Informatica® MDM - Relate 360
10.0 HotFix 8

User Guide

Informatica MDM - Relate 360 User Guide
10.0 HotFix 8
November 2018

Publication Date: 2018-11-30

Table of Contents

Preface

The *Informatica MDM - Relate 360 User Guide* provides information about how to install and configure Relate 360 for system administrators and data stewards. This guide assumes that you are familiar with the interface requirements for the Hadoop environment.

Informatica Resources

Informatica Network

Informatica Network hosts Informatica Global Customer Support, the Informatica Knowledge Base, and other product resources. To access Informatica Network, visit https://network.informatica.com.

As a member, you can:

- Access all of your Informatica resources in one place.
- Search the Knowledge Base for product resources, including documentation, FAQs, and best practices.
- View product availability information.
- Review your support cases.
- Find your local Informatica User Group Network and collaborate with your peers.

Informatica Knowledge Base

Use the Informatica Knowledge Base to search Informatica Network for product resources such as documentation, how-to articles, best practices, and PAMs.

To access the Knowledge Base, visit https://kb.informatica.com. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team at KB_Feedback@informatica.com.

Informatica Documentation

To get the latest documentation for your product, browse the Informatica Knowledge Base at https://kb.informatica.com/_layouts/ProductDocumentation/Page/ProductDocumentSearch.aspx.

If you have questions, comments, or ideas about this documentation, contact the Informatica Documentation team through email at infa_documentation@informatica.com.

Informatica Product Availability Matrixes

Product Availability Matrixes (PAMs) indicate the versions of operating systems, databases, and other types of data sources and targets that a product release supports. If you are an Informatica Network member, you can access PAMs at https://network.informatica.com/community/informatica-network/product-availability-matrices.

Informatica Velocity

Informatica Velocity is a collection of tips and best practices developed by Informatica Professional Services. Developed from the real-world experience of hundreds of data management projects, Informatica Velocity represents the collective knowledge of our consultants who have worked with organizations from around the world to plan, develop, deploy, and maintain successful data management solutions.

If you are an Informatica Network member, you can access Informatica Velocity resources at http://velocity.informatica.com.

If you have questions, comments, or ideas about Informatica Velocity, contact Informatica Professional Services at ips@informatica.com.

Informatica Marketplace

The Informatica Marketplace is a forum where you can find solutions that augment, extend, or enhance your Informatica implementations. By leveraging any of the hundreds of solutions from Informatica developers and partners, you can improve your productivity and speed up time to implementation on your projects. You can access Informatica Marketplace at https://marketplace.informatica.com.

Informatica Global Customer Support

You can contact a Global Support Center by telephone or through Online Support on Informatica Network.

To find your local Informatica Global Customer Support telephone number, visit the Informatica website at the following link:
http://www.informatica.com/us/services-and-training/support-services/global-support-centers.

If you are an Informatica Network member, you can use Online Support at http://network.informatica.com.

CHAPTER 1

Introduction to Informatica MDM - Relate 360

This chapter includes the following topics:

Informatica MDM - Relate 360 Overview

Use Informatica MDM - Relate 360 to link or tokenize the input data in a Hadoop environment. You can consolidate the linked data to get preferred records. You can also create relationships between the records and view the relationships in a relationship graph.

Linking is a process of grouping related records into clusters based on the matching rules. Tokenization is a process of adding a fuzzy token, which is an encoded key, to each input record. Consolidation is a process of merging data from different records in a cluster to create a preferred or golden record based on the consolidation rules. A relationship graph connects all the related records of a record through relationships.

You can use the linked and consolidated data to perform business analytics and reporting by using a third-party tool. You can use the tokenized data to perform search operations.

The input data can be batch data or streaming data. You can persist the linked or tokenized data in HDFS or in a repository.

Relate 360 Components

Relate 360 contains the following components:

SSA-NAME3

Builds keys, defines match criteria to link the records, and identifies the matching records based on the keys. SSA-NAME3 uses algorithms that can handle initials, aliases, common variations, prefixes, suffixes, transpositions, and word order.

Batch Jobs

Link or tokenize the input data, load the linked or tokenized data into a repository, consolidate the linked data, and search for matching records in the indexed or tokenized data.

Enabler

Loads the linked or consolidated data from HDFS or repository into Hive.

RESTful Web Services and Command-Line Commands

Perform fuzzy searches that can return relevant matching records or exact searches that can return exact matching records.

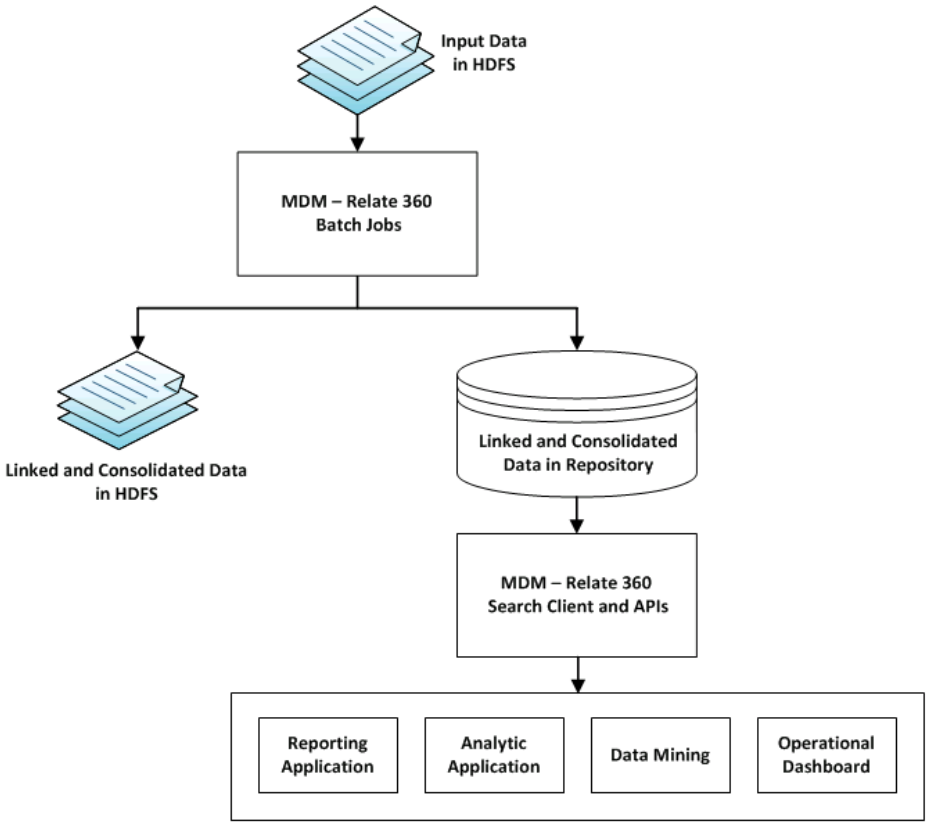
Linking Process

You can link batch data or streaming data. To link batch data, store the input data in HDFS. Relate 360 reads the input data from HDFS and links the input data. You can then persist the linked data in a repository or data warehouse. You can also consolidate the linked data that you persist in HDFS or repository.

To link streaming data, stream the data in the JSON format. Relate 360 links the input data and persists the linked data in a repository. You can also consolidate the streaming data.

Linking Batch Data

The following image shows how Relate 360 links the batch data and persists the linked data in HDFS or in a repository:

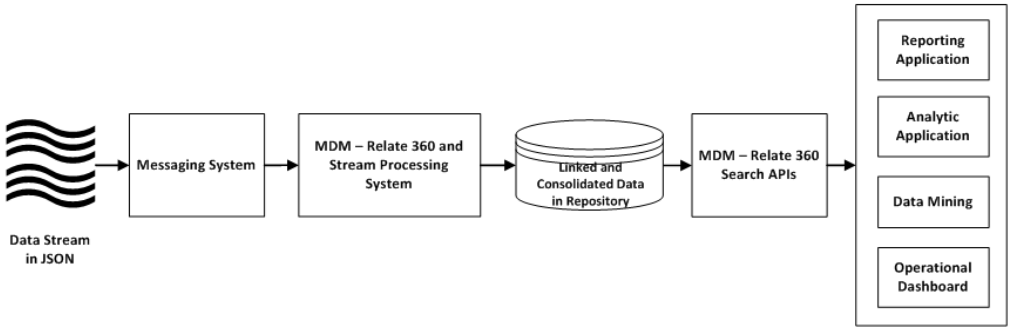


To link the input batch data, perform the following tasks:

1. Run the Relate 360 batch jobs to read the input data from HDFS and link the input data.
2. Optionally, load the linked data into the repository.
3. Optionally, consolidate the linked data.
4. If you want to perform search operation on the linked or consolidated data, use the RESTful web services or the command-line commands.
5. Optionally, if you want to perform analytics, load the linked data into a data warehouse.

Linking Streaming Data

The following image shows how Relate 360 links the streaming data and persists the linked data in a repository:



To link and consolidate the input streaming data, perform the following tasks:

1. Stream the input data in the JSON format.
2. Use the Relate 360 RESTful web services or the command-line commands to read the input data, link and consolidate the input data, and load the linked and consolidated data into the repository.
3. Optionally, if you want to search for matching records in the repository, use the RESTful web services or the command-line commands.
4. Optionally, if you want to perform analytics, load the linked and consolidated data into a data warehouse and use third-party tools to perform business analytics on the linked and consolidated data.

Use Case for the Linking Process

You work for a large insurance organization that wants to enrich the existing customer database with data from third-party data provider services in a Hadoop environment. The organization wants to compare the existing data with the third-party data to identify potential business prospects. The organization wants a 360-degree view of the customers to understand the relationship between them and come up with targeted marketing programs.

To identify potential business prospects and develop targeted marketing programs, perform the following tasks:

1. Use a data integration tool to read data from different data sources and write the data to HDFS in the fixed-width format.
2. Use Relate 360 to link the input records.
3. Load the linked data into HBase.
4. To identify potential business prospects, identify the group of records that do not have records from the existing customer database.
5. View the details of the potential business prospects to understand the relationship between them.
6. Based on the relationship between the potential business prospects, develop targeted marketing programs.

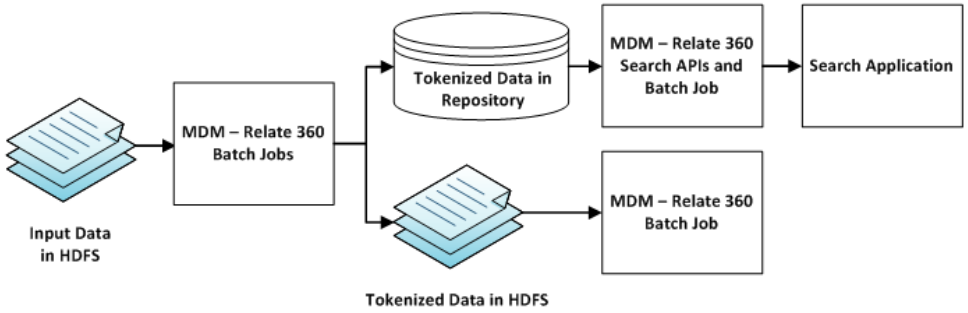
Tokenization Process

You can tokenize batch data or streaming data. To tokenize batch data, store the input data in HDFS. Relate 360 reads the input data from HDFS and tokenizes the input data. You can then persist the tokenized data in a repository. To tokenize streaming data, stream the input data in the JSON format. Relate 360 tokenizes the input data and persists the tokenized data in a repository.

The tokenized data contains input data and encoded tokens for the input data. You can perform searches on the tokenized data.

Tokenizing Batch Data

The following image shows how you can tokenize data in HDFS and persist the tokenized data in HDFS or in a repository:

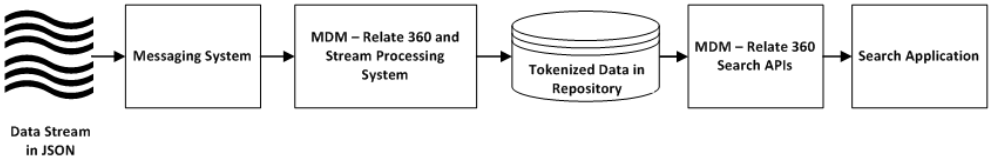


To tokenize the input data, perform the following tasks:

1. Run the Relate 360 batch jobs to read the input data in HDFS, add tokens to the input data, and persist the tokenized data in HDFS or in a repository.
2. To search the tokenized data for matching records, use the RESTful web services, and the batch job.

Tokenizing Streaming Data

The following image shows how Relate 360 tokenizes the streaming data and persists the tokenized data in a repository:



To tokenize the input streaming data, perform the following tasks:

1. Stream the input data in the JSON format.
2. Use the Relate 360 RESTful web services or the command-line command to read the input data, link the input data, and persist the tokenized data in a repository.
3. To perform search operation on the tokenized data, use the RESTful web services to search the tokenized data for matching records.

Use Case for the Tokenization Process

You work for a law enforcement agency that has a large data set. The agency uses the data to identify people or organization for fraud detection, intelligence, and screening purposes. The agency requires a product that is scalable to screen a large data set with the capability to manage all the variations and errors in the data.

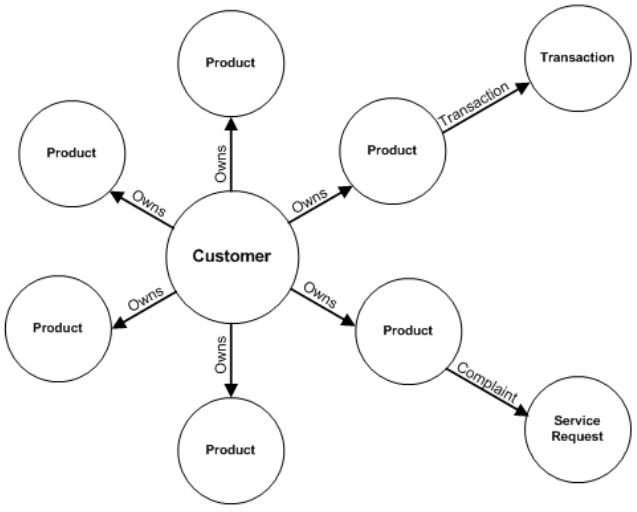
For example, if you search for an address, the product must return all the records that contain the address irrespective of all the variations in the address. Based on the search results, you can identify the person who can be a potential threat to the national security.

You can use Relate 360 to tokenize a large data set and search for matching records.

Relationship Graph

You can create relationships between the processed records. The processed records can be customer data, transaction data, product data, and other types of data. You can consider each type of data as a business entity type. When you create relationships between two or more business entity types, the relationships result in a relationship graph. A relationship graph displays all the related records of a record and its relationships with the related records.

The following image shows a representational relationship graph:



In the representational relationship graph, Customer, Product, Transaction, and Service Request are business entity types, and owns, transaction, and complaint are relationships that connect the business entity types. The representational graph can display all the products a customer owns. The graph can also display the service requests and the transaction details of each product the customer owns.

Use Case for the Relationship Graph

You work for a large insurance organization that uses different data sources to store its data, such as customer data, product data, and transaction data. The organization wants to relate the product and transaction data to the customer data to have a 360-degree view of the customers. The organization wants to understand their customers better and come up with targeted marketing programs.

To relate the customer data with the product and transaction data, perform the following tasks:

1. Use a data integration tool to read data from different data sources and write the data to HDFS in the fixed-width format.
2. Link the customer data to identify the household members.
3. Tokenize the product and transaction data.
4. Load the linked and tokenized data into the repository.
5. Create relationships between the customer and product data and the customer and transaction data.
6. View the details of the customers in the relationship graph to understand the customers and come up with targeted marketing programs.

CHAPTER 2

Linking Batch Data

This chapter includes the following topics:

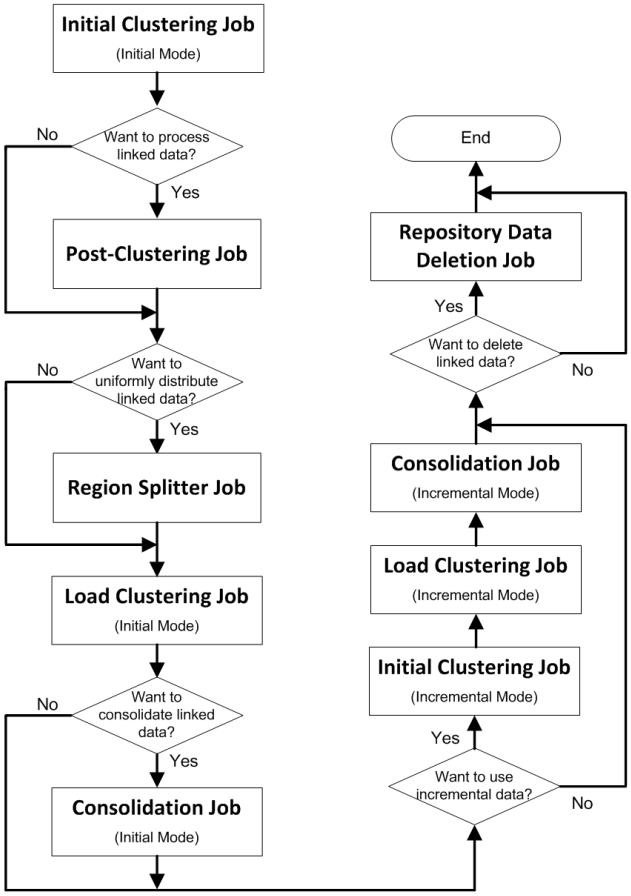
Linking Batch Data Overview

Use the Informatica MDM - Relate 360 batch jobs to link the batch data. Relate 360 reads the input data from HDFS and links the input data. You can consolidate the linked data to create the preferred records. You can then persist the linked and consolidated data in a repository or data warehouse.

Linking Data and Persisting the Linked Data in a Repository

You can link the input data based on the matching rules and consolidate the linked data based on the consolidation rules. You can then persist the linked and consolidated data in a repository so that you can perform data analytics or searches on the data.

The following image shows the batch jobs that you can run to link the input data, consolidate the linked data, and persist the linked data in a repository:



To persist the linked data in a repository, perform the following tasks:

1. Run the initial clustering job.

   The job links the input data and creates clusters for the input data in HDFS.

2. If you want to process the output files of the initial clustering job, run the post-clustering job.

   The post-clustering job reads the output files that the initial clustering job creates in HDFS and processes it based on the mode that you set.

3. If you want to uniformly distribute the linked data across all the regions in the repository, run the region splitter job.

   The job analyzes the input linked data and identifies the split points for all the regions in the repository.

4. Run the load clustering job.

   The job loads the linked data from HDFS into the repository.

5. If you want to consolidate the linked data, run the consolidation job.

   The consolidation job creates a preferred records table with a preferred record for each cluster.

6. To add an incremental data to the repository, run the initial clustering job in the incremental mode to link the incremental data and run the load clustering job in the incremental mode to add the linked data to the repository.

   If you consolidate the linked data, you can also run the consolidation job in the incremental mode to update the incremental data.

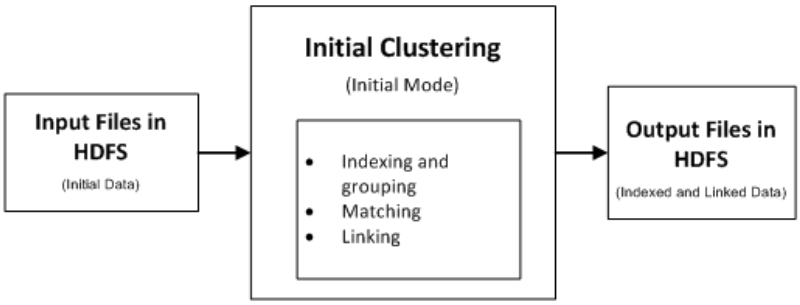
7. To delete records from the repository, run the repository data deletion job with the `--useIndexId` parameter.

Initial Clustering Job

The initial clustering job indexes and links the input data based on the rules in the matching rules file and the parameters in the configuration file. The initial clustering job reads the input data in HDFS and persists the indexed and linked data in HDFS. You can also run the initial clustering job in the incremental mode to incrementally update the indexed and linked data in HDFS.

Initial Mode

Use the initial mode to index and link the data for the first time. The following image shows how the initial clustering job indexes and links the input data when you run the job in the initial mode:



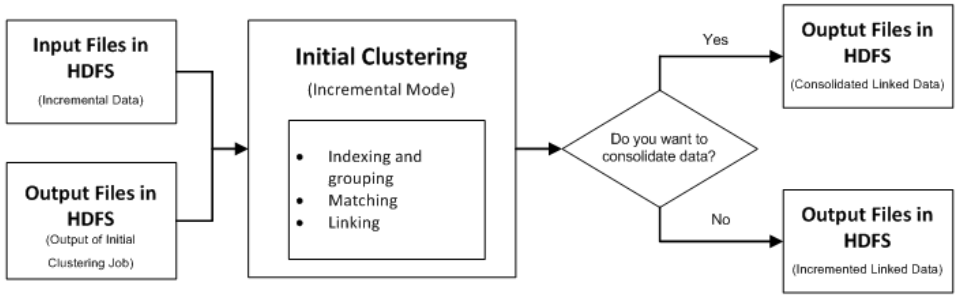
When you run the initial clustering job in the initial mode, the job performs the following tasks:

1. Reads the input files from HDFS.
2. Indexes and groups the input data based on the rules in the matching rules file.
3. Matches the grouped records and links the matched records.
4. Creates clusters for the matched and unmatched records.
5. Writes the indexed and linked records to the output files in HDFS.

   Note: The number of output files depends on the number of reducers that you run.

Incremental Mode

Use the incremental mode to incrementally update the existing indexed and linked data. The following image shows how the initial clustering job indexes and links the incremental data when you run the job in the incremental mode:



When you run the initial clustering job in the incremental mode, the job performs the following tasks:

1. Reads each record from the input files and generates keys and ranges based on the matching rules file.

2. Searches and matches the generated keys with the keys that the previous run of the initial clustering job creates.
3. Links the matched records to an existing cluster and creates clusters for the unmatched records.
4. If you configure the job to consolidate data, the job consolidates the incremental data with the indexed and linked data and writes the consolidated data to HDFS. If you configure the job not to consolidate data, the job indexes and links the incremental data and writes the incremental data to HDFS.

