IBM WebSphere Commerce V7 Quarterly Releases – 2Q2015

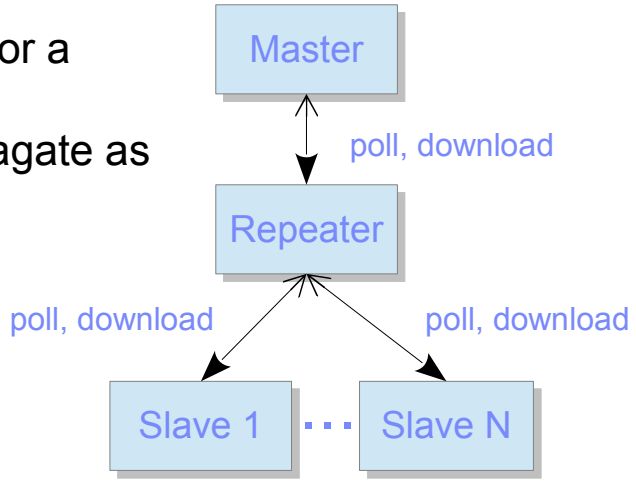
Index Lifecycle Automation

Index Lifecycle Automation

- This release adds a number of WebSphere Commerce Search index lifecycle improvements, including:
  - Index validation:
    - Automated tools to validate replicated index from a master/repeater
    - Support for adding post-validation actions, e.g. backup verified indexes to support roll-back-type functionality in case of corruption
    - Prevent an unverified index to be pushed from master to repeater when using indexProp
  - Change to use table truncation for temporary tables (TI_*), instead of dropping and recreating the tables (subject to DB support)
  - Better handling of configuration file templates between GA and interim fixes, allowing the administrator to choose the most appropriate template when running the setupSearchIndex script

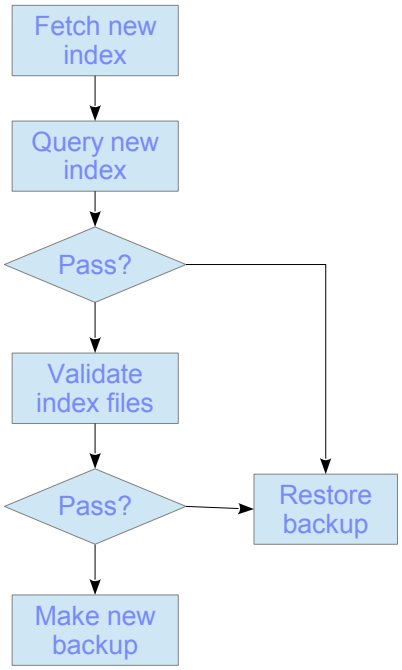
Index lifecycle for a production system

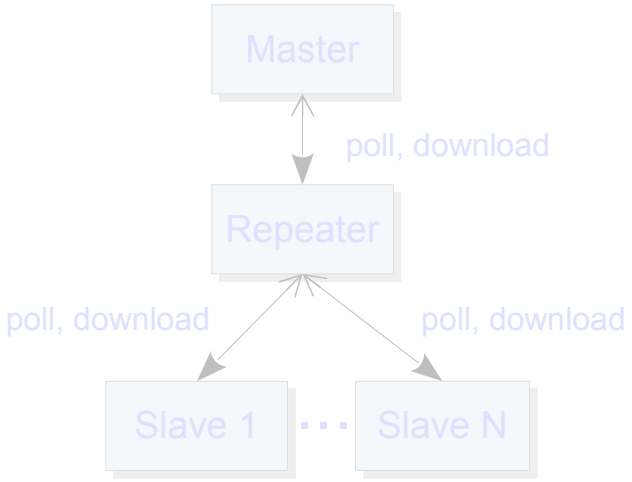
- The recommended production architecture for WebSphere Commerce Search comprises a search master, a repeater, and a number of slaves.
- The master is typically indexing the data from the staging server
- The repeater is responsible for getting the master's search index propagated into production when staging propagation happens
  - This is typically not automatic, but scripted to happen after a successful stage propagation
- The slaves will poll the repeater at regular intervals to check for a new index and download it if one is available
  - This is typically automated to ensure index changes propagate as quickly as possible
- We use Apache Solr's built-in index replication system for propagating indexes



