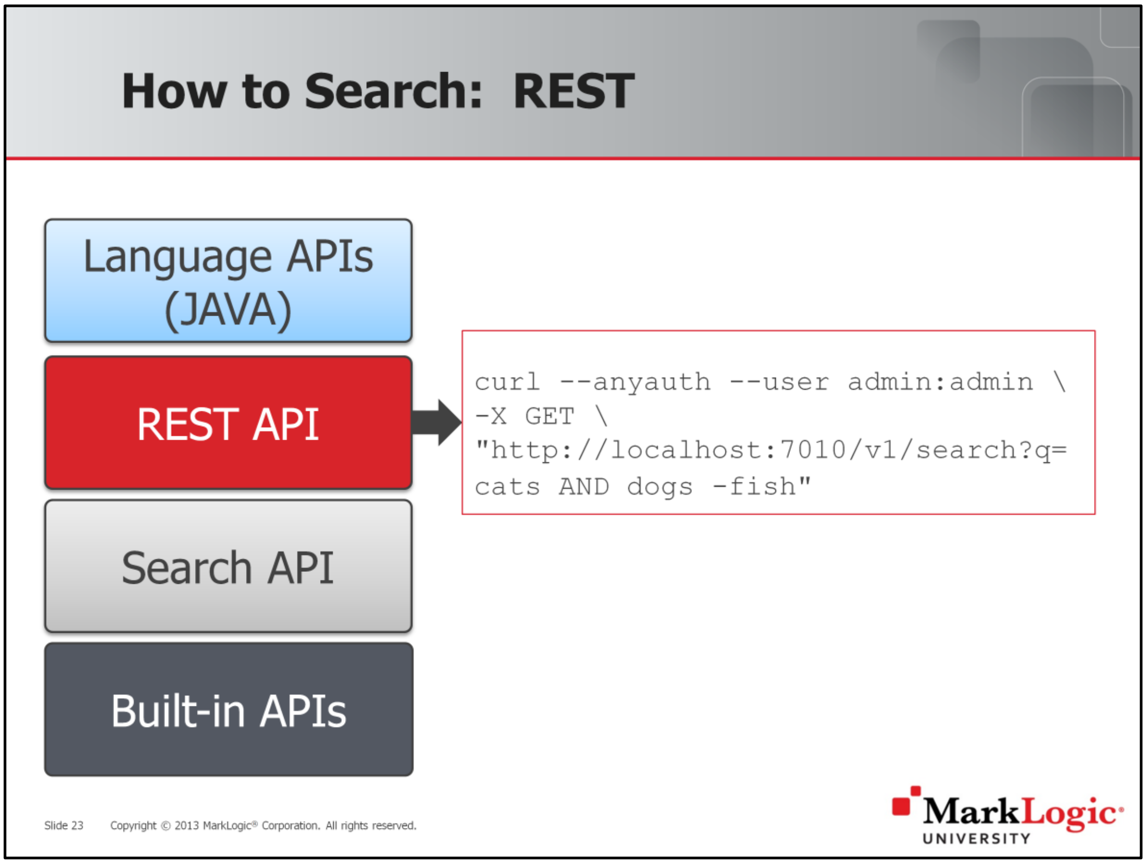
Take advantage of MarkLogic search and database capabilities from your Java Applications using MarkLogic’s Java API.



How to Search: Java API

```
String query = “cats AND dogs -fish”;

DatabaseClient client =
        DatabaseClientFactory.newClient(“top-songs-appserver”, 7010,
                “rest-writer-user”, “training”, Authentication.DIGEST);

//the start of any search is defining a QueryManager
QueryManager queryMgr = client.newQueryManager();

//it’s a string search
StringQueryDefinition queryDef = queryMgr.newStringDefinition();

//specify the desired search string
queryDef.setCriteria(query);

//run the search and put the results in a handle
SearchHandle results = queryMgr.search(queryDef, new SearchHandle());

//…do something with the results…

client.release();
```

Slide 22   Copyright © 2013 MarkLogic® Corporation. All rights reserved.

