

MetaMatrix Solr Connector

Design Document

Bank of America

Prepared by: Michael Walker

February 22, 2010

Contents

[1. Confidentiality Clause 5](#_Toc254690984)

[2. Copyright Protection 5](#_Toc254690985)

[3. Document Purpose 6](#_Toc254690986)

[4. Solution Overview 6](#_Toc254690987)

[4.1 Phase I 7](#_Toc254690988)

[4.1.1 Full Index Scenario 7](#_Toc254690989)

[4.1.2 Delta Index Scenario 8](#_Toc254690990)

[4.2 Solr Connector 9](#_Toc254690991)

[4.3 Phase II 10](#_Toc254690992)

[5. Solr Connector Requirements 10](#_Toc254690993)

[5.1 Common Use Case Behavior 10](#_Toc254690994)

[6. Software Components 11](#_Toc254690995)

[7. Solr Connector Design 11](#_Toc254690996)

[7.1 Connector Design Details 11](#_Toc254690997)

[7.1.1 SolrConnector 12](#_Toc254690998)

[7.1.2 SolrConnection 12](#_Toc254690999)

[7.1.3 SolrExecution 12](#_Toc254691000)

[7.1.4 SolrVisitor 13](#_Toc254691001)

[7.2 Connector Pooling Support 13](#_Toc254691002)

[7.3 Result Set Caching Support 13](#_Toc254691003)

[7.4 Monitoring Support 13](#_Toc254691004)

[8. Connector Query Capabilities 13](#_Toc254691005)

[8.1 Limit 14](#_Toc254691006)

[8.2 Offset 14](#_Toc254691007)

[8.3 Criteria 14](#_Toc254691008)

[8.4 Processing Literals 14](#_Toc254691009)

[8.4.1 Wildcards 14](#_Toc254691010)

[8.4.2 Lower Case Search 15](#_Toc254691011)

[8.4.3 Equals Criteria 15](#_Toc254691012)

[8.5 Compound Criteria 15](#_Toc254691013)

[8.5.1 AND 15](#_Toc254691014)

[8.5.2 OR 15](#_Toc254691015)

[8.6 NOT 15](#_Toc254691016)

[8.7 LIKE 15](#_Toc254691017)

[8.8 Non-supported capabilities 16](#_Toc254691018)

[9. Connector Properties 16](#_Toc254691019)

[9.1 SolrServerURL 16](#_Toc254691020)

[9.2 SoTimeout 16](#_Toc254691021)

[9.3 ConnectionTimeout 16](#_Toc254691022)

[9.4 DefaultMaxConnectionsPerHost 17](#_Toc254691023)

[9.5 AllowCompression 17](#_Toc254691024)

[9.6 MaxRetries 17](#_Toc254691025)

[9.7 LowerCaseSearch 17](#_Toc254691026)

[9.8 Max In Criteria Size 17](#_Toc254691027)

[9.9 Connector Pool Cleaning Interval 17](#_Toc254691028)

[9.10 Connector Pool Live and Unused Time 17](#_Toc254691029)

[9.11 Connector Pool Max Connections 17](#_Toc254691030)

[9.12 Connector Pool Wait For Source Time 17](#_Toc254691031)

[9.13 Result Set Cache Enabled 17](#_Toc254691032)

[9.14 Result Set Cache Maximum Age 17](#_Toc254691033)

[9.15 Result Set Cache Maximum Size 17](#_Toc254691034)

[9.16 Data Source Monitoring Enabled 18](#_Toc254691035)

[10. Unit Test Examples 18](#_Toc254691036)

[10.1 Test Environment 18](#_Toc254691037)

[10.1.1 MMX 18](#_Toc254691038)

[10.1.2 MC-Solr 18](#_Toc254691039)

[10.1.3 MDEX-Solr 19](#_Toc254691040)

[10.2 MMX > MC-Solr Tests 19](#_Toc254691041)

[10.3 MDEX-Solr > MMX > MC-Solr Tests 20](#_Toc254691042)

[11. Appendix: Future Considerations and Known Issues 20](#_Toc254691043)

[11.1 Authentication 20](#_Toc254691044)

[11.2 Greater Than/Less Than 21](#_Toc254691045)

[11.3 Reserved Characters 21](#_Toc254691046)

[11.4 Offset and ORDER BY 21](#_Toc254691047)

[11.5 Scoring and ORDER BY 21](#_Toc254691048)