Note: If an input record matches with records that belong to two different clusters, the initial clustering job links the input record to any one of the clusters.

Running the Initial Clustering Job

Use the initial clustering job to read data from the input files and then index and link the input data in HDFS. You can also use the initial clustering job to incrementally update the indexed and linked data with additional data.

To run the initial clustering job, run the `run_genclusters.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Initial Mode

Use the following command to run the `run_genclusters.sh` script in the initial mode:

```
run_genclusters.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--rule=matching_rules_file_name
[--outputpath=directory_for_output_files]
[--reducer=number_of_reducers]
[--keeptemp=true|false]
[--compression=true|false]
```

The following table describes the options and the arguments that you can specify to run the `run_genclusters.sh` script in the initial mode:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the input files in HDFS.
--reducer	number_of_reducers	Optional. Number of reducer jobs that you want to run to perform initial clustering. Default is 1.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The initial clustering job uses the working directory to store the output and library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.

Option	Argument	Description
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.
--keeptemp	true|false	Optional. Indicates whether to retain the intermediate output tables that the initial clustering job creates. You can use the intermediate output tables for troubleshooting purposes. Set to true to retain the intermediate output tables, and set to false to remove the intermediate output tables after the successful run of the job. Default is false.
--compression	true|false	Optional. Indicates whether to compress the output files that the initial clustering job creates. You can compress the output files to avoid any storage issues. Set to true to compress the output files, and set to false to retain the original size of the output files. Default is false.

For example, the following command runs the initial clustering job in the initial mode:

```
run_genclusters.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/
Source10Million --reducer=16 --outputpath=/usr/hdfs/outputfolder --hdfsdir=/usr/hdfs/
workingdir --rule=/usr/local/conf/matching_rules.xml
```

If you run the initial clustering job without the `--outputpath` parameter, you can find the linked data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/output/dir/pass-join`

Each job generates a unique ID, and you can identify the job ID based on the time stamp of the `<Job ID>` folder.

If you run the initial clustering job with the `--outputpath` parameter, you can find the linked data in the following directory: `<Output Directory in HDFS>/batch-cluster/output/dir/pass-join`

The following sample output of an initial clustering job shows the cluster ID, the field values, match rule name, and the metadata related to the cluster:

```
683e6e41-9174-4a22-b08e-d5d4adc9b2ee        0000000007ERP        3M        3M
Center        St. Paul        USA        ARC SOFT        07_UK_Rule1       00000001ZZB>$$
$$01000004NAH-C$$$QVM$*K$-N?H-C$$-NAH$$$$-
```

Incremental Mode

Use the following command to run the `run_genclusters.sh` script to incrementally update the indexed and linked data with additional data:

```
run_genclusters.sh

--config=configuration_file_name

--input=input_file_in_HDFS

--hdfsdir=working_directory_in_HDFS

--rule=matching_rules_file_name

--incremental
```

```
--clustereddirs=indexed_linked_data_directory
[--reducer=number_of_reducers]
[--outputpath=directory_for_output_files]
[--consolidate]
[--keeptemp=true|false]
[--compression=true|false]
```

The following table describes the options and the arguments that you can specify to run the `run_genclusters.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the input files in HDFS.
--reducer	number_of_reducers	Optional. Number of reducer jobs that you want to run to perform initial clustering. Default is 1.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The initial clustering job uses the working directory to store the output and library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.
--incremental		Runs the initial clustering job in the incremental mode. If you want to incrementally update the indexed and linked data in HDFS, run the job in the incremental mode. By default, the initial clustering job runs in the initial mode.
--clustereddirs	indexed_linked_data_directory	Absolute path to the directory that contains linked data. If you run the initial clustering job without the `--outputpath` parameter, you can find the linked data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/output/dir/pass-join` If you run the initial clustering job with the `--outputpath` parameter, you can find the linked data in the following directory: `<Output Directory in HDFS>/batch-cluster/output/dir/pass-join`
--consolidate		Consolidates the incremental data with the existing indexed and linked data in HDFS. By default, the initial clustering job indexes and links only the incremental data.
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.

Option	Argument	Description
--keeptemp	true|false	Optional. Indicates whether to retain the intermediate output tables that the initial clustering job creates. You can use the intermediate output tables for troubleshooting purposes. Set to true to retain the intermediate output tables, and set to false to remove the intermediate output tables after the successful run of the job. Default is false.
--compression	true|false	Optional. Indicates whether to compress the output files that the initial clustering job creates. You can compress the output files to avoid any storage issues. Set to true to compress the output files, and set to false to retain the original size of the output files. Default is false.

For example, the following command runs the initial clustering job in the incremental mode:

```
run_genclusters.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/
Source10Million --reducer=16 --hdfsdir=/usr/hdfs/workingdir --outputpath=/usr/hdfs/
outputfolder --rule=/usr/local/conf/matching_rules.xml --clustereddirs=/usr/hdfs/
workingdir/batch-cluster/MDMBDRM_931211654144593570/output/dir/pass-join --incremental
```

Post-Clustering Job

Use the post-clustering job to read the output files of an initial clustering job in HDFS and process the input data based on the mode that you configure. The input data can be linked data or poor quality data.

You can run the post-clustering job in one of the following modes:

Skip

Skips the records in the high-volume clusters that contain more than the specified number of records.

Recluster

Re-links the records in the high-volume clusters that contain more than the specified number of records.

Longtail

Decrypts the poor quality records that the initial clustering job identifies to the original input format. You can cleanse the decrypted data, and use it as the input data for the initial clustering job.

Export

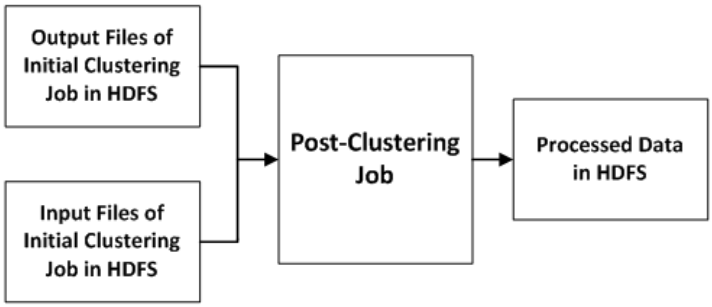
Exports the linked data in the CSV format.

The following image shows how the post-clustering job processes the input data in the skip, recluster, and longtail modes:

The post-clustering job performs the following tasks:

1. Reads the output files of an initial clustering job in HDFS.
2. Processes the input data based on the mode that you configure.
3. Writes the processed data in HDFS.

The following image shows how the post-clustering job processes the input data in the export mode:



The post-clustering job performs the following tasks:

1. Reads the input and output files of an initial clustering job in HDFS.
2. Exports the linked data in the CSV format.

Running the Post-Clustering Job

The post-clustering job uses the output files of an initial clustering job, so ensure that you run the initial clustering job before you run the post-clustering job.

To run the post-clustering job, run the `run_postprocess.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Skip and Recluster Modes

To run the `run_postprocess.sh` script in the skip or recluster mode, use the following command format:

```
run_postprocess.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--rule=matching_rules_file_name
--maxcluster=maximum_number_of_records
--mode=SKIP_LARGE_CLUSTER|RECLUSTER_TRANSITIVE
--threshold=matching_score_to_recluster
[--reducer=number_of_reducer_jobs]
```

The following table describes the options and arguments that you can specify to run the `run_postprocess.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file.
--input	input_file_in_HDFS	Absolute path to the directory that contains linked data. If you run the initial clustering job without the `--outputpath` parameter, you can find the linked data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/output/dir/pass-join` If you run the initial clustering job with the `--outputpath` parameter, you can find the linked data in the following directory: `<Output Directory in HDFS>/batch-cluster/output/dir/pass-join`
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The post-clustering job uses the working directory to store the library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file. The values in the matching rules file override the values in the configuration file.
--maxcluster	maximum_number_of_records	Maximum number of records in a cluster. If the number of records in a cluster exceeds the specified maximum number of records, the cluster becomes a high-volume cluster, and the post-clustering job processes the cluster.
--mode	SKIP_LARGE_CLUSTER| RECLUSTER_TRANSITIVE	Indicates the mode to process the input data. Use one of the following modes: - `SKIP_LARGE_CLUSTER`. Indicates to skip all the records in the high-volume clusters. - `RECLUSTER_TRANSITIVE`. Indicates to re-link all the records in the high-volume clusters.
--threshold	matching_score_to_recluster	Minimum score required for the records to be part of the same cluster when you run the post-clustering job in the recluster mode. The job creates a separate cluster for each record whose score is less than the threshold value.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run. Default is 1.

For example, the following command runs the post-clustering job in the skip mode:

```
run_postprocess.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/workingdir/
batch-cluster/MDMBDE0063_1602999447744334391/output/dir/pass-join --hdfsdir=/usr/hdfs/
workingdir --rule=/usr/local/conf/matching_rules.xml --maxcluster=1200 --threshold=90 --
mode=SKIP_LARGE_CLUSTER
```

If you run the post-clustering job without the `--outputpath` parameter, you can find the processed data in the following directory: `<Working Directory in HDFS>/ClusterPostProcessing/<Job ID>/output`

If you run the post-clustering job with the `--outputpath` parameter, you can find the processed data in the following directory: `<Output Directory in HDFS>/ClusterPostProcessing/output`

Longtail Mode

The initial clustering job identifies the poor quality data from the input data and loads it to a directory in an encrypted format. The post-clustering job decrypts the poor quality data to the original input format.

To run the `run_postprocess.sh` script in the longtail mode, use the following command format:

```
run_postprocess.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--rule=matching_rules_file_name
--mode=LONGTAIL_CLUSTERS
[--reducer=number_of_reducer_jobs]
```

The following table describes the options and arguments that you can specify to run the `run_postprocess.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file.
--input	input_file_in_HDFS	Absolute path to the directory that contains the poor quality data. If you run the initial clustering job without the `--outputpath` parameter, you can find the poor quality data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/poorqualitydata` If you run the initial clustering job with the `--outputpath` parameter, you can find the poor quality data in the following directory: `<Output Directory in HDFS>/batch-cluster/poorqualitydata`
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The post-clustering job uses the working directory to store the library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file. The values in the matching rules file override the values in the configuration file.
--mode	LONGTAIL_CLUSTERS	Indicates that the job processes the poor quality data.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run. Default is 1.

For example, the following command runs the post-clustering job in the longtail mode:

```
run_postprocess.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/workingdir/
batch-cluster/MDMBDE0063_1602999447744334391/poorqualitydata --hdfsdir=/usr/hdfs/
workingdir --rule=/usr/local/conf/matching_rules.xml --mode=LONGTAIL_CLUSTERS
```

If you run the post-clustering job without the `--outputpath` parameter, you can find the decrypted data in the following directory: `<Working Directory in HDFS>/ClusterPostProcessing/<Job ID>/output`

If you run the post-clustering job with the `--outputpath` parameter, you can find the decrypted data in the following directory: `<Output Directory in HDFS>/ClusterPostProcessing/output`

Export Mode

To run the `run_postprocess.sh` script in the export mode, use the following command format:

```
run_postprocess.sh
--config=configuration_file_name
--input=input_file_in_HDFS
```

```
--matchinput=match_output_in_HDFS

--hdfsdir=working_directory_in_HDFS

--rule=matching_rules_file_name

--mode=CSV_OUTPUT

[--reducer=number_of_reducer_jobs]
```

The following table describes the options and arguments that you can specify to run the `run_postprocess.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file.
--input	input_file_in_HDFS	Absolute path to the directory that contains the input files of the initial clustering job.
--matchinput	match_output_in_HDFS	Absolute path to the directory that contains the linked data. If you run the initial clustering job without the `--outputpath` parameter, you can find the linked data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/match/dir` If you run the initial clustering job with the `--outputpath` parameter, you can find the linked data in the following directory: `<Output Directory in HDFS>/batch-cluster/match/dir`
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The post-clustering job uses the working directory to store the library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file. The values in the matching rules file override the values in the configuration file.
--mode	CSV_OUTPUT	Indicates to export the input data in the CSV format.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run. Default is 1.

For example, the following command runs the post-clustering job in the export mode:

```
run_postprocess.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/
Source10Million --matchinput=/usr/hdfs/workingdir/batch-cluster/
MDMBDE0063_1602999447744334391/match/dir/ --hdfsdir=/usr/hdfs/workingdir --rule=/usr/
local/conf/matching_rules.xml --mode=CSV_OUTPUT
```

If you run the post-clustering job without the `--outputpath` parameter, you can find the CSV files in the following directory: `<Working Directory in HDFS>/ClusterPostProcessing/<Job ID>/output-match`

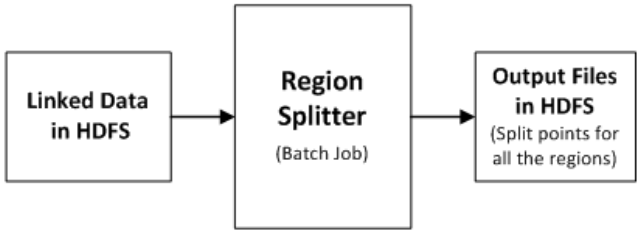
If you run the post-clustering job with the `--outputpath` parameter, you can find the CSV files in the following directory: `<Output Directory in HDFS>/ClusterPostProcessing/output-match`

Region Splitter Job

Use the region splitter job to analyze the input linked data and identify the split points to uniformly distribute the linked data across all the regions in the repository. The uniform distribution of the linked data optimally utilizes the resources and improves the search performance.

A load clustering job uses the output files of a region splitter job to distribute the linked data. Run the region splitter job before you run the load clustering job for the first time.

The following image shows how the region splitter job identifies the split points based on the input data:



The region splitter job performs the following tasks:

1. Reads the linked data in HDFS.
2. Identifies the split points for the number of regions that you specify.

Running the Region Splitter Job

The region splitter job identifies the split points for the input linked data to uniformly distribute the linked data across all the regions.

To run the region splitter job, run the `run_hbase_region_analysis.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_hbase_region_analysis.sh` script:

```
run_hbase_region_analysis.sh

--config=configuration_file_name

--input=input_file_in_HDFS

--hdfsdir=working_directory_in_HDFS

--rule=matching_rules_file_name

--regions=number_of_regions

[--outputpath=directory_for_output_files]

[--reducer=number_of_reducer_jobs]
```

The following table describes the options and arguments that you can specify to run the `run_hbase_region_analysis.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the directory that contains linked data. If you run the initial clustering job without the `--outputpath` parameter, you can find the linked data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/output/dir/pass-join` If you run the initial clustering job with the `--outputpath` parameter, you can find the linked data in the following directory: `<Output Directory in HDFS>/batch-cluster/output/dir/pass-join`
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The region splitter job uses the working directory to store the library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create. The values in the matching rules file override the values in the configuration file.
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.
--regions	number_of_regions	Number of regions that you want to use for the input data. The optimal number of regions depends on your environment and resources. For more information about regions and split points, see the repository documentation.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run. Default is 1.

For example, the following command runs the region splitter job:

```
run_hbase_region_analysis.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/
workingdir/MDMBDRMInitialBatch/MDMBDE0063_1602999447744334391/output/dir/pass-join --
hdfsdir=/usr/hdfs/workingdir --rule=/usr/local/conf/matching_rules.xml --regions=14 --
```

If you run the region splitter job without the `--outputpath` parameter, you can find the output files in the following directory: `<Working Directory in HDFS>/MDMBDRMRegionAnalysis/<Job ID>`

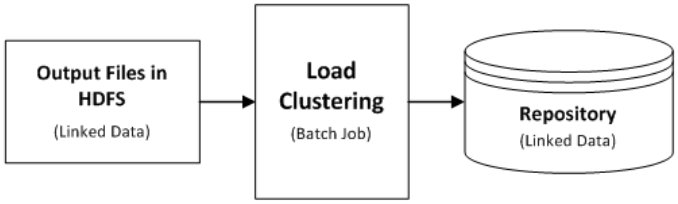
If you run the region splitter job with the `--outputpath` parameter, you can find the output files in the following directory: `<Output Directory in HDFS>/MDMBDRMRegionAnalysis`

Load Clustering Job

The load clustering job loads the output files of an initial clustering job from HDFS into the repository. Before you run the load clustering job, you can run the region splitter job to identify the split points for the input data.

You can also run the load clustering job in the incremental mode to load the incremental data into the repository.

The following image shows how the load clustering job loads the data into the repository:



The load clustering job performs the following tasks:

1. Reads the linked data from the output files of an initial clustering job in HDFS.
2. Loads the linked data into the repository.

Running the Load Clustering Job

The load clustering job loads the linked data from HDFS into the repository. The linked data can be initial data or incremental data.

To run the load clustering job, run the `run_clusterload.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Initial Mode

Use the following command to run the `run_clusterload.sh` script in the initial mode:

```
run_clusterload.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--rule=matching_rules_file_name
[--outputpath=directory_for_output_files]
[--reducer=number_of_reducer_jobs]
[--hbaseregionsplitpath=output_directory_of_region_splitter_job]
```

The following table describes the options and arguments that you can specify to run the `run_clusterload.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the directory that contains linked data. If you run the initial clustering job without the `--outputpath` parameter, you can find the linked data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/output/dir/pass-join` If you run the initial clustering job with the `--outputpath` parameter, you can find the linked data in the following directory: `<Output Directory in HDFS>/batch-cluster/output/dir/pass-join`
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run to perform load clustering. Default is 1.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The load clustering process uses the working directory to store the library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create. The values in the matching rules file override the values in the configuration file.

Option	Argument	Description
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.
--hbaseregionsplitpath	output_directory_of_region_splitter_job	Optional. Absolute path to the output files of a region splitter job. Based on the analysis of the region splitter job, the load clustering job uniformly distributes the linked data across all the regions. If you run the region splitter job without the `--outputpath` parameter, you can find the output files in the following directory: `<Working Directory in HDFS>/MDMBDRMRegionAnalysis/<Job ID>` If you run the region splitter job with the `--outputpath` parameter, you can find the output files in the following directory: `<Output Directory in HDFS>/MDMBDRMRegionAnalysis` By default, the load clustering job randomly distributes the linked data and might result in inconsistent distribution of data across the regions.

For example, the following command runs the load clustering job:

```
run_clusterload.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/workingdir/
MDMBDEInitialBatch/MDMBDE0063_1602999447744334391/output/dir/pass-join --reducer=16 --
hdfsdir=/usr/hdfs/workingdir --rule=/usr/local/conf/matching_rules.xml --
hbaseregionsplitpath=/usr/hdfs/workingdir/MDMBDRMRegionAnalysis/
MDMBDRM0063_8862443019807752334 --outputpath=/usr/hdfs/outputfolder
```

The following sample output of the load clustering job shows index ID, fuzzy keys, and the values from the column family:

```
ROW                                                COLUMN
+CELL

 00KCKSHX$$    SALESFO column=aml_link_columns:CLUSTERNUMBER, timestamp=1454327424272,
value=5fb1e9b8-1b51-47a3-bd45-d885bdc6bbcf   RCE0000000066
 00KCKSHX$$    SALESFO column=aml_link_columns:LMT_MATCHED_PK, timestamp=1454327424272,
value=                                     RCE0000000066
 00KCKSHX$$    SALESFO column=aml_link_columns:LMT_MATCHED_RECORD_SOURCE,
timestamp=1454327424272, value=                            RCE0000000066
 00KCKSHX$$    SALESFO column=aml_link_columns:LMT_MATCHED_SCORE,
timestamp=1454327424272, value=0                                  RCE0000000066
 00KCKSHX$$    SALESFO column=aml_link_columns:LMT_SOURCE_NAME, timestamp=1454327424272,
value=                      SALESFORCE     RCE0000000066
 00KCKSHX$$    SALESFO column=aml_link_columns:NAME, timestamp=1454327424272,
value=Abbott Laboratories                             RCE0000000066
 00KCKSHX$$    SALESFO column=aml_link_columns:ROWID, timestamp=1454327424272,
value=0000000066
```

Incremental Mode

Use the following command to run the `run_clusterload.sh` script in the incremental mode:

```
run_clusterload.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--rule=matching_rules_file_name
--incremental
[--outputpath=directory_for_output_files]
[--reducer=number_of_reducer_jobs]
```

The following table describes the options and arguments that you can specify to run the `run_clusterload.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the directory that contains linked data. If you run the initial clustering job without the `--outputpath` parameter, you can find the linked data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/output/dir/pass-join` If you run the initial clustering job with the `--outputpath` parameter, you can find the linked data in the following directory: `<Output Directory in HDFS>/batch-cluster/output/dir/pass-join`
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run to perform load clustering. Default is 1.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The load clustering process uses the working directory to store the library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create. The values in the matching rules file override the values in the configuration file.
--incremental		Runs the job in the incremental mode. Use the `--incremental` option when you load the incremental linked data. The incremental linked data can be the output of the initial clustering job that you run in the incremental mode without setting the `--consolidate` option.
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.

For example, the following command runs the load clustering job:

```
run_clusterload.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/
outputfolder/batch-cluster/output/dir/pass-join --reducer=16 --hdfsdir=/usr/hdfs/
workingdir --rule=/usr/local/conf/matching_rules.xml --incremental --outputpath=/usr/
hdfs/outputload
```

The following sample output of the load clustering job shows index ID, fuzzy keys, and the values from the column family:

```
ROW                                                COLUMN
+CELL

 00KCKSHX$$    SALESFO column=aml_link_columns:CLUSTERNUMBER, timestamp=1454327424272,
value=5fb1e9b8-1b51-47a3-bd45-d885bdc6bbcf   RCE0000000066
 00KCKSHX$$    SALESFO column=aml_link_columns:LMT_MATCHED_PK, timestamp=1454327424272,
value=                                     RCE0000000066
 00KCKSHX$$    SALESFO column=aml_link_columns:LMT_MATCHED_RECORD_SOURCE,
timestamp=1454327424272, value=                          RCE0000000066
 00KCKSHX$$    SALESFO column=aml_link_columns:LMT_MATCHED_SCORE,
timestamp=1454327424272, value=0                                    RCE0000000066
 00KCKSHX$$    SALESFO column=aml_link_columns:LMT_SOURCE_NAME, timestamp=1454327424272,
value=                      SALESFORCE     RCE0000000066
 00KCKSHX$$    SALESFO column=aml_link_columns:NAME, timestamp=1454327424272,
value=Abbott Laboratories                            RCE0000000066
 00KCKSHX$$    SALESFO column=aml_link_columns:ROWID, timestamp=1454327424272,
value=0000000066
```

Consolidation Job

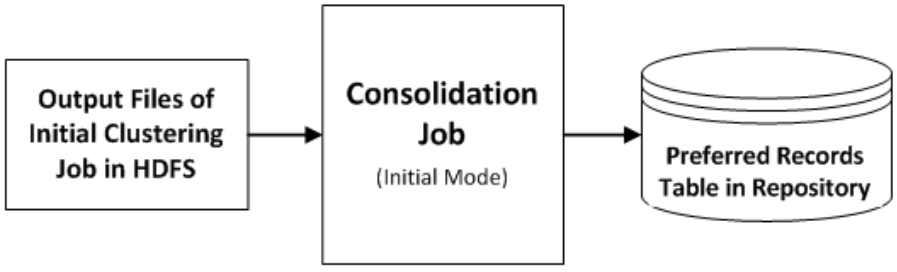
Use the consolidation job to consolidate the linked data and create a preferred records table with a preferred record for each cluster in the repository. The consolidation job creates the preferred records based on the rules defined in the consolidation rules file. You can run the consolidation job in the initial, incremental, and regenerate modes.