Challenges for index replication and mitigations

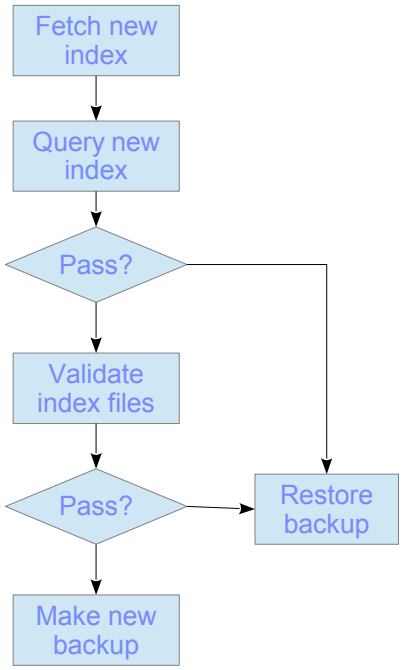
- Solr does implements a checksum algorithm to validate replicated data before switching to use it
- However, there have been incidents of index corruption, especially caused by:
  - Network problems
  - Running out of disk space

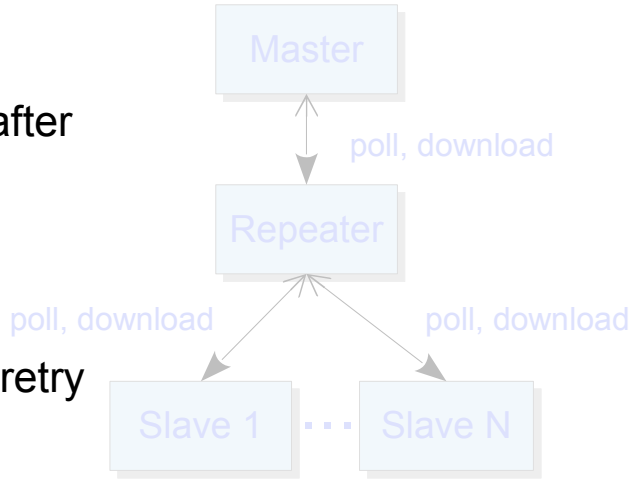




Index validation improvements

- To address the issues mentioned on the previous slide, we introduce support for additional validations before an index is used by a slave or repeater, including:
  - Pre-validation of index before replication from a master to a repeater
    - This is an indexProp option
  - Blocking replication of a repeater's index until replication *and* validation is complete
    - This ensures that a corrupt index cannot be replicated to slaves before validation
    - This is an indexProp option
  - Support for a query-based index validation (*test queries*) after replication
    - This is available for all types of replications
  - Extension point to allow for:
    - Success operations, e.g. backup of validated index
    - Failure operations, e.g. restore to last-known-good or retry
    - Modification of validation logic





Main validation configuration options

- To enable the validation logic, you must have a section like the following in the replication section of solrconfig.xml

```
<lst name="healthCheckOps">
   <str name="enable">${healthCheckOps.enable:false}</str>
   <str name="checkOps">
      ${healthCheckOps.checkOps:com.ibm.commerce.foundation.solr.operation.SolrDoQueryCheckOperation}
   </str>
   <str name="uponSuccessOps">${healthCheckOps.uponSuccessOps:}</str>
   <str name="uponFailureOps">${healthCheckOps.uponFailureOps:}</str>
</lst>
```

- The options refer to properties that can be controlled at core-level in solr.xml, e.g.:

```
<property name="healthCheckOps.enable" value="true"/>
```

- The actual validation and success/failure logic can be modified as well
- The actual checks to perform are configured in a query check properties file as described on the following slide