[11.6 Faceted search 21](#_Toc254691049)

[11.7 Dynamic Fields and All-field search 21](#_Toc254691050)

[11.7.1 Name/Value Models 22](#_Toc254691051)

[11.7.2 Dynamic Field Column 22](#_Toc254691052)

[11.8 Multi-valued Fields 23](#_Toc254691053)

[11.8.1 Multivalue-concat attribute 23](#_Toc254691054)

[11.8.2 Multivalue-expand attribute 23](#_Toc254691055)

[11.8.3 Name/Value models 23](#_Toc254691056)

# Confidentiality Clause

The information presented in this proposal is exclusively confidential to Amentra, Inc. In no event shall all or any part of this document be disclosed or disseminated without the expressed written permission of Amentra, Inc.

# Copyright Protection

Copyright © 2010 by Amentra, Inc. of Richmond, Virginia. All rights reserved. No part of the work covered by the copyright herein may be reproduced or used in any form or by any means- graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems- without permission in writing from Amentra.

# Document Purpose

This document articulates the requirements and general behavior of a custom MetaMatrix connector, written to provide connectivity to Solr as a data source.

# Solution Overview

Bank of America plans to build a Metadata Exchange (MDEX) platform to provide unified access to metadata found in multiple metadata repositories across the enterprise. MDEX will leverage MetaMatrix as the federation platform to provide access to each repository’s contents. This set of repositories includes multiple instances of DAG MetaCenter, which uses Solr internally to catalog metadata information from various systems.



Figure Solution Architecture Logical Diagram

In the first phase of implementation (represented in blue above), MetaMatrix will provide access to each MetaCenter Solr Index (“MC-Solr”), and map the index contents to a common data model that represents the contents of the Index. The Metadata Exchange Solr Server instance (“MDEX-Solr”) will connect to MetaMatrix as a client, using the SolrJ DataSourceImportHandler interface over JDBC.

The common data model used by the VDB and MDEX is detailed below:



Figure Common Data model

The Object entity represents the data coming from Solr Indexes. The remaining entities will come from Phase 2 implementation, and a separate Object API from MetaCenter.

## Phase I

There are two use case scenarios where MDEX will pull data from the MetaCenter Solr indexes:

1. Full Indexing
2. Delta Indexing

### Full Index Scenario

In the Full Index scenario, MDEX will query the common data model to retrieve the entire contents of all indexes, across all instances of MetaCenter. MDEX will re-index the contents, creating a master index with content that spans multiple repositories. This will be used for initial indexing.

The Solution Sequence Diagram, steps 1-4 below, illustrates this first phase:

Figure Full Indexing Solution Sequence Diagram

1. MDEX-Solr queries MetaMatrix by issuing a select all statement:

SELECT id, name, type, path, lastupdatedtime FROM object

1. SolrConnector will issue a query to MC-Solr:

Q=\*:\*

Field=Id, Name, Type, Path, LastUpdatedTime

1. MC-Solr will return all the documents that have been indexed
2. SolrConnector will extract the id, name, type, path, lastupdatedtime from the search result and return the result back to MDEX-SOLR

Steps 5-11 will be covered in Phase II

### Delta Index Scenario

In the Delta Indexing use case scenario, MDEX-Solr will query MetaMatrix to determine which objects have been updated since the last indexing was run. It will then issue a query to pull the new information for each updated item, and re-index them.

Because MetaCenter updates all objects on a per-data source basis, MDEX can simply query MetaMatrix for the set of data source IDs that have been updated. These data sources are known internally in MetaCenter as Repositories. MDEX can then fetch all the Objects that match the repository IDs in order to re-index them. Since the number of repositories (data sources) are fairly small, this will reduce the number of queries sent to MMX and to MC-Solr, when compared to individually querying for each object.

The sequence diagram below illustrates this scenario.

Figure Delta Indexing Solution Sequence Diagram

1. MDEX-Solr will query MMX for MC repositories (i.e. data sources) that have been updated since the last indexing:

SELECT repositoryid from commonobjectmodel.objects where lastupdateddate > '${dataimporter.last\_index\_time}'

1. SolrConnector will issue a query to MC-Solr to determine what objects have been updated:

Field=repository\_id, lastupdateddate

1. MC-Solr will return the repository\_ids of each updated repository, along with the update time.
2. SolrConnector will return the updated repository information to MDEX-Solr
3. MDEX-Solr will query MMX for all objectswith repository ids that have been updated:

SELECT id, name, type, path, lastupdatedtime FROM object WHERE repository\_id=<repository ID>

1. SolrConnector will issue a query to MC-Solr to fetch the object info
2. MC-Solr will retrieve the object data
3. SolrConnector will return the object data as a result set

## Solr Connector

In order to achieve both use cases shown in phase 1, a Solr Connector will be created, to provide the ability to extract the contents of multiple Solr indexes using standard SQL statements.