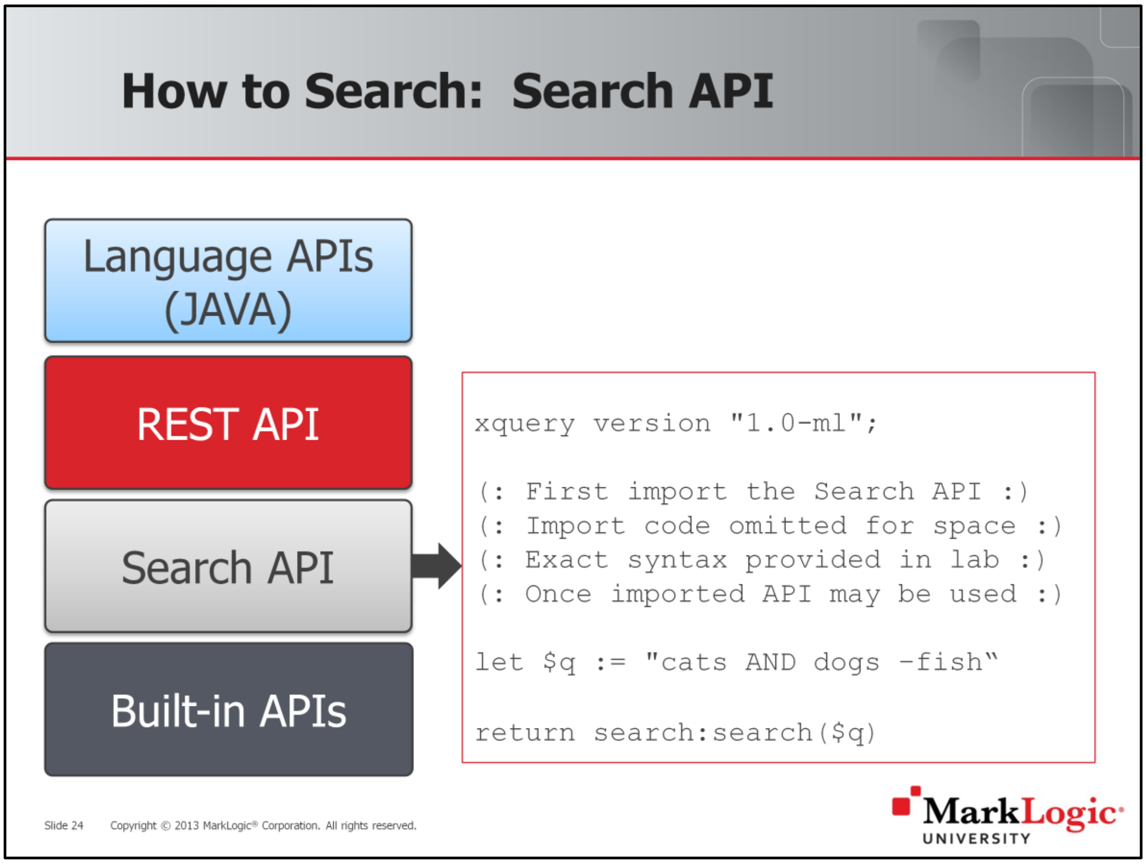


How to Search: REST

Language APIs (JAVA)
REST API
Search API
Built-in APIs

```
curl --anyauth --user admin:admin \
-X GET \
"http://localhost:7010/v1/search?q=
cats AND dogs -fish"
```

Search via the REST API is an excellent way to take advantage of everything that MarkLogic offers as a database and search engine, but without being tied to writing your applications in XQuery code. The REST API enables the developer to engage MarkLogic from the language of their choice via an HTTP request to a REST-ful web service.



