RDB via Table Scanner

The *RDBMS Scanner* component performs full and incremental scans over a database. Currently the RDBMS scanner performs full and incremental scanning using SQL statements to extract the data from one or more *data* tables. The incremental scanning will typically use an *update* table to hold information about which rows in the *data* tables have been updated. In due course, the scanner will be upgraded to use snapshot files and maintain a snapshot of the database and to compare with the current content to establish what content has been updated.

Updated content is submitted to the configured pipeline in AspireObjects attached to Jobs, with every column extracted from the *data* tables added to the AspireObject. Updated content is split into three types - add, update, and delete. Each type of content is published on a different event so that it may be handled by different Aspire pipelines.

The scanner reacts to an incoming job. This job may instruct the scanner to *start*, *stop*, *pause* or *resume*. The *start* job will contain the database connection parameters and SQL for data extraction, although this may be defaulted in the component configuration in the application.xml file. When pausing or stopping, the scanner will wait until all the jobs it published have completed before itself completing.

RDB via Table Scanner	
Factory Name	com.searchtechnologies.aspire:aspire-rdb-connector
subType	default
Inputs	AspireObject from a content source submitter holding all the information required for a crawl
Outputs	Jobs from the crawl

? Unknown Attachment

Configuration

This section lists all configuration parameters available to configure the RDB via Table Scanner component.

General Scanner Component Configuration

Basic Scanner Configuration

Element	Type	Default	Description
snapshotDir	String	snapshots	The directory for snapshot files.
numOfSnapshotBackups	int	2	The number of snapshots to keep after processing.
waitForSubJobsTimeout	long	600000 (=10 mins)	Scanner timeout while waiting for published jobs to complete.
maxOutstandingTimeStatistics	long	1m	The max about of time to wait before updating the statistics file. Whichever happens first between this property and maxOutstandingUpdatesStatistics will trigger an update to the statistics file.
maxOutstandingUpdatesStatistics	long	1000	The max number of files to process before updating the statistics file. Whichever happens first between this property and maxOutstandingTimeStatistics will trigger an update to the statistics file.
usesDomain	boolean	true	Indicates if the group expansion request will use a domain\user format (useful for connectors that does not support domain in the group expander).

Branch Handler Configuration

This component publishes to the *onAdd*, *onDelete* and *onUpdate*, so a branch must be configured for each of these three events.

Element	Type	Description
branches/branch/@event	string	The event to configure - *onAdd*, *onDelete* or *onUpdate*.
branches/branch/@pipelineManager	string	The name of the pipeline manager to publish to. Can be relative.
branches/branch/@pipeline	string	The name of the pipeline to publish to. If missing, publishes to the default pipeline for the pipeline manager.
branches/branch/@allowRemote	boolean	Indicates if this pipeline can be found on remote servers (see Distributed Processing for details).
branches/branch/@batching	boolean	Indicates if the jobs processed by this pipeline should be marked for batch processing (useful for publishers or other components that support batch processing).
branches/branch/@batchSize	int	The max size of the batches that the branch handler will created.
branches/branch/@batchTimeout	long	Time to wait before the batch is closed if the batchSize hasn't been reached.

branches/branch /@simultaneousBatches	int	The max number of simultanous batches that will be handled by the branch handler.

File System Specific Configuration

Element	Type	Default	Description
rdb	string		The Multi RDBMS Connection Pool component to use for RDB connections.