Query Check Properties File

- The actual test queries to run are specified in a properties file
  - This file can either be defined specific for a core, or common for all cores
- The format is a set of lines of the structure:
  - query.<*id*>.indexType=<*indexName*>
  - query.<*id*>.indexSubType=<*indexSubType*>
  - query.<*id*>.minResultCount=<*minimum expected rows*>
  - query.<*id*>.q=<*the search term ("q" parameter in Solr)*>
    - This is optional. If omitted, the query "*:*" is assumed
  - query.<*id*>.fq=<*filter queries*>
- For example:

```
# Check that we have at least 100 results for "sofa" in MC 10001 and stores 10051 and 10152
query.check1.indexType=CatalogEntry
query.check1.indexSubType=Structured
query.check1.minResultCount=100
query.check1.q=sofa
query.check1.fq=catalog_id:10001 AND storeent_id:(10051 10152)
# At least 1000 results for "dress" in the "name" field in MC 10001, store 10051
# and parent category ID 10006
query.2.indexType=CatalogEntry
query.2.minResultCount=1000
query.2.q=name:dress
query.2.fq=catalog_id:10001 AND parentCatgroup_id_search:(10001_10006) AND storeent_id:10051
```

KC link

http://www.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/refs/rsdsearchquerycheckproperties.htm

Index Propagation Changes

- To support the controlled propagation of indexes, the indexProp utility has been updated
- Two new options have been added:
  - queryCheckPropFile – specifies the location of the query check properties file with the checks to perform before propagating the index
  - disableReplicationOnRepeaterUntilIndexPropSuccessful – ensures that the repeater will not release the updated index to any slaves before index checks have been performed

http://www.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/tasks/tsdsearchindexprop_fep7.htm

Customization Points

- All actions in the index check lifecycle can be customized or extended
- For example, to provide a custom action to perform after a successful validation, create an implementation of com.ibm.commerce.foundation.solr.operation.SolrDoCoreOperation and refer to it in solr.xml:

```
<property name="healthCheckOps.uponSuccessOps" value=" com.mycompany.MySuccessOperation"/>

<property name="healthCheckOps.uponFailureOps" value=" com.mycompany.MyFailureOperation"/>
```

http://www.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/tasks/tsdsearchcoreops.htm

Temporary table truncation

- Reindexing of data for WebSphere Commerce Search is a two-step process:
  - Preprocessing
    - This involves de-normalizing and otherwise preprocessing the WebSphere Commerce data into a set of temporary tables, prefixed by “TI_”
  - Data Import
    - This is the step where data is copied to Apache Solr using the Solr Data Import Handler (DIH)
- The preprocessing logic currently drops all temporary tables, recreates the tables, and then does the preprocessing
- This can be inefficient when the structure of the temporary tables has not changed
- We have added support for using the TRUNCATE TABLE DDL command
  - TRUNCATE TABLE will simply mark the underlying table extents ready for deallocation and is as such very efficient
  - To use this feature, your database must support the TRUNCATE TABLE command:
    - All versions of Oracle supported by WC
    - IBM DB2 9.7 and later

http://www.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/tasks/tsdsearchbuildpre.htm

Using the TRUNCATE option

- A new optional -dropTempTable paramter to the di-preprocess utility
  - If specified as “-dropTempTable false”, TRUNCATE will be used instead
- If using the properties file with options, use the new “DropTempTable” property instead
- Note that if you change the preprocessing configuration to change temporary table structure, or have not run preprocessing before, you must run the utility without this option (or set to “true”)
- The following statement will be used to truncate temporary tables, if enabled:
  - DB2: `TRUNCATE TABLE <tablename> IMMEDIATE`
  - Oracle: `TRUNCATE TABLE <tablename>`
- If required by your DBA, a different statement can be added via configuration options in wc-component.xml or shading input file, if parallel indexing is used

Optional template updates

- Previously, changes to search-related samples and configuration template files were applied directly to the files in the WebSphere Commerce installation directory
- This meant that a customer at a particular version, say V7 FEPx, who then installed an interim fix, say JR12345 were forced to adopt any template and sample changes in JR12345 when setting up a new index
- What we are introducing in this release is a new approach where updates to configuration template files and samples are added to a separate directory
- The setupSearchIndex script has been amended to allow the operator to specify which version of the templates to use when creating the index
  - includeUpdate: specifies to use templates from interim fixes
  - configWcforSolrUpdate: apply search updates from interim fixes to WC
  - configsolrCoresUpdate: apply search updates from interim fixes to Solr