Note: The consolidation process ignores any null values.

Initial Mode

Use the initial mode to consolidate the records and create preferred records for the first time. The initial mode uses the output files of an initial clustering job in HDFS as input.

The following image shows how the consolidation job processes the input data in the initial mode:



The consolidation job performs the following tasks:

1. Reads the output files of an initial clustering job in HDFS.

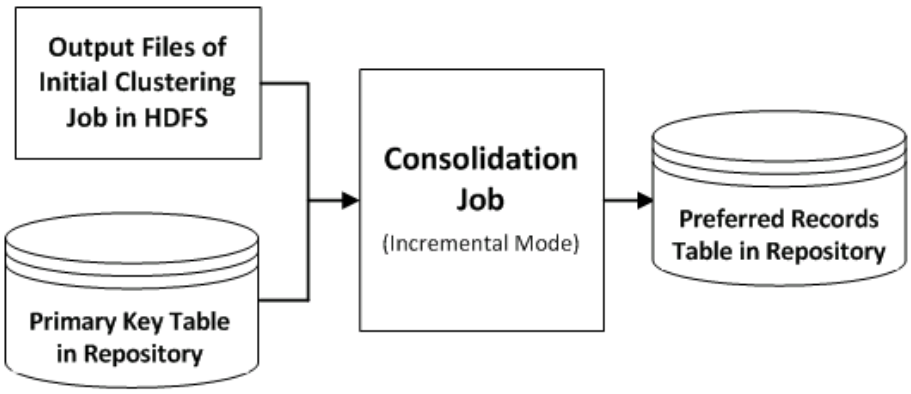
2. Creates a preferred records table that contains a preferred record for each cluster in the repository based on the rules defined in the consolidation rules file.

   Note: The consolidation job sets the same time stamp for all the records in the repository.

Incremental Mode

Use the incremental mode to create and update the preferred records for the incremental linked data. The incremental mode uses the output files of an initial clustering job in HDFS and the primary key table in the repository as input. Before you run the consolidation job in the incremental mode, ensure that the initial linked data is loaded into the repository.

The following image shows how the consolidation job processes the input data in the incremental mode:



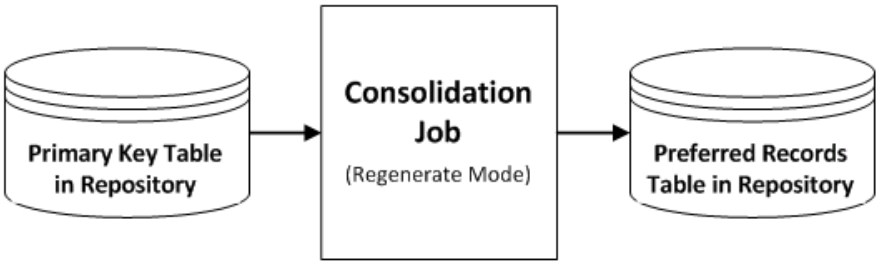
The consolidation job performs the following tasks:

1. Reads the output files of an initial clustering job in HDFS and the primary key table in the repository.
2. Creates and updates the preferred records for the incremental data in the existing preferred records table based on the rules defined in the consolidation rules file.

Regenerate Mode

Use the regenerate mode to consolidate the records and regenerate all the preferred records in the existing preferred records table. The regenerate mode uses the primary key table in the repository as input. Before you run the consolidation job in the regenerate mode, ensure that the linked data is loaded into the repository.

The following image shows how the consolidation job processes the input data in the regenerate mode:



The consolidation job performs the following tasks:

1. Reads the linked data from the primary key table in the repository.
2. Regenerates preferred records for all the clusters in the existing preferred records table based on the rules defined in the consolidation rules file.

Running the Consolidation Job

The consolidation job consolidates the linked data and creates preferred records for all the clusters in the repository. You can run the consolidation job in the initial, incremental, or regenerate mode.

To run the consolidation job, run the `run_consolidate.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Initial Mode

Use the following command to run the `run_consolidate.sh` script in the initial mode:

```
run_consolidate.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--consolidate=consolidation_rules_file_name
[--reducer=number_of_reducer_jobs]
[--help]
```

The following table describes the options and arguments that you can specify to run the `run_consolidate.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--consolidate	consolidation_rules_file_name	Absolute path and file name of the consolidation rules file.
--input	input_file_in_HDFS	Absolute path to the directory that contains linked data. If you run the initial clustering job without the `--outputpath` parameter, you can find the linked data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/output/dir/pass-join` If you run the initial clustering job with the `--outputpath` parameter, you can find the linked data in the following directory: `<Output Directory in HDFS>/batch-cluster/output/dir/pass-join`
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The consolidation process uses the working directory to store the library files.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run. Default is 1.
--help		Optional. Displays the supported arguments for the script.

For example, the following command runs the consolidation job in the initial mode:

```
run_consolidate.sh --config=/usr/local/conf/config_big.xml --consolidate=/usr/local/conf/
consolidate_rules.xml --input=/usr/hdfs/workingdir/MDMBDEInitialBatch/
MDMBDE0063_1602999447744334391/output/dir/pass-join --reducer=16 --hdfsdir=/usr/hdfs/
workingdir
```

Incremental Mode

Use the following command to run the `run_consolidate.sh` script in the incremental mode:

```
run_consolidate.sh

--config=configuration_file_name

--input=input_file_in_HDFS

--hdfsdir=working_directory_in_HDFS

--consolidate=consolidation_rules_file_name

--incremental

[--reducer=number_of_reducer_jobs]

[--help]
```

The following table describes the options and arguments that you can specify to run the `run_consolidate.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--consolidate	consolidation_rules_file_name	Absolute path and file name of the consolidation rules file.
--input	input_file_in_HDFS	Absolute path to the directory that contains linked data. If you run the initial clustering job without the `--outputpath` parameter, you can find the linked data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/output/dir/pass-join` If you run the initial clustering job with the `--outputpath` parameter, you can find the linked data in the following directory: `<Output Directory in HDFS>/batch-cluster/output/dir/pass-join`
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The consolidation process uses the working directory to store the library files.
--incremental		Runs the job in the incremental mode. Use the `--incremental` option when you want to update the preferred records table with the incremental linked data. The incremental linked data can be the output of an initial clustering job that you run in the incremental mode without setting the `--consolidate` option.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run to perform consolidation. Default is 1.
--help		Optional. Displays the supported arguments for the script.

For example, the following command runs the consolidation job in the incremental mode:

```
run_consolidate.sh --config=/usr/local/conf/config_big.xml --consolidate=/usr/local/conf/
consolidate_rules.xml --input=/usr/hdfs/workingdir/MDMBDEInitialBatch/
MDMBDE0063_1602999447744334391/output/dir/pass-join --reducer=16 --hdfsdir=/usr/hdfs/
workingdir --incremental
```

Regenerate Mode

Use the following command to run the `run_consolidate.sh` script in the regenerate mode:

```
run_consolidate.sh

--config=configuration_file_name

--hdfsdir=working_directory_in_HDFS

--consolidate=consolidation_rules_file_name

--regenerate

[--reducer=number_of_reducer_jobs]

[--help]
```

The following table describes the options and arguments that you can specify to run the `run_consolidate.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--consolidate	consolidation_rules_file_name	Absolute path and file name of the consolidation rules file.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The consolidation process uses the working directory to store the library files.
--regenerate		Runs the job in the regenerate mode. Use the `--regenerate` option when you want to regenerate the preferred records for all the clusters.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run to perform consolidation. Default is 1.
--help		Optional. Displays the supported arguments for the script.

For example, the following command runs the consolidation job in the regenerate mode:

```
run_consolidate.sh --config=/usr/local/conf/config_big.xml --consolidate=/usr/local/conf/
consolidate_rules.xml --reducer=16 --hdfsdir=/usr/hdfs/workingdir --regenerate
```

Preferred Record Status

After creating a preferred record for a cluster, the consolidation process sets an indicator value that represents the status of the preferred record in the repository. The LMT_DIRTY_IND column in the preferred records table stores the indicator value.

The indicator uses the following values:

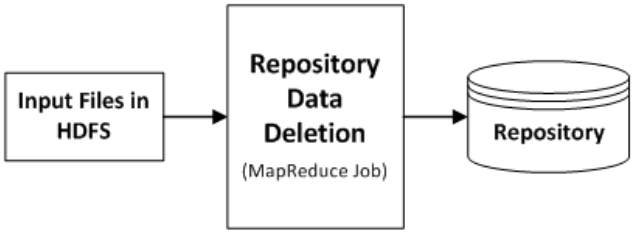
- 0. Indicates that the preferred record is up to date.
- 1. Indicates that the preferred record is dirty and not up to date.

A preferred record might become dirty when one or more records in the cluster have changed. When a preferred record is dirty, run the consolidation job in the incremental or regenerate mode to update the preferred record.

Repository Data Deletion Job

The repository data deletion job matches the input data in HDFS with the repository based on the column that you set as primary key and deletes the matching records from the repository.

The following image shows how the data is deleted from the repository when you run the repository data deletion job:



The repository data deletion job performs the following tasks:

1. Reads the input files from HDFS.
2. Matches the input data with the repository based on the column that you set as primary key.
3. Deletes the matching records from the repository.

Running the Repository Data Deletion Job

The repository data deletion job matches the input data in HDFS with the repository data based on the column that you set as primary key and deletes the matching records from the repository.

To run the repository data deletion job, use the `run_delete.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_delete.sh` script:

```
run_delete.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--forceCopy
[--outputpath=directory_for_output_files]
[--reducer=number_of_reducer_jobs]
[--useIndexId]
```

The following table describes the options and arguments that you can specify to run the `run_delete.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the input files in HDFS.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run. Default is 1.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The repository data deletion job uses the working directory to store the library files.
--useIndexId		Optional. Indicates to use the index identifiers to identify the input data in the repository. You must use the `--useIndexId` option to delete the linked data or the tokenized data.
-- forceCopy		Copies the dependent library files to HDFS. Use this option only when you run the repository data deletion job for the first time.
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.

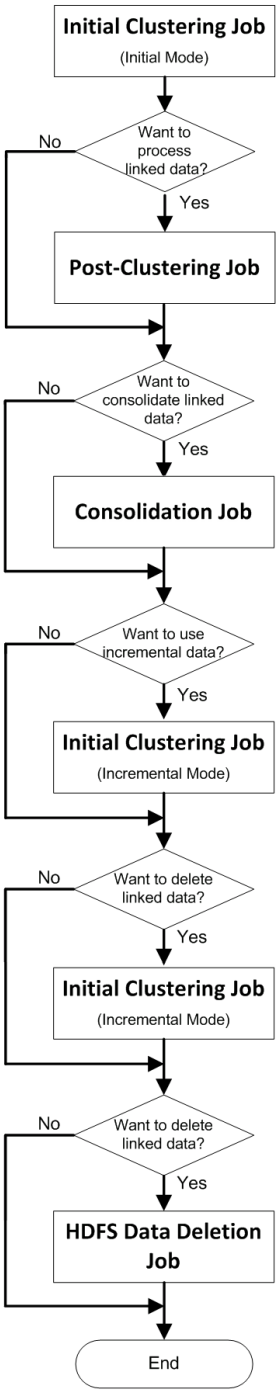
For example, the following command deletes the matching records from the repository:

```
run_delete.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/Source10Million
--reducer=16 --hdfsdir=/usr/hdfs/workingdir --useIndexId --forceCopy --outputpath=/usr/
hdfs/outputfolder
```

Linking Data and Persisting the Linked Data in HDFS

You can link the input data based on the matching rules and consolidate the linked data based on the consolidation rules. You can persist the linked and consolidated data in HDFS.

The following image shows the batch jobs that you can run to link data, consolidate the linked data, and persist the data in HDFS:



To persist the linked and consolidated data in HDFS, perform the following tasks:

1. Run the initial clustering job.

   The job links the input data and creates clusters for the input data in HDFS.

2. If you want to process the output files of the initial clustering job, run the post-clustering job.

   The post-clustering job reads the output files that the initial clustering job creates in HDFS and processes it based on the mode that you set.

3. If you want to consolidate the linked data, run the consolidation job.

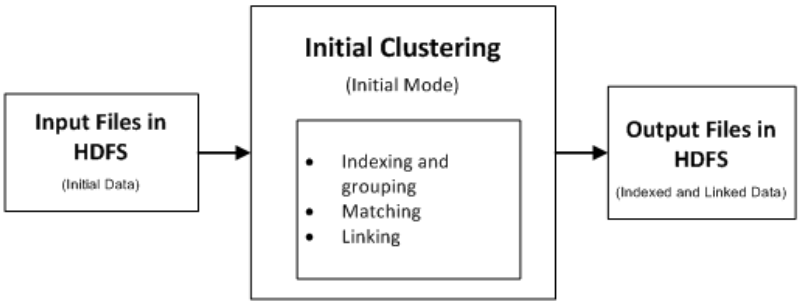
   The consolidation job creates a preferred record for each cluster.
4. To add incremental data to the linked data and link the incremental data, run the initial clustering job in the incremental mode.
5. To delete records from the linked data in HDFS, run the HDFS data deletion job.

Initial Clustering Job

The initial clustering job indexes and links the input data based on the rules in the matching rules file and the parameters in the configuration file. The initial clustering job reads the input data in HDFS and persists the indexed and linked data in HDFS. You can also run the initial clustering job in the incremental mode to incrementally update the indexed and linked data in HDFS.

Initial Mode

Use the initial mode to index and link the data for the first time. The following image shows how the initial clustering job indexes and links the input data when you run the job in the initial mode:



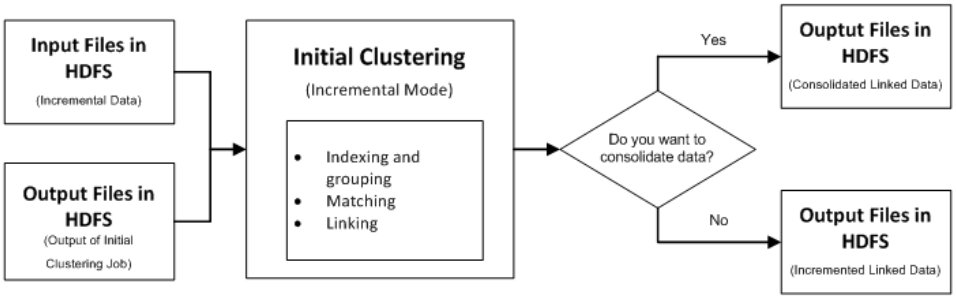
When you run the initial clustering job in the initial mode, the job performs the following tasks:

1. Reads the input files from HDFS.
2. Indexes and groups the input data based on the rules in the matching rules file.
3. Matches the grouped records and links the matched records.
4. Creates clusters for the matched and unmatched records.
5. Writes the indexed and linked records to the output files in HDFS.

   Note: The number of output files depends on the number of reducers that you run.

Incremental Mode

Use the incremental mode to incrementally update the existing indexed and linked data. The following image shows how the initial clustering job indexes and links the incremental data when you run the job in the incremental mode:



When you run the initial clustering job in the incremental mode, the job performs the following tasks:

1. Reads each record from the input files and generates keys and ranges based on the matching rules file.
2. Searches and matches the generated keys with the keys that the previous run of the initial clustering job creates.
3. Links the matched records to an existing cluster and creates clusters for the unmatched records.
4. If you configure the job to consolidate data, the job consolidates the incremental data with the indexed and linked data and writes the consolidated data to HDFS. If you configure the job not to consolidate data, the job indexes and links the incremental data and writes the incremental data to HDFS.

Note: If an input record matches with records that belong to two different clusters, the initial clustering job links the input record to any one of the clusters.

Running the Initial Clustering Job

Use the initial clustering job to read data from the input files and then index and link the input data in HDFS. You can also use the initial clustering job to incrementally update the indexed and linked data with additional data.

To run the initial clustering job, run the `run_genclusters.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Initial Mode

Use the following command to run the `run_genclusters.sh` script in the initial mode:

```
run_genclusters.sh

--config=configuration_file_name

--input=input_file_in_HDFS

--hdfsdir=working_directory_in_HDFS

--rule=matching_rules_file_name

[--outputpath=directory_for_output_files]

[--reducer=number_of_reducers]

[--keeptemp=true|false]

[--compression=true|false]
```

The following table describes the options and the arguments that you can specify to run the `run_genclusters.sh` script in the initial mode:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the input files in HDFS.
--reducer	number_of_reducers	Optional. Number of reducer jobs that you want to run to perform initial clustering. Default is 1.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The initial clustering job uses the working directory to store the output and library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.
--keeptemp	true|false	Optional. Indicates whether to retain the intermediate output tables that the initial clustering job creates. You can use the intermediate output tables for troubleshooting purposes. Set to true to retain the intermediate output tables, and set to false to remove the intermediate output tables after the successful run of the job. Default is false.
--compression	true|false	Optional. Indicates whether to compress the output files that the initial clustering job creates. You can compress the output files to avoid any storage issues. Set to true to compress the output files, and set to false to retain the original size of the output files. Default is false.

For example, the following command runs the initial clustering job in the initial mode:

```
run_genclusters.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/
Source10Million --reducer=16 --outputpath=/usr/hdfs/outputfolder --hdfsdir=/usr/hdfs/
workingdir --rule=/usr/local/conf/matching_rules.xml
```

If you run the initial clustering job without the `--outputpath` parameter, you can find the linked data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/output/dir/pass-join`

Each job generates a unique ID, and you can identify the job ID based on the time stamp of the `<Job ID>` folder.

If you run the initial clustering job with the `--outputpath` parameter, you can find the linked data in the following directory: `<Output Directory in HDFS>/batch-cluster/output/dir/pass-join`

The following sample output of an initial clustering job shows the cluster ID, the field values, match rule name, and the metadata related to the cluster:

```
683e6e41-9174-4a22-b08e-d5d4adc9b2ee        0000000007ERP        3M        3M
Center        St. Paul        USA        ARC SOFT        07_UK_Rule1       00000001ZZB>$$
$$01000004NAH-C$$$QVM$*K$-N?H-C$$-NAH$$$$-
```

Incremental Mode

Use the following command to run the `run_genclusters.sh` script to incrementally update the indexed and linked data with additional data:

```
run_genclusters.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--rule=matching_rules_file_name
--incremental
--clustereddirs=indexed_linked_data_directory
[--reducer=number_of_reducers]
[--outputpath=directory_for_output_files]
[--consolidate]
[--keeptemp=true|false]
[--compression=true|false]
```

The following table describes the options and the arguments that you can specify to run the `run_genclusters.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the input files in HDFS.
--reducer	number_of_reducers	Optional. Number of reducer jobs that you want to run to perform initial clustering. Default is 1.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The initial clustering job uses the working directory to store the output and library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.
--incremental		Runs the initial clustering job in the incremental mode. If you want to incrementally update the indexed and linked data in HDFS, run the job in the incremental mode. By default, the initial clustering job runs in the initial mode.

Option	Argument	Description
--clustereddirs	indexed_linked_data_directory	Absolute path to the directory that contains linked data. If you run the initial clustering job without the `--outputpath` parameter, you can find the linked data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/output/dir/pass-join` If you run the initial clustering job with the `--outputpath` parameter, you can find the linked data in the following directory: `<Output Directory in HDFS>/batch-cluster/output/dir/pass-join`
--consolidate		Consolidates the incremental data with the existing indexed and linked data in HDFS. By default, the initial clustering job indexes and links only the incremental data.
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.
--keeptemp	true|false	Optional. Indicates whether to retain the intermediate output tables that the initial clustering job creates. You can use the intermediate output tables for troubleshooting purposes. Set to true to retain the intermediate output tables, and set to false to remove the intermediate output tables after the successful run of the job. Default is false.
--compression	true|false	Optional. Indicates whether to compress the output files that the initial clustering job creates. You can compress the output files to avoid any storage issues. Set to true to compress the output files, and set to false to retain the original size of the output files. Default is false.

For example, the following command runs the initial clustering job in the incremental mode:

```
run_genclusters.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/
Source10Million --reducer=16 --hdfsdir=/usr/hdfs/workingdir --outputpath=/usr/hdfs/
outputfolder --rule=/usr/local/conf/matching_rules.xml --clustereddirs=/usr/hdfs/
workingdir/batch-cluster/MDMBDRM_931211654144593570/output/dir/pass-join --incremental
```

Post-Clustering Job

Use the post-clustering job to read the output files of an initial clustering job in HDFS and process the input data based on the mode that you configure. The input data can be linked data or poor quality data.

You can run the post-clustering job in one of the following modes:

Skip

Skips the records in the high-volume clusters that contain more than the specified number of records.

Recluster

Re-links the records in the high-volume clusters that contain more than the specified number of records.

Longtail

Decrypts the poor quality records that the initial clustering job identifies to the original input format. You can cleanse the decrypted data, and use it as the input data for the initial clustering job.