Configuration Example

```
<component name="RDBMSRDB" subType="default" factoryName="aspire-multiple-rdb">
  <debug>true</debug>
  <timeout>30m</timeout>
  <purgePoll>5m</purgePoll>
</component>
```

```
<component name="Scanner" subType="default" factoryName="aspire-rdb-connector">
  <debug>true</debug>
  <rdb>./RDBMSRDB</rdb>
  <branches>
    <branch event="onAdd" pipelineManager="../ProcessPipelineManager"
      pipeline="addUpdatePipeline" allowRemote="true" batching="true"
      batchSize="50" batchTimeout="60000" simultaneousBatches="2" />
    <branch event="onUpdate" pipelineManager="../ProcessPipelineManager"
      pipeline="addUpdatePipeline" allowRemote="true" batching="true"
      batchSize="50" batchTimeout="60000" simultaneousBatches="2" />
    <branch event="onDelete" pipelineManager="../ProcessPipelineManager"
      pipeline="deletePipeline" allowRemote="true" batching="true"
      batchSize="50" batchTimeout="60000" simultaneousBatches="2" />
  </branches>
</component>
```

Source Configuration

Scanner Control Configuration

The following table describes the list of attributes that the AspireObject of the incoming scanner job requires to correctly execute and control the flow of a scan process.

Element	Type	Options	Description
@action	string	start, stop, pause, resume, abort	Control command to tell the scanner which operation to perform. Use **start** option to launch a new crawl.
@actionProperties	string	full, incremental	When a **start** @action is received, it will tell the scanner to either run a **full** or an **incremental** crawl.
@normalizedCSName	string		Unique identifier name for the content source that will be crawled.
displayName	string		Display or friendly name for the content source that will be crawled.

Header Example

```
  <doc action="start" actionProperties="full" actionType="manual" crawlId="0" dbId="0" jobNumber="0"
normalizedCSName="FeedOne_Connector"
   scheduleId="0" scheduler="##AspireSystemScheduler##" sourceName="ContentSourceName">
    ...
    <displayName>testSource</displayName>
    ...
  </doc>
```

All configuration properties described in this section are relative to /doc/connectorSource of the AspireObject of the incoming Job.

Property	Type	Default	Description
url	string		The JDBC URL for your RDBMS server and database. For example, "jdbc:mysql://192.168.40.27/mydb" (MySQL). This will vary depending on the type of RDBMS.
username	string		The name of a database user with read-only access to all of the tables which need to be indexed, and write access to the necessary update tables (if update management is handled through the RDB).
password	string		The database password
dbDriverJar	string		Path to the JDBC driver JAR file for your RDBMS. Typically this is placed in the "lib" directory inside your Aspire Home, for example "lib/myjdbcdriver.jar".
dbDriverClass	string		(Optional)The name of the JDBC driver class if the class name from the META-INF/services/java.sql.Driver file in the driver JAR file should not be used, or that file does not exist in the driver JAR file.
fullSelectSQL	string		This SQL will be executed when the user clicks on "full crawl". Each record produced by this statement will be indexed as a separate document. Some field names have special meaning (such as 'title', 'content', 'url', 'aspire_id', etc.)
idColumn	string	id	The name of the column which contains unique identifiers for each row. This is used for both full and incremental crawls and must match the name returned by the SQL. If the column is aliased using the SQL "AS" construct, you should provide the alias name here.
useSlices	boolean	false	True if you want to divide the Full crawl SQL into multiple slices. Only works when the id column is an integer.
slicesNum	integer	10	The number of SQL slices to split Full crawl SQL. Only works when **idColumn** is an integer.
preUpdateSQL	string		SQL to run before an incremental crawl. This SQL can be used to mark documents for update, save timestamps, clear update tables, etc. as needed to prepare for an incremental crawl. Can be left blank if you never do an incremental crawl.
updateSQL	string		SQL to run for an incremental crawl. This SQL should provide a list of all adds and deletes to the documents in the index. Some field names have special meaning (such as 'title', 'content', 'url', 'aspire_id', etc.) see the wiki for more information. Note the special column, 'aspire_action' should report 'I' (for inserts), 'U' (for updates, typically the same as updates for most search engines), and 'D' (for deletes).
postUpdateSQL	string		SQL to run after each record processed. This SQL can be used un-mark / delete each document from the tables after it is complete.
postUpdateFailedSQL	string		SQL to run after each record if processing fails. If this SQL is left blank, the 'Post incremental crawl SQL' will be run instead
seqColumn	string	seq	The name of the column in the returned data which holds the sequence number of the update.
actionColumn	string	action	The name of the column in the returned data which holds action of the update (ie Insert, Update or Delete).