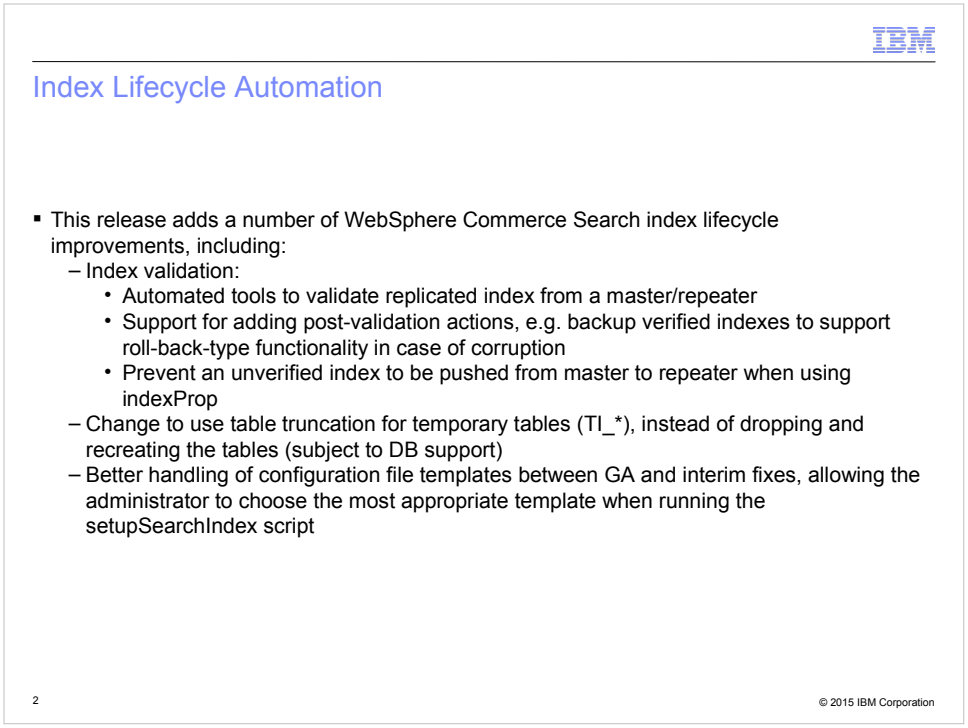
http://www.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.admin.doc/refs/rsdsearchindexsetup.htm

Trademarks, disclaimer, and copyright information

IBM WebSphere Commerce V7 Quarterly Releases – 2Q2015

Index Lifecycle Automation

Welcome to the index lifecycle automation module of the technical enablement for the second quarter release 2015 of IBM WebSphere Commerce version 7.



We are releasing a number of improvements in the area of search index automation for this release of WebSphere Commerce.

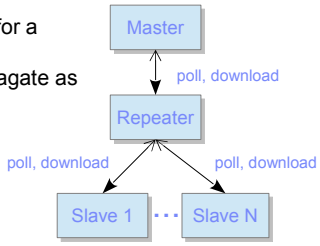
The improvements mainly falls in three areas:

- Validation of indexes before and after replication
- Added support for using table truncation for temporary preprocessing tables; and
- Better handling of search configuration templates between GA versions and fixpack and interim fixes

The index validation improvements comprises a number of individual features, including pre- and post-replication index verification, and support for adding custom actions after a index verification, depending on the outcome of the validation.

Index lifecycle for a production system

- The recommended production architecture for WebSphere Commerce Search comprises a search master, a repeater, and a number of slaves.
- The master is typically indexing the data from the staging server
- The repeater is responsible for getting the master's search index propagated into production when staging propagation happens
  - This is typically not automatic, but scripted to happen after a successful stage propagation
- The slaves will poll the repeater at regular intervals to check for a new index and download it if one is available
  - This is typically automated to ensure index changes propagate as quickly as possible
- We use Apache Solr's built-in index replication system for propagating indexes



Before we cover the index verification improvements, let spend a few minutes discussing the index lifecycle for a typical production system.

The recommended production architecture for WebSphere Commerce Search includes a master search server, a repeater, and a number of slaves, or subordinates, to handle the actual search requests.

