Summit Lab Workbook

L4080 – Adobe Experience Manager search demystified

Danny Gordon, AEM Technical Marketing
David Gonzalez, AEM Technical Marketing
Sean Schnoor, AEM Technical Marketing

March 23, 2017

Table of Contents

Chapter 0

Using Chrome, log in to AEM Author at http://localhost:4502/ as admin.

- User name: admin
- Password: admin

Chapters 1 – 8 have prepared pages available via **Sites > Adobe Summit 2017 > L4080. Select the Chapter page**, and click **Edit** in the top action bar.

Adobe Developer Tools

Index Manager
Web console that facilitates re-indexing of Oak indices and reviewing high-level Oak index configurations.

- AEM > Tools > Operations > Diagnosis > Index Manager
- http://localhost:4502/libs/granite/operations/content/diagnosis/tool.html/granite_oakindexmanager

Query Performance
Web console that lists recent slow and popular queries.

- AEM > Tools > Operations > Diagnosis > Query Performance
- http://localhost:4502/libs/granite/operations/content/diagnosis/tool.html/granite_queryperformance

Explain Query
Web console that provides detailed execution details for a specific query.

- AEM > Tools > Operations > Diagnosis > Query Performance > Explain Query tab
- http://localhost:4502/libs/granite/operations/content/diagnosis/tool.html/granite_queryperformance

QueryBuilder Debugger
Executes QueryBuilder-based queries, provides the derived XPath expression and results.

- http://localhost:4502/libs/cq/search/content/querydebug.html

Community Developer Tools

Not supported by Adobe

Index Definition Analyzer
Upload an Oak index definition XML or JSON definitions and outputs an easy-to-consume visual report.

- http://oakutils.appspot.com/analyze/index

Oak Index Definition Generator
Provide a query and generate the appropriate index to satisfy the query.

- http://oakutils.appspot.com/generate/index

AEM Chrome Plug-in
Developer Tools plug-in for the Chrome Web browser that uses Sling Log Tracer to exposed detailed logging directly in the browser.

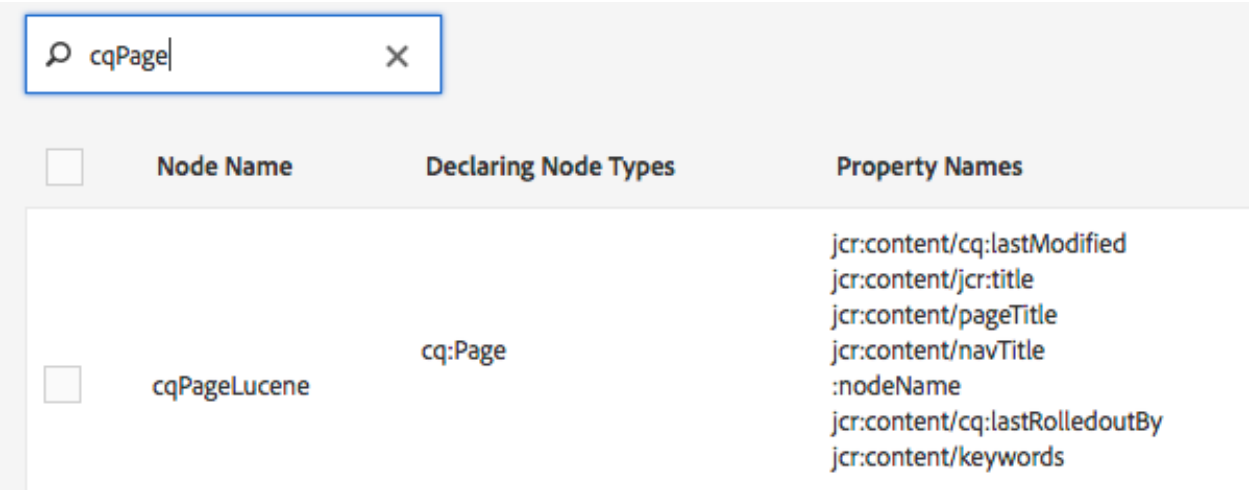
- http://adobe-consulting-services.github.io/acs-aem-tools/aem-chrome-plugin/

Re-indexing Oak Indexes via Index Manager

Throughout this lab, re-indexing of the /oak:index/cqPageLucene will be required to make configuration changes to take effect.

Below are the steps required to re-index the cqPageLucene index,

1. From the AEM Start Menu > Tools (hammer icon) > Diagnosis -> Index Manager
   - http://localhost:4502/libs/granite/operations/content/diagnosis/tool.html/granite_oakindexmanager
2. In the Filter field type **cqPageLucene** to filter the indices



3. Click the **re-index** button in the right most column to trigger re-indexing of the **cqPageLucene** index.
4. The cqPageLucene index row will be colored red while the indexing is occurring.
5. The index will re-index, the UI might time out but it should take less than a minute to finish.

Chapter 1: Full-text & search fundamentals

AEM search supports robust full-text search, provided by the Apache Lucene.

Lucene property indices are at the core of AEM Search and must be well understood. This exercise covers:

- Definition of the OOTB **cqPageLucene** Oak Lucene property index
- Search query inspection
- Full-text search operators
- Search result excerpts

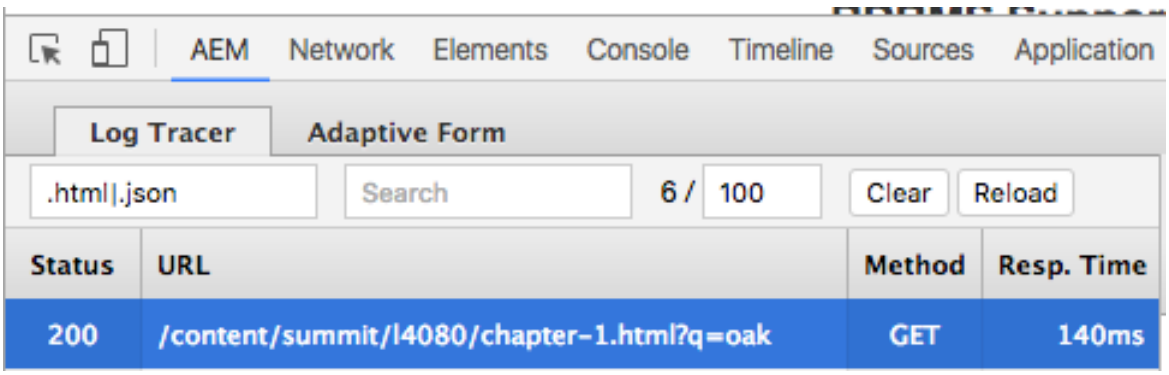
Exercise

Configuring this lab's Search component

1. Edit the page at **Sites > Adobe Summit 2017 > L4080 > Chapter 1 – Full-text**
   - http://localhost:4502/editor.html/content/summit/l4080/chapter-1.html
2. Ensure the page is in **Edit mode**, by selecting **Edit** from the dropdown in the top right
3. Select the Search component and **click the wrench icon** to edit
4. In the Search component dialog, set the following:
   a. Search path: **/content/docs/en/aem/6-3**
   b. Click the checkbox icon to **save** dialog changes
5. Switch to **Preview mode**, by clicking Preview in the top right
6. Enter the term **oak** in the search box and press Go
7. Click through to the first few search results and note the existence of the term **oak** through-out the content

Inspect the query

1. Return the **Chapter 1 – Full-text** page
   - http://localhost:4502/editor.html/content/summit/l4080/chapter-1.html
2. Open **AEM Chrome Plug-in**
   - Chrome > View > Developer > Developer Tools > AEM tab
   - Or on macOS press Option – Command – I
3. On the Chapter 1 – Full-text page (which now has the AEM Chrome Plug-in docked to the bottom), enter the term **oak** in the search box and press **Go**



4. In the left pane of AEM Chrome Plug-in **click** the row for
   - **/content/summit/l4080/chapter-1.html?q=oak**
5. The right panel will update to with logging and query information for this request.
6. Click the **Queries** tab, and the executed query displays:

```
/jcr:root/content/docs/en/aem/_x0036_-3//element(*,
cq:Page)[(jcr:contains(., 'oak'))]
```

with the plan below it

```
[cq:Page] as [a] /* lucene:cqPageLucene(/oak:index/cqPageLucene)
+:fulltext:oak +:ancestors:/content/docs/en/aem/6-3 ft:("oak")
where (contains([a].[*], 'oak')) and (isdescendantnode([a],
[/content/docs/en/aem/6-3])) */
```

7. The plan describes what Oak index will be used to execute this query; in this case the Lucene index named **cqPageLucene** is selected for use.

Inspecting the cqPageLucene index definition

1. Open **CRXDE Lite**
   - http://localhost:4502/crx/de
2. Select **/oak:index/cqPageLucene** node
3. Core index configurations are on **cqPageLucene**
4. Full-text aggregate configuration are defined under **cqPageLucene/aggregates**
5. Property specific configurations are defined under **cqPageLucene/indexRules**

Full-text operations

1. Return the **Chapter 1 – Full-text** page

2. Try out the following full text searches using the supported operators and note the changes in results:
    a. Phrases
        i. Group phrases with using double-quotes; compare the results of the following:
            1. **"sites assets"**
            2. **sites assets**
    b. OR
        i. sites OR assets
    c. AND
        i. sites AND assets

Excerpts operations

1. Ensure the page is **in Edit mode**, by clicking **Edit** in the top right
2. Click the Search component
3. Select the Search component and **click the wrench icon** to edit
4. In the Search component dialog, set the following:
    a. Use Excepts: checked
    b. Click the checkbox icon to **save** dialog changes
5. Switch to **Preview mode**, by clicking Preview in the top right
6. Enter the term **sites** in the search box and press **Go**
    a. Note the excerpts with term **highlighting** in the results
7. Inspect the query with **AEM Chrome Plug-in** and note the **rep:excerpt(.)** function in the query.

Solution Package

1. Navigate to CRX Package Manager: **AEM Start** > **Tools** > **Deployment** > **Packages**
    - http://localhost:4502/crx/packmgr/index.jsp
2. Search for **Chapter-1**
3. Click the package to expand: L4080-Chapter1.zip
4. Click install

Chapter 2: Filtering

Property matches, to fulfill common requirements of result filtering, can restrict search. Supported property-based filtering include:

- Equals
- Not equals
- Ranges

Exercise

Define an Oak index rule for Tag-based property filtering.