Scanner Configuration Example

```
<doc action="start" actionProperties="full" normalizedCSName="My_RDB_Source">
  <connectorSource>
    <url>jdbc:postgresql://localhost:5433/RDBMS_Test_DB</url>
    <username>aspire_user</username>
    <password>encrypted:062062C25293A7A3BA08D29385F6C05A</password>
    <dbDriverJar>lib\postgresql-9.3-1100.jdbc41.jar</dbDriverJar>
    <fullSelectSQL>SELECT "id", "name", "company", "email",  "birthday"
            FROM "Test_table" WHERE "id" <= 1000 AND {SLICES}</fullSelectSQL>
    <idColumn>id</idColumn>
    <useSlices>true</useSlices>
    <slicesNum>10</slicesNum>
    <incrementalIndex>true</incrementalIndex>
    <preUpdateSQL>UPDATE update_table SET status='I' WHERE action='I'</preUpdateSQL>
    <updateSQL>SELECT updates_table.seq_id AS seq_id, updates_table.id AS id,
               updates_table.action AS update_action, main_data.col1 AS cola,
               main_data.col2 AS colb, main_data.col3 AS coly,
               main_data.col4 AS colz
          FROM main_data RIGHT OUTER JOIN updates_table ON main_data.id = updates_table.id
          WHERE updates_table.ACTION = 'I' ORDER BY updates_table.seq_id ASC</updateSQL>
    <postUpdateSQL>UPDATE update_table SET status='A' WHERE status='I'</postUpdateSQL>
    <postUpdateFailedSQL/>
    <seqColumn>seq_id</seqColumn>
    <actionColumn>action</actionColumn>
  </connectorSource>
  <displayName>My RDB Source</displayName>
</doc>
```

Output

```
<doc>
  <docType>item</docType>
  <url>10</url>
  <id>10</id>
  <fetchUrl>10</fetchUrl>
  <displayUrl>10</displayUrl>
  <snapshotUrl>001 10</snapshotUrl>
  <sourceType>rdb</sourceType>
  <sourceName>My_RDB_Source</sourceName>
  <id>10</id>
  <connectorSpecific type="rdb">
    <field name="name">Meghan1</field>
    <field name="company">Interdum Enim Non LLC</field>
    <field name="email">enim.gravida@diam.net</field>
    <field name="birthday">10/21/13</field>
  </connectorSpecific>
  <connectorSource>
    <url>jdbc:postgresql://localhost:5433/RDBMS_Test_DB</url>
    <username>aspire_user</username>
    <password>encrypted:062062C25293A7A3BA08D29385F6C05A</password>
    <dbDriverJar>lib\postgresql-9.3-1100.jdbc41.jar</dbDriverJar>
    <fullSelectSQL>SELECT "id", "name", "company", "email",  "birthday"
                    FROM "Test_table" WHERE "id" <= 1000 AND {SLICES}</fullSelectSQL>
    <idColumn>id</idColumn>
    <useSlices>true</useSlices>
    <slicesNum>10</slicesNum>
    <incrementalIndex>true</incrementalIndex>
    <preUpdateSQL>UPDATE update_table SET status='I' WHERE action='I'</preUpdateSQL>
    <updateSQL>SELECT updates_table.seq_id AS seq_id, updates_table.id AS id,
               updates_table.action AS update_action, main_data.col1 AS cola,
               main_data.col2 AS colb, main_data.col3 AS coly,
               main_data.col4 AS colz
            FROM main_data RIGHT OUTER JOIN updates_table ON main_data.id = updates_table.id
            WHERE updates_table.ACTION = 'I' ORDER BY updates_table.seq_id ASC</updateSQL>
    <postUpdateSQL>post incremental</postUpdateSQL>
    <postUpdateFailedSQL>UPDATE update_table SET status='A' WHERE status='I'</postUpdateFailedSQL>
    <seqColumn>seq_id</seqColumn>
    <actionColumn>action</actionColumn>
    <displayName>My RDB Source</displayName>
  </connectorSource>
  <action>add</action>
  <hierarchy>
    <item id="D3D9446802A44259755D38E6D163E820" level="1" name="My_RDB_Source" url="10"/>
  </hierarchy>
</doc>
```