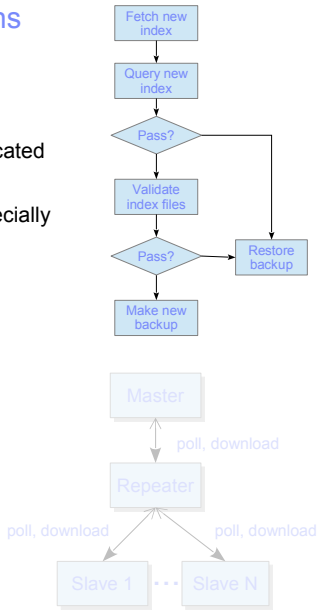
The Master indexes the data on the staging or authoring server. Its index is pushed to the repeater when a stage propagation job is run, via the indexProp utility.

The Repeater is part of the production search cluster and is responsible for getting the index to the slaves in the cluster, who handles that live search traffic.

The replication of the index from the master to the repeater and the repeater to the slaves is done using Solr's built-in index replication mechanisms.

Challenges for index replication and mitigations

- Solr does implements a checksum algorithm to validate replicated data before switching to use it
- However, there have been incidents of index corruption, especially caused by:
  - Network problems
  - Running out of disk space

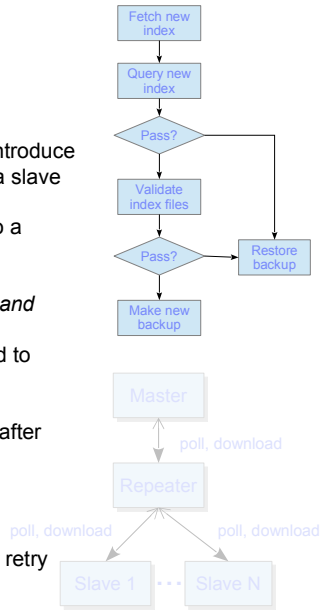


In the field, we have noticed some incidents of corruption of search indexes, typically caused by disk space issues, or network problems. For example, if the master runs out of disk space, or has some other disk I/O issue during the building of the index, the index may become corrupted, and a corrupt index can thus end up being replicated to the live traffic search servers.

Although Solr's index replication feature does use checksums to help guard from these issues, it is not 100% safe, as we have seen instances of corruption in isolated cases.

Index validation improvements

- To address the issues mentioned on the previous slide, we introduce support for additional validations before an index is used by a slave or repeater, including:
  - Pre-validation of index before replication from a master to a repeater
    - This is an indexProp option
  - Blocking replication of a repeater's index until replication *and* validation is complete
    - This ensures that a corrupt index cannot be replicated to slaves before validation
    - This is an indexProp option
  - Support for a query-based index validation (*test queries*) after replication
    - This is available for all types of replications
  - Extension point to allow for:
    - Success operations, e.g. backup of validated index
    - Failure operations, e.g. restore to last-known-good or retry
    - Modification of validation logic