1. Open the **Sites > Adobe Summit 2017 > L4080 > Chapter 2 – Filtering**
    a. http://localhost:4502/editor.html/content/summit/l4080/chapter-2.html
2. Ensure the page is **in Edit mode**, by clicking **Edit** in the top right
3. Click the Search component
4. Select the Search component and **click the wrench icon** to edit
5. In the Search component dialog, set the following:
    a. Facets > Show Facets: checked
    b. Click the checkbox icon to **save** dialog changes
6. Switch to **Preview mode**, by clicking Preview in the top right
7. Click to filter by facet
    a. Click **Versions > AEM 6.3**
    b. Click **Products > Commerce**
    c. Click **Audience > Developer**
    d. Click **Go** several times to get a sense of the average **Time Taken** by the query
8. Inspect the Query Plan using **AEM Chrome Plug-in > Queries**

```
[cq:Page] as [a] /* lucene:cqPageLucene(/oak:index/cqPageLucene)
:ancestors:/content/docs/en/aem where ([a].[jcr:content/cq:tags]
in('version:aem63', '/etc/tags/version/aem63')) and
([a].[jcr:content/cq:tags] in('product:sites',
'/etc/tags/product/sites')) and (isdescendantnode([a],
[/content/docs/en/aem])) */
```

    a. Note **/oak:index/cqPageLucene** is used to evaluate the query, and there are property restrictions on **jcr:content/cq:tags.**
9. Open **CRXDE Lite**
    - http://localhost:4502/crx/de
6. Select **/oak:index/cqPageLucene/indexRules/cq:Page/properties** node
    - Each node under **properties** defines how a specific property under the cq:Page

hierarchy is indexed.

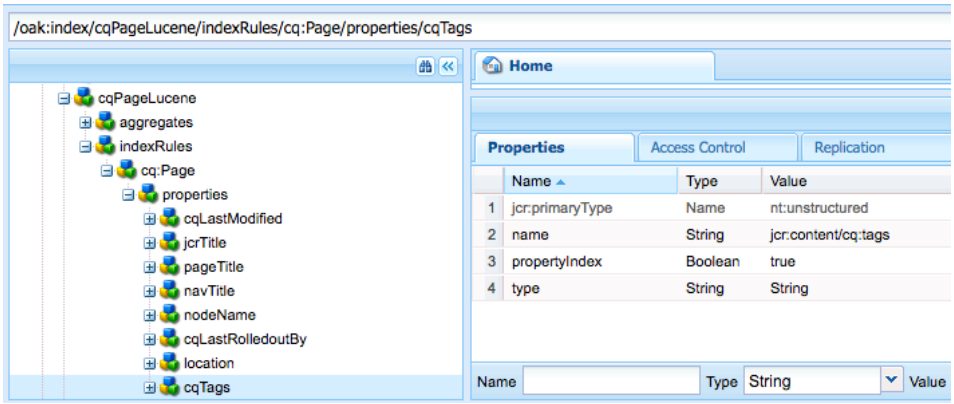
- Note there is no property index rules for **jcr:content/cq:tags**

7. **Create a index rule for the property [cq:Page]/jcr:content/cq:tags**
   - While **/oak:index/cqPageLucene/indexRules/cq:Page/properties** is selected
   - Click **Create… > Create Node**

Node Name	Node Type
`cqTags`	`nt:unstructured`

   - Add the following properties to the new **cqTags** node

Property Name	Property Type	Property Value
`propertyIndex`	`Boolean`	`true`
`type`	`String`	`String`
`name`	`String`	`jcr:content/cq:tags`



8. Click **Save All** in the top left to save changes
9. **Re-index cqPageLucene** via Index Manager
   - http://localhost:4502/libs/granite/operations/content/diagnosis/tool.html/granite_oakindexmanager
   - ***Note:*** *the index manager might time out while it re-indexes. It should take less than one minute to finish. Simply refresh the page until the cqPageLucene row is no longer red and the re-index icon is no longer spinning.*
10. Return to **Chapter 2 – Filtering**
11. Click **Go** several times to re-issue the query and note the average **Time Taken**
    - This new Time Taken should be noticeably less than the **Time Taken** in Step 7.

Pro-tip: For small content sets like this Lab, the cqPageLucene/aggregates configuration covers [cq:Page]/jcr:content/cq:tags making the property restrictions fast (100-200ms). As the body of content grows large (10k's to millions of pages) the index rule greatly increases performance.

Solution Package

1. Navigate to CRX Package Manager: AEM Start > Tools > Deployment > Packages
    a. http://localhost:4502/crx/packmgr/index.jsp
2. Search for **Chapter-2**
3. Click the package to expand: L4080-Chapter2.zip
4. Click install

Chapter 3: Pagination

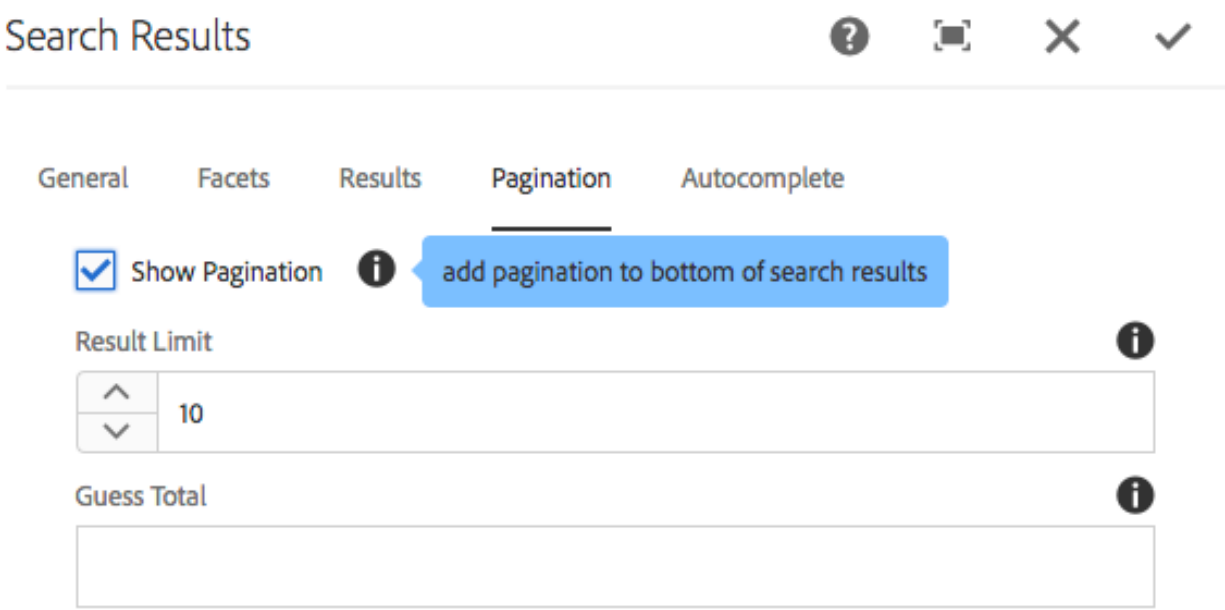
Pagination of search results is an important component in any search implementation. AEM's QueryBuilder includes several options to make pagination easier to implement. We will be working with the following QueryBuilder properties behind the scenes when working with pagination:

- **p.limit**: Defines the number of results to return for a given query. The default is **10** and **-1** will return all results.
- **p.offset:** a 0-based value that defines where the search results start. Changing this parameter is used to go to the "next" page of search results.
- **p.guessTotal**: Using this parameter can significantly improve the performance of queries that return large result sets because Oak does not need to calculate the exact number of the result set. The disadvantage is that since we do not know the exact results implementing pagination can be difficult.

Exercise

Turn on Pagination

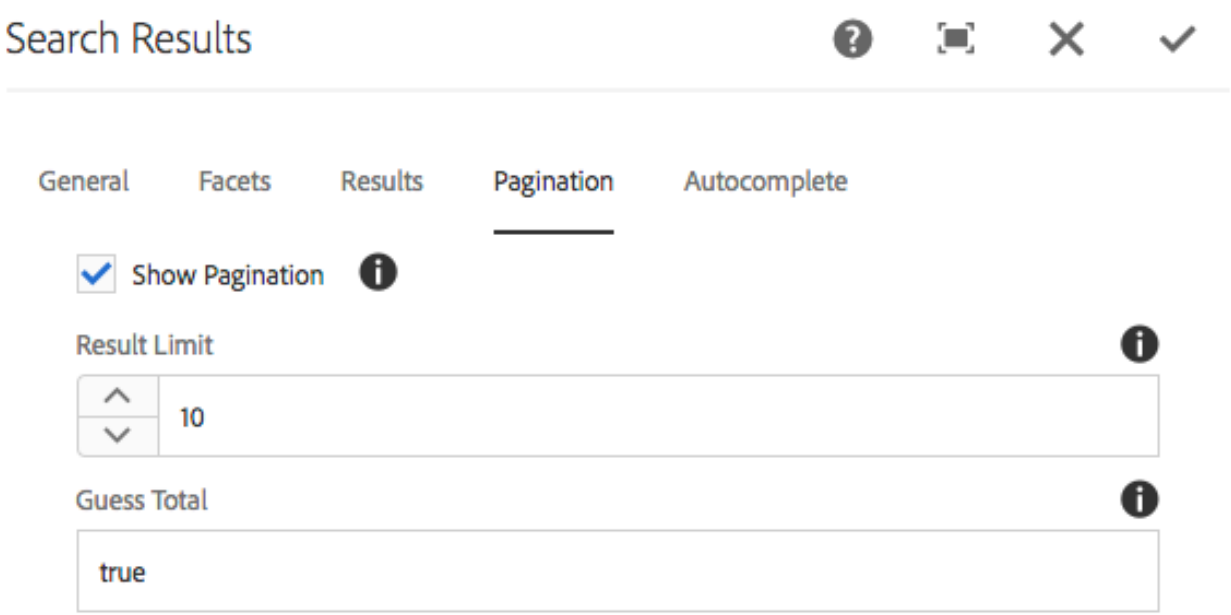
1. Open the **Sites** > **Adobe Summit 2017** > **L4080** > **Chapter 3 – Pagination**
    - http://localhost:4502/editor.html/content/summit/l4080/chapter-3.html
2. In **Edit** mode select the Search component and open the component dialog by clicking the **wrench** icon.
3. Select the **Pagination** Tab
4. Click the checkbox to **Show Pagination** (leave the other fields as is) and save the dialog by clicking the **checkmark**.



5. Switch the page mode to **Preview** (upper right hand corner) and perform a full-text search that should result in multiple pages of results.
    i. Search for ***template development***
6. Notice the pagination at the bottom of the page. Click through the pagination to view multiple pages of results. Click to the last page (you might have to repeatedly click higher numbered pages to get the end. Notice that the 'Next' button has disappeared. Make a mental note of the Time taken in milliseconds.