Export

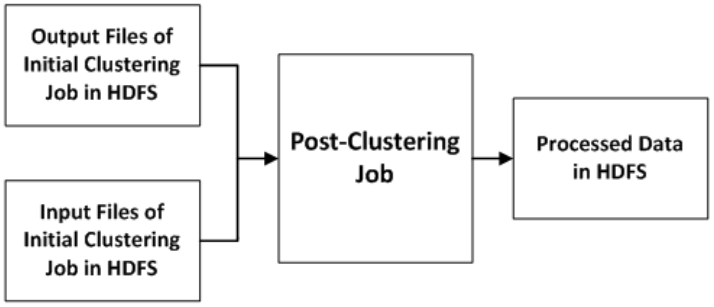
Exports the linked data in the CSV format.

The following image shows how the post-clustering job processes the input data in the skip, recluster, and longtail modes:

The post-clustering job performs the following tasks:

1. Reads the output files of an initial clustering job in HDFS.
2. Processes the input data based on the mode that you configure.
3. Writes the processed data in HDFS.

The following image shows how the post-clustering job processes the input data in the export mode:



The post-clustering job performs the following tasks:

1. Reads the input and output files of an initial clustering job in HDFS.
2. Exports the linked data in the CSV format.

Running the Post-Clustering Job

The post-clustering job uses the output files of an initial clustering job, so ensure that you run the initial clustering job before you run the post-clustering job.

To run the post-clustering job, run the `run_postprocess.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Skip and Recluster Modes

To run the `run_postprocess.sh` script in the skip or recluster mode, use the following command format:

```
run_postprocess.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--rule=matching_rules_file_name
--maxcluster=maximum_number_of_records
--mode=SKIP_LARGE_CLUSTER|RECLUSTER_TRANSITIVE
--threshold=matching_score_to_recluster
[--reducer=number_of_reducer_jobs]
```

The following table describes the options and arguments that you can specify to run the `run_postprocess.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file.
--input	input_file_in_HDFS	Absolute path to the directory that contains linked data. If you run the initial clustering job without the `--outputpath` parameter, you can find the linked data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/output/dir/pass-join` If you run the initial clustering job with the `--outputpath` parameter, you can find the linked data in the following directory: `<Output Directory in HDFS>/batch-cluster/output/dir/pass-join`
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The post-clustering job uses the working directory to store the library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file. The values in the matching rules file override the values in the configuration file.
--maxcluster	maximum_number_of_records	Maximum number of records in a cluster. If the number of records in a cluster exceeds the specified maximum number of records, the cluster becomes a high-volume cluster, and the post-clustering job processes the cluster.
--mode	SKIP_LARGE_CLUSTER| RECLUSTER_TRANSITIVE	Indicates the mode to process the input data. Use one of the following modes: - `SKIP_LARGE_CLUSTER`. Indicates to skip all the records in the high-volume clusters. - `RECLUSTER_TRANSITIVE`. Indicates to re-link all the records in the high-volume clusters.

Option	Argument	Description
--threshold	matching_score_to_recluster	Minimum score required for the records to be part of the same cluster when you run the post-clustering job in the recluster mode. The job creates a separate cluster for each record whose score is less than the threshold value.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run. Default is 1.

For example, the following command runs the post-clustering job in the skip mode:

```
run_postprocess.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/workingdir/
batch-cluster/MDMBDE0063_1602999447744334391/output/dir/pass-join --hdfsdir=/usr/hdfs/
workingdir --rule=/usr/local/conf/matching_rules.xml --maxcluster=1200 --threshold=90 --
mode=SKIP_LARGE_CLUSTER
```

If you run the post-clustering job without the `--outputpath` parameter, you can find the processed data in the following directory: `<Working Directory in HDFS>/ClusterPostProcessing/<Job ID>/output`

If you run the post-clustering job with the `--outputpath` parameter, you can find the processed data in the following directory: `<Output Directory in HDFS>/ClusterPostProcessing/output`

Longtail Mode

The initial clustering job identifies the poor quality data from the input data and loads it to a directory in an encrypted format. The post-clustering job decrypts the poor quality data to the original input format.

To run the `run_postprocess.sh` script in the longtail mode, use the following command format:

```
run_postprocess.sh

--config=configuration_file_name

--input=input_file_in_HDFS

--hdfsdir=working_directory_in_HDFS

--rule=matching_rules_file_name

--mode=LONGTAIL_CLUSTERS

[--reducer=number_of_reducer_jobs]
```

The following table describes the options and arguments that you can specify to run the `run_postprocess.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file.
--input	input_file_in_HDFS	Absolute path to the directory that contains the poor quality data. If you run the initial clustering job without the `--outputpath` parameter, you can find the poor quality data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/poorqualitydata` If you run the initial clustering job with the `--outputpath` parameter, you can find the poor quality data in the following directory: `<Output Directory in HDFS>/batch-cluster/poorqualitydata`
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The post-clustering job uses the working directory to store the library files.

Option	Argument	Description
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file. The values in the matching rules file override the values in the configuration file.
--mode	LONGTAIL_CLUSTERS	Indicates that the job processes the poor quality data.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run. Default is 1.

For example, the following command runs the post-clustering job in the longtail mode:

```
run_postprocess.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/workingdir/
batch-cluster/MDMBDE0063_1602999447744334391/poorqualitydata --hdfsdir=/usr/hdfs/
workingdir --rule=/usr/local/conf/matching_rules.xml --mode=LONGTAIL_CLUSTERS
```

If you run the post-clustering job without the `--outputpath` parameter, you can find the decrypted data in the following directory: `<Working Directory in HDFS>/ClusterPostProcessing/<Job ID>/output`

If you run the post-clustering job with the `--outputpath` parameter, you can find the decrypted data in the following directory: `<Output Directory in HDFS>/ClusterPostProcessing/output`

Export Mode

To run the `run_postprocess.sh` script in the export mode, use the following command format:

```
run_postprocess.sh

--config=configuration_file_name

--input=input_file_in_HDFS

--matchinput=match_output_in_HDFS

--hdfsdir=working_directory_in_HDFS

--rule=matching_rules_file_name

--mode=CSV_OUTPUT

[--reducer=number_of_reducer_jobs]
```

The following table describes the options and arguments that you can specify to run the `run_postprocess.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file.
--input	input_file_in_HDFS	Absolute path to the directory that contains the input files of the initial clustering job.
--matchinput	match_output_in_HDFS	Absolute path to the directory that contains the linked data. If you run the initial clustering job without the `--outputpath` parameter, you can find the linked data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/match/dir` If you run the initial clustering job with the `--outputpath` parameter, you can find the linked data in the following directory: `<Output Directory in HDFS>/batch-cluster/match/dir`

Option	Argument	Description
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The post-clustering job uses the working directory to store the library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file. The values in the matching rules file override the values in the configuration file.
--mode	CSV_OUTPUT	Indicates to export the input data in the CSV format.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run. Default is 1.

For example, the following command runs the post-clustering job in the export mode:

```
run_postprocess.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/
Source10Million --matchinput=/usr/hdfs/workingdir/batch-cluster/
MDMBDE0063_1602999447744334391/match/dir/ --hdfsdir=/usr/hdfs/workingdir --rule=/usr/
local/conf/matching_rules.xml --mode=CSV_OUTPUT
```

If you run the post-clustering job without the `--outputpath` parameter, you can find the CSV files in the following directory: `<Working Directory in HDFS>/ClusterPostProcessing/<Job ID>/output-match`

If you run the post-clustering job with the `--outputpath` parameter, you can find the CSV files in the following directory: `<Output Directory in HDFS>/ClusterPostProcessing/output-match`

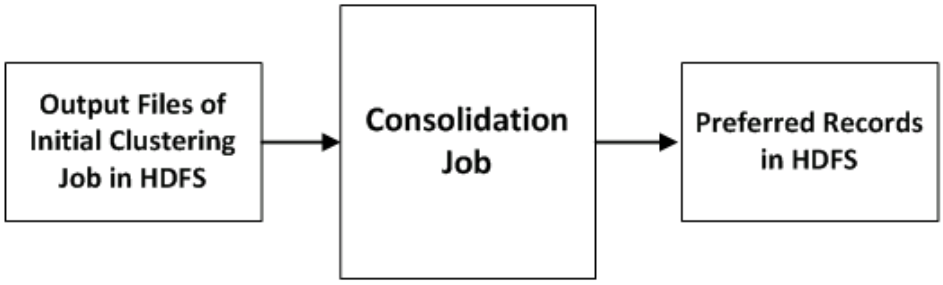
Consolidation Job

Use the consolidation job to consolidate the linked data and create a preferred record for each cluster in HDFS. The consolidation job uses the output files of an initial clustering job in HDFS as input. The consolidation job creates the preferred records based on the rules defined in the consolidation rules file.

For incremental data, use the output files of an initial clustering job that you run in the incremental mode with the `--consolidate` option as input for the consolidation job.

Note: The consolidation process ignores any null values.

The following image shows how the consolidation job processes the input data:



The consolidation job performs the following tasks:

1. Reads the output files of an initial clustering job in HDFS.
2. Creates preferred records for all the clusters based on the rules defined in the consolidation rules file.

Running the Consolidation Job

The consolidation job consolidates the linked data and creates preferred records for all the clusters in HDFS.

To run the consolidation job, run the `run_consolidate.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_consolidate.sh` script:

```
run_consolidate.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--consolidate=consolidation_rules_file_name
--hdfsOnly
[--reducer=number_of_reducer_jobs]
[--help]
```

The following table describes the options and arguments that you can specify to run the `run_consolidate.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--consolidate	consolidation_rules_file_name	Absolute path and file name of the consolidation rules file.
--input	input_file_in_HDFS	Absolute path to the directory that contains linked data. If you run the initial clustering job without the `--outputpath` parameter, you can find the linked data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/output/dir/pass-join` If you run the initial clustering job with the `--outputpath` parameter, you can find the linked data in the following directory: `<Output Directory in HDFS>/batch-cluster/output/dir/pass-join` For incremental data, use the output files of an initial clustering job that you run in the incremental mode with the `--consolidate` option.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The consolidation process uses the working directory to store the library files.
--hdfsOnly		Indicates to persist the preferred records in HDFS.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run. Default is 1.
--help		Optional. Displays the supported arguments for the script.

For example, the following command runs the consolidation job in the initial mode:

```
run_consolidate.sh --config=/usr/local/conf/config_big.xml --consolidate=/usr/local/conf/
consolidate_rules.xml --input=/usr/hdfs/workingdir/MDMBDEInitialBatch/
MDMBDE0063_1602999447744334391/output/dir/pass-join --reducer=16 --hdfsdir=/usr/hdfs/
workingdir --hdfsOnly
```

If you run the consolidation job without the `--outputpath` parameter, you can find the preferred records in the following directory: `<Working Directory in HDFS>/batch-consolidation/<Job ID>/consolidation-output`

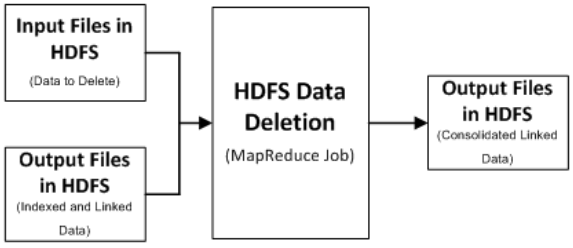
Each job generates a unique ID, and you can identify the job ID based on the time stamp of the `<Job ID>` folder.

If you run the consolidation job with the `--outputpath` parameter, you can find the preferred records in the following directory: `<Output Directory in HDFS>/batch-consolidation/<Job ID>/consolidation-output`

HDFS Data Deletion Job

The HDFS data deletion job matches the input data in HDFS with the linked or tokenized data in HDFS that the initial clustering or HDFS tokenization job creates. The job deletes the matching data and writes the consolidated data to an output directory in HDFS.

The following image shows how the HDFS data deletion job deletes the input data:



The HDFS data deletion job performs the following tasks:

1. Reads the input files from HDFS.
2. Matches the input data with the linked or tokenized data in HDFS.
3. Deletes the matching records from the linked or tokenized data in HDFS.

Running the HDFS Data Deletion Job

The HDFS data deletion job matches the input data in HDFS with the linked or tokenized data in HDFS, deletes the matching data, and writes the consolidated data to an output directory in HDFS.

To run the HDFS data deletion job, use the `run_clusterdel.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_clusterdel.sh` script:

```
run_clusterdel.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--clustereddirs=indexed_linked_data_directory
[--reducer=number_of_reducer_jobs]
[--outputpath=directory_for_output_files]
```

The following table describes the options and arguments that you can specify to run the `run_clusterdel.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the input files in HDFS.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run. Default is 1.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The HDFS data deletion job uses the working directory to store the library files.
--clustereddirs	indexed_linked_data_directory	Absolute path to the output files that an initial clustering job or a HDFS tokenization job creates. If you run the initial clustering job without the `--outputpath` parameter, you can find the linked data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/output/dir/pass-join` If you run the initial clustering job with the `--outputpath` parameter, you can find the linked data in the following directory: `<Output Directory in HDFS>/batch-cluster/output/dir/pass-join` If you run the HDFS tokenization job without the `--outputpath` parameter, you can find the tokenized data in the following directory: `<Working Directory in HDFS>/batch-tokenize/<Job ID>/tokenize` If you run the HDFS tokenization job with the `--outputpath` parameter, you can find the tokenized data in the following directory: `<Output Directory in HDFS>/batch-tokenize/tokenize`
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.

For example, the following command deletes the matching records from the linked data in HDFS and writes the consolidated data to the output files in HDFS:

```
run_clusterdel.sh  --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/
Source10Million --reducer=16 --hdfsdir=/usr/hdfs/workingdir --clustereddirs=/usr/hdfs/
workingdir/batch-cluster/MDMBDRM_931211654144593570/output/dir/pass-join
```

If you run the HDFS data deletion job without the `--outputpath` parameter, you can find the consolidated data in the following directory: `<Working Directory in HDFS>/BDRMClusterDel/<Job ID>/output/dir/consolidated-dir`

If you run the HDFS data deletion job with the `--outputpath` parameter, you can find the consolidated data in the following directory: `<Output Directory in HDFS>/BDRMClusterDel/output/dir/consolidated-dir`

CHAPTER 3

Tokenizing Batch Data

This chapter includes the following topics:

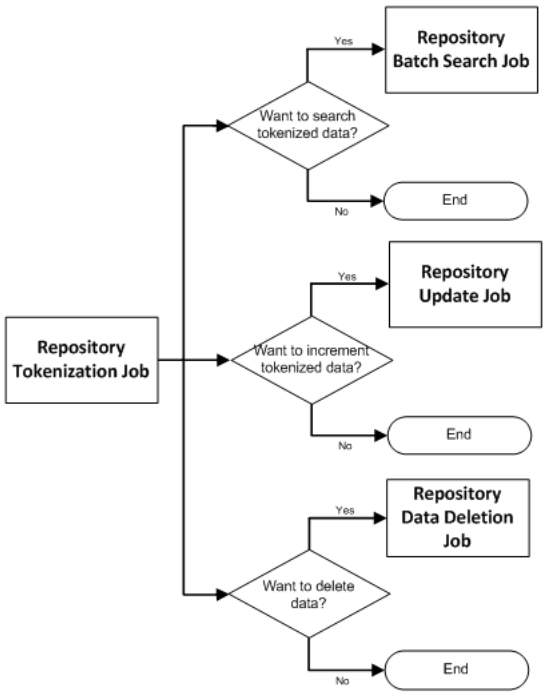
Tokenizing Batch Data Overview

Use the Informatica MDM - Relate 360 batch jobs to tokenize the batch data. Relate 360 reads the input data from HDFS, adds a fuzzy token for each input record, and writes the tokenized data to HDFS. You can persist the tokenized data in HDFS or in a repository.

Tokenizing Data and Persisting the Tokenized Data in a Repository

You can create fuzzy tokens for the input data based on the matching rules and persist the tokenized data in a repository so that you can perform searches on the tokenized data.

The following image shows the batch jobs that you can run to tokenize data and persist the tokenized data in a repository:



To persist the tokenized data in a repository, perform the following tasks:

1. Run the repository tokenization job.

   The job creates fuzzy tokens for the input data in HDFS and loads the tokenized data into the repository.

   Alternatively, you can perform the following tasks:

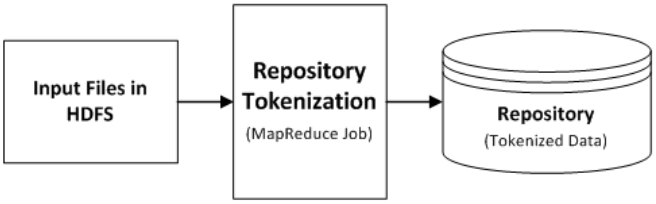
   1. Run the HDFS tokenization job to create fuzzy tokens for the input data.
   2. Optionally, run the region splitter job to analyze the input tokenized data and identify the split points for all the regions in the repository.
   3. Run the load clustering job to load the tokenized data into the repository.
2. To search for records in the repository, run the repository batch search job.
3. To add an incremental data to the tokenized data in the repository, run the repository update job.
4. To delete records from the tokenized data in the repository, run the repository data deletion job.

Repository Tokenization Job

The repository tokenization job creates match tokens for the input data in HDFS and loads the tokenized data into the repository. The repository tokenization job uses the columns that you configure as index fields to generate the match tokens.

The repository tokenization job performs the tasks of a HDFS tokenization job and a load clustering job. The repository tokenization job reads the input data in HDFS, creates tokenized data in HDFS, and loads the tokenized data into the repository. The tokenized data includes input records and their match tokens. You must use the repository update job to incrementally update the tokenized data in the repository.

The following image shows how the repository tokenization job creates match tokens for the input data and loads the tokenized data into a repository:



When you run the repository tokenization job, the job performs the following tasks:

1. Reads the input files in HDFS.
2. Generates match tokens for the input data.
3. Writes the tokenized data to the output files in HDFS.
   The tokenized data includes input records and their match tokens.

   Note: The number of output files depends on the number of reducers that you run.
4. Loads the tokenized data into the repository.

Running the Repository Tokenization Job

The repository tokenization job reads the input data from HDFS, creates match tokens for the input data in HDFS, and loads the tokenized data into the repository.

To run the repository tokenization job, run the `run_tokenloader.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_tokenloader.sh` script:

```
run_tokenloader.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--rule=matching_rules_file_name
--tmpdir=temporary_working_directory
[--outputpath=directory_for_output_files]
[--reducer=number_of_reducer_jobs]
```

The following table describes the options and arguments that you can specify to run the `run_tokenloader.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create. In the configuration file, if you set the `StoreAllFields` parameter to false, the repository does not persist all the columns but persists only the columns that you use to index the input data. If you want to persist all the columns in the repository, ensure that you set the `StoreAllFields` parameter to true in the configuration file before you tokenize the input data.
--input	input_file_in_HDFS	Absolute path to the input files in HDFS.

Option	Argument	Description
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run. Default is 1.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The repository tokenization process uses the working directory to store the library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create. The values in the matching rules file override the values in the configuration file.
--tmpdir	temporary_working_directory	Absolute path to a temporary directory to which you have write permission in the local file system. The repository tokenization job uses the directory to store the intermediate files.
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.

For example, the following command runs the repository tokenization job:

```
run_tokenloader.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/
GenerateTokens --reducer=16 --hdfsdir=/usr/hdfs/workingdir --rule=/usr/local/conf/
matching_rules.xml --tmpdir=/tmp
```

The following sample output of the repository tokenization job shows index ID, token, and the values from the column family:

```
ROW                                              COLUMN
+CELL

 00KCKSHX$$    SALESFORCE0000000 column=aml_link_columns:CLUSTERNUMBER,
timestamp=1454406691384, value=f9febe82-8f55-4b7e-98de-a11290ae2807   066
 00KCKSHX$$    SALESFORCE0000000 column=aml_link_columns:LMT_MATCHED_PK,
timestamp=1454406691384, value=                                        066
 00KCKSHX$$    SALESFORCE0000000 column=aml_link_columns:LMT_MATCHED_RECORD_SOURCE,
timestamp=1454406691384, value=                              066
 00KCKSHX$$    SALESFORCE0000000 column=aml_link_columns:LMT_MATCHED_SCORE,
timestamp=1454406691384, value=0                                    066
 00KCKSHX$$    SALESFORCE0000000 column=aml_link_columns:LMT_SOURCE_NAME,
timestamp=1454406691384, value=                                      066
 00KCKSHX$$    SALESFORCE0000000 column=aml_link_columns:NAME, timestamp=1454406691384,
value=Abbott Laboratories                          066
 00KCKSHX$$    SALESFORCE0000000 column=aml_link_columns:ROWID, timestamp=1454406691384,
value=0000000066                                   066
```

HDFS Tokenization Job

The HDFS tokenization job creates fuzzy tokens for the input data based on the rules in the matching rules file and the parameters in the configuration file. The HDFS tokenization job uses the columns that you configure as index fields to generate the fuzzy tokens.

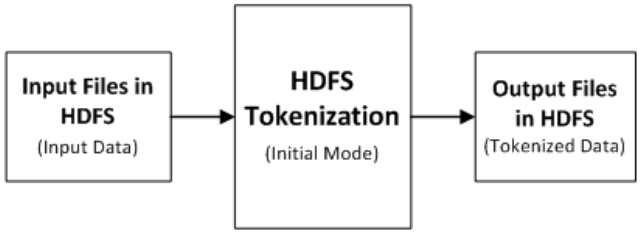
The HDFS tokenization job reads the input data in HDFS and creates the output files that contain tokenized data in HDFS. The tokenized data includes input records and the fuzzy tokens of the input records. You can

also run the HDFS tokenization job in the incremental mode to update the tokenized data that a HDFS tokenization job creates.

Initial Mode

Use the initial mode to create fuzzy tokens for the first time.

The following image shows how the HDFS tokenization job creates fuzzy tokens for the input data in the initial mode:



When you run the HDFS tokenization job in the initial mode, the job performs the following tasks:

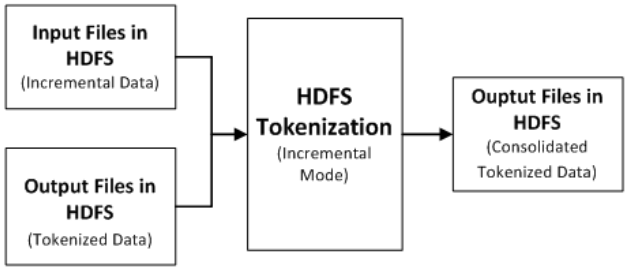
1. Reads the input files in HDFS.
2. Generates fuzzy tokens for the input data.
3. Writes the tokenized data to the output files in HDFS.
   The tokenized data includes input records and the fuzzy tokens of the input records.