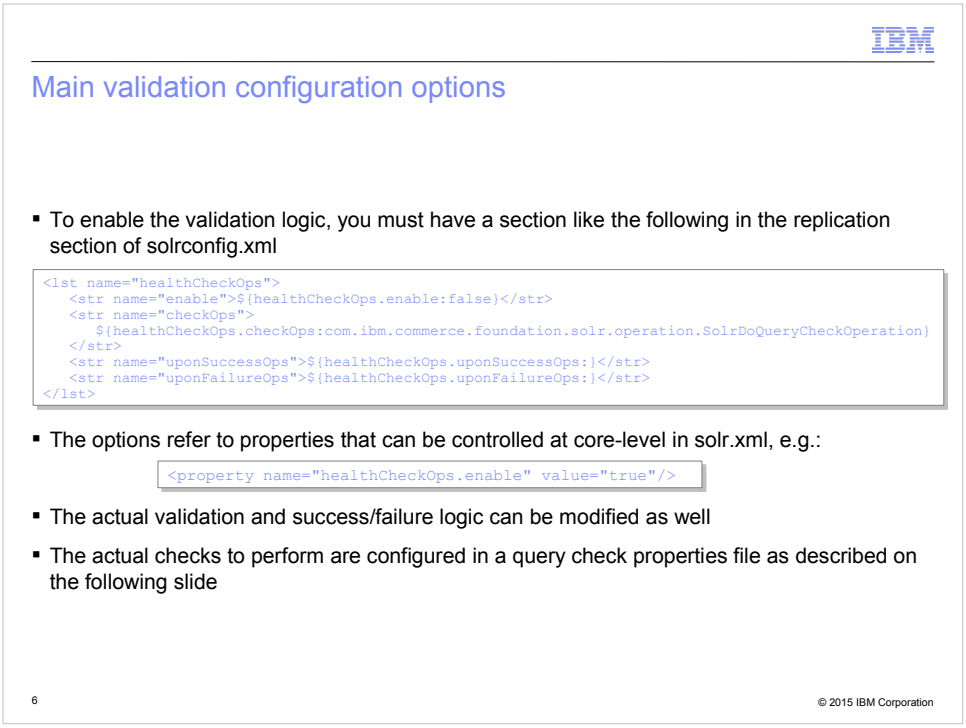


To lower the risk of future incidents, we introduce a new index validation mechanism. This works at several layers:

- First, it allows for the configuration of a number of test queries to be configured. Unless these provide the desired results, which in the default implementation means that the expectede number of results are returned, the index is considered corrupt.

- Secondarily, it allows for the replication from the master to the repeater to be blocked until a successful validation of the index on the master has taken place. This guards against a corrupt index even leaving the master server.

- Finally, there are clear extension points provided for clients to implement custom actions based on the outcome of of a validation. These extensions could, for instance, back up a validated index and restore a previous backup in case of a failed validation, thus providing an effective roll-back mechanism in case of index corruption on any server.



This slide summarizes the configuration options for the validation feature.

First off, to enable validation, you must add the first snippet to the replication section of a node's solrconfig.xml file. This file defines the default settings for the various options:

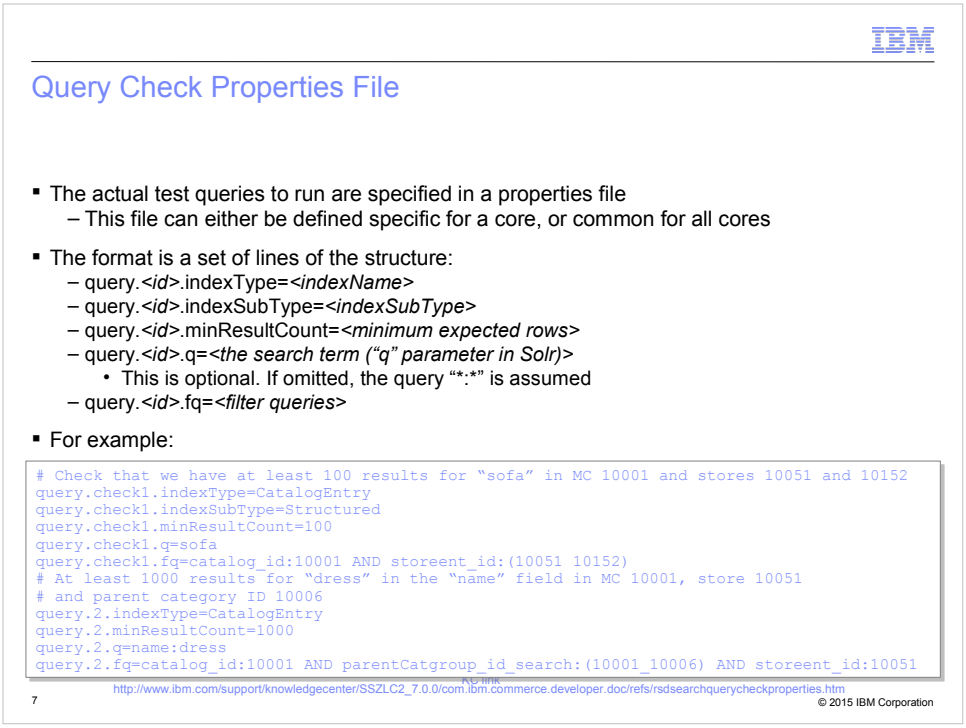
- The healthCheckOps variable turns validation on and off. As you can see here, its default setting is "off"

- The checkOps variable references an implementation class that performs the validation. Its default value, as you can see, points to the built-in SolrDoQueryCheckOperation class. This class is the one that can execute a number of test queries against the index and verify the number of returned documents.