Use guessTotal=true to increase performance

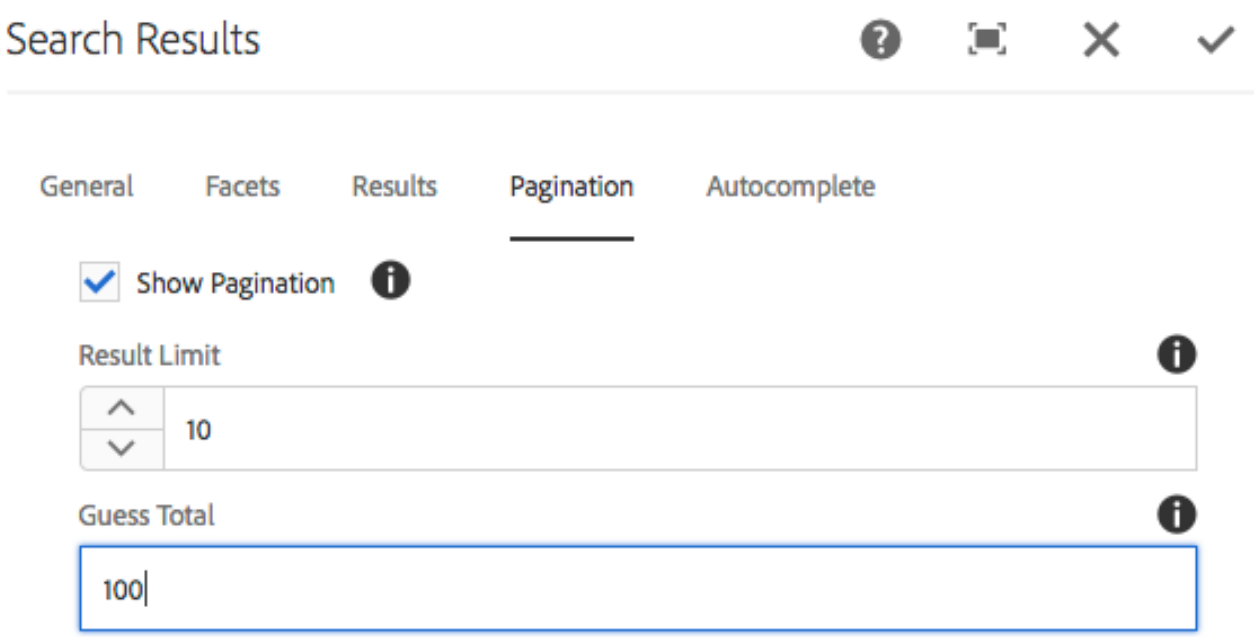
1. Switch the page mode to **Edit** and re-open the Search component dialog. Navigate to the Pagination tab and type **true** in the ***Guess Total*** field. Save and close the dialog.



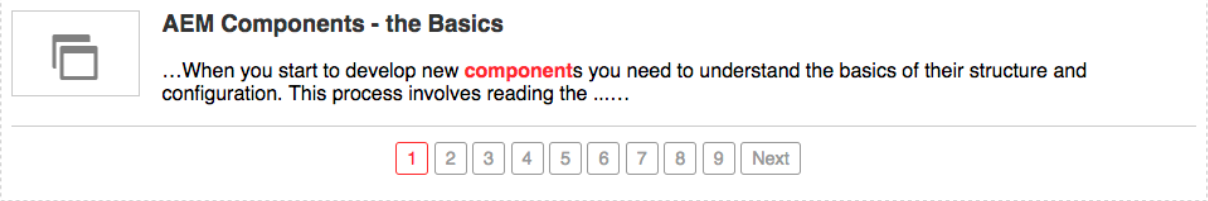
2. Switch the page mode to **Preview** and perform the same keyword search as in the previous step.
    - Search ***template development***.
3. Notice that the pagination has changed to only show the next page of results. You should be able to click through to the last page of results, but you will need to click 'Next' many more times. The time taken should be faster than in previous steps.

Use guessTotal=100 to read the first 100 results

1. Switch the page mode to **Edit** and open the Search Component dialog.
2. Navigate to the **Pagination** Tab and update the **Guess Total** field value to **100**. Save and close the dialog.

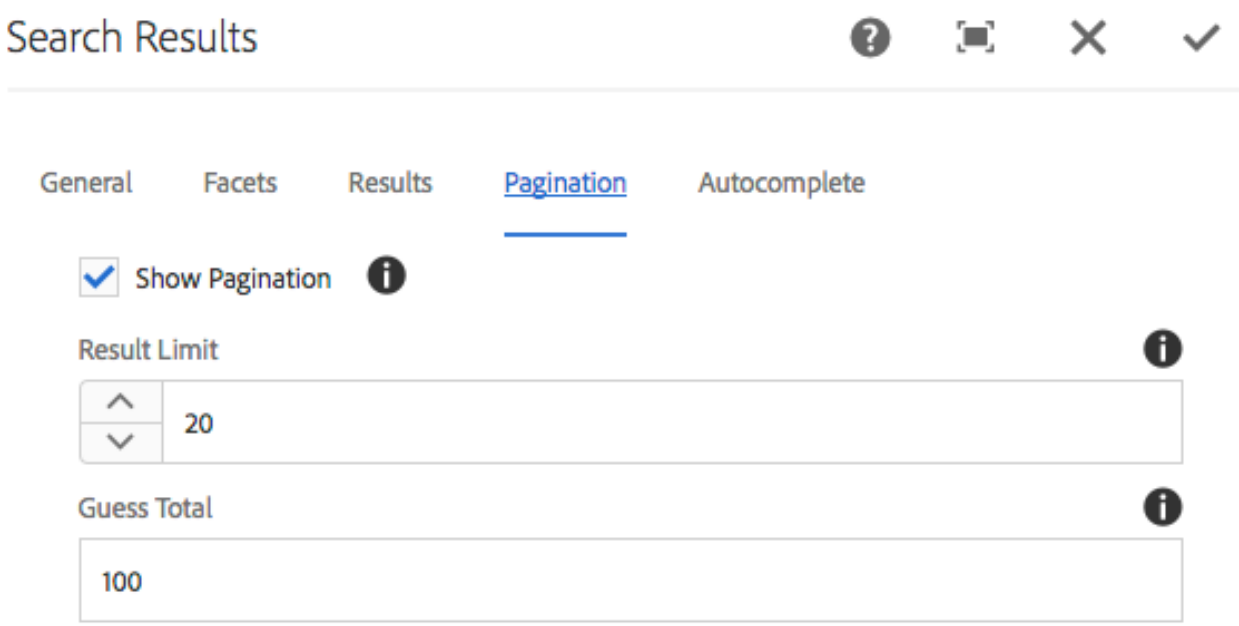


3. Switch the page mode to **Preview** and perform the same keyword search as in previous steps
   - Search ***template development***.
4. Notice that the pagination now shows more than 2 pages. However, since **guessTotal** is being used there is the potential for an extra page to be shown. Click immediately to the last page. Depending on the size of the result set you may end up on a page with no results because the calculated offset exceeded the result set total size.



Update the search result size

1. Switch the page mode to **Edit** and open the Search Component dialog.
2. Change the limit field to increase or decrease the number of results displayed per page. Save and close the dialog.



3. Switch the mode to 'Preview' and perform the same keyword search as previously.
    a. Search ***template development***.
4. Notice that the number of results per page has changed and the pagination has been updated to match.

Pro-tip: For small content sets like this Lab you might not see a huge disparity between turning on guessTotal but for larger content sets and more complex queries it can significantly speed query performance. guessTotal=true should always be used when returning a fixed limit or pagination is not needed.

Bonus Exercise

1. Open the QueryBuilder debugger: http://localhost:4502/libs/cq/search/content/querydebug.html
2. Type the following text in the text area and execute the search:

```
fulltext=aem sites
type=cq:Page
```

3. Notice the number of results and the time it took to execute.

4. Perform the same query but with **guessTotal** turned on:

```
fulltext=aem sites
type=cq:Page
p.guessTotal=true
```

5. Notice that you no longer get the exact total number of results but the time taken to return should have decreased.

Solution Package

1. Navigate to CRX Package Manager: AEM Start > Tools > Deployment > Packages
    a. http://localhost:4502/crx/packmgr/index.jsp
2. Search for **Chapter-3**
3. Click the package to expand: L4080-Chapter3.zip
4. Click install

Reference Links

https://docs.adobe.com/docs/en/aem/6-2/develop/search/querybuilder-api.html

Chapter 4: Suggestions

Suggestions provide lists of terms or phrases that exist in the content match a use-provided initial search term.

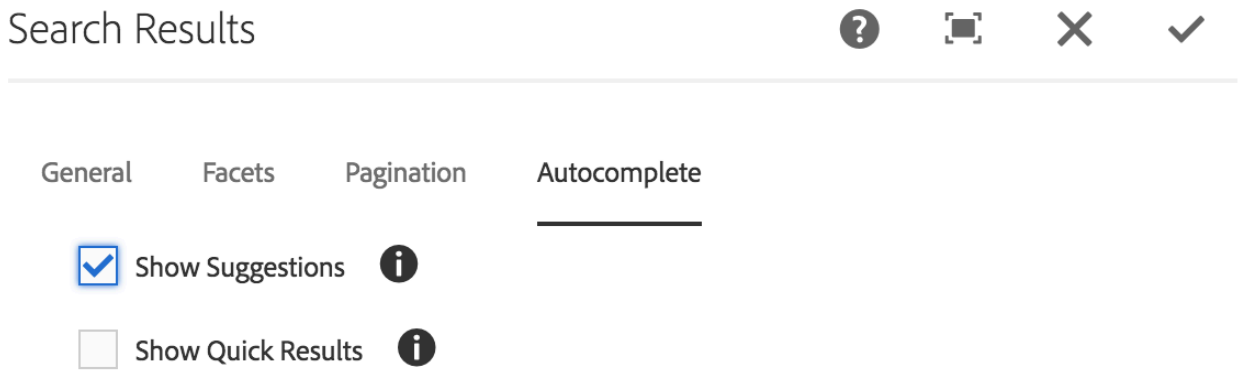
There are two types of suggestion configurations:

- Property-based
  - Returns the entire value (multi-word) of a property as a suggested term.
- Aggregate-based
  - Returns a list of single-word terms that match the user-provided search term.

Exercise

Turn on suggestions

1. Open the **Sites** > **Adobe Summit 2017** > **L4080** > **Chapter 4 – Suggestions**
   - http://localhost:4502/editor.html/content/summit/l4080/chapter-4.html
2. In **Edit** mode select the Search component and open the component dialog by clicking the wrench icon.
3. Select the **Autocomplete** Tab
4. Check the box to **Show Suggestions**



5. Save and close the dialog

Test search suggestions

1. Switch the mode to **Preview**
2. Start typing the search term **metadata** into the search field
   a. Note the multi-word suggestions as the term is typed in. This is because, OOTB property-based suggestions are used.

   b. Click on a suggestion to search and note the number of search results.
3. Open CRXDE Lite
   a. http://localhost:4502/crx/de
4. Navigate to **/oak:index/cqPageLucene/indexRules/cq:Page/properties**
   a. Review the properties for the nodes
      i. **.../properties/jcrTitle**
      ii. **... /properties/nodeName**
   b. Note they both have the **useInSuggest** property set to **true** which is why the current suggestions are the multi-word values of these 2 properties.
5. Create a new node named **suggestion**, under **/oak:index/cqPageLucene**

Node Name	Node Type
`suggestion`	`nt:unstructured`

6. Add the following properties to the **suggestion** node

Property Name	Property Type	Property Value
`suggestAnalyzed`	`Boolean`	`true`

7. Click **Save All** in the top left to save changes
8. **Re-index cqPageLucene** via Index Manager
   a. http://localhost:4502/libs/granite/operations/content/diagnosis/tool.html/granite_oakindexmanager
9. Return to **Chapter 4 – Suggestions** page
10. Start typing in a search query and note how the suggestions are 1 word and click on a suggestion to search.