   Note: The number of output files depends on the number of reducers that you run.

Incremental Mode

Use the incremental mode to update the tokenized data that a HDFS tokenization job creates.

The following image shows how the HDFS tokenization job updates the tokenized data in the incremental mode:



When you run the HDFS tokenization job in the incremental mode, the job performs the following tasks:

1. Reads the input files in HDFS.
2. Reads the output files of a HDFS tokenization job in HDFS.
3. Merges the input data from the input files and the tokenized data from the HDFS tokenization job.
4. Identifies the duplicate records in the tokenized data based on the fuzzy tokens and updates them with the records from the input files.
5. Generates fuzzy tokens for the input data and adds them to the merged data.
6. Writes the merged data to the output files in HDFS.

Running the HDFS Tokenization Job

Use the HDFS tokenization job to read data from the input files in HDFS, create match tokens for the input data, and write the tokenized data to the output files in HDFS.

To run the HDFS tokenization job, run the `run_tokenizer.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Initial Mode

Use the following command to run the `run_tokenizer.sh` script in the initial mode:

```
run_tokenizer.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--rule=matching_rules_file_name
[--outputpath=directory_for_output_files]
[--reducer=number_of_reducers]
```

The following table describes the options and the arguments that you can specify to run the `run_tokenizer.sh` script in the initial mode:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create. In the configuration file, if you set the `StoreAllFields` parameter to false, the output files of the job does not include all the columns but includes only the columns that you use to index the input data. If you want to include all the columns in the output files, ensure that you set the `StoreAllFields` parameter to true in the configuration file before you run the job.
--input	input_file_in_HDFS	Absolute path to the input files in HDFS.
--reducer	number_of_reducers	Optional. Number of reducer jobs that you want to run. Default is 1.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The HDFS tokenization job uses the working directory to store the output and library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.

For example, the following command runs the HDFS tokenization job in the initial mode:

```
run_tokenizer.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/
GenerateTokens --reducer=16 --hdfsdir=/usr/hdfs/workingdir --rule=/usr/local/conf/
matching_rules.xml
```

If you run the HDFS tokenization job without the `--outputpath` parameter, you can find the tokenized data in the following directory: `<Working Directory in HDFS>/batch-tokenize/<Job ID>/tokenize`

Each job generates a unique ID, and you can identify the job ID based on the time stamp of the `<Job ID>` folder.

If you run the HDFS tokenization job with the `--outputpath` parameter, you can find the tokenized data in the following directory: `<Output Directory in HDFS>/batch-tokenize/tokenize`

The following sample output of the HDFS tokenization job shows the cluster ID, the field values, and the token for an input record:

```
a3ebff6d-c578-4ace-b6d1-2805788d78a6        0000000007ERP        3M        3M
Center        St. Paul        USA        ARC SOFT        00000001ZZB>$$$$01000004N?H-C$$-
NAH$$$$-NAH-C$$$QVM$*K$-
```

Incremental Mode

Use the following command to run the `run_tokenizer.sh` script in the incremental mode:

```
run_tokenizer.sh

--config=configuration_file_name

--input=input_file_in_HDFS

--hdfsdir=working_directory_in_HDFS

--rule=matching_rules_file_name

--incremental

--clustereddirs=tokenize_output_data_directory

[--outputpath=directory_for_output_files]

[--reducer=number_of_reducers]
```

The following table describes the options and the arguments that you can specify to run the `run_tokenizer.sh` script in the initial mode:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the input files in HDFS.
--reducer	number_of_reducers	Optional. Number of reducer jobs that you want to run. Default is 1.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The HDFS tokenization job uses the working directory to store the output and library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.
--incremental		Runs the HDFS tokenization job in the incremental mode. If you want to incrementally update the output files of a HDFS tokenization job, run the job in the incremental mode. By default, the HDFS tokenization job runs in the initial mode.

Option	Argument	Description
--clustereddirs	tokenize_output_data_directory	Absolute path to the directory that contains tokenized data. If you run the HDFS tokenization job without the `--outputpath` parameter, you can find the tokenized data in the following directory: `<Working Directory in HDFS>/batch-tokenize/<Job ID>/tokenize`If you run the HDFS tokenization job with the `--outputpath` parameter, you can find the tokenized data in the following directory: `<Output Directory in HDFS>/batch-tokenize/tokenize`
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.

For example, the following command runs the HDFS tokenization job in the incremental mode:

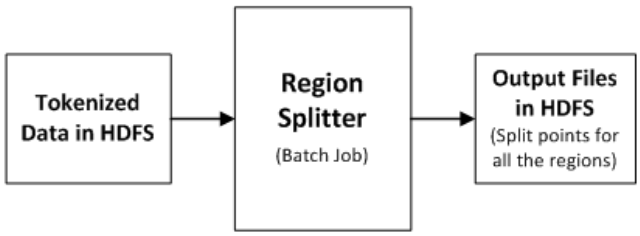
```
run_tokenizer.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/
GenerateTokens --reducer=16 --hdfsdir=/usr/hdfs/workingdir --rule=/usr/local/conf/
matching_rules.xml --clustereddirs=/usr/hdfs/workingdir/batch-tokenize/
MDMBDRM_931211654144593570/tokenize --incremental
```

Region Splitter Job

Use the region splitter job to analyze the input tokenized data and identify the split points to uniformly distribute the tokenized data across all the regions in the repository. The uniform distribution of the tokenized data optimally utilizes the resources and improves the search performance.

A load clustering job uses the output files of a region splitter job to distribute the linked data. Run the region splitter job before you run the load clustering job for the first time.

The following image shows how the region splitter job identifies the split points based on the input data:



The region splitter job performs the following tasks:

1. Reads the tokenized data in HDFS.
2. Identifies the split points for the number of regions that you specify.

Running the Region Splitter Job

The region splitter job identifies the split points for the input tokenized data to uniformly distribute the tokenized data across all the regions.

To run the region splitter job, run the `run_hbase_region_analysis.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_hbase_region_analysis.sh` script:

```
run_hbase_region_analysis.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--rule=matching_rules_file_name
--regions=number_of_regions
[--reducer=number_of_reducer_jobs]
[--outputpath=directory_for_output_files]
```

The following table describes the options and arguments that you can specify to run the `run_hbase_region_analysis.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the directory that contains tokenized data. If you run the HDFS tokenization job without the `--outputpath` parameter, you can find the tokenized data in the following directory: `<Working Directory in HDFS>/batch-tokenize/<Job ID>/tokenize` If you run the HDFS tokenization job with the `--outputpath` parameter, you can find the tokenized data in the following directory: `<Output Directory in HDFS>/batch-tokenize/tokenize`
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The region splitter job uses the working directory to store the library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create. The values in the matching rules file override the values in the configuration file.
--regions	number_of_regions	Number of regions that you want to use for the input data. The optimal number of regions depends on your environment and resources. For more information about regions and split points, see the repository documentation.

Option	Argument	Description
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run. Default is 1.
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.

For example, the following command runs the region splitter job:

```
run_hbase_region_analysis.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/
workingdir/MDMBDRMInitialBatch/MDMBDE0063_1602999447744334391/output/dir/pass-join --
hdfsdir=/usr/hdfs/workingdir --rule=/usr/local/conf/matching_rules.xml --regions=14
```

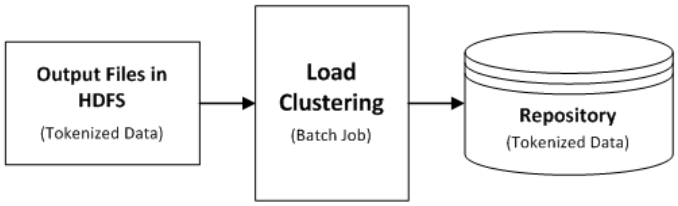
If you run the region splitter job without the `--outputpath` parameter, you can find the output files in the following directory: `<Working Directory in HDFS>/MDMBDRMRegionAnalysis/<Job ID>`

If you run the region splitter job with the `--outputpath` parameter, you can find the output files in the following directory: `<Output Directory in HDFS>/MDMBDRMRegionAnalysis`

Load Clustering Job

The load clustering job loads the output files of a HDFS tokenization job from HDFS into the repository. Before you run the load clustering job, you can run the region splitter job to identify the split points for the input data.

The following image shows how the load clustering job loads the data into the repository:



The load clustering job performs the following tasks:

1. Reads the data from the output files of a HDFS tokenization job in HDFS.
2. Loads the tokenized data into the repository.

Running the Load Clustering Job

The load clustering job loads the tokenized data from HDFS into the repository.

To run the load clustering job, run the `run_clusterload.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_clusterload.sh` script:

```
run_clusterload.sh

--config=configuration_file_name
```

```
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--rule=matching_rules_file_name
[--reducer=number_of_reducer_jobs]
[--hbaseregionsplitpath=output_directory_of_region_splitter_job]
[--outputpath=directory_for_output_files]
```

The following table describes the options and arguments that you can specify to run the `run_clusterload.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the directory that contains tokenized data. If you run the HDFS tokenization job without the `--outputpath` parameter, you can find the tokenized data in the following directory: `<Working Directory in HDFS>/batch-tokenize/<Job ID>/tokenize` If you run the HDFS tokenization job with the `--outputpath` parameter, you can find the tokenized data in the following directory: `<Output Directory in HDFS>/batch-tokenize/tokenize`
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run to perform load clustering. Default is 1.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The load clustering process uses the working directory to store the library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create. The values in the matching rules file override the values in the configuration file.

Option	Argument	Description
--hbaseregionsplitpath	output_directory_of_region_splitter_job	Optional. Absolute path to the output files of the region splitter job. If you run the region splitter job without the `--outputpath` parameter, you can find the output files in the following directory: `<Working Directory in HDFS>/MDMBDRMRegionAnalysis/<Job ID>` If you run the region splitter job with the `--outputpath` parameter, you can find the output files in the following directory: `<Output Directory in HDFS>/MDMBDRMRegionAnalysis` By default, the load clustering job randomly distributes the tokenized data and might result in inconsistent distribution of data across the regions.
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.

For example, the following command runs the load clustering job:

```
run_clusterload.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/workingdir/
batch-tokenize/MDMBDE0063_1602999447744334391/tokenize --reducer=16 --hdfsdir=/usr/hdfs/
workingdir --rule=/usr/local/conf/matching_rules.xml --hbaseregionsplitpath=/usr/hdfs/
workingdir/MDMBDRMRegionAnalysis/MDMBDRM0063_8862443019807752334
```

The following sample output of the load clustering job shows index ID, fuzzy keys, and the values from the column family:

```
ROW                                                  COLUMN
+CELL

 00KCKSHX$$    SALESFO column=aml_link_columns:CLUSTERNUMBER, timestamp=1454327424272,
value=5fb1e9b8-1b51-47a3-bd45-d885bdc6bbcf   RCE0000000066
 00KCKSHX$$    SALESFO column=aml_link_columns:LMT_MATCHED_PK, timestamp=1454327424272,
value=                                     RCE0000000066
 00KCKSHX$$    SALESFO column=aml_link_columns:LMT_MATCHED_RECORD_SOURCE,
timestamp=1454327424272, value=                          RCE0000000066
 00KCKSHX$$    SALESFO column=aml_link_columns:LMT_MATCHED_SCORE,
timestamp=1454327424272, value=0                                  RCE0000000066
 00KCKSHX$$    SALESFO column=aml_link_columns:LMT_SOURCE_NAME, timestamp=1454327424272,
value=                    SALESFORCE     RCE0000000066
 00KCKSHX$$    SALESFO column=aml_link_columns:NAME, timestamp=1454327424272,
value=Abbott Laboratories                             RCE0000000066
 00KCKSHX$$    SALESFO column=aml_link_columns:ROWID, timestamp=1454327424272,
value=0000000066
```

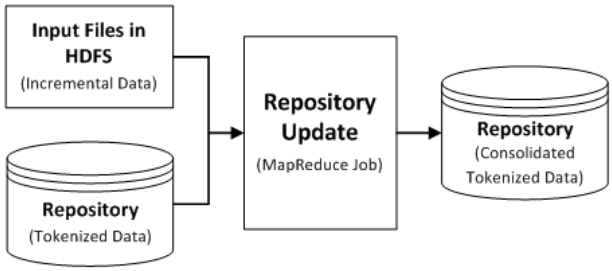
Repository Update Job

The repository update job updates the tokenized data in the repository with the input data and creates match tokens for the input data in the repository. During the update process, the repository update job matches the

input data with the repository data, deletes the matching records from the repository, and adds the input data to the repository.

Note: Before you run the repository update job, ensure that the repository contains tokenized data.

The following image shows how the repository update job updates the repository data:



The repository update job performs the following tasks:

1. Reads the input files in HDFS.
2. Matches the input data with the repository data.
3. Deletes the matching records from the repository.
4. Generates match tokens for the input data.
5. Loads the input data with the match tokens to the repository.

Running the Repository Update Job

The repository update job updates the repository data with the input data and creates match tokens for the input data in the repository.

To run the repository update job, run the `run_updatesync.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_updatesync.sh` script:

```
run_updatesync.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--rule=matching_rules_file_name
[--outputpath=directory_for_output_files]
[--reducer=number_of_reducer_jobs]
```

The following table describes the options and arguments that you can specify to run the `run_updatesync.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the input files in HDFS.

Option	Argument	Description
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run to update the repository. Default is 1.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The repository update job uses the working directory to store the library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create. The values in the matching rules file override the values in the configuration file.
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.

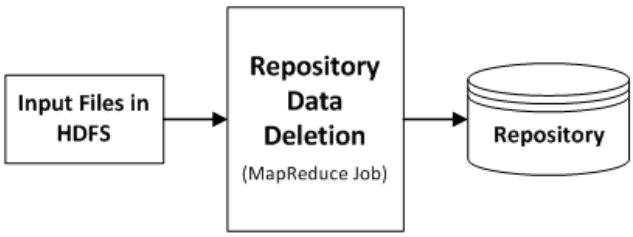
For example, the following command runs the repository update job:

```
run_updatesync.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/
IncrementalData --reducer=16 --hdfsdir=/usr/hdfs/workingdir --rule=/usr/local/conf/
matching_rules.xml
```

Repository Data Deletion Job

The repository data deletion job matches the input data in HDFS with the repository based on the column that you set as primary key and deletes the matching records from the repository.

The following image shows how the data is deleted from the repository when you run the repository data deletion job:



The repository data deletion job performs the following tasks:

1. Reads the input files from HDFS.
2. Matches the input data with the repository based on the column that you set as primary key.
3. Deletes the matching records from the repository.

Running the Repository Data Deletion Job

The repository data deletion job matches the input data in HDFS with the repository data based on the column that you set as primary key and deletes the matching records from the repository.

To run the repository data deletion job, use the `run_delete.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_delete.sh` script:

```
run_delete.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--forceCopy
[--outputpath=directory_for_output_files]
[--reducer=number_of_reducer_jobs]
[--useIndexId]
```

The following table describes the options and arguments that you can specify to run the `run_delete.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the input files in HDFS.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run. Default is 1.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The repository data deletion job uses the working directory to store the library files.
--useIndexId		Optional. Indicates to use the index identifiers to identify the input data in the repository. You must use the `--useIndexId` option to delete the linked data or the tokenized data.
-- forceCopy		Copies the dependent library files to HDFS. Use this option only when you run the repository data deletion job for the first time.
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.

For example, the following command deletes the matching records from the repository:

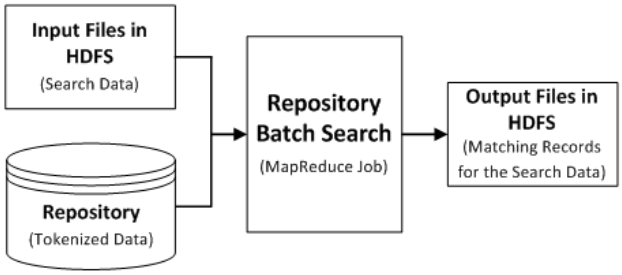
```
run_delete.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/Source10Million
--reducer=16 --hdfsdir=/usr/hdfs/workingdir --useIndexId --forceCopy --outputpath=/usr/
hdfs/outputfolder
```

Repository Batch Search Job

The repository batch search job identifies the matching records for the input data in the repository based on the match tokens. The repository batch search job reads the input data in HDFS and creates the output files that contain the matching records for the input data in HDFS.

The repository batch search job requires the repository to contain all the columns with the match tokens. You must set the `StoreAllFields` parameter to true in the configuration file when you tokenize the input data to include all the columns.

The following image shows how the repository batch search job searches for the matching records in the repository:



When you run the repository batch search job, the job performs the following tasks:

1. Reads the input files in HDFS.
2. Compares the input data against the tokenized data in the repository based on the match tokens.
3. Writes the matching records for the input data to the output files in HDFS.
   The number of output files depends on the number of reducers that you run.

Running the Repository Batch Search Job

Use the repository batch search job to identify the matching records for the input data in the repository based on the match tokens.

To run the repository batch search job, run the `run_dbrelate.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_dbrelate.sh` script:

```
run_dbrelate.sh

--config=configuration_file_name

--input=input_file_in_HDFS

--hdfsdir=working_directory_in_HDFS

--rule=matching_rules_file_name

--tmpdir=temporary_working_directory

[--reducer=number_of_reducers]

[--outputpath=directory_for_output_files]
```

The following table describes the options and the arguments that you can specify to run the `run_dbrelate.sh` script in the initial mode:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the input files in HDFS.
--reducer	number_of_reducers	Optional. Number of reducer jobs that you want to run. Default is 1.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The repository batch search job uses the working directory to store the output and library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.
--tmpdir	temporary_working_directory	Absolute path to a temporary directory to which you have write permission in the local file system. The repository tokenization job uses the directory to store the intermediate files.
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.

For example, the following command runs the repository batch search job:

```
run_dbrelate.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/SearchRecords
--reducer=16 --hdfsdir=/usr/hdfs/workingdir --rule=/usr/local/conf/matching_rules.xml
```

If you run the repository batch search job without the `--outputpath` parameter, you can find the matching records in the following directory: `<Working Directory in HDFS>/BDRMRelate/<Job ID>/relate-output/dir`

If you run the repository batch search job with the `--outputpath` parameter, you can find the matching records in the following directory: `<Output Directory in HDFS>/BDRMRelate/relate-output/dir`

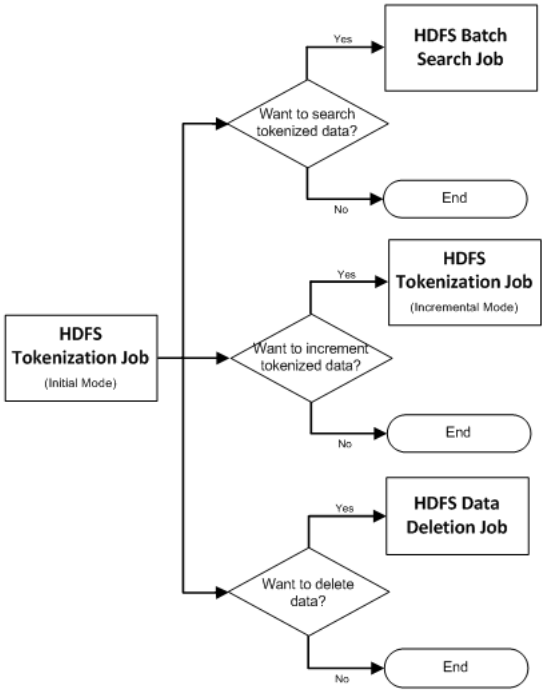
The following sample output of a repository batch search job shows the search record and the matching record with the matching score:

```
*************** a18f250f-4e2f-4b4e-b014-837b428d94a3     ***************
100AML.ADDR    a18f250f-4e2f-4b4e-b014-837b428d94a3    0000009514    ERPWright Medical
Technology    5677 Airline Road    Arlington    USA    ARC SOFT
---            a18f250f-4e2f-4b4e-b014-837b428d94a3    0000009514    ERPWright Medical
Technology    5677 Airline Road    Arlington    USA    ARC SOFT
```

Tokenizing Data and Persisting the Tokenized Data in HDFS

You can create fuzzy tokens for the input data based on the matching rules and persist the tokenized data in HDFS so that you can perform searches on the tokenized data.

The following image shows the batch jobs that you can run to tokenize data and persist the tokenized data in HDFS:



To persist the tokenized data in HDFS, perform the following tasks:

1. Run the HDFS tokenization job.

   The job creates fuzzy tokens for the input data in HDFS.
2. To search for records, run the HDFS batch search job.
3. To add an incremental data to the tokenized data in HDFS, run the HDFS tokenization job in the incremental mode.
4. To delete records from the tokenized data in HDFS, run the HDFS data deletion job.

HDFS Tokenization Job

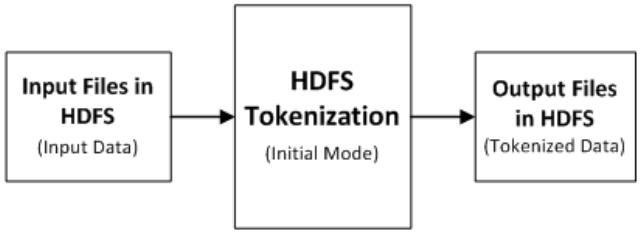
The HDFS tokenization job creates fuzzy tokens for the input data based on the rules in the matching rules file and the parameters in the configuration file. The HDFS tokenization job uses the columns that you configure as index fields to generate the fuzzy tokens.

The HDFS tokenization job reads the input data in HDFS and creates the output files that contain tokenized data in HDFS. The tokenized data includes input records and the fuzzy tokens of the input records. You can also run the HDFS tokenization job in the incremental mode to update the tokenized data that a HDFS tokenization job creates.

Initial Mode

Use the initial mode to create fuzzy tokens for the first time.