How to Search: Search API

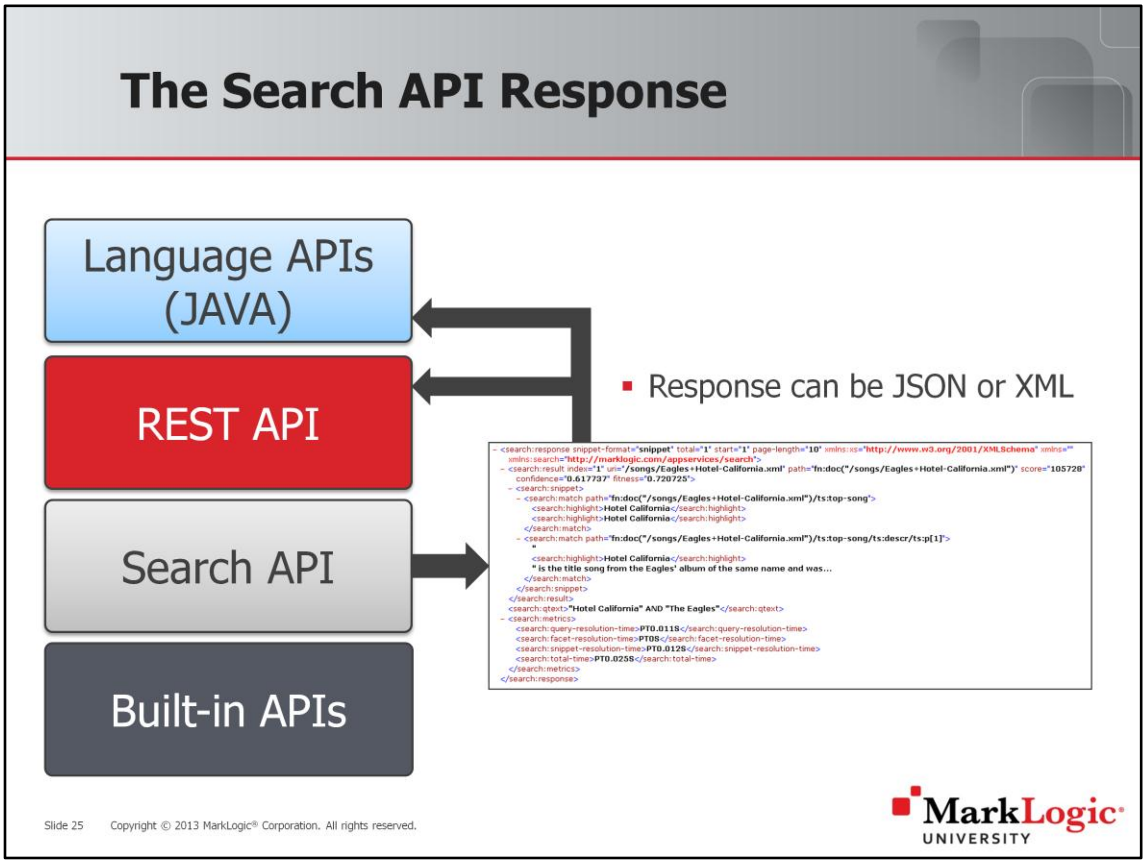
```
xquery version "1.0-ml";

(: First import the Search API :)
(: Import code omitted for space :)
(: Exact syntax provided in lab :)
(: Once imported API may be used :)

let $q := "cats AND dogs -fish"

return search:search($q)
```

Slide 24 Copyright © 2013 MarkLogic® Corporation. All rights reserved.

The search API makes it very easy to do things like search grammar because it was built to understand a certain grammar. This makes it easy for the developer to simply grab the query as a string from a web based form and pass it to the API. The response that comes back from the search API is consistent to the example we saw earlier, and can be specified as XML or JSON.

Customization to search API behavior is done using query options and constraints.

Under the covers, the search API uses the underlying low level CTS search functions.