Figure Solr Connector Architecture Diagram

This document contains the details of the design of this Solr Connector.

## Phase II

In a second phase of implementation, MetaMatrix will use the MetaCenter Object API to provide access to repository-specific content about the objects, such as object properties, and object relationships. The details of this work are outside the scope of this document.

# Solr Connector Requirements

The following requirements are necessary to achieve the goals put forth in the solution overview:

1. Represent fields in a Solr Index as if it were columns in a relational table
2. Provide read access to Solr Index
3. Provide ability to translate certain ANSI SQL queries into SolrQuery, including:

SELECT id, name, type, path, lastupdatedtime FROM Index

1. Provide ability to batch results, retrieving results from Solr at the time each batch is requested

## Common Use Case Behavior

A common query from the Solr client in the Bank of America use case may be:

SELECT id, name, type FROM Index;

This query should select 5 fields from all documents in the index, with no search criteria specified.

The equivalent search in Solr should be:

SolrQuery solrQuery = **new** SolrQuery();

solrQuery.addField(“id”);

solrQuery.addField(“name”);

solrQuery.addField(“type”);

solrQuery.setQuery(“\*:\*”);

solrQuery.setRows(batchSize);

QueryResponse rsp = server.query(solrQuery);

Fields will be individually added to the query, to ensure that unused fields are not returned, thereby increasing performance.

The query supplied to Solr will be “\*:\*”, which instructs Solr to return all documents.

# Software Components

MetaMatrix Connector SDK will be used to develop the custom connector. The MetaMatrix Connector SDK leverages a framework for quickly developing connections to non-standard sources, and provides facilities for connection pooling, monitoring, and result set caching.

SolrJ will be used as the API to interface with Solr indexes. Specifically, the CommonsHttpSolrServer class will be used to represent a connection to a Solr server over HTTP. The DAG team has advised using this approach for efficiency and ease-of-use.

# Solr Connector Design

The following classes will be implemented using the MetaMatrix Connector SDK:

* SolrConnector
  + Initializes the environment and creates a Connection Pool of SolrConnections
* SolrConnection
  + Represents a connection to a Solr Server. Contains CommonsHttpSolrServer.
* SolrConectionFactory
  + Used by the connection pool to generate new SolrConnections
* SolrExecution
  + Creates a SolrQuery and executes it, retrieving a Document list from which results can be read.
* SolrVisitor
  + Translates an ANSI SQL query into a SolrQuery (search string)
* SolrCapabilities
  + Describes the SQL supported by the connector, e.g. SELECT, FROM

## Connector Design Details

The following sequence diagram illustrates the typical sequence of events when executing a query:



Figure Connector Sequence Diagram

The following classes and methods will be implemented to support connectivity to Solr.

### SolrConnector

getConnection(context)

Creates a CommonsHttpSolrServer connection, and configures the properties for the SolrServer connection, such as timeout, max connections, compression, etc. Necessary properties can be exposed as properties of the SolrConnector, and defined at deploy time, making them available as part of the context.

### SolrConnection

createExecution(executionMode, executionContext, metadata)

Creates and returns a SolrExecution.

### SolrExecution

Implements SyncQueryExecution, supporting read-only queries.

Execute(query, maxBatchSize)

Creates a SolrVisitor to parse the SQL statement and create a Solr Query string. Creates a SolrQuery object and executes the query. Saves the document results.

nextBatch()

Retrieves the next batch of results, as limited by the batch property of the connector. Maps the fields in the document to the columns expected in the results. Creates a RowSet containing the results for that batch, and returns the results. Sets lastBatch if at end of document list.

### SolrVisitor

visitNode(Query)

Walks through the MetaMatrix SQL Query object and constructs the appropriate search string formatted for Solr.

getStringQuery()

Returns the search string formatted for Solr.

Javadoc shipped with the code contains additional information on connector behavior.