Scanner Operation

Table Driven Operation

Full feed

Full mode uses SQL taken from the job (*<connectorSource/fullSelectSQL>*) or configuration)(*<sql/fullSelect>*) and executes this against the database configured via a connection pool stage.

- Each resulting row is formed into an AspireObject using the column names as document elements, and this document is submitted to a pipeline manager using the event configured for inserts.
- As the document is created, the value of the column identified in the job (*<connectorSource/idColumn>*) is noted as the primary key of the document.
- The value *insert* will be placed in the *action* attribute of the document.

Column names from SQL queries are added to the AspireObject inside the "connectorSpecific" field. If the column names are standard AspireObject fields, they are added to the root level.

See Connector AspireObject Metadata for further details on which are standard fields.

Example AspireObject from a full feed

```
<doc>
  <docType>item</docType>
  <url>436</url>
  <id>436</id>
  <fetchUrl>436</fetchUrl>
  <snapshotUrl>001 436</snapshotUrl>
  <sourceType>rdb</sourceType>
  <sourceName>TestRDB</sourceName>
  <connectorSpecific type="rdb">
    <field name="id">436</field>
    <field name="title">Mmounting S3 on EC2</field>
    <field name="url">https://wiki.searchtechnologies.com/mediawiki/index.php/Mmounting_S3_on_EC2</field>
    <field name="page_namespace">0</field>
    <field name="page_is_redirect">1</field>
    <field name="old_id">3818</field>
    <field name="content">#REDIRECT [[Mounting S3 on EC2]]</field>
  </connectorSpecific>
  <connectorSource>
    <url>jdbc:mysql://192.168.40.27/wikidb</url>
    <username>aspire_crawl</username>
    <password>encrypted:C0FA502DA95D855623E02465F95F94D8</password>
    <dbDriverJar>lib/mysql-connector-java-5.1.18-bin.jar</dbDriverJar>
    <fullSelectSQL>SELECT P.page_id as id,
                    replace(P.page_title,"_"," ") as title,
                    concat('https://wiki.searchtechnologies.com/mediawiki/index.php/',P.page_title) as url,
                    P.page_namespace,
                    P.page_is_redirect,
                    T.old_id,
                    T.old_text as content
                FROM mw_page P, mw_revision R, mw_text T
                WHERE R.rev_id = page_latest AND T.old_id = R.rev_text_id</fullSelectSQL>
    <idColumn>id</idColumn>
    <incrementalIndex>false</incrementalIndex>
    <useSlices>false</useSlices>
    <displayName>TestRDB</displayName>
  </connectorSource>
  <action>add</action>
  <hierarchy>
    <item id="D3D9446802A44259755D38E6D163E820" level="1" name="TestRDB" url="436"/>
  </hierarchy>
</doc>
```


Incremental feed

Incremental mode also uses SQL from the job (*<connectorSource/updateSQL>*) or configuration file (*<sql/update>*). However, this time a "pre-update" command is run (optional - *<connectorSource/preUpdateSQL>* from the job or *<sql/preUpdate>* from the configuration file), followed by the SQL statement to extract the data. Once jobs return from the pipeline, a "post-update" SQL statement is run for the individual job. This SQL can be different for successful and failed jobs (*<connectorSource/postUpdateSQL>* or *<connectorSource/postUpdateFailedSQL>* from the job or *<sql/postUpdateSQL>* or *<sql/postUpdateFailed>* from the configuration file).