The Search API Response

Language APIs (JAVA)
REST API
Search API
Built-in APIs

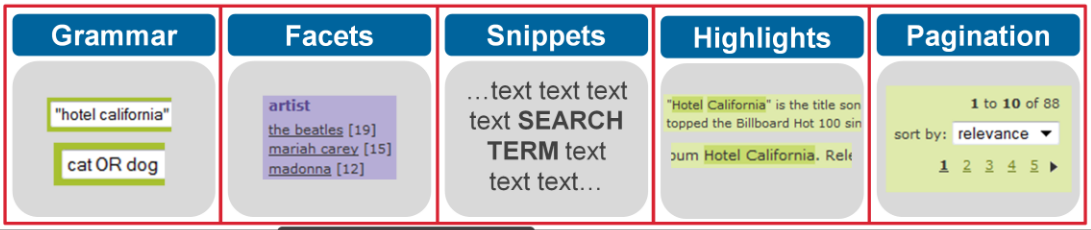
- Response can be JSON or XML

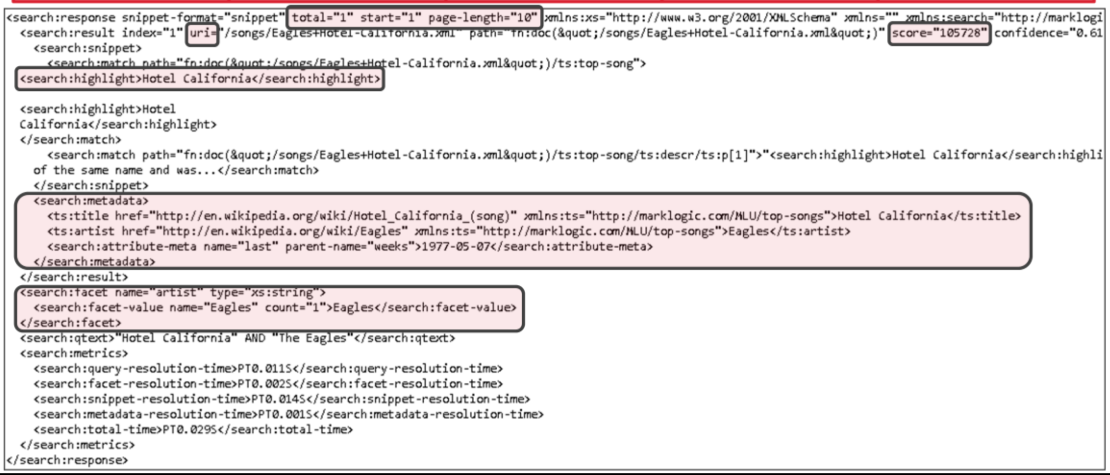
```
- <search:response snippet-format="snippet" total="1" start="1" page-length="10" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns=""
    xmlns:search="http://marklogic.com/appservices/search">
  - <search:result index="1" uri="/songs/Eagles+Hotel-California.xml" path="fn:doc("/songs/Eagles+Hotel-California.xml")" score="105728"
      confidence="0.617737" fitness="0.720725">
    - <search:snippet>
      - <search:match path="fn:doc("/songs/Eagles+Hotel-California.xml")/ts:top-song">
          <search:highlight>Hotel California</search:highlight>
          <search:highlight>Hotel California</search:highlight>
        </search:match>
      - <search:match path="fn:doc("/songs/Eagles+Hotel-California.xml")/ts:top-song/ts:descr/ts:p[1]">
          "
          <search:highlight>Hotel California</search:highlight>
          " is the title song from the Eagles' album of the same name and was...
        </search:match>
      </search:snippet>
    </search:result>
    <search:qtext>"Hotel California" AND "The Eagles"</search:qtext>
  - <search:metrics>
      <search:query-resolution-time>PT0.011S</search:query-resolution-time>
      <search:facet-resolution-time>PT0S</search:facet-resolution-time>
      <search:snippet-resolution-time>PT0.012S</search:snippet-resolution-time>
      <search:total-time>PT0.025S</search:total-time>
    </search:metrics>
  </search:response>
```

Slide 25 Copyright © 2013 MarkLogic® Corporation. All rights reserved.

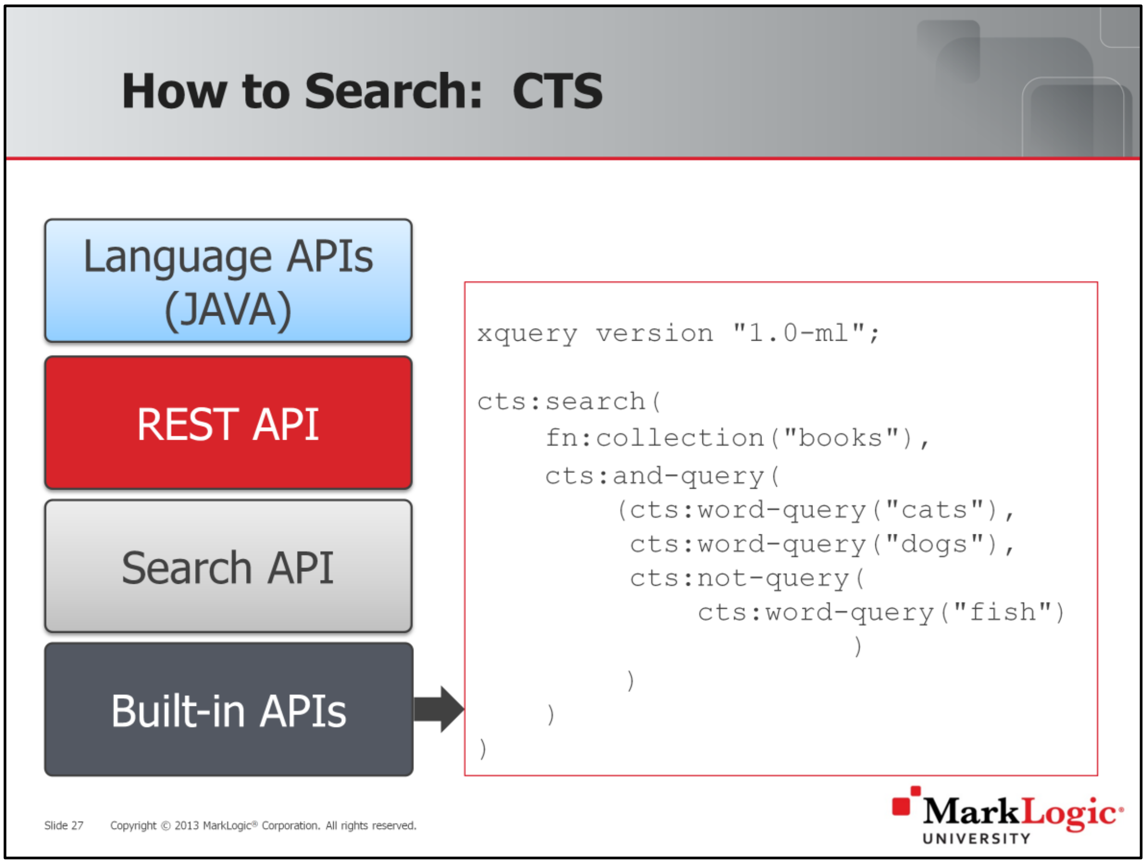
The Search API generates a standard response for the Developer to use. This response can come back as XML or JSON. On the next slide we will see an example of this standard response and how the information it contains enables developers to easily build “Google-like” search applications.