## Connector Pooling Support

The SolrConnector will be designed to support single-user connection pooling provided by the MetaMatrix framework. The connector pool allows for the reuse of valid connections, which reduces the overhead in establishing a new connection. The pool can yield better performance, especially in high transaction and multi-user concurrent query scenarios.

The pool can be configured using the connector binding properties described in Section 9.

## Result Set Caching Support

The SolrConnector will be designed to support the result set cache capabilities provided by the MetaMatrix framework. The result set cache is an in-memory cache used to store results of particular queries sent to the system. Result Set caching can improve performance when the backend data does not change rapidly, and when the exact same queries are being sent to the system repeatedly.

See the MetaMatrix Data Caching Guide.pdf for details on Result Set Caching.

The cache can be configured using the connector binding properties described in Section 9.

## Monitoring Support

The SolrConnector will implement the Connection.isAlive method, in order to allow the connection pool to be monitored at a regularly scheduled interval by the built-in monitoring service. This will allow the MetaMatrix Console to reflect the current connectivity to the Solr data source, and expose the information via the Admin API.

# Connector Query Capabilities

The capabilities define what SQL query constructs will be translated and incorporated into the Solr search. Any capabilities that are not explicitly supported by the connector will typically be applied by the MetaMatrix query engine in a post-processing step. So, increasing the capabilities set will improve performance by increasing the amount of work done by Solr to filter the results.

The following list enumerates the capabilities that will be supported, and how they wlil be mapped to Solr functionality.

## Limit

This value will map to SolrQuery.setRows(int), and control the total number of results returned.

## Offset

This value will map to SolrQuery.setStart(int), and control the starting point of the results returned.

NOTE: Using offset in combination with ORDER BY may produce unexpected results, since offset is supported by the connector, but Order By is not. Offset will jump through the unordered results in Solr, and ORDER BY will be evaluated in MetaMatrix as a post-processing step.

## Criteria

Criteria in the WHERE clause will be supported as time permits. It is not a strict requirement of the project, but would provide useful, additional functionality, since the criteria would be translated into Solr search criteria, thereby allowing Solr to be searched via SQL statements.

## Processing Literals

Whenever a literal is passed in as search criteria, it will be processed in the following manner:

1. SQL Wildcards will be translated into the Solr equivalent.
2. The string will be shifted to lower case, if the connector binding is set to request it.

### Wildcards

If the literal contains Solr wildcards, they will be passed down in the search, e.g.:

WHERE fieldA = “accou\*”

Will become the search:

fieldA:accou\*

If the literal contains SQL wildcards, they will be translated to the following Solr equivalents:

|  |  |  |
| --- | --- | --- |
| SQL Wildcard | Solr Equivalent | Meaning |
| % | \* | Zero or more chars |
| \_ | ? | One char |

e.g.:

WHERE fieldA = “accou%”

Will become the search:

fieldA:accou\*

### Lower Case Search

Among the default set of filters used by MetaCenter is the solr.LowerCaseFilterFactory, which translates all field values to lower case when building the index. A result of this factory is that all searches should be performed in lower case, and if upper-case values are provided, no matches will be found.

An optional property of the connector binding will be exposed to automatically translate all Iliteral string searches to lower-case. Or, the client can take care to ensure criterion are always passed in lower case values.

### Equals Criteria

Criteria of the following form will be supported:

IElement = Iliteral

Where IElement is a field in the physical model, and ILiteral is a String value to be passed down to the search.

e.g.

WHERE fieldA = “account\_id”

## Compound Criteria

ICompoundCriteria objects will be processed by adding parentheses before and after the criteria contained within, by recursively processing each criterion contained within it, and by adding the appropriate conjunction operator in Solr (e.g. AND or OR).

### AND

AND criteria will be processed in the following manner to generate a Solr search string:

( <processCriteriaA> AND <processCriteriaB> )

### OR

OR criteria will be processed in the following manner to generate a Solr search string:

( <processCriteriaA> OR <processCriteriaB> )

## NOT

NOT criteria will be processed in the following manner to generate a Solr search string:

NOT <processCriteriaA>

## LIKE

LIKE clauses of the following form will be supported:

WHERE fieldA LIKE “acc%”

The LIKE clause will be treated as if it were equals criteria, since the equals criteria performs wildcard replacement.

## Non-supported capabilities

Since Solr is not a relational database, the Solr Connector translates SQL queries to appropriate Solr searches. Some SQL does not have a matching equivalent in Solr, or was considered out-of-scope for the development effort. However, any valid SQL statement can be issued against a Solr Connector-bound data source.