In order to handle updates:

1. Configure a *queue* table in your database.
    a. The actual name of the table can be anything, as long as it is referenced consistently in the SQL statements used in the job or configuration file.
    b. When a row changes in the table, you should use a configured database trigger to fire and insert a row into this *queue* table.
    c. The *queue* table must have (at least) four columns which contain values for *sequence*, *id*, *action* and *status*.
    d. Again, these columns can be named anything you want; the first three are set using job or configuration parameters, and the **status** is used only in the SQL statements below.
        i. *sequence* is a unique incrementing ID for this change.
        ii. *id* indicates the ID (primary key) of the row in the main data table that changed.
        iii. *action* is a single character indicating the type of change (**I** for *insert*, **U** for *update*, and **D** for *delete*, or **N** for *no change*)
        iv. *status* is a flag indicating the current status of the update (suggested values are **W** for *waiting to be processed*, **I** for *in progress* and **C** for *completed*.
2. Your *pre-update* SQL can then update the status (from **W** to **I**) for those updates you wish to process on this pass.
    a. This could be all the updates that are not complete, or a subset (the first *n*).
    b. Typically you should process the updates in the order they are received - i.e. increasing *sequence*.
3. Your data extraction SQL should then join this update table with the main data table,
    a. selecting the columns you wish to index (those with a status of **I**) and the **id**, **sequence** and **action** at a minimum,
    b. ordering the rows by increasing **sequence** number.
    The column names returned for **id**, **sequence** and **action** must match the column names specified in the job parameters, configuration parameters or defaults - you may need to use the SQL **AS** construct.
4. AspireObjects will then be formed for each row returned and submitted to the pipeline manager using the event defined for the appropriate *action*.
    a. You can configure events for *insert*, *update* and *delete*, which means that these actions can use differing pipelines.
    b. As the document is created, the value of the column identified in the job (*<connectorSource/idColumn>*) or configuration (*<columns/@id>*) is noted as the primary key of the document.
    c. The value *insert* will be placed in the *action* attribute of the document.
5. The feeder will ensure that jobs created for a particular document *id* are finished before another for the same *id* is started, but jobs for differing *id*s could be submitted out of *sequence* order.

Post Update SQL

When a job is completed, the "post-update" SQL (*<connectorSource/postUpdateSQL>* or *<connectorSource/postUpdateFailedSQL>* from the job or *<sql/postUpdateSQL>* or *<sql/postUpdateFailed>* from the configuration file) will be executed. This SQL statement will undergo substitution using Simple Templates. The possible substitutions are shown below:

Placeholder	Description
{documentId}	The *document id* from the column identified by *<connectorSource/idColumn>* from the job or *<columns/@id>* of the configuration.
{action}	The *action* from the column identified by *<connectorSource/actionColumn>* from the job or *<columns/@action>* of the configuration.
{sequenceId}	The *sequence* from the column identified by *<connectorSource/seqColumn>* from the job or *<columns/@sequence>* of the configuration.
{failed}	*true* if the submitted job failed, *false* otherwise.

The "post-update" SQL should update the **status** in the *queue* table (from **I** to **C**) to indicate the update has been processed.

Alternative Process

If you wish to keep the number of rows in the *queue* table to a minimum, you may use the "post-update" SQL to delete the row. You may then not need to use the "pre-update" SQL. In this case, this could be omitted from the configuration.