Search API Response





```
<search:response snippet-format="snippet" total="1" start="1" page-length="10" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="" xmlns:search="http://marklogi
  <search:result index="1" uri="/songs/Eagles+Hotel-California.xml" path="fn:doc(&quot;/songs/Eagles+Hotel-California.xml&quot;)" score="105728" confidence="0.61
    <search:snippet>
      <search:match path="fn:doc(&quot;/songs/Eagles+Hotel-California.xml&quot;)/ts:top-song">
 <search:highlight>Hotel California</search:highlight>

  <search:highlight>Hotel
  California</search:highlight>
  </search:match>
      <search:match path="fn:doc(&quot;/songs/Eagles+Hotel-California.xml&quot;)/ts:top-song/ts:descr/ts:p[1]">"<search:highlight>Hotel California</search:highli
    of the same name and was...</search:match>
    </search:snippet>
    <search:metadata>
      <ts:title href="http://en.wikipedia.org/wiki/Hotel_California_(song)" xmlns:ts="http://marklogic.com/MLU/top-songs">Hotel California</ts:title>
      <ts:artist href="http://en.wikipedia.org/wiki/Eagles" xmlns:ts="http://marklogic.com/MLU/top-songs">Eagles</ts:artist>
      <search:attribute-meta name="last" parent-name="weeks">1977-05-07</search:attribute-meta>
    </search:metadata>
  </search:result>
 <search:facet name="artist" type="xs:string">
    <search:facet-value name="Eagles" count="1">Eagles</search:facet-value>
 </search:facet>
  <search:qtext>"Hotel California" AND "The Eagles"</search:qtext>
  <search:metrics>
    <search:query-resolution-time>PT0.011S</search:query-resolution-time>
    <search:facet-resolution-time>PT0.002S</search:facet-resolution-time>
    <search:snippet-resolution-time>PT0.014S</search:snippet-resolution-time>
    <search:metadata-resolution-time>PT0.001S</search:metadata-resolution-time>
    <search:total-time>PT0.029S</search:total-time>
  </search:metrics>
</search:response>
```