MetaMatrix will use the connector’s *capabilities* to determine what SQL can be processed by the Solr Server. This is known as “query pushdown”, because the query is being pushed down to the source. SQL that is not supported by these capabilities will be performed by MetaMatrix in a post-processing step.

It is important to note that pushdown results in the best performance, since the work is performed at the data source, and only the minimum amount of data is retrieved. If a query statement is not supported by the connector capabilities, then the performance may be impacted, since more data may need to be fetched and processed in MetaMatrix. Therefore, it is useful to know what capabilities are supported by the connector, and what will be processed by MetaMatrix.

The following capabilities are explicitly not supported by the connector:

* JOIN
* BETWEEN
* Aggregations (DISTINCT, SUM, COUNT, MAX, MIN, etc)
* Functions
* ORDER BY
* GROUP BY
* Greater Than, Less than, Not Equal To

If the non-supported capabilities are used in queries, it is advisable to view the query plan and examine the performance, etc., to ensure the MMX system is not overtaxed.

# Connector Properties

## SolrServerURL

URL of the SolrServer, e.g.:

http://localhost:8080/solr/search

## SoTimeout

Sets SolrServer.setSoTimeout(Integer).

Defaults to NULL (uses Solr default).

## ConnectionTimeout

Sets SolrServer.setConnectionTimeout(Integer).

Defaults to NULL (uses Solr default).

## DefaultMaxConnectionsPerHost

Sets SolrServer.setMaxTotalConnections(Integer).

Defaults to NULL (uses Solr default).

## AllowCompression

Sets SolrServer.setAllowCompression(boolean).

Defaults to TRUE.

## MaxRetries

Sets SolrServer.setMaxRetries(Integer).

Defaults to NULL (uses Solr default).

## LowerCaseSearch

Automatically translates all string search criteria to lower-case, to accommodate for Solr filter built on lower-case values.

Defaults to TRUE.

## Max In Criteria Size

Determines the maximum number of clauses to use in a single IN criteria statement. MetaMatrix will break up queries that exceed this limit into multiple queries.

## Connector Pool Cleaning Interval

Interval of time before the pool is checked for connections that have been alive and unused for longer than the Connector Pool Live and Unused Time.

## Connector Pool Live and Unused Time

Amount of time before a connection in the pool is considered closable.

## Connector Pool Max Connections

Maximum number of connections in the pool. If a connection is requested and the pool is at a maximum, the Connector Pool Wait For Source Time will determine how long the thread will wait until an error is thrown.

## Connector Pool Wait For Source Time

Determines how long a thread will wait for a connection (e.g. if the connection pool is temporarily full).

## Result Set Cache Enabled

Activates result set caching.

Default=FALSE

## Result Set Cache Maximum Age

Maximum age, in milliseconds, of data in the result set cache before it can be considered stale.

## Result Set Cache Maximum Size

Maximum size of the result cache before data is evicted.

## Data Source Monitoring Enabled

Provides status information about the source connection when enabled.

# Unit Test Examples

## Test Environment

### MMX

A test VDB named “MDEX.vdb” was created to simulate the Metadata Exchange-to-MetaCenter integration use case. MDEX contains the following models:

|  |  |  |
| --- | --- | --- |
| Model Name | Model Type | Description |
| MC001\_Index | Physical Relational | Represents the backend Solr Index maintained by MetaCenter, which contains all the basic metadata of interest (name, type, etc). |
| MC001\_Refresh | Physical Relational | Represents the Internal Solr Index that contains the last updated time for each data source. |
| CommonObjectModel | Virtual Relational | Represents the common data model to provide a single view of multiple metadata sources. The Object table maps to the metadata in MetaCenter, joining each item to the appropriate refresh date from the Internal table. |

This example VDB can be found in the deliverables package.

### MC-Solr

A Tomcat 6 installation was created to represent the MetaCenter Solr indexes. To configure Tomcat to host the MetaCenter Solr indexes:

* Install the MC-Solr indexes and config directories somewhere on disk, e.g. c:\metacenter\test
* Install Solr.war into the Tomcat webapps directory
* Add the following line to the Tomcat startup JVM options in catalina.bat:
  + -Dsolr.solr.home c:\metacenter\test\config

This results in two Solr Cores: Search and Internal. MC001\_Index and MC001\_Refresh were bound to the corresponding Solr cores, using the Solr Connector and the appropriate URL. Examples of the bindings can be found in the MDEX VDB.