Solution Package

1. Navigate to CRX Package Manager: AEM Start > Tools > Deployment > Packages
   a. http://localhost:4502/crx/packmgr/index.jsp
2. Search for **Chapter-4**
3. Click the package to expand: L4080-Chapter4.zip
4. Click install

Chapter 5: Analyzers

AEM search allows Analyzers to be configured per index. Analyzers dictate how content is indexed into the search indexes, and can also augment how queries are executed against them. This exercise set up Stemming, Synonyms, Stop words and HTML Stripping.

Setup Package

For this series of exercises, install the Package **L4080-Chapter5-Setup.zip** via CRX Package Manager. This package augments the **/oak:index/cqPageLucene** index with a basic analyzer configurations.

- Standard character mapping
- Standard tokenizer
- Lower-case token filter

1. Navigate to CRX Package Manager: AEM Start > Tools > Deployment > Packages
    a. http://localhost:4502/crx/packmgr/index.jsp
2. Search for **Chapter-5**
3. Select the **package L4080-Chapter5-Setup.zip**
4. Click install

IMPORTANT DO NOT INSTALL **L4080-Chapter5-Solution.zip** package.

For the following exercises

1. Open the **Sites > Adobe Summit 2017 > L4080 > Chapter 5 – Analyzers**
    - http://localhost:4502/editor.html/content/summit/l4080/chapter-5.html

Stemming

Stemming converts user-provided search words into their linguistic "root" thereby intelligently expanding the scope of the full-text search.

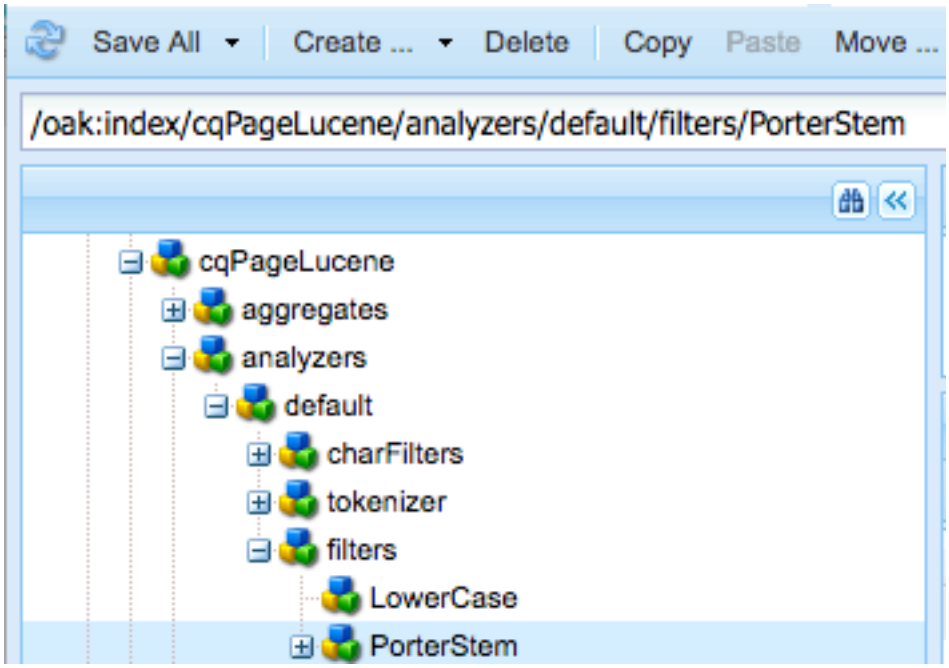
Stemming both an index time and query time activity. At index time, stemmed terms (rather than full terms) are stored in the full text index. At query time, the user provided search terms are stemmed and passed in as the full-text term.

For example

- Given the provided term: **developing**
- The stemmer will derive the root word: **develop**
- Which includes content that contains derived forms such as "**developer**", and "**development**".

Stemming exercise

Add the **PorterStemFilter** to the **cqPageLucene** index.



1. Perform three searches using the keywords
    - **development (~134 results)**
    - **developer (~350 results)**
    - **developing (~110 results)**

and note how the results are **different** between all three searches.

2. Open CRXDE Lite
    - http://localhost:4502/crx/de
3. Select **/oak:index/cqPageLucene** and click **refresh** in the top left
4. Create a new node under **/oak:index/cqPageLucene/analyzers/default/filters**

Node Name	Node Type
`PorterStem`	`nt:unstructured`

5. Move the new PorterStem node to be below the LowerCase filter
6. Click **Save all** in the top left
7. Re-index **cqPageLucene** via Index Manager
    - http://localhost:4502/libs/granite/operations/content/diagnosis/tool.html/granite_oakindexmanager
8. Perform the three searches in Step #1 and note that the results are the **same (~449 results)**.
    - This is because the PorterStemmer stemmed both the indexed terms and query terms to the stem **develop**.

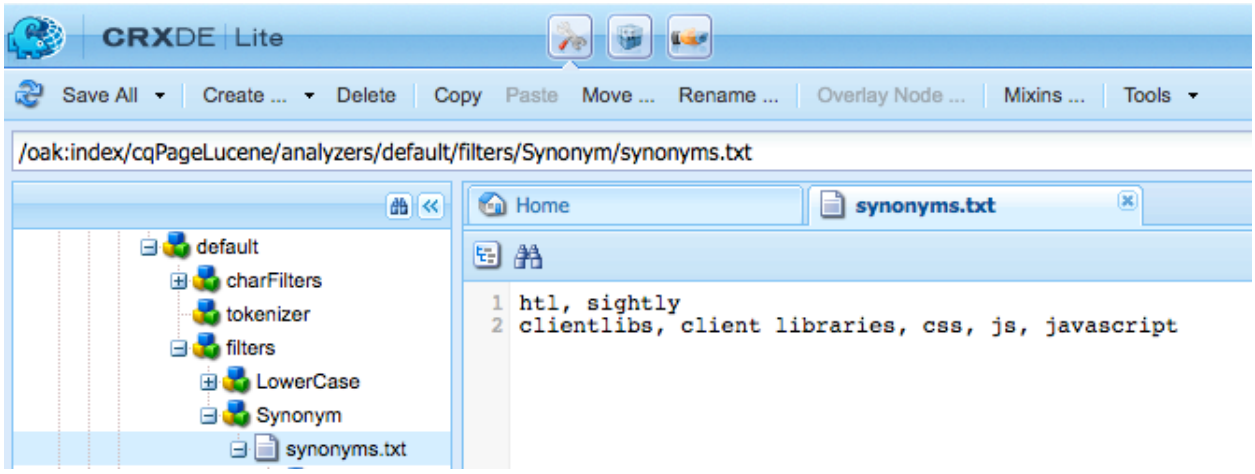
Synonyms

Synonyms allow different terms with equivalent meaning to be considered the same by full-text search.

Pro-tip: Place the Synonym filter node after LowerCase but BEFORE PorterStem

Synonyms exercise

Create a custom synonym list for the **cqPageLucene** index.



1. Perform a search using the keyword **sightly** and note the lack of results, perform a search using the keyword **HTL** and note there are many results.

2. Open CRXDE Lite
   - http://localhost:4502/crx/de
3. Create a new node named **Synonym** under **/oak:index/cqPageLucene/analyzers/default/filters**

Node Name	Node Type
`Synonym`	`nt:unstructured`

4. Move the new Synonym node to be AFTER LowerCase and BEFORE PorterStem
5. Click **Save all** in the top left
6. Create a File named **synonyms.txt** under **/oak:index/cqPageLucene/analyzers/default/filters/Synonym**

Node Name	Node Type
`synonyms.txt`	`nt:file`

7. Double-click to **edit synonyms.txt**, and enter the synonyms:

```
htl, sightly
```

8. Add the property **synonyms** to node **/oak:index/cqPageLucene/indexRules/analyzers/default/filters/Synonym**

Property Name	Property Type	Property Value
`synonyms`	`String`	`synonyms.txt`

9. Click **Save all** in the top left
10. Re-index **cqPageLucene** via Index Manager
11. Perform a search using the keyword **sightly** and note all the HTL-centric results
    - Searching for either term **sightly** or **htl** should yield the same **~17 results** as they are now considered equivalent terms.

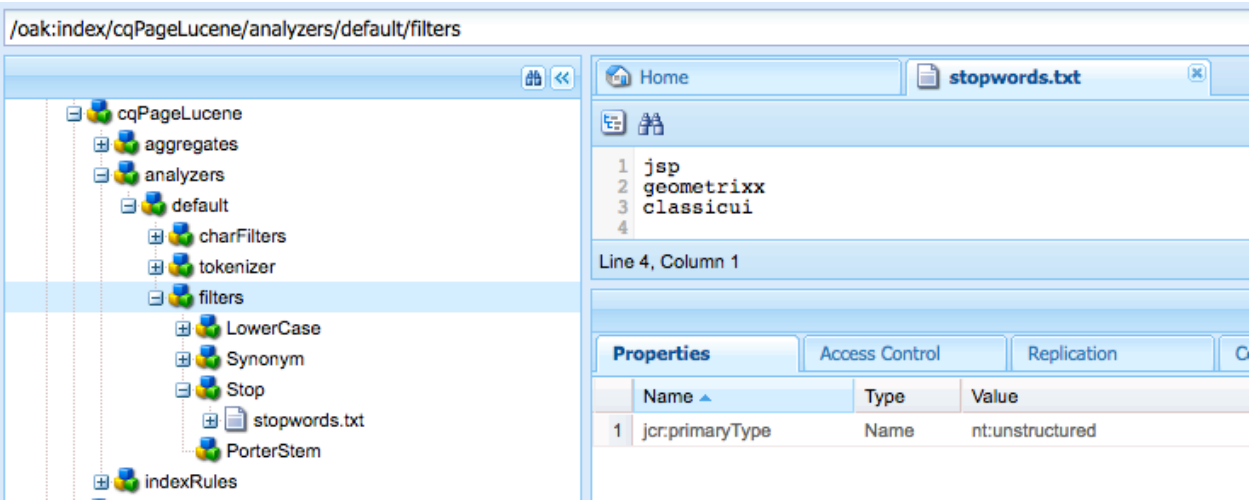
Stop words

Stop words are effectively a black list of words that will not be added to the search index and thus unsearchable. Managed industries may add subjective terms as stop terms, or search over user-generated content may leverage them to keep profanities being searchable.

Pro-tip: Place the Stop filter node after both the LowerCase and Synonym filter nodes

Stop words exercise

Create a custom Stop Words Filter for the **cqPageLucene** index.