How to Search: CTS

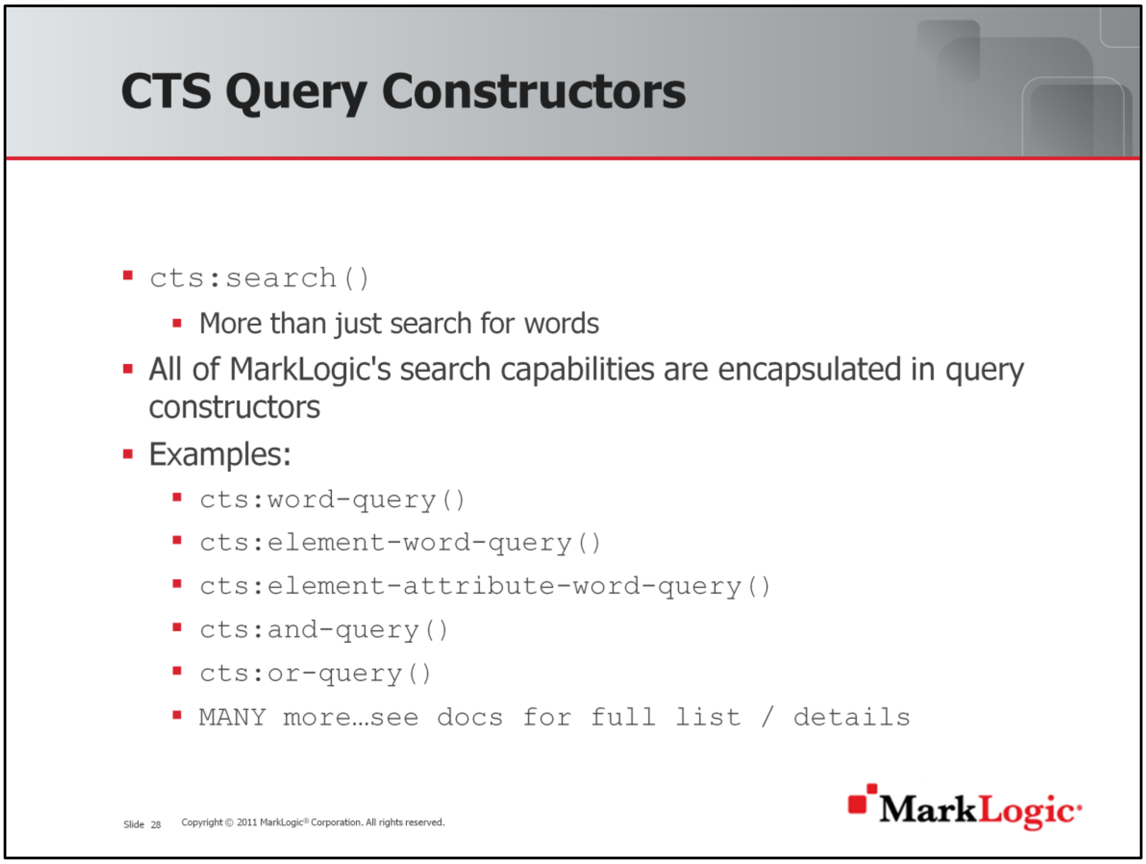
Language APIs (JAVA)
REST API
Search API
Built-in APIs

```
xquery version "1.0-ml";

cts:search(
     fn:collection("books"),
     cts:and-query(
          (cts:word-query("cats"),
           cts:word-query("dogs"),
           cts:not-query(
                cts:word-query("fish")
                           )
           )
     )
)
```

Slide 27 Copyright © 2013 MarkLogic® Corporation. All rights reserved.

This example shows the lowest level API used for search in MarkLogic: CTS or Core Text Search. This API is available to XQuery Developers and enables very powerful, customized search queries.

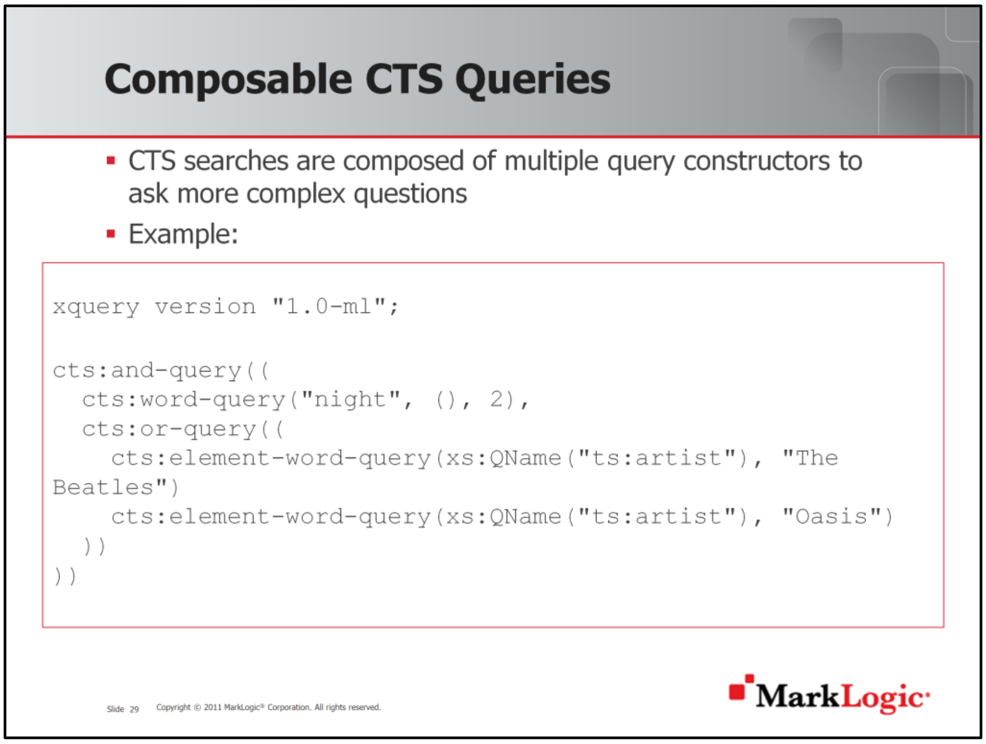
The goal of this query is to search any document that is in the "books" collection where the document contains the word "cats" AND the word "dogs" but does NOT contain the word "fish".