### MDEX-Solr

A standard Solr 1.4 installation was created to represent the Metadata Exchange Solr client. The Solr 1.4 package was downloaded from Apache:

<http://www.apache.org/dyn/closer.cgi/lucene/solr/>

The following files were modified or added to the base installation:

|  |  |
| --- | --- |
| File | Description |
| Schema.xml | Modified to add fields to represent Object fields |
| Solrconfig.xml | Added a DataImportHandler to connect to MetaMatrix as a client |
| mmx-dag-data-config.xml | Configuration file describing the connection to MetaMatrix, and the queries for full import and delta import scenarios. |

Examples of these files can be found in the deliverable package.

## MMX > MC-Solr Tests

The SquirrelSQL JDBC client was connected to MMX’s MDEX VDB, in order to run queries against MC-Solr as a source and validate the connector. The following queries are a *few* examples of what was run:

|  |  |
| --- | --- |
| Purpose | Query |
| Test all columns modeled in MC-Solr Index | select \* from MC001\_Index.SolrIndex |
| Test all columns modeled in MC-Solr Internal | select \* from MC001\_Refresh.RepoRefreshIndex |
| Test all columns in common object model | select \* from CommonObjectModel.Objects |
| Test limit below batch size of 2000 | select display\_name from MC001\_Index.SolrIndex LIMIT 100 |
| Test limit above batch size of 2000 | select display\_name from MC001\_Index.SolrIndex LIMIT 4000 |
| Test equals criteria with uppercase translation to lowercase turned on | select \* from MC001\_Index.SolrIndex where display\_name='ACCOUNT\_DIM' |
| Test equals criteria with lowercase | select \* from MC001\_Index.SolrIndex where display\_name='account\_dim' |
| Test LIKE and wildcard % | select display\_name from MC001\_Index.SolrIndex where display\_name LIKE 'acc%' |
| Test \* wildcard | select display\_name from MC001\_Index.SolrIndex where display\_name='acc\*' |
| Test \_ wildcard | select display\_name from MC001\_Index.SolrIndex where display\_name LIKE 'acc\_UNT\_FACT' |
| Test AND | select \* from MC001\_Index.SolrIndex where display\_name='acc\*' and content='pub\*' |
| Test OR | select \* from MC001\_Index.SolrIndex where display\_name='acc\*' or content='pub\*' |
| Test compound criteria | select \* from MC001\_Index.SolrIndex where (display\_name='public\*' and content='pub') OR display\_name='acc\*' |
| Test IN | select \* from MC001\_Index.SolrIndex where display\_name IN ('account\_dim', 'account\_dim\_id', 'ACCOUNT\_type\_DIM\_id', 'asset\_STR\*') |
| Test NOT, compound criteria | select \* from MC001\_Index.SolrIndex where display\_name IN ('account\_dim', 'account\_dim\_id', 'ACCOUNT\_type\_DIM\_id', 'asset\_STR\*') and content not like 'PUBLIC%' |
| Test unsupported functions GROUP BY, COUNT, ORDER BY | select display\_name, count(\*) mycount from MC001\_Index.SolrIndex where display\_name='ac\*' group by display\_name order by mycount desc |

## MDEX-Solr > MMX > MC-Solr Tests

To simulate the MDEX use case, the MDEX-Solr client was used to execute two built-in scenarios:

|  |  |
| --- | --- |
| Scenario | URL Command |
| Full Import | http://localhost:8983/solr/dataimport?command=full-import |
| Delta Import | http://localhost:8983/solr/dataimport?command=delta-import |

NOTE: In order to simulate the delta import scenario, the Solr client was stopped, and the dataimport.properties file was modified to artifically rollback the last refresh date. This ensured that when the delta import was run, new data would be fetched.

# Appendix: Future Considerations and Known Issues

## Authentication

The Solr Connector does not perform authentication with the Solr server, since the current implementation of MetaCenter/Solr does not use it. If this is added, then the connector should be extended to support authentication over HTTP (i.e. HTTP Basic security).

## Greater Than/Less Than

Greater Than and Less Than operators (>, <) are not suppoted as WHERE clause criteria. This is particularly relevant for the Delta Index scenario, which searches for all Repository IDs that have been updated after a certain date. In this case, MetaMatrix will fetch all RepositoryIDs and RefreshDates, and perform the comparison as a post-processing step.