1. Perform searches using the keywords, and note the large number of results.
   - **jsp**
   - **geometrixx**
   - **classic ui**
2. Open CRXDE Lite
   - http://localhost:4502/crx/de
3. Create a node named **Stop** under **/oak:index/cqPageLucene/analyzers/default/filters**

Node Name	Node Type
`Stop`	`nt:unstructured`

4. Move the Stop node below the Synonym node
5. Click **Save all** in the top left
6. Create a File named **stopwords.txt** under

/oak:index/cqPageLucene/analyzers/default/filters/Stop

Node Name	Node Type
`stopwords.txt`	`nt:file`

7. Double click to **edit stopwords.txt**, and enter a few stop words, one per line, for example:

```
jsp
geometrixx
classicui
```

8. Add the property **words** to **/oak:index/cqPageLucene/indexRules/analyzers/default/filters/Stop**

Property Name	Property Type	Property Value
`words`	`String`	`stopwords.txt`

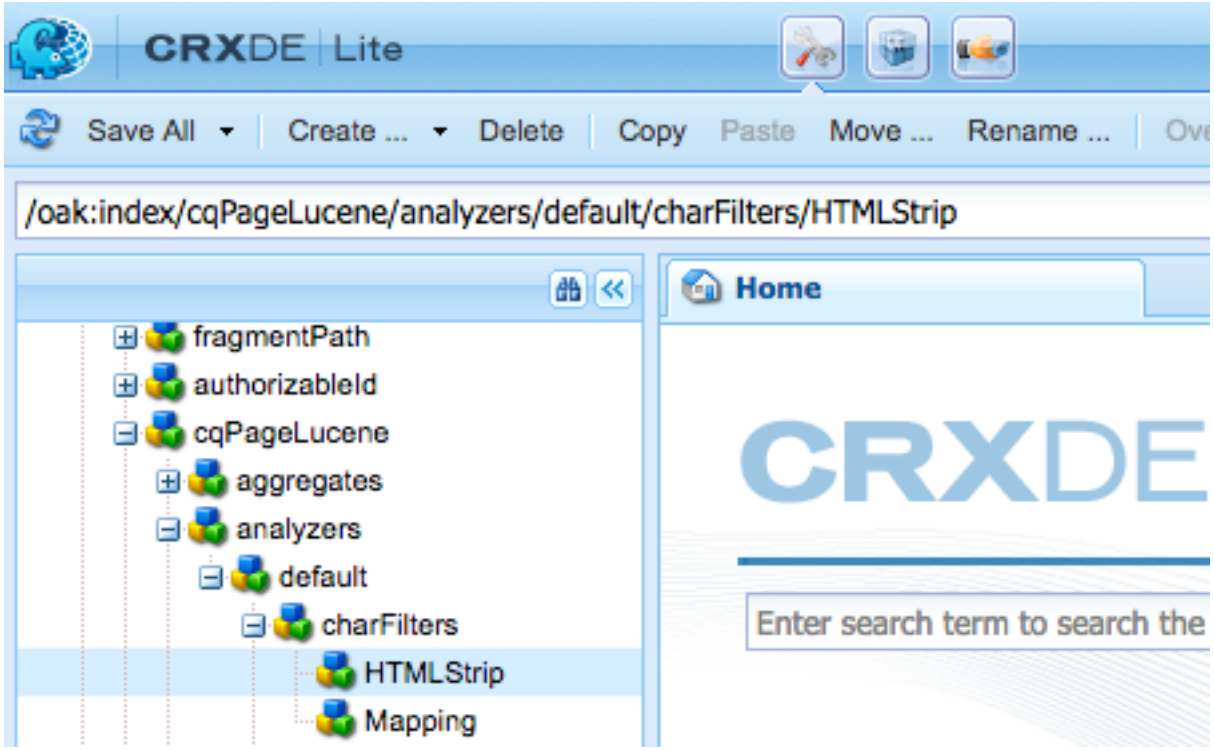
9. Click **Save All** in the top left to save changes
10. Re-index **cqPageLucene** via Index Manager
    - http://localhost:4502/libs/granite/operations/content/diagnosis/tool.html/granite_oakindexmanager
11. Perform the three searches in Step #1 and note now there are **no results**

HTML Strip

HTML can be automatically removed from the search index; so as non-content elements don't populate the content search space. This can be helpful when HTML it stored in page properties, such as with Rich Text editors, Table or Content Fragment components.

HTML Strip exercise

Create an HTMLStrip CharFilter to the cqPageLucene index to prevent any HTML artifacts on the Page node from influencing search results.



1. Perform a search using the keyword **tahoma**
2. Click on any result, view source on the result page, and search for **tahoma** in the HTML source and note it only appears as part of a HTML attribute.
3. Open CRXDE Lite
   - http://localhost:4502/crx/de
4. Create a new node named **HTMLStrip** under **/oak:index/cqPageLucene/analyzers/default**

Node Name	Node Type
`HTMLStrip`	`nt:unstructured`

5. Move the new HTMLStrip node to be ABOVE the Mapping node
6. Click **Save All** in the top left
7. Re-index cqPageLucene via Index Manager
    - http://localhost:4502/libs/granite/operations/content/diagnosis/tool.html/granite_oakindexmanager
8. Perform the Search from step #1 and notice there are no results!

Pro-tip: When possible, avoid storing HTML, CSS or JavaScript in jcr properties!

Solution Package

1. Navigate to CRX Package Manager: AEM Start > Tools > Deployment > Packages
    a. http://localhost:4502/crx/packmgr/index.jsp
2. Search for **Chapter-5**
3. Click the package to expand: L4080-Chapter5-Solution.zip
4. Click install

Chapter 6: Boosting

Lucene full-text indexing supports the ability to boost or weight specific metadata properties. This allows specified properties to be ranked higher than others, thus when a search term is found in a boosted property the result is moved up in the search results.

*Note Lucene does a decent job of ranking metadata properties as it considers the length of the property when evaluating the result score. A title field is typically shorter than a description and thus search terms found in the title would typically be ranked higher by default.

Exercise

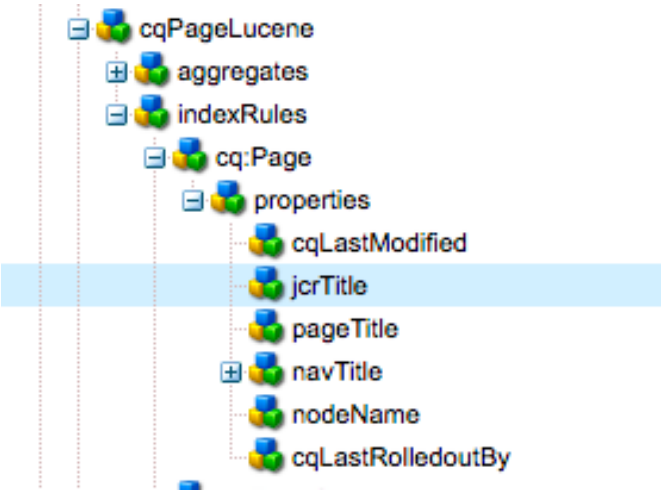
Perform search without boosting

1. Open **Sites / Adobe Summit 2017 / L4080 / Chapter 6 – Boosting**
   - http://localhost:4502/editor.html/content/summit/l4080/chapter-6.html
2. Perform a search with the search term: **forms**
   - Note the ~4th result titled **Overview;** this does not have the keyword **forms** in the title
3. Keep this tab open for comparing results after we enable boosting

Update cqPageLucene index to enable boosting

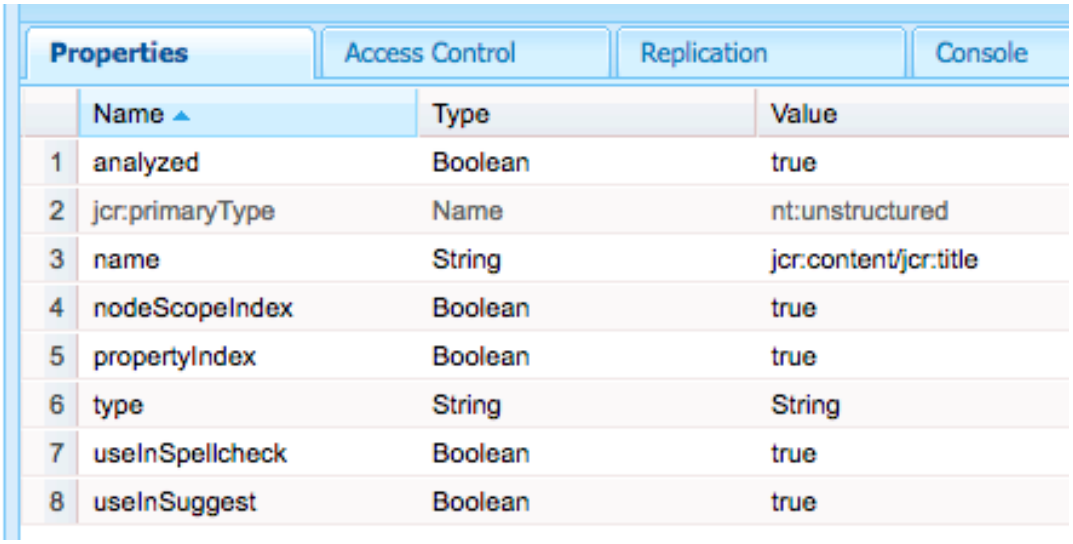
1. In a new tab navigate to CRXDE Lite:
   - http://localhost:4502/crx/de/index.jsp
2. In the left side panel expand the oak:index tree and navigate to the **cqPageLucene** index
   - Expand the cqPageLucene > indexRules > cq:Page > properties > and select jcrTitle

```
/oak:index/cqPageLucene/indexRules/cq:Page/properties/jcrTitle
```



3. Add two properties to the **jcrTitle** node with the following values:

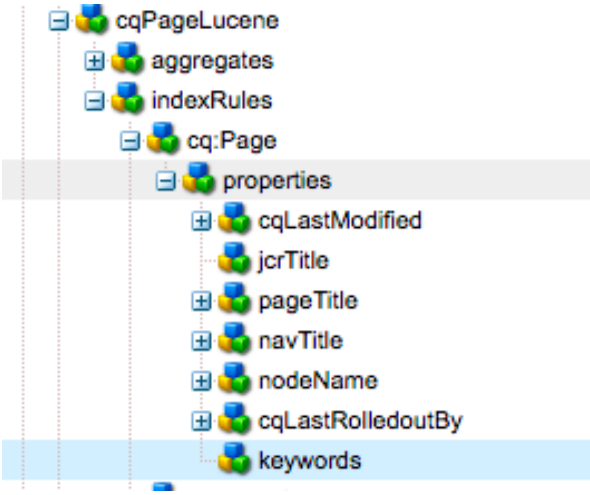
Property Name	Property Type	Property Value
`analyzed`	`Boolean`	`true`
`nodeScopeIndex`	`Boolean`	`true`