- Finally there are two variables, uponSuccessOps and uponFailureOps, who point to the Java classes to execute upon successful and unsuccessful validation, respectively. As you can see, we do not provide default implementations of these, but the Knowledge Center has information on how you can implement your own custom logic for these events.

The intent is to provide node-wide configuration options in the solrconfig.xml file and then override the settings in the solr.xml file at a core-level. The code to enable the validation is shown on this slide as well.

As you may notice, we have not yet defined exactly which queries to execute. This is specified in a special properties file, which is covered next.



This slide describes the syntax of the properties file that provides the actual test queries to run against a core in order to verify its validity.

The file is a standard Java properties file with a number of properties prefixed by the string "query", followed by a period, followed by an identifier, and then a property setting for that test query.

For each test query, you can specify the index type, e.g. CatalogEntry, the index subtype, for example "Structured" or "Unstructured", the minimum number of expected results, the actual query, i.e. the value of the "q" parameter sent to the DisMax handler, and then an optional filter query expression.

This slide has an example file, where we define two queries:

- One that searches for "sofa" in the default search field and expects at least 100 results for catalog ID 10001 and stores 10051 and 10152.

- The second query searches for "dress" in the "name" field and expects at least 1000 results in the category ID 10006.

Index Propagation Changes

- To support the controlled propagation of indexes, the indexProp utility has been updated
- Two new options have been added:
  - queryCheckPropFile – specifies the location of the query check properties file with the checks to perform before propagating the index
  - disableReplicationOnRepeaterUntilIndexPropSuccessful – ensures that the repeater will not release the updated index to any slaves before index checks have been performed

http://www.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/tasks/tsdsearchindexprop_fep7.htm

In addition to allowing for index verification, we have also amended the indexProp utility to specify the check properties file and to block replication to the repeater until checks have completed.

The two command line options for this are outlined on this slide.

Customization Points

- All actions in the index check lifecycle can be customized or extended
- For example, to provide a custom action to perform after a successful validation, create an implementation of com.ibm.commerce.foundation.solr.operation.SolrDoCoreOperation and refer to it in solr.xml:

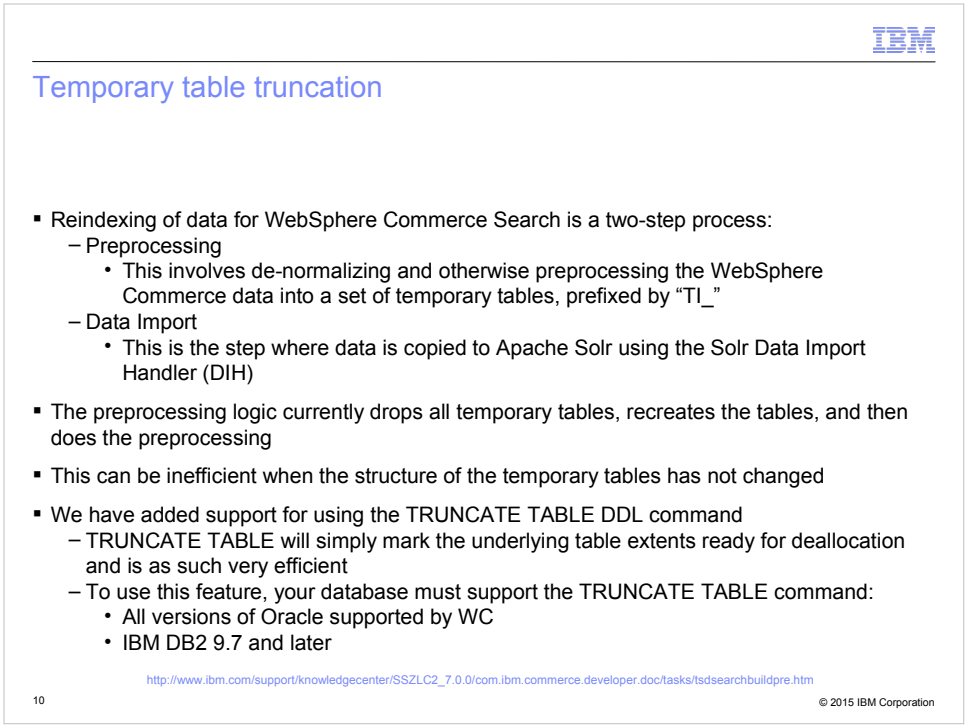
```
<property name="healthCheckOps.uponSuccessOps" value=" com.mycompany.MySuccessOperation"/>
<property name="healthCheckOps.uponFailureOps" value=" com.mycompany.MyFailureOperation"/>
```