CTS Query Constructors

- `cts:search()`
  - More than just search for words
- All of MarkLogic's search capabilities are encapsulated in query constructors
- Examples:
  - `cts:word-query()`
  - `cts:element-word-query()`
  - `cts:element-attribute-word-query()`
  - `cts:and-query()`
  - `cts:or-query()`
  - `MANY more…see docs for full list / details`

Slide 28 Copyright © 2011 MarkLogic® Corporation. All rights reserved. MarkLogic



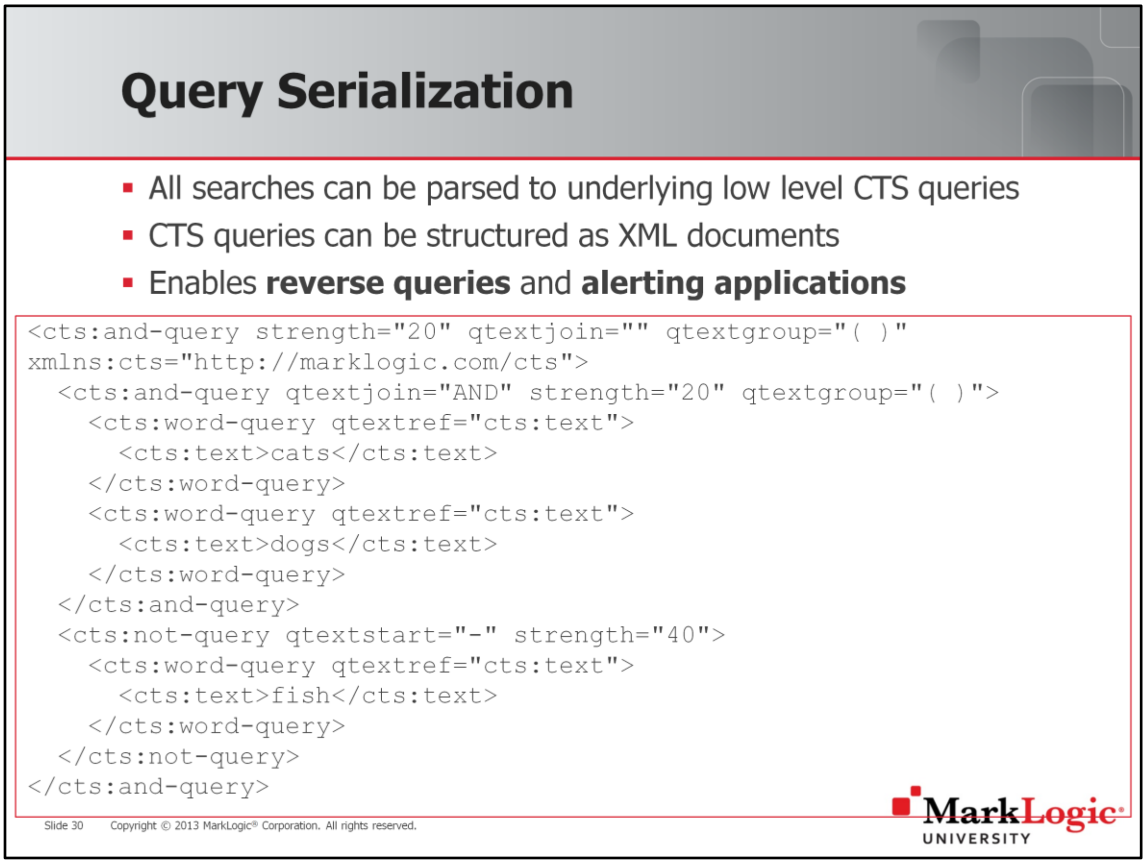
Composable CTS Queries

- CTS searches are composed of multiple query constructors to ask more complex questions
- Example:

```
xquery version "1.0-ml";

cts:and-query((
  cts:word-query("night", (), 2),
  cts:or-query((
    cts:element-word-query(xs:QName("ts:artist"), "The
Beatles")
    cts:element-word-query(xs:QName("ts:artist"), "Oasis")
  ))
))
```

Slide 29 Copyright © 2011 MarkLogic® Corporation. All rights reserved.