Properties | Access Control | Replication | Console

	Name	Type	Value
1	analyzed	Boolean	true
2	jcr:primaryType	Name	nt:unstructured
3	name	String	jcr:content/jcr:title
4	nodeScopeIndex	Boolean	true
5	propertyIndex	Boolean	true
6	type	String	String
7	useInSpellcheck	Boolean	true
8	useInSuggest	Boolean	true

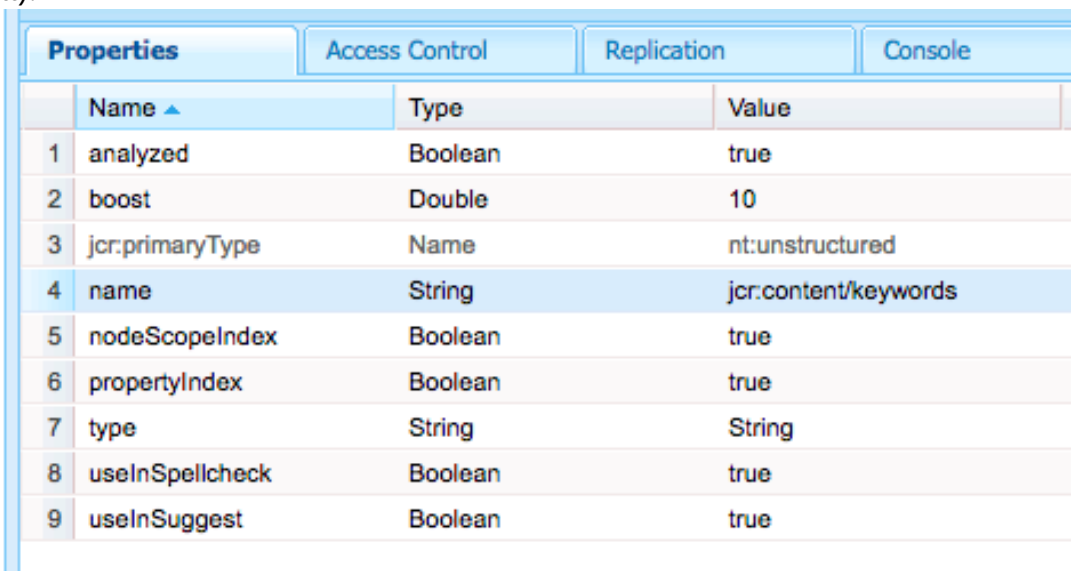
4. Click **Save All** in the upper left hand corner to save the changes to the index (the red marks should disappear)
5. Right-click the **jcrTitle** node and select **Copy** from the menu
6. Right-click the **properties** node (parent of the jcrTitle node) and click **Paste**
7. A new node will be created named **Copy of jcrTitle**.
8. Right-click this node and rename to **keywords**



9. Right-click the keywords node and click **Refresh**. The properties of the node should now appear in the center console.
10. Update the **name** property from **jcr:content/jcr:title** to **jcr:content/keywords**
11. Add a new property with the following values

Property Name	Property Type	Property Value
boost	Double	10

12. The keyword node should now have the following properties (make sure to click **Save All**):



	Name	Type	Value
1	analyzed	Boolean	true
2	boost	Double	10
3	jcr:primaryType	Name	nt:unstructured
4	name	String	jcr:content/keywords
5	nodeScopeIndex	Boolean	true
6	propertyIndex	Boolean	true
7	type	String	String
8	useInSpellcheck	Boolean	true
9	useInSuggest	Boolean	true

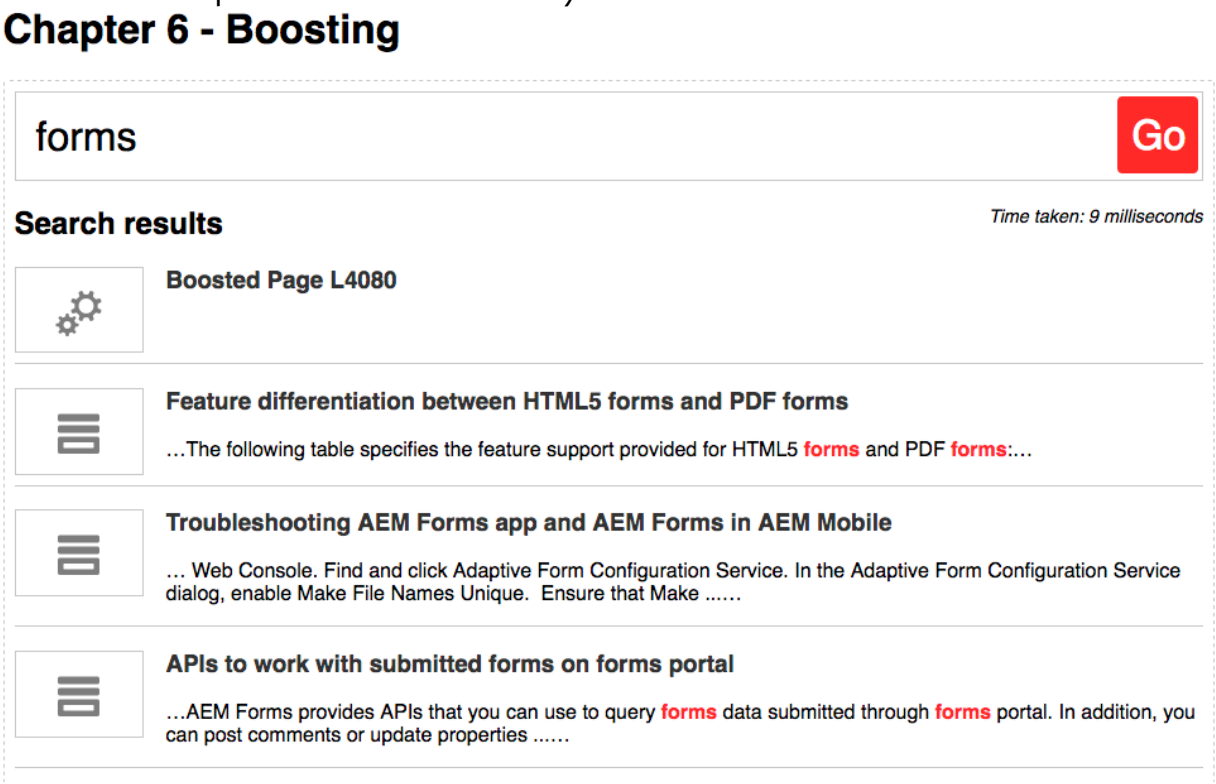
13. Re-index **cqPageLucene** via Index Manager

Perform Search with boosting

1. Open new tab and navigate to the Chapter 6 boosting page (use a different tab then at the beginning of the exercise so you can compare results)

- http://localhost:4502/editor.html/content/summit/l4080/chapter-6.html

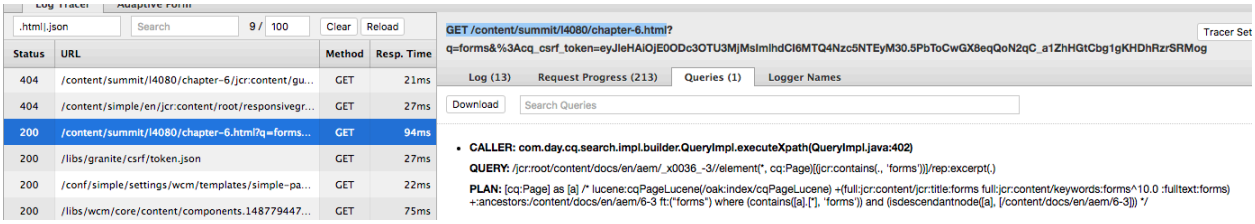
2. In Preview mode perform a search of the keyword **forms**



3. The first page of results should be different then at the beginning of the exercise.
4. The first result should be a page named **Boosted Page L4080**.
   - *This page does not have any content except for a **keywords** metadata property with a value of **forms***
5. Notice that the other search results on the first page all have the term **forms** in the title.

Inspect the Query Plan using AEM Chrome Plug-in

1. Open Chrome Developer Tools
2. Navigate to the AEM tab
3. Perform a full-text search with the term **forms**
4. Select the /content/summit/l4080/chapter-6.html request in the left-hand panel.
5. In the center panel select the Queries (1) tab



6.

   In the Plan you should now see full-text applied to the **jcr:title** property and boosting of **10** applied to **keywords** property

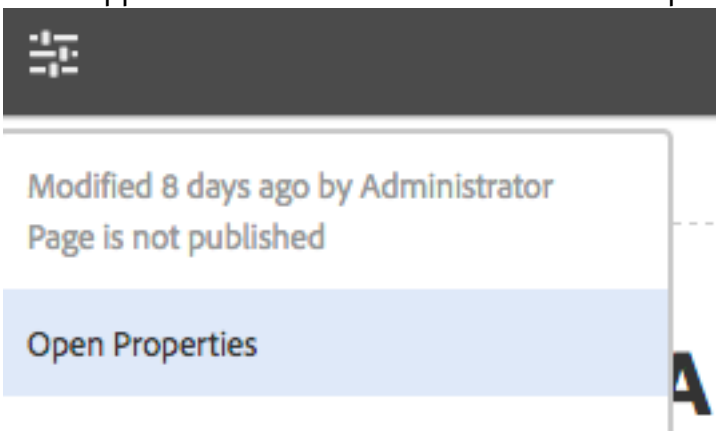
```
PLAN: [cq:Page] as [a] /*
lucene:cqPageLucene(/oak:index/cqPageLucene)
+(full:jcr:content/jcr:title:forms
full:jcr:content/keywords:forms^10.0 :fulltext:forms)
+:ancestors:/content/docs/en/aem/6-3 ft:("forms") where
(contains([a].[*], 'forms')) and (isdescendantnode([a],
[/content/docs/en/aem/6-3])) */
```

Pro-tip: Use explicit boosting sparingly. In most cases simply adding the property to the full-text index and setting analyzed=true will suffice. The Lucene algorithm already does a good job of evaluating what properties are more important based on text length.