http://www.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/tasks/tsdsearchcoreops.htm

As mentioned earlier, we have made all events in the index replication feature customizable.

You can specify custom actions for the index validation checks, as well as specify custom actions to be taken, depending on the outcome of the validation.

This slide shows how to specify properties in solr.xml to point to custom success and failure actions.



Another feature delivered to help index lifecycle management is the support for truncating temporary tables instead of dropping and recreating them.

As you may be aware of, the WebSphere Commerce Search indexing process currently comprise two steps:

- First, the data in the highly normalized database is de-normalized into a set of temporary tables. This is known as preprocessing.

- Then the Solr data import handler is called to load the denormalized data into Solr. This is known as the data import step.

The current implementation of the preprocesser will drop all temporary tables before recreating them. This can have a negative impact on database performance, and as such, we are now adding support for using the TRUNCATE DDL statement instead. This statement is available in all versions of Oracle supported by WebSphere Commerce version 7, as well as all IBM DB2 version 9.7 and up.

Using the TRUNCATE option

- A new optional -dropTempTable paramter to the di-preprocess utility
  - If specified as “-dropTempTable false”, TRUNCATE will be used instead
- If using the properties file with options, use the new “DropTempTable” property instead
- Note that if you change the preprocessing configuration to change temporary table structure, or have not run preprocessing before, you must run the utility without this option (or set to “true”)
- The following statement will be used to truncate temporary tables, if enabled:
  - DB2: `TRUNCATE TABLE <tablename> IMMEDIATE`
  - Oracle: `TRUNCATE TABLE <tablename>`
- If required by your DBA, a different statement can be added via configuration options in wc-component.xml or shading input file, if parallel indexing is used

The feature is enabled by adding the parameter "dropTempTable" to the preprocess script with a value of "false"

If you use the option for the preprocess script to pass in parameters in a properties file, there is a property for this as well.

Note that since the tables will not be recreated, you must omit this parameter, or specify it as "true" if you change the structure of a temporary table in the preprocessing configuration files. Otherwise preprocessing will fail.

Also, you have the option of customizing the actual DDL statement used. The default ones for DB2 and Oracle are shown here.

Optional template updates

- Previously, changes to search-related samples and configuration template files were applied directly to the files in the WebSphere Commerce installation directory
- This meant that a customer at a particular version, say V7 FEPx, who then installed an interim fix, say JR12345 were forced to adopt any template and sample changes in JR12345 when setting up a new index
- What we are introducing in this release is a new approach where updates to configuration template files and samples are added to a separate directory
- The setupSearchIndex script has been amended to allow the operator to specify which version of the templates to use when creating the index
  - includeUpdate: specifies to use templates from interim fixes
  - configWcforSolrUpdate: apply search updates from interim fixes to WC
  - configsolrCoresUpdate: apply search updates from interim fixes to Solr

http://www.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.admin.doc/refs/rsdsearchindexsetup.htm

The final feature to help in the search area is a different handling of search configuration template files.

Previously, interim fixes would update the base templates when installed. This meant that if a customer installed a cumulative fix and then recreated an index, the new index would include all the template changes introduced in the interim fixes from the cumulative fix.

This could cause issues with clients that were not interested in taking advantage of all the new search features in the cumulative fix.

To help address this, we now keep updates to template files in special "update" sub-directories under the template directories. When a client runs the setupSearchIndex script, they can now specify which version of the templates to use. You can also specify which updates to apply to WebSphere Commerce and Solr.

Trademarks, disclaimer, and copyright information