Example AspireObject for an update

```
<doc>
  <docType>item</docType>
  <url>436</url>
  <id>436</id>
  <fetchUrl>436</fetchUrl>
  <snapshotUrl>001 436</snapshotUrl>
  <sourceType>rdb</sourceType>
  <sourceName>TestRDB</sourceName>
  <connectorSpecific type="rdb">
    <field name="id">436</field>
    <field name="title">Mmounting S3 on EC2</field>
    <field name="url">https://wiki.searchtechnologies.com/mediawiki/index.php/Mmounting_S3_on_EC2</field>
    <field name="page_namespace">0</field>
    <field name="page_is_redirect">1</field>
    <field name="old_id">3818</field>
    <field name="content">#REDIRECT [[Mounting S3 on EC2]]</field>
  </connectorSpecific>
  <connectorSource>
    <dbDriverJar>lib/mysql-connector-java-5.1.18-bin.jar</dbDriverJar>
    <dbPassword>encrypted:C0FA502DA95D855623E02465F95F94D8</dbPassword>
    <dbUrl>jdbc:mysql://192.168.40.27/wikidb</dbUrl>
    <dbUser>aspire_crawl</dbUser>
    <fullSelectSQL>SELECT P.page_id as id,
                    replace(P.page_title,"_"," ") as title,
                    concat('https://wiki.searchtechnologies.com/mediawiki/index.php/',P.page_title) as url,
                    P.page_namespace,
                    P.page_is_redirect,
                    T.old_id,
                    T.old_text as content
                FROM mw_page P, mw_revision R, mw_text T
                WHERE R.rev_id = page_latest AND T.old_id = R.rev_text_id</fullSelectSQL>
    <displayName>TestRDB</displayName>
  </connectorSource>
  <action>update</action>
</doc>
```

Example AspireObject for a delete

```
<doc>
  <docType>item</docType>
  <url>436</url>
  <id>436</id>
  <fetchUrl>436</fetchUrl>
  <snapshotUrl>001 436</snapshotUrl>
  <sourceType>rdb</sourceType>
  <sourceName>TestRDB</sourceName>
  <connectorSpecific type="rdb">
    <field name="id">436</field>
    <field name="title">Mmounting S3 on EC2</field>
    <field name="url">https://wiki.searchtechnologies.com/mediawiki/index.php/Mmounting_S3_on_EC2</field>
    <field name="page_namespace">0</field>
    <field name="page_is_redirect">1</field>
    <field name="old_id">3818</field>
    <field name="content">#REDIRECT [[Mounting S3 on EC2]]</field>
  </connectorSpecific>
  <connectorSource>
    <dbDriverJar>lib/mysql-connector-java-5.1.18-bin.jar</dbDriverJar>
    <dbPassword>encrypted:C0FA502DA95D855623E02465F95F94D8</dbPassword>
    <dbUrl>jdbc:mysql://192.168.40.27/wikidb</dbUrl>
    <dbUser>aspire_crawl</dbUser>
    <fullSelectSQL>SELECT P.page_id as id,
                    replace(P.page_title,"_"," ") as title,
                    concat('https://wiki.searchtechnologies.com/mediawiki/index.php/',P.page_title) as url,
                    P.page_namespace,
                    P.page_is_redirect,
                    T.old_id,
                    T.old_text as content
                FROM mw_page P, mw_revision R, mw_text T
                WHERE R.rev_id = page_latest AND T.old_id = R.rev_text_id</fullSelectSQL>
    <displayName>TestRDB</displayName>
  </connectorSource>
  <action>delete</action>
</doc>
```


Slicing Full Crawl SQL

This feature consists in splitting the Full crawl SQL into multiple queries, for the scanning process to be faster. Slicing the *fullSelectSQL* should significantly improve the performance when scanning large databases. Only works when **idColumn is an integer**.

Full crawl SQL Example (with slicing)

Having the following fullSelectSQL:

```
SELECT C.page_id, C.modified_date,
  FROM mw_content C, mw_published P, mw_old T
  WHERE P.rev_id = C.page_latest AND T.old_id = P.rev_text_id;
```

For the connector to support slices you should add the **{SLICES}** tag into your *WHERE* clause in your *fullSelectSQL* as shown in the next example:

```
SELECT C.page_id, C.modified_date,
  FROM mw_content C, mw_published P, mw_old T
  WHERE P.rev_id = C.page_latest AND T.old_id = P.rev_text_id AND {SLICES};
```

In this case, there already was a *WHERE* clause, so we added "*AND {SLICES}*". In the case you don't need any condition, your query must contain *W HERE {SLICES}* for slices to work.