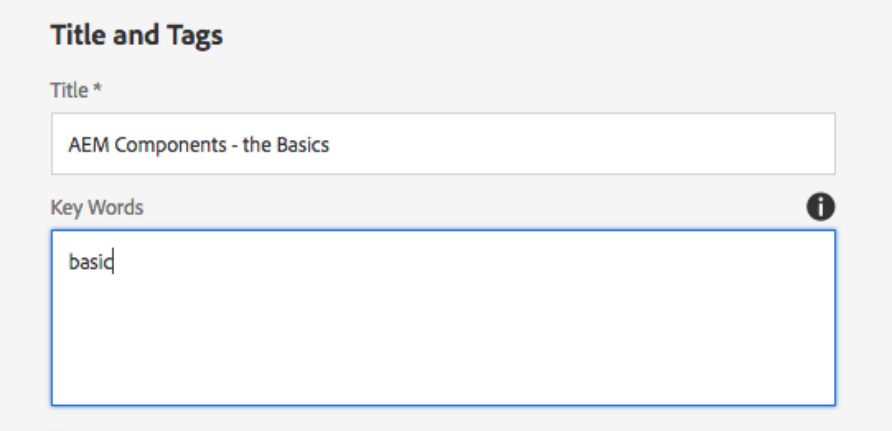
Bonus Exercise

Update the keywords property of a different page

1. Open a page from the search results. Make sure to add the /editor.html prefix to the beginning of the URL to allow editing of Page Properties
   - http://localhost:4502/editor.html/content/docs/en/aem/6-3/develop/components/components-basics.html
2. In the upper left select the menu and from the dropdown click Page Properties



3. Add a new value to the Keywords field i.e **basic**



4. Save and close the dialog
5. Return to the Chapter 6 page
   - http://localhost:4502/editor.html/content/summit/l4080/chapter-6.html
6. Switch the page mode to 'Preview'
7. Perform a search with the term **basic**
8. The page modified in Step 1 should be the first result

Solution Package

1. Navigate to CRX Package Manager: AEM Start > Tools > Deployment > Packages
   a. http://localhost:4502/crx/packmgr/index.jsp
2. Search for **Chapter-6**
3. Click the package to expand: L4080-Chapter6-Solution.zip
4. Click install

Reference Links

https://docs.adobe.com/docs/pt-br/aem/6-2/deploy/best-practices/best-practices-for-queries-and-indexing.html#Tips%20for%20Creating%20Efficient%20Indexes

http://jackrabbit.apache.org/oak/docs/query/lucene.html#boost

https://wiki.apache.org/lucene-java/LuceneFAQ#How_do_I_make_sure_that_a_match_in_a_document_title_has_greater_weight_than_a_match_in_a_document_body.3F

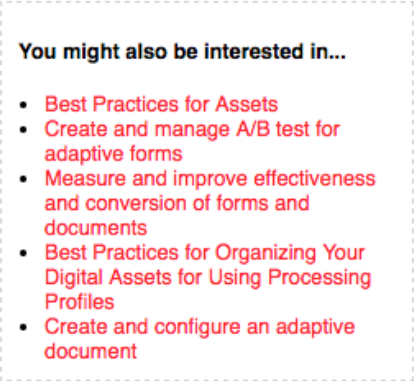
Chapter 7: Similarity

Oak Lucene indexes also support Similarity Queries. The idea behind the similarity query is that it will return nodes that have similar content to the node specified in the query. This can be useful when attempting to implement a "More Like this…" component.

Exercise

View the Similar Results Component

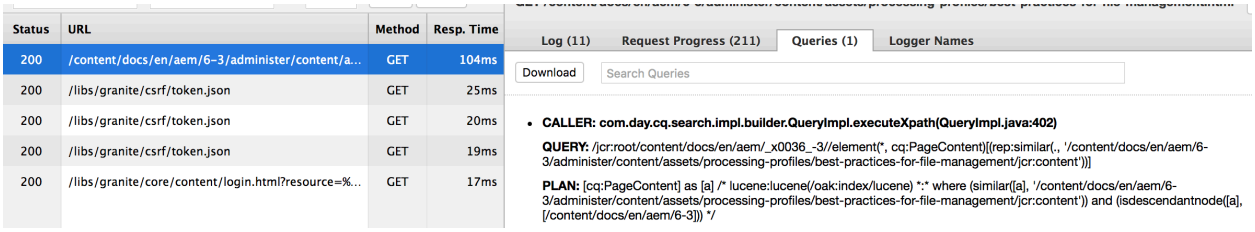
1. Open **Sites / Adobe Summit 2017 / L4080 / Chapter 7 – Similarity**
   - http://localhost:4502/editor.html/content/summit/l4080/chapter-7.html
2. Perform a full-text search with term **asset metadata**
3. Click one of the search results to open
   - http://localhost:4502/content/docs/en/aem/6-3/administer/content/assets/metadata.html
4. Notice the box in the side bar with the heading **You might also be interested in…**



5. There should be 5 links that have similar content as the current page.
6. Click some of the links in the component and see how the similar results change as you navigate to other pages

Inspect the Similar Results Query

1. While on one of the results pages open the Chrome Developer tools -> AEM Tab
2. Refresh the page and select the page request in the left-hand panel
3. In the main panel select the Queries (1) tab.



4. Notice the **rep:similar** function in the Query. The query searches for other jcr:content nodes that are similar to the one beneath the current page.

Pro-tip: We found the easiest way to take advantage of similarity search was to search against the cq:PageContent (jcr:content) node beneath a page. Then before returning our results to our HTL script we moved the hit result path up one level to return the cq:Page.

Reference Links

http://jackrabbit.apache.org/oak/docs/query/query-engine.html#Similarity_Queries

Chapter 8: Putting it all together

We have included a Search Page with all the features enabled in previous exercises. Experiment with the different search capabilities and use the tools in previous exercises to analyze the queries running behind the scenes.

Exercise

4. Navigate to **Sites / Adobe Summit 2017 / L4080 / Chapter 8 – Putting it all together**
    a. http://localhost:4502/editor.html/content/summit/l4080/chapter-8.html
2. Search!

Note that this final implementation enables Quick Results that display below the Suggestions. This makes use of the SearchResults Sling Model exposed as JSON via Sling Model Exporter.

Download the code

Visit the public github.com repository for the code-base used in this lab.

- https://www.github.com/Adobe-Marketing-Cloud/aem-guides/tree/master/simple-search-guide

Chapter 9: Traversing queries

For this exercise, we will attempt to execute a QueryBuilder-based query lists all the component nodes that were rolled out by the msm-service user and order the nodes descending by their roll-out date.

1. Double-click to **open the error.log** from the AEM Logs folder on the Desktop
    - The log should open in Console app for macOS
2. In **Chrome**, navigate to **QueryBuilder Debugger**
    - http://localhost:4502/libs/cq/search/content/querydebug.html
3. Execute the QueryBuilder query

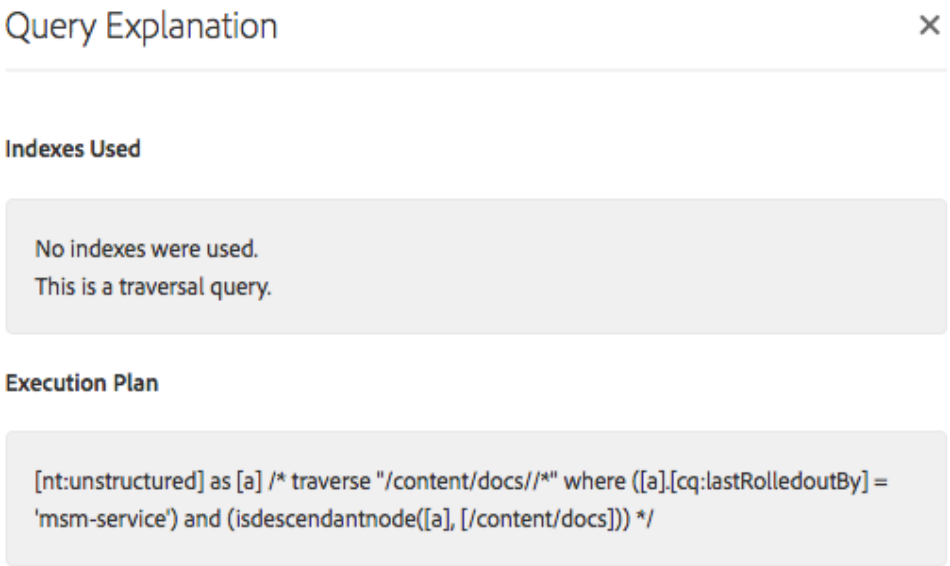
```
type=nt:unstructured
path=/content/docs
property=cq:lastRolledoutBy
property.value=msm-service
orderby=@cq:lastRolledout
orderby.sort=desc
```

    - Or, to avoid typing, click on the Chrome bookmark:
        i. **Chapters > Chapter 9**
    - While the query executes, watch the logs in Console app.
    - After a few seconds the query will fail, and an exception will appear on the QueryBuilder Debugger web page.

4. Locate the executed XPath query in the logs in the Console App
    - Or, re-run the Query and use AEM Chrome Plug-in to find the query.

```
/jcr:root/content/docs//element(*,
nt:unstructured)[(@cq:lastRolledoutBy = 'msm-service')] order by
@cq:lastRolledout descending
```

5. In a new browser tab, navigate to **Explain Query**
    - http://localhost:4502/libs/granite/operations/content/diagnosis/tool.html/granite_queryperformance > Explain Query tab
6. Select **xPath** as the **Language**
7. Paste the **query from Step #4** into the **Query text box**, and click **Explain**
    - The explanation calls out this is a traversal query and no indexes are used.



8. In a new browser tab, navigate to **Oak Index Definition Generator**
    - http://oakutils.appspot.com/generate/index
9. Replace contents of the **Queries text box** with the XPath statement located in Step #4 and click **Generate**
10. The second text box populates with the Oak index definition required to satisfy the query
    - The resulting index definition is provided as an installable AEM package:
        i. **L4080-Chapter9.zip**
11. Repeat Step #3 and note the results and lack of Traversal warning
    - Inspect the Query Plan for the new query using the techniques learned in this lab, so see it hitting the new index.

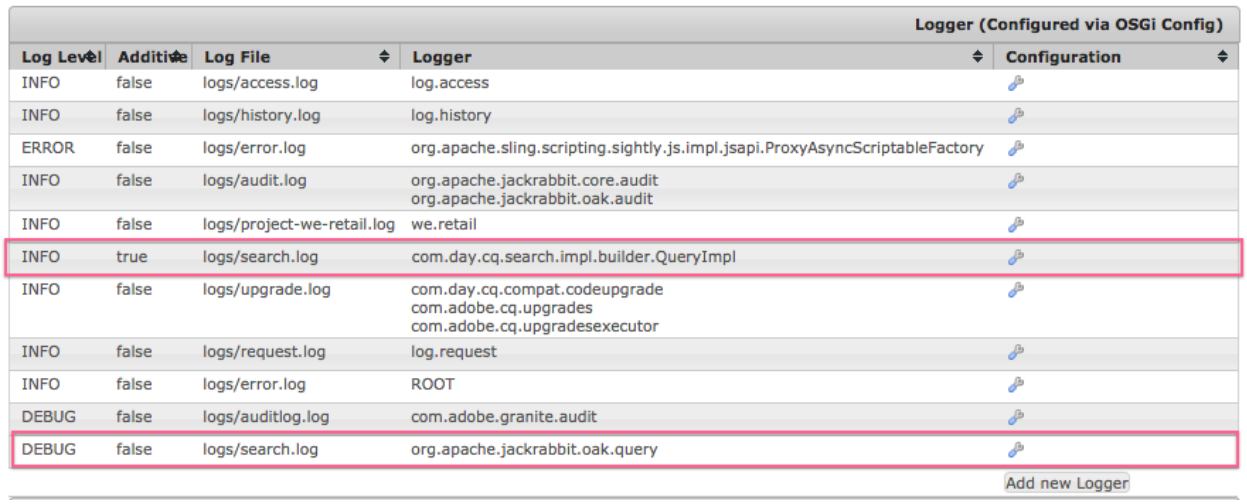
Pro-Tip: On real projects, the XML node definition can be copy/pasted into the AEM Code Project for controlled deployment.