Since the number of repositories (data sources) in MetaCenter is managably small (say, under 100,000), the cost of this work is very small, and will not be noticable to the end client. However, it is important to note that this work will be evaluated as a post-processing step, especially if the connector is used for other purposes.

## Reserved Characters

To search for reserved characters, the characters should be double-quoted, e.g.:

SELECT fieldA from index WHERE fieldB=’my\”\*\”string’

Will return documents where fieldB is set to the string literal “my\*string”. However:

SELECT fieldA from index WHERE fieldB=’my\*string’

Will return documents where fieldB is set to “mystring”, “my123string”, etc.

Documentation on reserved characters can be found at:

<http://lucene.apache.org/java/2_3_2/queryparsersyntax.html#Escaping%20Special%20Characters>

The current set of reserved characters are:

+ - && || ! ( ) { } [ ] ^ " ~ \* ? : \

## Offset and ORDER BY

Using offset in combination with ORDER BY may produce unexpected results, since offset is supported by the connector, but Order By is not. Offset will jump through the unordered results in Solr, and ORDER BY will be evaluated in MetaMatrix as a post-processing step.

## Scoring and ORDER BY

In Lucene, it is possible to rank and sort searches. Solr does not appear to support this capability – all matching documents get an equal score, according to the online Wiki (<http://wiki.apache.org/solr/SolrQuerySyntax#Differences_From_Lucene_Query_Parser>). If it is exposed in Solr, it could be added to the connector. Special fields could be exposed to support sorting by score.

## Faceted search

Facets were not exposed in this version of the connector, since it is unused by MetaCenter.

## Dynamic Fields and All-field search

In Solr, it is possible to search across all fields, e.g.:

\*:account

Will return all documents with “account” in any field name.

This is not supported in the connector. The fields must be explicitly named, and the connector will pull the data for only those explicitly-named fields. When new fields are added to a Solr schema, they should also be added to the Solr data model.

The ability to set or determine a field at runtime could be added as an enhancement. Two possible ways of implementing this feature are included below for reference purposes.

### Name/Value Models

A relational table could be created with a few columns:

DocId(string)

FieldName(string)

FieldType(string)

FieldValue(object)

SearchString(string)

A special metadata property on the table could indicate that this table should be treated differently by the connector. The column name should not be used as the field name. Instead, the connector would pull ALL fields from the result, and use the value of SearchString as the search criteria, if it has been set.

The results would be a flattened list of all fields for every object matching the search criteria.

If FieldName is set in the criteria, then the connector could dynamically set the SolrQuery to search only for those fields.

If SearchString is unset, then the search string defaults to “\*:\*”.

For example:

SELECT DocId, FieldName, FieldType, FieldValue WHERE SearchString=”\*:account”;

Might return:

101, contents, string, “The last account specified.”

101, display\_name, string, “last\_account”

122, description, string “Use this when the account is present”

Example #2:

SELECT DocId, FieldName, FieldType, FieldValue WHERE SearchString=”\*:account” and FieldName IN (‘contents’, ‘description’);

Might return:

101, contents, string, “The last account specified.”

122, description, string “Use this when the account is present”

### Dynamic Field Column

A special field called “DynamicField” could be reserved, and optionally added to a physical source model. In cases where criteria are set against this field, the appropriate search string could be created to pull the field specified in the criteria. The returned ResultSet would set the values of that field to the return value of the DynamicField column.

## Multi-valued Fields

In Solr, it is possible for a single field to have multiple values. The MetaCenter usage of Solr does not use this feature.

The connector is designed to return the first value retrieved by Solr, and ignore all other field values.

If multi-value fields are used in the future, and all values need to be retrieved, three possible approaches follow:

### Multivalue-concat attribute

Fields that could contain multiple values are marked with metadata in Designer that designate them as “multivalue-concat”. Optionally, a delimiter is specified in this property.

The connector will retrieve all values for these fields, and concatenate the results, using the specified delimiter, or the default delimiter if none is specified.

### Multivalue-expand attribute

Fields that could contain multiple values are marked with metadata in Designer that designate them as “multivalue-expand”.

The connector will retrieve all values for these fields, and return a new row for each value, thereby repeating the rest of the row. If a row has multiple multivalue-expand columns, then the connector will recursively repeat the cross-combination of all values. In this way, the results are denormalized and flattened.

### Name/Value models

A name/value model (see 11.5.1) could be created. The connector would retrieve all values of all fields and return each field/value pair as a separate row.