What is this query asking?
Show me all documents that contain the word “night” AND the artist is either “The Beatles” OR “Oasis”.



Query Serialization

- All searches can be parsed to underlying low level CTS queries
- CTS queries can be structured as XML documents
- Enables **reverse queries** and **alerting applications**

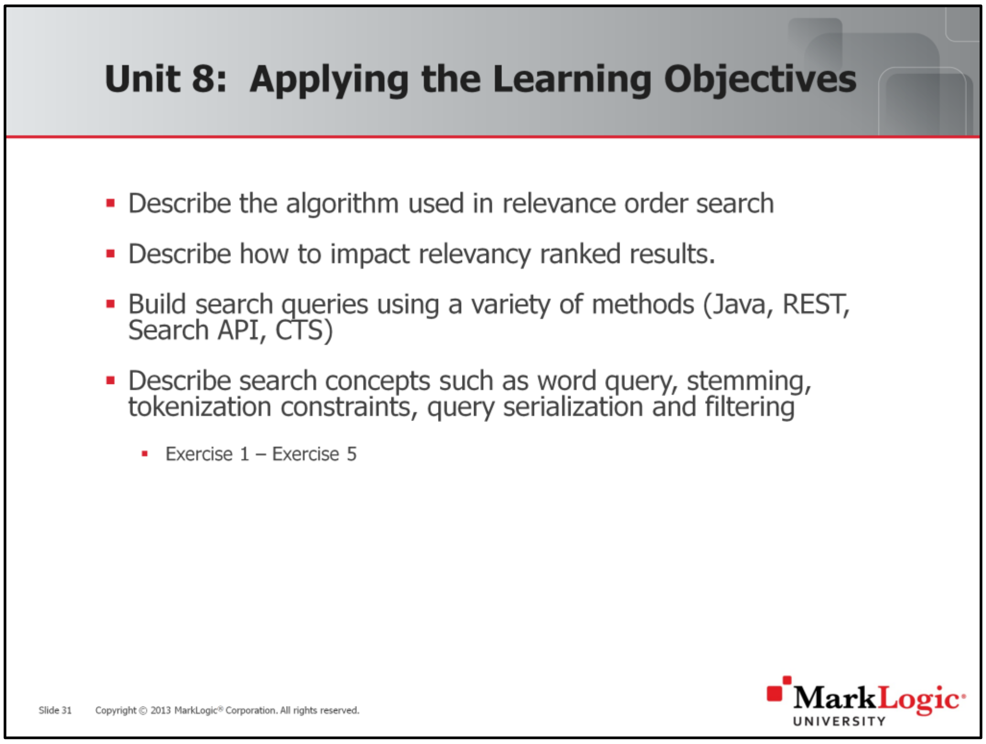
```
<cts:and-query strength="20" qtextjoin="" qtextgroup="( )"
xmlns:cts="http://marklogic.com/cts">
  <cts:and-query qtextjoin="AND" strength="20" qtextgroup="( )">
    <cts:word-query qtextref="cts:text">
      <cts:text>cats</cts:text>
    </cts:word-query>
    <cts:word-query qtextref="cts:text">
      <cts:text>dogs</cts:text>
    </cts:word-query>
  </cts:and-query>
  <cts:not-query qtextstart="-" strength="40">
    <cts:word-query qtextref="cts:text">
      <cts:text>fish</cts:text>
    </cts:word-query>
  </cts:not-query>
</cts:and-query>
```

Slide 30 Copyright © 2013 MarkLogic® Corporation. All rights reserved.

MarkLogic UNIVERSITY

Because a search can be serialized as an XML document it enables us to store queries as documents in the database. This enables the concept of a reverse query: your search query sits in the database and searches any new incoming documents. If an incoming document matches the query, an alerting application could take some action.

A special reverse query index can be turned on to optimize these types of applications.



Unit 8: Applying the Learning Objectives

- Describe the algorithm used in relevance order search
- Describe how to impact relevancy ranked results.
- Build search queries using a variety of methods (Java, REST, Search API, CTS)
- Describe search concepts such as word query, stemming, tokenization constraints, query serialization and filtering
  - Exercise 1 – Exercise 5

Slide 31