Bonus Exercises

Logging for Search

AEM Chrome Plug-in is an efficient view into AEM search logging, however it is not always available. The same level of information can be obtained via standard AEM logging.

1. As admin, navigate to AEM's OSGi Web console
    a. http://localhost:4502/system/console
2. In the top menu bar, click **Sling** > **Log Support**
    a. http://localhost:4502/system/console/slinglog
3. Click the **Add new logger** button
4. Configure the new logger to expose Oak query execution details
    a. Log level: DEBUG
    b. Additive: false
    c. Log file: logs/search.log
    d. Logger: org.apache.jackrabbit.oak.query
5. Click **Save**
6. Create a second logger by clicking the **Add new logger** button
7. Configure the new logger to expose Query Builder details
    a. Log level: INFO
    b. Additive: true
    c. Log file: logs/search.log
    d. Logger: com.day.cq.search.impl.builder.QueryImpl
8. Click **Save**



Logger (Configured via OSGi Config)

Log Level	Additive	Log File	Logger	Configuration
INFO	false	logs/access.log	log.access	
INFO	false	logs/history.log	log.history	
ERROR	false	logs/error.log	org.apache.sling.scripting.sightly.js.impl.jsapi.ProxyAsyncScriptableFactory	
INFO	false	logs/audit.log	org.apache.jackrabbit.core.audit org.apache.jackrabbit.oak.audit	
INFO	false	logs/project-we-retail.log	we.retail	
INFO	true	logs/search.log	com.day.cq.search.impl.builder.QueryImpl	
INFO	false	logs/upgrade.log	com.day.cq.compat.codeupgrade com.adobe.cq.upgrades com.adobe.cq.upgradesexecutor	
INFO	false	logs/request.log	log.request	
INFO	false	logs/error.log	ROOT	
DEBUG	false	logs/auditlog.log	com.adobe.granite.audit	
DEBUG	false	logs/search.log	org.apache.jackrabbit.oak.query	

Add new Logger

9. Open the new search.log file in Console (or tail –f from the command line) and perform several searches using Chapter 8.

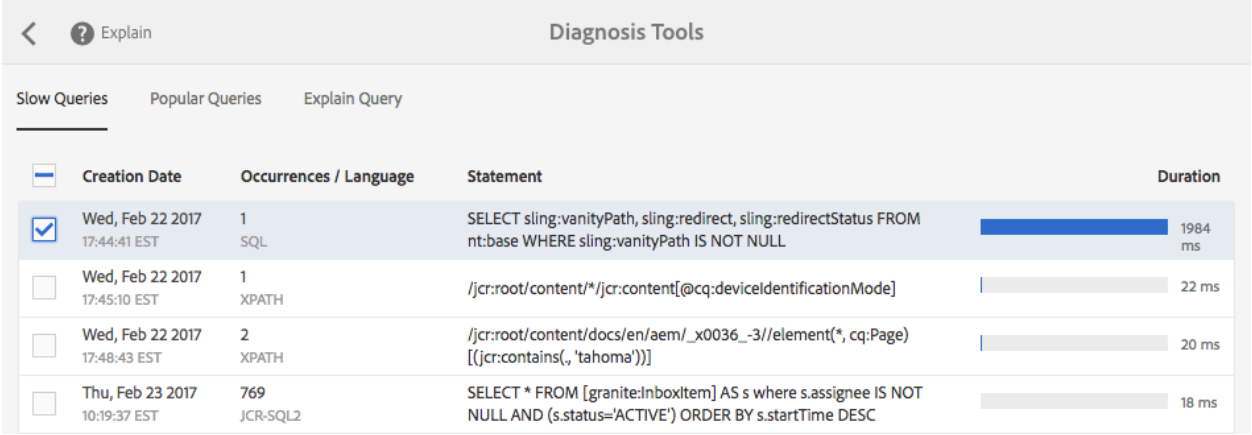
The new logging shows

- QueryBuilder predicate definition
- The query executed by the Oak query engine
    - This query can be used in Explain Query as described in the Explain Query section
- The Oak index cost evaluation
- The query plan the Oak query engine generates based on the provided query

Pro-tip: Traversal queries will show up prominently in the logs. Any traversal queries MUST be fixed prior to production deployment to avoid performance issues.

Slow and Popular Queries

AEM maintains a list of slow and popular queries that have been recently executed. These lists can be helpful in locating and identifying problematic queries or simply queries that should be tuned for maximum efficiency (optimize the common case).



1. As admin, navigate to AEM's Query Performance console
    a. AEM > Tools > Operations > Diagnostics > Query Performance
2. Slow Queries tab
    a. Show the recent slowest queries, their query time and execution count
    b. Note that Traversal queries will likely not show on this list as in AEM 6.3 they auto-terminate
3. Popular Queries tab
    a. Show the recent slowest queries, their execution count and query time
4. Any of the queries can be selected and Explained via the Explain Query button in the top left
    - *Explain Query explained in more detail below*

Slow and Popular Queries JMX MBean

Slow and Popular queries can be cleared via the JMX MBean "QueryStat" available in the AEM OSGi Web console

1. As admin, navigate to AEM's OSGi Web console
    - http://localhost:4502/system/console
2. In the top menu bar, click **Main** > **JMX**
    - http://localhost:4502/system/console/jmx
3. Scroll down and click on the row
    - org.apache.jackrabbit.org QueryStat Oak Query Statistics
4. Slow and Popular queries display at the top of the page
5. Scroll to Operations the bottom of the page and click the appropriate method name to clear the corresponding lists.
    - Clear slow queries lists: clearSlowQueriesQueue()
    - Clear popular queries lists: clearPopularQueriesQueue()

Operations

Return Type	Name
void	clearSlowQueriesQueue() Operation exposed for management
void	clearPopularQueriesQueue() Operation exposed for management

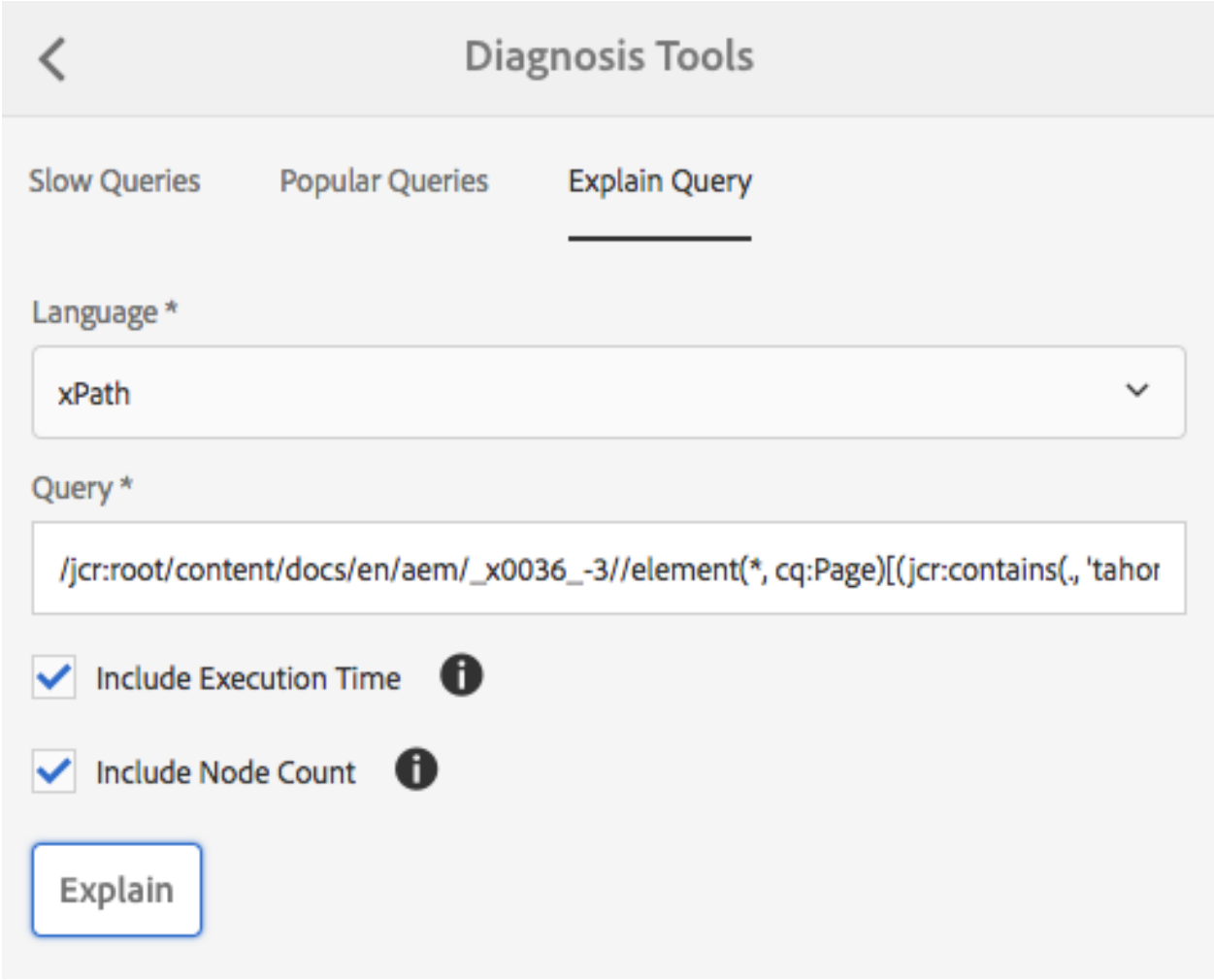
Explain Query

Explain Query is a powerful tool that provides detailed information on how Oak evaluates and executes search queries. Explain Query shows the

- Executed query
- Execution cost per index
- Query plan
    - The Oak indices are used
    - What query constraints are evaluated
- Query time
- Number of results

Explain query accepts XPath, JCR-SQL and JCR-SQL2 queries. QueryBuilder queries must be evaluated to their XPath query statement, and said XPath query statement is to be provided to Explain Query.

1. As admin, navigate to AEM's Query Performance console and click on the Explain Query tab
    - AEM > Tools > Operations > Diagnostics > Query Performance > Explain Query
2. Select the **query language**
    - XPath, JCR-SQL, JCR-SQL2
3. Provide the **query statement**
4. Optionally include **Execution Time** and **Node Count**



Pro-tip: Execution Time and Node Count will execute the provided query which depending on the query could be resource intensive. It is best to only Explain the query first and ensure it is not a Traversal prior to including Execution Time and Node Count.

Upon explaining, Explain Query will provide the query explanation in a modal overlay.