The following image shows how the HDFS tokenization job creates fuzzy tokens for the input data in the initial mode:



When you run the HDFS tokenization job in the initial mode, the job performs the following tasks:

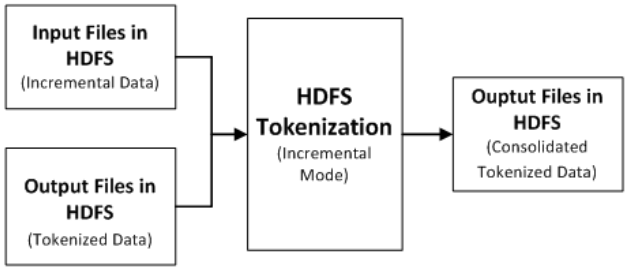
1. Reads the input files in HDFS.
2. Generates fuzzy tokens for the input data.
3. Writes the tokenized data to the output files in HDFS.
   The tokenized data includes input records and the fuzzy tokens of the input records.

   Note: The number of output files depends on the number of reducers that you run.

Incremental Mode

Use the incremental mode to update the tokenized data that a HDFS tokenization job creates.

The following image shows how the HDFS tokenization job updates the tokenized data in the incremental mode:



When you run the HDFS tokenization job in the incremental mode, the job performs the following tasks:

1. Reads the input files in HDFS.
2. Reads the output files of a HDFS tokenization job in HDFS.
3. Merges the input data from the input files and the tokenized data from the HDFS tokenization job.
4. Identifies the duplicate records in the tokenized data based on the fuzzy tokens and updates them with the records from the input files.
5. Generates fuzzy tokens for the input data and adds them to the merged data.
6. Writes the merged data to the output files in HDFS.

Running the HDFS Tokenization Job

Use the HDFS tokenization job to read data from the input files in HDFS, create match tokens for the input data, and write the tokenized data to the output files in HDFS.

To run the HDFS tokenization job, run the `run_tokenizer.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Initial Mode

Use the following command to run the `run_tokenizer.sh` script in the initial mode:

```
run_tokenizer.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--rule=matching_rules_file_name
[--outputpath=directory_for_output_files]
[--reducer=number_of_reducers]
```

The following table describes the options and the arguments that you can specify to run the `run_tokenizer.sh` script in the initial mode:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create. In the configuration file, if you set the `StoreAllFields` parameter to false, the output files of the job does not include all the columns but includes only the columns that you use to index the input data. If you want to include all the columns in the output files, ensure that you set the `StoreAllFields` parameter to true in the configuration file before you run the job.
--input	input_file_in_HDFS	Absolute path to the input files in HDFS.
--reducer	number_of_reducers	Optional. Number of reducer jobs that you want to run. Default is 1.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The HDFS tokenization job uses the working directory to store the output and library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.

For example, the following command runs the HDFS tokenization job in the initial mode:

```
run_tokenizer.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/
GenerateTokens --reducer=16 --hdfsdir=/usr/hdfs/workingdir --rule=/usr/local/conf/
matching_rules.xml
```

If you run the HDFS tokenization job without the `--outputpath` parameter, you can find the tokenized data in the following directory: `<Working Directory in HDFS>/batch-tokenize/<Job ID>/tokenize`

Each job generates a unique ID, and you can identify the job ID based on the time stamp of the `<Job ID>` folder.

If you run the HDFS tokenization job with the `--outputpath` parameter, you can find the tokenized data in the following directory: `<Output Directory in HDFS>/batch-tokenize/tokenize`

The following sample output of the HDFS tokenization job shows the cluster ID, the field values, and the token for an input record:

```
a3ebff6d-c578-4ace-b6d1-2805788d78a6        0000000007ERP        3M        3M
Center        St. Paul        USA        ARC SOFT        00000001ZZB>$$$$01000004N?H-C$$-
NAH$$$$-NAH-C$$$QVM$*K$-
```

Incremental Mode

Use the following command to run the `run_tokenizer.sh` script in the incremental mode:

```
run_tokenizer.sh

--config=configuration_file_name

--input=input_file_in_HDFS

--hdfsdir=working_directory_in_HDFS

--rule=matching_rules_file_name

--incremental

--clustereddirs=tokenize_output_data_directory

[--outputpath=directory_for_output_files]

[--reducer=number_of_reducers]
```

The following table describes the options and the arguments that you can specify to run the `run_tokenizer.sh` script in the initial mode:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the input files in HDFS.
--reducer	number_of_reducers	Optional. Number of reducer jobs that you want to run. Default is 1.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The HDFS tokenization job uses the working directory to store the output and library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.
--incremental		Runs the HDFS tokenization job in the incremental mode. If you want to incrementally update the output files of a HDFS tokenization job, run the job in the incremental mode. By default, the HDFS tokenization job runs in the initial mode.

Option	Argument	Description
--clustereddirs	tokenize_output_data_directory	Absolute path to the directory that contains tokenized data. If you run the HDFS tokenization job without the `--outputpath` parameter, you can find the tokenized data in the following directory: `<Working Directory in HDFS>/batch-tokenize/<Job ID>/tokenize`If you run the HDFS tokenization job with the `--outputpath` parameter, you can find the tokenized data in the following directory: `<Output Directory in HDFS>/batch-tokenize/tokenize`
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.

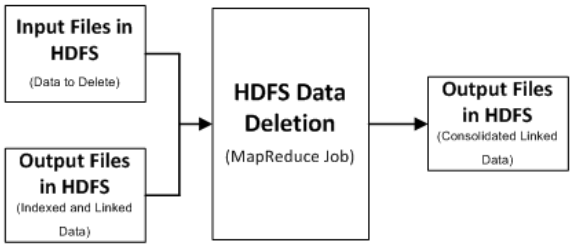
For example, the following command runs the HDFS tokenization job in the incremental mode:

```
run_tokenizer.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/
GenerateTokens --reducer=16 --hdfsdir=/usr/hdfs/workingdir --rule=/usr/local/conf/
matching_rules.xml --clustereddirs=/usr/hdfs/workingdir/batch-tokenize/
MDMBDRM_931211654144593570/tokenize --incremental
```

HDFS Data Deletion Job

The HDFS data deletion job matches the input data in HDFS with the linked or tokenized data in HDFS that the initial clustering or HDFS tokenization job creates. The job deletes the matching data and writes the consolidated data to an output directory in HDFS.

The following image shows how the HDFS data deletion job deletes the input data:



The HDFS data deletion job performs the following tasks:

1. Reads the input files from HDFS.
2. Matches the input data with the linked or tokenized data in HDFS.
3. Deletes the matching records from the linked or tokenized data in HDFS.

Running the HDFS Data Deletion Job

The HDFS data deletion job matches the input data in HDFS with the linked or tokenized data in HDFS, deletes the matching data, and writes the consolidated data to an output directory in HDFS.

To run the HDFS data deletion job, use the `run_clusterdel.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_clusterdel.sh` script:

```
run_clusterdel.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--clustereddirs=indexed_linked_data_directory
[--reducer=number_of_reducer_jobs]
[--outputpath=directory_for_output_files]
```

The following table describes the options and arguments that you can specify to run the `run_clusterdel.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the input files in HDFS.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run. Default is 1.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The HDFS data deletion job uses the working directory to store the library files.
--clustereddirs	indexed_linked_data_directory	Absolute path to the output files that an initial clustering job or a HDFS tokenization job creates. If you run the initial clustering job without the `--outputpath` parameter, you can find the linked data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/output/dir/pass-join` If you run the initial clustering job with the `--outputpath` parameter, you can find the linked data in the following directory: `<Output Directory in HDFS>/batch-cluster/output/dir/pass-join` If you run the HDFS tokenization job without the `--outputpath` parameter, you can find the tokenized data in the following directory: `<Working Directory in HDFS>/batch-tokenize/<Job ID>/tokenize` If you run the HDFS tokenization job with the `--outputpath` parameter, you can find the tokenized data in the following directory: `<Output Directory in HDFS>/batch-tokenize/tokenize`
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.

For example, the following command deletes the matching records from the linked data in HDFS and writes the consolidated data to the output files in HDFS:

```
run_clusterdel.sh  --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/
Source10Million --reducer=16 --hdfsdir=/usr/hdfs/workingdir --clustereddirs=/usr/hdfs/
workingdir/batch-cluster/MDMBDRM_931211654144593570/output/dir/pass-join
```

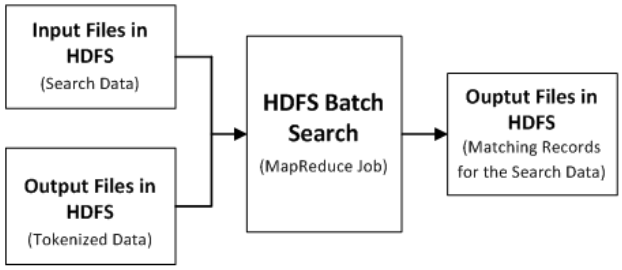
If you run the HDFS data deletion job without the `--outputpath` parameter, you can find the consolidated data in the following directory: `<Working Directory in HDFS>/BDRMClusterDel/<Job ID>/output/dir/consolidated-dir`

If you run the HDFS data deletion job with the `--outputpath` parameter, you can find the consolidated data in the following directory: `<Output Directory in HDFS>/BDRMClusterDel/output/dir/consolidated-dir`

HDFS Batch Search Job

The HDFS batch search job identifies the matching records for the input data in the output files of a HDFS tokenization job. The HDFS batch search job reads the input data in HDFS and creates the output files that contain the matching records for the input data in HDFS.

The following image shows how the HDFS batch search job searches for the matching records:



When you run the HDFS batch search job, the job performs the following tasks:

1. Reads the input files in HDFS.
2. Compares the input data against the tokenized data that a HDFS tokenization job creates.
3. Writes the matching records for the input data to the output files in HDFS.

   Note: The number of output files depends on the number of reducers that you run.

Running the HDFS Batch Search Job

Use the HDFS batch search job to identify the matching records for the input data in the tokenized data that a HDFS tokenization job creates.

To run the HDFS batch search job, run the `run_relate.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_relate.sh` script:

```
run_relate.sh

--config=configuration_file_name

--input=input_file_in_HDFS

--hdfsdir=working_directory_in_HDFS
```

```
--rule=matching_rules_file_name

--clustereddirs=tokenize_output_data_directory

[--reducer=number_of_reducers]

[--outputpath=directory_for_output_files]
```

The following table describes the options and the arguments that you can specify to run the `run_relate.sh` script in the initial mode:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the input files in HDFS.
--reducer	number_of_reducers	Optional. Number of reducer jobs that you want to run to perform initial clustering. Default is 1.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The HDFS batch search job uses the working directory to store the output and library files.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.
--clustereddirs	tokenize_output_data_directory	Absolute path to the directory that contains tokenized data. If you run the HDFS tokenization job without the `--outputpath` parameter, you can find the tokenized data in the following directory: `<Working Directory in HDFS>/batch-tokenize/<Job ID>/tokenize` If you run the HDFS tokenization job with the `--outputpath` parameter, you can find the tokenized data in the following directory: `<Output Directory in HDFS>/batch-tokenize/tokenize`
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.

For example, the following command runs the HDFS batch search job:

```
run_relate.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/SearchData --
reducer=16 --hdfsdir=/usr/hdfs/workingdir --rule=/usr/local/conf/matching_rules.xml --
clustereddirs=/usr/hdfs/workingdir/batch-tokenize/MDMBDRM_931211654144593570/tokenize
```

If you run the HDFS batch search job without the `--outputpath` parameter, you can find the matching records in the following directory: `<Working Directory in HDFS>/BDRMRelate/<Job ID>/relate-output/dir`

If you run the HDFS batch search job with the `--outputpath` parameter, you can find the matching records in the following directory: `<Output Directory in HDFS>/BDRMRelate/relate-output/dir`

The following sample output of the HDFS batch search job shows the search record and the matching record:

```
*************** 13489618-64bc-44a6-8ef4-fb57d63894ff     ***************
100AML.ADDR    13489618-64bc-44a6-8ef4-fb57d63894ff    0000009514ERP    Wright Medical
Technology    5677 Airline Road    Arlington    USA    ARC SOFT
```

```
---             13489618-64bc-44a6-8ef4-fb57d63894ff    0000009514ERP    Wright Medical
Technology    5677 Airline Road    Arlington    USA    ARC SOFT
```

CHAPTER 4

Processing Streaming Data

This chapter includes the following topics:

Processing Streaming Data Overview

You can link and consolidate or tokenize the streaming data. Use the JSON format to stream the input data into Kafka. Spark or Storm processes the input data in Kafka and then loads it to the repository. After loading the processed data to the repository, Informatica MDM - Relate 360 publishes relevant messages to the output topic in Kafka that you configure when you deploy Relate 360 on Spark or Storm.

You can use the following methods to stream the input data:

- RESTful web services
- Command line
- Informatica Vibe® Data Stream

Prerequisites

Before you process the input data, ensure that you perform the following tasks:

1. Configure Relate 360 to process streaming data.

   For more information about configuring Relate 360 to process streaming data, see the *Informatica MDM - Relate 360 Installation and Configuration Guide*.
2. If you use Spark, ensure that the `setup_realtime.sh` script runs.
3. Create the required tables in the repository.

Creating the Required Tables in the Repository

Before you process the input data, you must create the required tables in the repository. Use the batch jobs to create the required tables in the repository.

1. If you plan to perform the linking process, perform the following tasks:
   a. Run the initial clustering job with at least one record.
   b. If you want to uniformly distribute the linked data across all the regions in the repository, run the region splitter job.

      The job analyzes the input linked data and identifies the split points for all the regions in the repository.
   c. Run the load clustering job.

      The job creates primary key table, link table, and index table in the repository.
   d. If you want to consolidate the linked data, run the `create_preferred_records_table.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

      The script creates an empty preferred records table in the repository.

      Use the following command to run the `create_preferred_records_table.sh` script:

      ```
      create_preferred_records_table.sh --config=<Configuration file name>
      ```

      The following sample command runs the `create_preferred_records_table.sh` script:

      ```
      create_preferred_records_table.sh --config=/usr/local/conf/config_big.xml
      ```

   For more information about the initial clustering, region splitter, load clustering, and consolidation jobs, see the Linking Data and Persisting the Linked Data in a Repository section.
2. If you plan to perform the tokenization process, perform one of the following tasks:
   - Run the repository tokenization job with at least one record.

     The job creates the required tables in the repository.
   - Perform the following tasks:
     1. Run the HDFS tokenization job with at least one record.
     2. If you want to uniformly distribute the linked data across all the regions in the repository, run the region splitter job.

        The job analyzes the input linked data and identifies the split points for all the regions in the repository.
     3. Run the load clustering job.

        The job creates the required tables in the repository.

     For more information about the repository tokenization, HDFS tokenization, region splitter, and load clustering jobs, see the Tokenizing Data and Persisting the Tokenized Data in a Repository section.

Related Topics:

- "Linking Data and Persisting the Linked Data in a Repository" on page 16
- "Tokenizing Data and Persisting the Tokenized Data in a Repository" on page 54

Streaming Data by Using the RESTful Web Services

You can use the RESTful web services to stream the input data in the JSON format for processing. Before you use the web services, ensure that you have packaged and deployed the web services.

For more information about packing and deploying the web services, see the *Informatica MDM - Relate 360 Installation and Configuration Guide*.

1. If you have secured the web services, run the AUTHENTICATE web service.

   The AUTHENTICATE web service authenticates the user credentials and generates an authentication token.

2. Run the GETINGESTLAYOUT web service.

   The GETINGESTLAYOUT web service gets a layout for the input data based on the configuration file. The layout contains a list of fields and their lengths that you can use to specify the input data in the JSON format.

3. Run the INGEST web service.

   The INGEST web service reads the input data, links and consolidates or tokenizes the data, and loads the linked and consolidated or tokenized data into the repository. If any input record is an update to an existing record, the INGEST web service updates the record in the repository.

4. To get or delete a record, perform the following tasks:

   a. Run the GETRECORDLAYOUT web service.

      The GETRECORDLAYOUT web service gets a layout that contains a list of fields required to identify a record. Use the layout to specify the input record for the GETRECORD or DELETERECORD web service.

   b. To get a record, run the GETRECORD web service.

      The GETRECORD web service gets the matching records for the input data from the repository.

   c. To delete a record, run the DELETERECORD web service.

      The DELETERECORD web service deletes the matching record from the repository.

5. To move records into any specific cluster, perform the following tasks:

   a. Run the GETMANAGECLUSTERLAYOUT web service.

      The GETMANAGECLUSTERLAYOUT web service gets a layout that contains a list of fields required to identify records. Use the layout to specify the records for the MANAGECLUSTER web service that removes the records from their current clusters and adds them to a new cluster or to the specified cluster number.

   b. Run the MANAGECLUSTER web service.

      The MANAGECLUSTER web service reads the input records, removes the input records from their respective clusters, and moves the records to the specified cluster or to a new cluster.

Authenticate Web Service

The Authenticate web service authenticates the user credentials that you specify in the web service request. After a successful authentication, the web service request returns an encoded authentication token that you can specify in the header of the subsequent web service requests. A web service request returns appropriate response only if the request contains a valid token or user credentials.

Note: Use the Authenticate web service only if you have secured the RESTful web services.

Request URL

Use the GET method to run the Authenticate web service.

To run the Authenticate web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
Authenticate
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/Authenticate
```

Request Header

Use the request header to specify the following headers:

Authorization

Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Sample Response

The following sample response shows the token generated after a successful authentication:

```
{
   "timeTaken":0.134,
   "token":"AQIC5wM2LY4Sfczqyo9VO8tpm3qq7vDAsf3BB9nzatkh_S8.*AAJTSQACMDEAAlNLABI4MzAzMj
   EwNzMzODg0ODAxNTkAAlMxAAA.*"
}
```

Note: An authentication token is valid for the web service requests that use the same WAR file and host name as that of the Authenticate web service request.

GETINGESTLAYOUT Web Service

The GETINGESTLAYOUT web service gets a layout for the input data based on the configuration file. The layout contains a list of fields and their lengths that you can use to specify the input data for the INGEST web service in the JSON format.

Request URL

Use the GET method to run the GETINGESTLAYOUT web service.

To run the GETINGESTLAYOUT web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
GETINGESTLAYOUT
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/GETINGESTLAYOUT
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Sample Response

The following response format shows a list of fields that you can use to specify the input data:

```
{
   "input":{
      "SOURCE":"MaxLen:20",
      "PERSON":"MaxLen:100",
      "DATE_OF_BIRTH":"MaxLen:10",
      "ADDRESS":"MaxLen:100",
      "POSTCODE":"MaxLen:10",
      "CITY":"MaxLen:100",
      "ROWID":"MaxLen:21"
   },
   "resultCount":0,
   "messages":{

   }
}
```

INGEST Web Service

The INGEST web service reads the input data and then links or tokenizes the input data based on the value that you set for the `ALTERNATETABLEFORGROUPINFO` parameter in the configuration file. The INGEST web service then loads the linked or tokenized data into the repository.

When you deploy Relate 360 on Spark or Storm, if you have specified the consolidation rules file, the INGEST web service consolidates the linked data and creates preferred records in the repository. If any input record is an update to an existing record, the INGEST web service updates the record in the repository.

During the linking process, when an input record matches with a record in the repository, the linking process links the input record to the cluster of the matching record. The linking process does not further match the input record with other records.

Note: When you link the input data, ensure that you do not configure the `MaxCandidateSet` parameter in the configuration file. The `MaxCandidateSet` parameter value impacts the maximum number of records that the INGEST web service can add to a cluster.

Before you run the INGEST web service, run the GETINGESTLAYOUT web service to get the layout for the input data in the JSON format based on the configuration file. Based on the layout, you can specify the input data for the INGEST web service.

Request URL

Use the POST method to run the INGEST web service.

To run the INGEST web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
INGEST
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/INGEST
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Content-Type

Format of the request body. The supported format is JSON. Specify `application/json` as the header value.

Request Body

Use the request body to specify the input data for the INGEST web service based on the layout that the GETINGESTLAYOUT web service returns.

The following sample body contains a record for the INGEST web service to process:

```
{
   "input":{
      "SOURCE":"AML",
      "PERSON":"Susan Shaw",
      "ADDRESS":"Castle Boulevard",
      "POSTCODE":"94061",
      "CITY":"Redwood City",
      "ROWID":"0000300002"
   },
   "resultCount":0,
   "messages":{

   }
```

Sample Response

The following sample response shows a record that the INGEST web service submits for processing:

```
{
   "input":{
      "SOURCE":"AML",
      "PERSON":"Susan Shaw",
      "ADDRESS":"Castle Boulevard",
      "POSTCODE":"94061",
      "CITY":"Redwood City",
      "ROWID":"0000300002"
   },
   "resultCount":1,
   "messages":{
      "Message.1":"Record submitted for Processing"
   }
}
```

GETRECORDLAYOUT Web Service

The GETRECORDLAYOUT web service gets a layout that contains a list of fields required to identify a record. Use the layout to specify the input data for the DELETERECORD web service that deletes the specified records from the repository.

Request URL

Use the GET method to run the GETRECORDLAYOUT web service.

To run the GETRECORDLAYOUT web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
GETRECORDLAYOUT
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/GETRECORDLAYOUT
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Sample Response

The following sample response shows a list of fields required to identity a record:

```
{
  "keyData": {
    "SOURCE": "mandatory",
    "ROWID": "mandatory"
  },
  "record": {},
  "resultCount": 0,
  "messages": {}
}
```

GETRECORD Web Service

The GETRECORD web service gets the matching records for the input data from the repository. Before you run the GETRECORD web service, run the GETRECORDLAYOUT web service to get the layout for the input data in the JSON format based on the configuration file. Based on the layout, you can specify the input data for the GETRECORD web service.

Request URL

Use the GET method to run the GETRECORD web service.

To run the GETRECORD web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
GETRECORD
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/GETRECORD
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Content-Type

Format of the request body. The supported format is JSON. Specify `application/json` as the header value.

Request Body

Use the request body to specify the input data for the GETRECORD web service based on the layout that the GETRECORDLAYOUT web service returns.

The following sample body contains the condition based on which you want to retrieve the matching records:

```
{
   "keyData":{
      "LMT_SOURCE_NAME":"AML",
      "ROWID":"0000300001"
   },
   "record":{

   },
   "resultCount":0,
   "messages":{

   }
}
```

Sample Response

The following sample response shows a record that the GETRECORD web service retrieved:

```
{
  "keyData": {
    "LMT_SOURCE_NAME": "AML",
    "ROWID": "0000300001"
  },
  "record": {
    "AGENCY_TYPE": "SDN",
    "NAME": "AL SAADI, FARAJ FARJ HASSAN",
    "COUNTRY": "ITALY",
    "ADDRESS": "VIALE BLIGNY 42",
    "CITY": "MILAN",
    "COUNTRYISO": "IT",
    "AGENCY": "FAC",
    "LMT_SOURCE_NAME": "                                AML",
    "ROWID": "0000300001"
  },
  "resultCount": 1,
  "messages": {}
}
```

DELETERECORD Web Service

The DELETERECORD web service indicates to read the input data and delete the matching records from the repository. Before you run the DELETERECORD web service, run the GETRECORDLAYOUT web service to get the layout for the input data in the JSON format based on the configuration file. Based on the layout, you can specify the input data for the DELETERECORD web service.

Request URL

Use the POST method to run the DELETERECORD web service.

To run the DELETERECORD web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
DELETERECORD
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/DELETERECORD
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Content-Type

Format of the request body. The supported format is JSON. Specify `application/json` as the header value.

Request Body

Use the request body to specify the input data for the DELETERECORD web service based on the layout that the GETRECORDLAYOUT web service returns.

The following sample body contains a record that you want to delete:

```
{
   "keyData":{
      "LMT_SOURCE_NAME":"AML",
      "ROWID":"0000300001"
   },
   "record":{

   },
   "resultCount":0,
   "messages":{

   }
}
```

Sample Response

The following sample response shows a record that the DELETERECORD web service submits for processing:

```
{
   "keyData":{
      "LMT_SOURCE_NAME":"AML",
      "ROWID":"0000300001"
   },
   "record":{

   },
   "resultCount":1,
   "messages":{
      "Message.1":"Record submitted for Processing"
   }
}
```

GETMANAGECLUSTERLAYOUT Web Service

The GETMANAGECLUSTERLAYOUT web service gets a layout that contains a list of fields required to identify records. Use the layout to specify the records for the MANAGECLUSTER web service that removes the records from their current clusters and adds them to a new cluster or to the specified cluster number.

Request URL

Use the GET method to run the GETMANAGECLUSTERLAYOUT web service.

To run the GETMANAGECLUSTERLAYOUT web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
GETMANAGECLUSTERLAYOUT
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/GETMANAGECLUSTERLAYOUT
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Sample Response

The following response format shows the list of fields that you can specify to manage clusters:

```
{
  "clusterNumber": "",
  "members": [
    {
      "LMT_SOURCE_NAME": "",
      "PK": ""
    }
  ],
  "resultCount": 0,
  "messages": {}
}
```

A cluster layout includes the following parameters:

clusterNumber

Identifier for the cluster. Run the GETCLUSTER web service to get the cluster number of a record.

LMT_SOURCE_NAME

Source name of the record.

PK

Column name that you set as primary key in the configuration file.

MANAGECLUSTER Web Service

The MANAGECLUSTER web service indicates to read the input records, remove the input records from their respective clusters, and move the records to the specified cluster or to a new cluster.

Before you run the MANAGECLUSTER web service, run the GETMANAGECLUSTERLAYOUT web service to get the layout for the input data in the JSON format based on the configuration file. Based on the layout, you can specify the input data for the MANAGECLUSTER web service.

Request URL

Use the POST method to run the MANAGECLUSTER web service.

To run the MANAGECLUSTER web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
MANAGECLUSTER
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/MANAGECLUSTER
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Content-Type

Format of the request body. The supported format is JSON. Specify `application/json` as the header value.

Request Body

Use the request body to specify the input data for the MANAGECLUSTER web service based on the layout that the GETMANAGECLUSTERLAYOUT web service returns.

If you do not specify the cluster number in the body, the MANAGECLUSTER web service indicates to create a cluster and move the specified records to the created cluster.

The following sample body contains a record that you want to add it to the specified cluster number:

```
{
   "clusterNumber":"12b1ea9f-4f5c-4168-828e-1b03edef0c6a    ",
   "members":[
      {
         "LMT_SOURCE_NAME":"AML",
         "PK":"0000300003"
      }
   ],
   "resultCount":0,
   "messages":{

   }
}
```

Sample Response

The following sample response shows a record that the MANAGECLUSTER web service submits for processing:

```
{
  "clusterNumber": "12b1ea9f-4f5c-4168-828e-1b03edef0c6a    ",
  "members": [
    {
      "LMT_SOURCE_NAME": "AML",
      "PK": "0000300003"
    }
  ],
  "resultCount": 1,
  "messages": {
    "Message.1": "Record submitted for Processing"
  }
}
```

Streaming Data by Using the Command-Line Command

You can use the command-line command to stream the input data in the JSON format for processing.

1. Perform the GETINGESTLAYOUT operation.

   The GETINGESTLAYOUT operation gets a layout for the input data based on the configuration file. The layout contains a list of fields and their lengths that you can use to specify the input data in the JSON format.

2. Perform the INGEST operation.

   The INGEST operation reads the input data, links or tokenizes the data, and loads the linked or tokenized data into the repository. If any input record is an update to an existing record, the INGEST operation updates the record in the repository.

3. To delete a record, perform the following tasks:

   a. Perform the GETRECORDLAYOUT operation.

      The GETRECORDLAYOUT operation gets a layout that contains a list of fields required to identify a record. Use the layout to specify the input record for the GETRECORD or DELETERECORD operation.

   b. To get a record, perform the GETRECORD operation.

      The GETRECORD operation gets the matching records for the input data from the repository.

c. To delete a record, perform the DELETERECORD operation.

The DELETERECORD operation deletes the matching record from the repository.

4. To move the records into a specific cluster, perform the following tasks:

a. Perform the GETMANAGECLUSTERLAYOUT operation.

The GETMANAGECLUSTERLAYOUT operation gets a layout that contains a list of fields required to identify records. Use the layout to specify the records for the MANAGECLUSTER operation that removes the records from their current clusters and adds them to a new cluster or to the specified cluster number.

b. Perform the MANAGECLUSTER operation.

The MANAGECLUSTER operation reads the input records, removes the input records from their respective clusters, and moves the records to the specified cluster or to a new cluster.

GETINGESTLAYOUT Operation

The GETINGESTLAYOUT operation gets a layout for the input data based on the configuration file. The layout contains a list of fields and their lengths that you can use to specify the input data for the INGEST web service in the JSON format.

Run the `run_client.sh` script located in the following directory to perform the GETINGESTLAYOUT operation: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_client.sh` script:

```
run_client.sh

--config=configuration_file_name

--rule=matching_rules_file_name

--operation=GETINGESTLAYOUT

[--outputfile=output_file_name]
```

The following table describes the options and arguments that you can specify to run the `run_client.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.
--operation	GETINGESTLAYOUT	Type of operation that you want to perform. Specify GETINGESTLAYOUT.
--outputfile	output_file_name	Absolute path and name of the output JSON file to which you want to load the layout.

For example:

```
run_client.sh --config=/usr/local/tree/configuration.xml --operation=GETINGESTLAYOUT --
outputfile=/usr/local/tree/output.json --rule=usr/local/conf/matching_rules.xml
```

The following response format shows a list of fields that you can use to specify the input data:

```
{
   "input":{
      "SOURCE":"MaxLen:20",
      "PERSON":"MaxLen:100",
```

```
        "DATE_OF_BIRTH":"MaxLen:10",
        "ADDRESS":"MaxLen:100",
        "POSTCODE":"MaxLen:10",
        "CITY":"MaxLen:100",
        "ROWID":"MaxLen:21"
    },
    "resultCount":0,
    "messages":{

    }
}
```

INGEST Operation

The INGEST operation reads the input data and then links or tokenizes the input data based on the value that you set for the `ALTERNATETABLEFORGROUPINFO` parameter in the configuration file. The INGEST operation then loads the linked or tokenized data into the repository.

When you deploy Relate 360 on Spark or Storm, if you have specified the consolidation rules file, the INGEST operation consolidates the linked data and creates preferred records in the repository. If any input record is an update to an existing record, the INGEST operation updates the record in the repository.

During the linking process, when an input record matches with a record in the repository, the linking process links the input record to the cluster of the matching record. The linking process does not further match the input record with other records.

Note: When you link the input data, ensure that you do not configure the `MaxCandidateSet` parameter in the configuration file. The `MaxCandidateSet` parameter value impacts the maximum number of records that the INGEST operation can add to a cluster.

Run the `run_client.sh` script located in the following directory to perform the INGEST operation: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_client.sh` script:

```
run_client.sh

--config=configuration_file_name

--rule=matching_rules_file_name

--operation=INGEST

--input=input_file_name

[--outputfile=output_file_name]
```

The following table describes the options and arguments that you can specify to run the `run_client.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.
--operation	INGEST	Type of operation that you want to perform. Specify INGEST.

Option	Argument	Description
--input	input_file_name	Absolute path and name of the input JSON file that contains the input data.
--outputfile	output_file_name	Optional. Absolute path and name of the output JSON file to which you want to load the processed data.

For example:

```
run_client.sh --config=/usr/local/tree/configuration.xml --operation=INGEST --input=/usr/
local/tree/input.json --rule=usr/local/conf/matching_rules.xml --outputfile=/usr/local/
tree/output.json
```

The following input data contains a record for the INGEST operation to process:

```
{
   "input":{
      "SOURCE":"AML",
      "PERSON":"Susan Shaw",
      "ADDRESS":"Castle Boulevard",
      "POSTCODE":"94061",
      "CITY":"Redwood City",
      "ROWID":"0000300002"
   },
   "resultCount":0,
   "messages":{

   }
```

The following sample output shows a record that the INGEST operation submits for processing:

```
{
   "input":{
      "SOURCE":"AML",
      "PERSON":"Susan Shaw",
      "ADDRESS":"Castle Boulevard",
      "POSTCODE":"94061",
      "CITY":"Redwood City",
      "ROWID":"0000300002"
   },
   "resultCount":1,
   "messages":{
      "Message.1":"Record submitted for Processing"
   }
}
```

GETRECORDLAYOUT Operation

The GETRECORDLAYOUT operation gets a layout that contains a list of fields required to identify a record. Use the layout to specify the input data for the DELETERECORD operation that deletes the specified records from the repository.

Run the `run_client.sh` script located in the following directory to perform the GETRECORDLAYOUT operation: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_client.sh` script:

```
run_client.sh

--config=configuration_file_name

--rule=matching_rules_file_name

--operation=GETRECORDLAYOUT
```

```
[--outputfile=output_file_name]
```

The following table describes the options and arguments that you can specify to run the `run_client.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.
--operation	GETRECORDLAYOUT	Type of operation that you want to perform. Specify GETRECORDLAYOUT.
--outputfile	output_file_name	Optional. Absolute path and name of the output JSON file to which you want to load the record layout.

For example:

```
run_client.sh --config=/usr/local/tree/configuration.xml --operation=GETRECORDLAYOUT --
outputfile=/usr/local/tree/output.json --rule=usr/local/conf/matching_rules.xml
```

The following output format shows the list of fields that you can specify the input data:

```
{
  "keyData": {
    "SOURCE": "mandatory",
    "ROWID": "mandatory"
  },
  "record": {},
  "resultCount": 0,
  "messages": {}
}
```

GETRECORD Operation

The GETRECORD operation gets the matching records for the input data from the repository. Before you run the GETRECORD operation, run the GETRECORDLAYOUT operation to get the layout for the input data in the JSON format based on the configuration file. Based on the layout, you can specify the input data for the GETRECORD operation.

Run the `run_client.sh` script located in the following directory to perform the GETRECORD operation: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_client.sh` script:

```
run_client.sh

--config=configuration_file_name

--rule=matching_rules_file_name

--operation=GETRECORD

--input=input_file_name

[--outputfile=output_file_name]
```

The following table describes the options and arguments that you can specify to run the `run_client.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.
--operation	GETRECORD	Type of operation that you want to perform. Specify GETRECORD.
--input	input_file_name	Absolute path and name of the input JSON file that contains the input data.
--outputfile	output_file_name	Optional. Absolute path and name of the output JSON file to which you want to load the record layout.

For example:

```
run_client.sh --config=/usr/local/tree/configuration.xml --operation=GETRECORD --
input=/usr/local/tree/input.json --rule=usr/local/conf/matching_rules.xml --
outputfile=/usr/local/tree/output.json
```

The following input data contains the condition based on which you want to retrieve the matching records:

```
{
   "keyData":{
      "LMT_SOURCE_NAME":"AML",
      "ROWID":"0000300001"
   },
   "record":{

   },
   "resultCount":0,
   "messages":{

   }
}
```

The following sample output shows a record that the GETRECORD operation retrieved:

```
{
  "keyData": {
    "LMT_SOURCE_NAME": "AML",
    "ROWID": "0000300001"
  },
  "record": {
    "AGENCY_TYPE": "SDN",
    "NAME": "AL SAADI, FARAJ FARJ HASSAN",
    "COUNTRY": "ITALY",
    "ADDRESS": "VIALE BLIGNY 42",
    "CITY": "MILAN",
    "COUNTRYISO": "IT",
    "AGENCY": "FAC",
    "LMT_SOURCE_NAME": "                                AML",
    "ROWID": "0000300001"
  },
  "resultCount": 1,
  "messages": {}
}
```

DELETERECORD Operation

The DELETERECORD operation indicates to read the input data and delete the matching records from the repository. Before you run the DELETERECORD operation, run the GETRECORDLAYOUT operation to get the

layout for the input data in the JSON format based on the configuration file. Based on the layout, you can specify the input data for the DELETERECORD operation.

Run the `run_client.sh` script located in the following directory to perform the DELETERECORD operation: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_client.sh` script:

```
run_client.sh

--config=configuration_file_name

--rule=matching_rules_file_name

--operation=DELETERECORD

--input=input_file_name

[--outputfile=output_file_name]
```

The following table describes the options and arguments that you can specify to run the `run_client.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.
--operation	DELETERECORD	Type of operation that you want to perform. Specify DELETERECORD.
--input	input_file_name	Absolute path and name of the input JSON file that contains the input data.
--outputfile	output_file_name	Optional. Absolute path and name of the output JSON file to which you want to load the deleted record.

For example:

```
run_client.sh --config=/usr/local/tree/configuration.xml --operation=DELETERECORD --
input=/usr/local/tree/input.json --rule=usr/local/conf/matching_rules.xml --
outputfile=/usr/local/tree/output.json
```

The following input data contains a record that you want to delete:

```
{
   "keyData":{
      "LMT_SOURCE_NAME":"AML",
      "ROWID":"0000300001"
   },
   "record":{

   },
   "resultCount":0,
   "messages":{

   }
}
```

The following sample output shows a record that the DELETERECORD operation submits for processing:

```
{
   "keyData":{
      "LMT_SOURCE_NAME":"AML",
      "ROWID":"0000300001"
   },
   "record":{
```

```
   },
   "resultCount":1,
   "messages":{
      "Message.1":"Record submitted for Processing"
   }
}
```

GETMANAGECLUSTERLAYOUT Operation

The GETMANAGECLUSTERLAYOUT operation gets a layout that contains a list of fields required to identify records. Use the layout to specify the records for the MANAGECLUSTER operation that removes the records from their current clusters and adds them to a new cluster or to the specified cluster number.

Run the `run_client.sh` script located in the following directory to perform the GETMANAGECLUSTERLAYOUT operation: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_client.sh` script:

```
run_client.sh

--config=configuration_file_name

--rule=matching_rules_file_name

--operation=GETMANAGECLUSTERLAYOUT

[--outputfile=output_file_name]
```

The following table describes the options and arguments that you can specify to run the `run_client.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.
--operation	GETMANAGECLUSTERLAYOUT	Type of operation that you want to perform. Specify GETMANAGECLUSTERLAYOUT.
--outputfile	output_file_name	Absolute path and name of the output JSON file to which you want to load the layout.
--outputfile	output_file_name	Optional. Absolute path and name of the output JSON file to which you want to load the layout.

For example:

```
run_client.sh --config=/usr/local/tree/configuration.xml --
operation=GETMANAGECLUSTERLAYOUT --outputfile=/usr/local/tree/output.json --rule=usr/
local/conf/matching_rules.xml --outputfile=/usr/local/tree/output.json
```

The following output format shows the list of fields that you can specify the input data:

```
{
  "clusterNumber": "",
  "members": [
    {
      "LMT_SOURCE_NAME": "",
      "PK": ""
    }
  ],
  "resultCount": 0,
```

```
  "messages": {}
}
```

A cluster layout includes the following parameters:

clusterNumber

Identifier for the cluster. Perform the GETCLUSTER operation to get the cluster number of a record.

LMT_SOURCE_NAME

Source name of the record.

PK

Column name that you set as primary key in the configuration file.

MANAGECLUSTER Operation

The MANAGECLUSTER operation indicates to read the input records, remove the input records from their respective clusters, and move the records to the specified cluster or to a new cluster. If you do not specify the cluster number in the input file, the MANAGECLUSTER operation indicates to create a cluster and move the specified records to the created cluster.

Run the `run_client.sh` script located in the following directory to perform the MANAGECLUSTER operation: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_client.sh` script:

```
run_client.sh

--config=configuration_file_name

--rule=matching_rules_file_name

--operation=MANAGECLUSTER

--input=input_file_name

[--outputfile=output_file_name]
```

The following table describes the options and arguments that you can specify to run the `run_client.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.
--operation	MANAGECLUSTER	Type of operation that you want to perform. Specify MANAGECLUSTER.
--input	input_file_name	Absolute path and name of the input JSON file that contains the input data.
--outputfile	output_file_name	Optional. Absolute path and name of the output JSON file to which you want to load the processed data.

For example:

```
run_client.sh --config=/usr/local/tree/configuration.xml --operation=MANAGECLUSTER --
input=/usr/local/tree/input.json --rule=usr/local/conf/matching_rules.xml --
outputfile=/usr/local/tree/output.json
```

The following input data contains a record that you want to add it to the specified cluster number:

```
{
   "clusterNumber":"12b1ea9f-4f5c-4168-828e-1b03edef0c6a    ",
   "members":[
      {
         "LMT_SOURCE_NAME":"AML",
         "PK":"0000300003"
      }
   ],
   "resultCount":0,
   "messages":{

   }
}
```

The following sample output shows a record that the MANAGECLUSTER operation submits for processing:

```
{
  "clusterNumber": "12b1ea9f-4f5c-4168-828e-1b03edef0c6a    ",
  "members": [
    {
      "LMT_SOURCE_NAME": "AML",
      "PK": "0000300003"
    }
  ],
  "resultCount": 1,
  "messages": {
    "Message.1": "Record submitted for Processing"
  }
}
```

Viewing the Output Messages

The RESTful web services or the command-line command publishes the input data to the input topic in Kafka. Spark or Storm processes the input data in Kafka. After processing the input data, Relate 360 publishes output messages to a topic in Kafka that you configure when you deploy Relate 360 on Spark or Storm.

For information about how to view messages in a Kafka topic, see the Kafka documentation.

The following sample message indicates successful INGEST and DELETERECORD operations:

```
{
   "Status":"SUCCESS",
   "CLUSTER":"5b53d62e-e87a-4618-9d97-a6c858d8bee8    ",
   "SOURCE":"AML",
   "PK":"0000300011",
   "OPERATION":"CREATE"
}
{
   "Status":"SUCCESS",
   "SOURCE":"AML",
   "PK":"0000300014",
   "OPERATION":"DELETE"
}
```

CHAPTER 5

Creating Relationship Graph

This chapter includes the following topics:

Relationship Graph Overview

You can create relationships between the processed records. The processed records can be customer data, transaction data, product data, and other types of data. You can consider each type of data as a business entity type. When you create relationships between two or more business entity types, the relationships result in a relationship graph. A relationship graph displays all the related records of a record and its relationships with the related records.

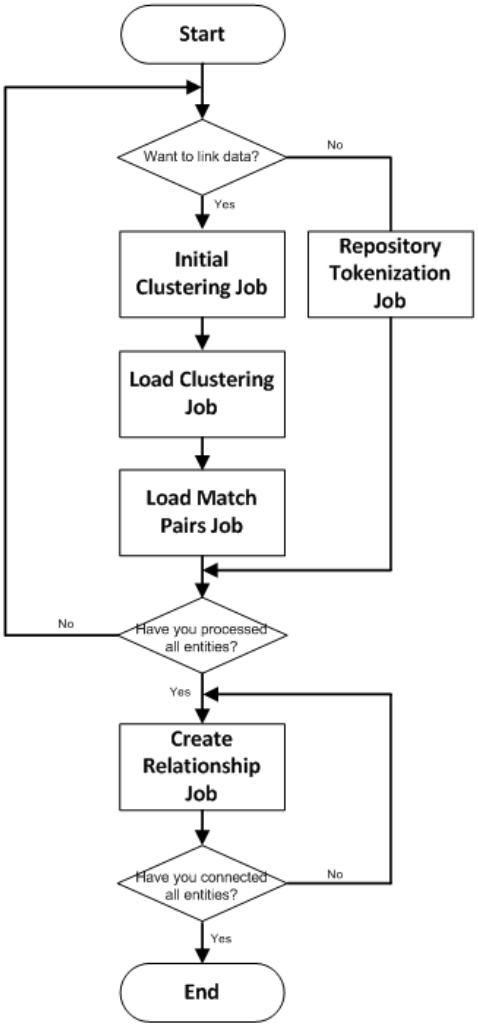
You create relationships between the business entity types based on the parameters in the relationship configuration file.

Creating Relationship Graph

A business entity type is a set of similar type of input data. The input data can be customer data, transaction data, product data, and other types of data. You can customize the `PZMAP` section of the configuration file and the match rule sets in the matching rules file for each type of input data. You can link or tokenize the input data based on the type of data.

For example, you can link the customer data to identify the household relationships. You do not require to link the transaction data. In this case, link the customer data and load the linked data into the repository, and tokenize the transaction data and load the tokenized data into the repository.

The following image shows the batch jobs that you can run to create the relationship graph:



To create a relationship graph, perform the following tasks:

1. If you want to link the input data, perform the following tasks:
   a. Run the initial clustering job.

      The job links the input data and creates linked and match-pair data in HDFS.

   b. Run the load clustering job.

      The job creates required tables in the repository and loads the linked data into the tables.

   c. Run the load match pairs job.

      The job loads the match-pair data of the business entity type into the repository.
2. If you do not want to link the input data, run the repository tokenization job.

   The job tokenizes the input data, creates required tables in the repository, and loads the tokenized data into the tables.
3. Similarly, process other business entity types and load the processed data into the repository.
4. Run the create relationship job.

   The job creates relationship between two entities.
5. Similarly, run the create relationship job for other entities to create the relationship graph.

Related Topics:

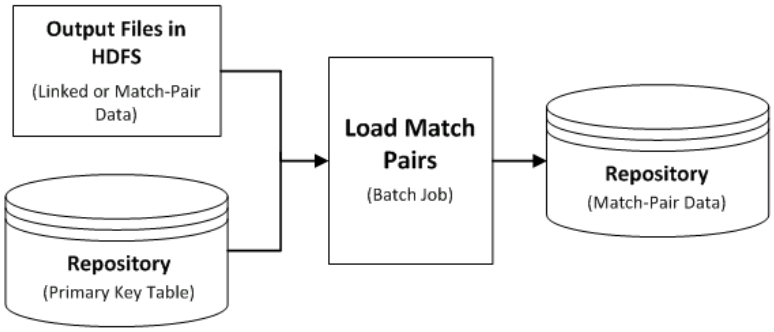
- "Initial Clustering Job" on page 18
- "Load Clustering Job" on page 28
- "Repository Tokenization Job" on page 55
- "Load Match Pairs Job" on page 104
- "Create Relationship Job" on page 106

Load Match Pairs Job

The load match pairs job creates a relationship table in the repository and loads the details of the match pairs into the relationship table. If the relationship table already exists, the load match pairs job appends the details of the match pairs to the relationship table. The input data can be linked or match-pair data that an initial clustering job creates.

Before you run the load match pairs job, you must have run the load clustering job at least once.

The following image shows how the load match pairs job loads the data into the repository:



The load match pairs job performs the following tasks:

1. Creates a relationship table in the repository if the table does not exist.
2. Loads the linked or match-pair data into the relationship table.

Run the Load Match Pairs Job

The load match pairs job loads the linked or match-pair data into the relationship table. If the relationship table does not exist, the job creates the table and then loads the data into it.

To run the load match pairs job, run the `run_graphloader.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_graphloader.sh` script:

```
run_graphloader.sh

--config=configuration_file_name

--relconfig=relationship_configuration_file_name

--input=input_file_in_HDFS

--hdfsdir=working_directory_in_HDFS
```

```
--entity=entity_name

[--outputpath=directory_for_output_files]

[--reducer=number_of_reducer_jobs]
```

The following table describes the options and arguments that you can specify to run the `run_graphloader.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file.
--relconfig	relationship_configuration_file_name	Absolute path and file name of the relationship configuration file.
--input	input_file_in_HDFS	Absolute path to the directory that contains match-pair or linked data. If you run the initial clustering job without the `--outputpath` parameter, you can find the processed data in the following directories: - Match-pair data. `<Working Directory in HDFS>/batch-cluster/<Job ID>/match/dir` - Linked data. `<Working Directory in HDFS>/batch-cluster/<Job ID>/output/dir/pass-join` If you run the initial clustering job with the `--outputpath` parameter, you can find the processed data in the following directories: - Match-pair data. `<Output Directory in HDFS>/batch-cluster/match/dir` - Linked data. `<Output Directory in HDFS>/batch-cluster/output/dir/pass-join`
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The job uses the working directory to store the library files.
--entity	entity_name	Name of the business entity type based on which the job retrieves the entity configuration from the relationship configuration file.
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run to load the match pairs. Default is 1.

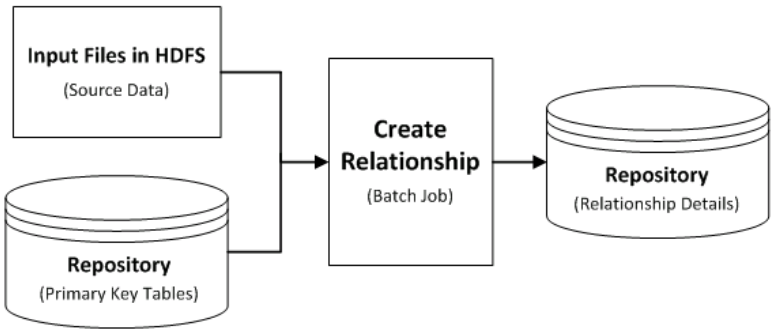
For example, the following command runs the load match pairs job:

```
run_graphloader.sh --config=/usr/local/conf/config_big.xml --relconfig=/usr/local/conf/
relconfig.xml --input=/usr/hdfs/workingdir/batch-cluster/MDMBDE0063_1602999447744334391/
match/dir --reducer=16 --hdfsdir=/usr/hdfs/workingdir --entity=Customer --
outputpath=/usr/hdfs/outputfolder
```

Create Relationship Job

The create relationship job creates relationship between two business entity types based on the parameters in the relationship configuration file and updates the relationship table with the relationship details. If the relationship table does not exist, the job creates the relationship table and then loads the relationship details into the relationship table. After you run the job, you can view the relationships in the relationship graph.

The following image shows how the create relationship job loads the data into the repository:



The create relationship job performs the following tasks:

1. Creates a relationship table in the repository if the table does not exist.
2. Creates relationship between two business entity types.
3. Updates the relationship table with the relationship details.

Run the Create Relationship Job

The create relationship job creates relationship between two entities based on the parameters in the relationship configuration file and updates the relationship table in the repository with the relationship details.

To run the create relationship job, run the `run_relloader.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_relloader.sh` script:

```
run_relloader.sh
--config=configuration_file_name
--relconfig=relationship_configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--sourceentity=source_entity_name
--targetentity=target_entity_name
--relationship=relationship_name
[--outputpath=directory_for_output_files]
[--reducer=number_of_reducer_jobs]
```

The following table describes the options and arguments that you can specify to run the `run_relloader.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file for the input files.
--relconfig	relationship_configuration_file_name	Absolute path and file name of the relationship configuration file.
--sourceentity	source_entity_name	Name of the source business entity type for which you want to create the relationship.
--targetentity	target_entity_name	Name of the target business entity type to which you want to connect the source business entity type.
--input	input_file_in_HDFS	Absolute path to the input files in HDFS. If the input files for the source and target business entity types are different, you must integrate the input files and use the integrated files as the input files for the create relationship job.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. The job uses the working directory to store the library files.
--relationship	relationship_name	Name of the outbound relationship for the source business entity type.
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. By default, the batch job loads the output files to the working directory in HDFS.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run. Default is 1.

For example, the following command runs the create relationship job:

```
run_relloader.sh --config=/usr/local/conf/config_big.xml --relconfig=/usr/local/conf/
relconfig.xml --sourceentity=Customer --targetentity=Products --relationship=Owns --
input=/usr/hdfs/Source10Million --reducer=16 --hdfsdir=/usr/hdfs/workingdir --
outputpath=/usr/hdfs/outputfolder
```

Retrieving the Relationship Details

After you create relationships between the business entity types, you can retrieve details about the business entity types, their relationships, and the relationship graph that you create.

To retrieve the relationship details, use the following RESTful web services:

- Get Graph Metadata

- Get Entity Metadata
- Get Entity Relationship
- Get All Relationship
- Get Entity Details

Note: You can access these RESTful web services only when you include the relationship configuration file in the WAR file that you generate for the RESTful web services.

Get Graph Metadata Web Service

The Get Graph Metadata web service gets the metadata information related to the relationship graph that you created. The metadata information includes details about each business entity type and its relationships with other entities.

Request URL

Use the GET method to run the Get Graph Metadata web service.

To run the Get Graph Metadata web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
GETGRAPHMETADATA
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/GETGRAPHMETADATA
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Sample Response

The following sample response shows the metadata information of a relationship graph named `Customer360`:

```
{
   "relationshipName":"Customer360",
   "entities":{
      "Organization":{
         "urlEndPoint":"",
         "displayField":"CompanyName",
         "parentEntity":"Customer",
```

```
            "parentEntityColumnName":"id",
            "parentlabelName":"Owns",
            "pkColumnLength":0,
            "inboundLabel":[
               "Employee"
            ],
            "outboundLabel":[
               "Employee"
            ],
            "viewfields":{
               "field":[
                  {
                     "value":"id",
                     "name":"id"
                  },
                  {
                     "value":"Source",
                     "name":"LMT_SOURCE_NAME"
                  }
               ]
            },
            "partOfLayout":false,
            "attributes":{
               "attribute":[
                  [
                     [
                        {
                           "type":"EXACT"
                        }
                     ],
                     "CompanyName",
                     100,
                     "true",
                     "false"
                  ],
                  [
                     [
                        {
                           "type":"EXACT"
                        }
                     ],
                     "City",
                     25,
                     "false",
                     "true"
                  ],
                  [
                     [
                        {
                           "type":"EXACT"
                        }
                     ],
                     "State",
                     3,
                     "false",
                     "true"
                  ]
               ]
            },
            "entityConfig":{
               "Hbase Master":"localhost:60000",
               "Hbase Zookeeper ClientPort":"2181",
               "Hbase Zookeeper
Quorum":"torarch2.informatica.com,torarch3.informatica.com,torarch4.informatica.com",
               "Hbase Distributed":"true",
               "Hbase Scan CacheSize":"5000",
               "Hbase Scan BatchSize":"25000"
            },
            "entityOptions":{
               "CompanyName":{
                  "options":{
```

```
                  "Aggregate":"false",
                  "Filterable":"true"
               }
            },
            "City":{
               "options":{
                  "Aggregate":"true",
                  "Filterable":"false"
               }
            }
         }
      }
   },
   "debug":false,
   "resultCount":0,
   "messages":{

   }
}
```


Get Entity Metadata Web Service

The Get Entity Metadata web service gets the metadata information related to a business entity type. The metadata information includes the relationship details of the specified business entity type with other business entity types.

Request URL

Use the POST method to run the Get Entity Metadata web service.

To run the Get Entity Metadata web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
GETENTITYMETADATA
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/GETENTITYMETADATA
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Request Body

Use the request body to specify the business entity type for which you want to retrieve the metadata information.

To specify the business entity type, use the following request body format:

```
{
   "input":{
      "EntityType":"<Entity Type>"
   }
}
```

The following sample request body specifies to retrieve the metadata information of the Customer business entity type:

```
{
   "input":{
      "EntityType":"Customer"
   }
}
```

Sample Response

The following sample response shows the metadata information of the Customer business entity type:

```
{
   "input":{
      "EntityType":"Customer"
   },
   "entity":{
      "urlEndPoint":"http://localhost:8080/MDMBDRMCustomer360/v4.0/Customer360/",
      "displayField":"Name",
      "pkColumnLength":0,
      "inboundLabel":[
         "Duplicate",
         "Household"
      ],
      "outboundLabel":[
         "Duplicate",
         "Household"
      ],
      "viewfields":{
         "field":[
            {
               "value":"id",
               "name":"id"
            },
            {
               "value":"Source",
               "name":"LMT_SOURCE_NAME"
            },
            {
               "value":"Name",
               "name":"Name"
            }
         ]
      },
      "partOfLayout":false,
      "attributes":{
         "attribute":[
            [
               [
                  {
                     "type":"EXACT"
                  }
               ],
               "Name",
               60,
               "true",
               "false"
```

```
                ],
                [
                   [
                      {
                         "facet":[
                            {
                               "value":"Male",
                               "id":"M"
                            },
                            {
                               "value":"Female",
                               "id":"F"
                            }
                         ],
                         "type":"Enum"
                      }
                   ],
                   "Gender",
                   2,
                   "true",
                   "true"
                ],
                [
                   [
                      {
                         "facet":[
                            {
                               "value":"",
                               "id":"0",
                               "min":"18",
                               "max":"100"
                            }
                         ],
                         "type":"Range"
                      }
                   ],
                   "Age",
                   5,
                   "true",
                   "false",
                   "integer"
                ]
             ]
          },
          "entityConfig":{
             "Hbase Master":"localhost:60000",
             "Hbase Zookeeper ClientPort":"2181",
             "Hbase Zookeeper
Quorum":"torarch2.informatica.com,torarch3.informatica.com,torarch4.informatica.com",
             "Hbase Distributed":"true",
             "Hbase Scan CacheSize":"5000",
             "Hbase Scan BatchSize":"25000"
          },
          "entityOptions":{
             "Name":{
                "options":{
                   "Aggregate":"false",
                   "Filterable":"true"
                }
             },
             "Gender":{
                "options":{
                   "Aggregate":"true",
                   "Filterable":"true"
                }
             },
             "Age":{
                "options":{
                   "Aggregate":"false",
                   "Filterable":"true"
                }
```

```
            }
         }
      },
      "debug":false,
      "resultCount":0,
      "messages":{

      }
   }
```

Get Entity Relationship Web Service

The Get Entity Relationship web service retrieves all the related records of a record.

Request URL

Use the POST method to run the Get Entity Relationship web service.

To run the Get Entity Relationship web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
GETENTITYRELATIONSHIP
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/GETENTITYRELATIONSHIP
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Request Body

Use the request body to specify the parameters for the input record.

To specify the input record, use the following request body format:

```
{
   "input":{
      "source":"<Source Column>",
      "id":"<Primary Key Column>",
      "entityType":"<Entity Type>",
      "nodeLimit":"<Number of Records>",
      "depth":"<Number of Child Levels>"
```

```
   },
   "aggregateEntityTypes":[
      "<Entity Type 1>",
      "<Entity Type 2>",
      ...
      "<Entity Type N>"
   ]
}
```

The request body format uses the following parameters:

Source Column

Source name of the input record.

Primary Key Column

Primary key column value of the input record.

Entity Type

Name of the business entity type to which the input record belongs.

Number of Records

Optional. Maximum number of related records that you want to retrieve. Default is 500.

Number of Child Levels

Optional. Maximum number of child levels to include in the response. Default is 5.

Entity Type 1, 2,..N

Optional. Name of the business entity types based on which you want to aggregate the related records.

The following sample request body specifies the input record and the business entity types based on which you want to aggregate the related records:

```
{
   "input":{
      "source":"CRM",
      "id":"10000003",
      "entityType":"Customer",
      "nodeLimit":"3",
      "depth":"2"
   },
   "aggregateEntityTypes":[
      "Customer",
      "Transaction"
   ]
}
```

Sample Response

The following response shows the related records of a record whose identifier is 10000003:

```
{
   "input":{
      "depth":"2",
      "entityType":"Customer",
      "nodeLimit":"3",
      "source":"CRM",
      "id":"10000003"
   },
   "excludeEntityTypes":[
      "Customer",
      "Transaction"
   ],
   "excludeRelationshipTypes":[

   ],
   "aggregateEntityTypes":[
      "Customer",
```

```
   "Transaction"
],
"aggregateRelationshipTypes":[

],
"entities":[
   {
      "id":"1",
      "entityType":"Customer",
      "displayValue":"Customer",
      "attributes":{

      },
      "isPivot":false,
      "hasMore":false,
      "isAggregate":true,
      "count":15,
      "aggregateList":[
         [
            "10000004",
            "CRM",
            "Suzzy Rapp",
            "6625 The Corners Parkway"
         ],
         [
            "10000005",
            "CRM",
            "David Atwood",
            "6625 The Corners Parkway"
         ]
      ]
   },
   {
      "id":"10000003",
      "source":"CRM",
      "entityType":"Customer",
      "displayValue":"James Rapp",
      "attributes":{
         "MaritalStatus":"",
         "Networth":"141",
         "State":"GA",
         "Gender":"M",
         "Age":"44",
         "Name":"James Rapp"
      },
      "isPivot":true,
      "hasMore":false,
      "isAggregate":false,
      "count":0
   },
   {
      "id":"7001007",
      "source":"SAP",
      "entityType":"Product",
      "displayValue":"Mortgage",
      "attributes":{
         "ProductName":"Mortgage"
      },
      "isPivot":false,
      "hasMore":false,
      "isAggregate":false,
      "count":0
   }
],
"relations":[
   {
      "id":"10000003#7001007",
      "label":"owns",
      "fromEntity":{
         "id":"10000003",
         "source":"CRM"
```

```
            },
            "toEntity":{
               "id":"7001007",
               "source":"SAP"
            },
            "isAggregate":false,
            "count":0
         },
         {
            "id":"10000003#7001010",
            "label":"owns",
            "fromEntity":{
               "id":"10000003",
               "source":"CRM"
            },
            "toEntity":{
               "id":"7001010",
               "source":"SAP"
            },
            "isAggregate":false,
            "count":0
         }
      ],
      "debug":false,
      "resultCount":0,
      "messages":{

      }
}
```


Get All Relationships Web Service

The Get All Relationships web service gets the relationship details between two records.

Request URL

Use the POST method to run the Get All Relationships web service.

To run the Get All Relationships web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
GETALLRELATIONSHIPS
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/GETALLRELATIONSHIPS
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Request Body

Use the request body to specify the input records for which you want to retrieve the relationship details.

To specify the input records, use the following request body format:

```
{
   "input":{
      "startEntitySource":"<Source Column 1>",
      "startEntityId":"<Primary Key Column 1>",
      "startEntityType":"<Entity Type 1>",
      "targetEntitySource":"<Source Column 2>",
      "targetEntityId":"<Primary Key Column 2>",
      "targetEntityType":"<Entity Type 2>",
      "nodeLimit":"<Number of Records>",
      "depth":"<Number of Child Levels>"
   }
}
```

The request body format uses the following parameters:

Source Column Name 1, 2

Source name of the input records.

Primary Key Column 1, 2

Primary key column value of the input records.

Entity Type 1, 2

Name of the business entity types to which the input records belong.

Number of Records

Optional. Maximum number of related records that you want to retrieve. Default is 500.

Number of Child Levels

Optional. Maximum number of child levels to include in the response. Default is 5.

The following sample request body specifies two input records whose identifiers are 10000003 and 60000026:

```
{
   "input":{
      "startEntitySource":"CRM",
      "startEntityId":"10000003",
      "startEntityType":"Customer",
      "targetEntitySource":"SAP",
      "targetEntityId":"60000026",
      "targetEntityType":"Transaction",
      "nodeLimit":"150",
      "depth":"5"
   }
}
```

Sample Response

The following response shows the relationship details between two records:

```
{
   "input":{
      "startEntityType":"Customer",
      "targetEntityId":"7001008",
      "depth":"5",
      "targetEntityType":"Product",
      "startEntityId":"10000003",
```

```
      "nodeLimit":"150",
      "startEntitySource":"CRM",
      "targetEntitySource":"SAP"
   },
   "excludeEntityTypes":[

   ],
   "excludeRelationshipTypes":[

   ],
   "aggregateEntityTypes":[

   ],
   "aggregateRelationshipTypes":[

   ],
   "entities":[
      {
         "id":"10000003",
         "source":"CRM",
         "entityType":"Customer",
         "displayValue":"James Rapp",
         "attributes":{
            "MaritalStatus":"",
            "Networth":"141",
            "State":"GA",
            "Gender":"M",
            "Age":"44",
            "Name":"James Rapp"
         },
         "isPivot":true,
         "hasMore":false,
         "isAggregate":false,
         "count":0
      },
      {
         "id":"7001008",
         "source":"SAP",
         "entityType":"Product",
         "displayValue":"Bank account or service",
         "attributes":{
            "ProductName":"Bank account or service"
         },
         "isPivot":false,
         "hasMore":false,
         "isAggregate":false,
         "count":0
      }
   ],
   "relations":[
      {
         "id":"10000003#7001008",
         "label":"owns",
         "fromEntity":{
            "id":"10000003",
            "source":"CRM"
         },
         "toEntity":{
            "id":"7001008",
            "source":"SAP"
         },
         "isAggregate":false,
         "count":0
      }
   ],
   "debug":false,
   "resultCount":0,
   "messages":{

   }
}
```

Get Entity Details Web Service

The Get Entity Details web service retrieves the details of a record that you specify.

Request URL

Use the POST method to run the Get Entity Details web service.

To run the Get Entity Details web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
GETENTITYDETAILS
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/GETENTITYDETAILS
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Request Body

Use the request body to specify the parameters for the input record.

To specify the input record, use the following request body format:

```
{
   "input":{
      "source":"<Source Column Name>",
      "id":"<Primary Key Column>",
      "entityType":"<Entity Type>"
   }
}
```

The request body format uses the following parameters:

Source Column Name

Source name of the input record.

Primary Key Column

Primary key column value of the input record.

Entity Type

Name of the business entity type to which the input record belongs.

The following sample request body specifies the input record that belongs to the Customer business entity type:

```
{
   "input":{
      "source":"CRM",
      "id":"10000005",
      "entityType":"Customer"
   }
}
```

Sample Response

The following response shows the input record details:

```
{
   "input":{
      "entityType":"Customer",
      "source":"CRM",
      "id":"10000005"
   },
   "output":{
      "Address":"6625 The Corners Parkway",
      "FirstName":"David",
      "GroupNumber":"bdaed85a-dcf8-4330-9716-e53ee19478fb",
      "LMT_SOURCE_NAME":"CRM",
      "City":"Norcross",
      "Gender":"M",
      "Prefix":"",
      "Postcode":"30092",
      "Name":"David Atwood",
      "MaritalStatus":"",
      "Networth":"198",
      "Phone":"(404) 448-5210",
      "State":"GA",
      "Country":"United States",
      "LastName":"Atwood",
      "id":"10000005",
      "Age":"17"
   },
   "debug":false,
   "resultCount":0,
   "messages":{

   }
}
```

Managing the Relationships

You can create a relationship or remove the relationship between two records.

To manage the relationship between two records, use the following RESTful web services:

- Create Relationship
- Remove Relationship

Note: You can access these RESTful web services only when you include the relationship configuration file in the WAR file that you generate for the RESTful web services.

Create Relationship Web Service

The Create Relationship web service creates relationship between two records.

Request URL

Use the POST method to run the Create Relationship web service.

To run the Create Relationship web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
CREATERELATIONSHIP
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/CREATERELATIONSHIP
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Request Body

Use the request body to specify the records that you want to relate.

To specify the business entity type, use the following request body format:

```
{
   "input":{
      "startEntitySource":"<Source Column 1>",
      "startEntityId":"<Primary Key Column 1>",
      "startEntityType":"<Entity Type 1>",
      "targetEntitySource":"<Source Column 2>",
      "targetEntityId":"<Primary Key Column 2>",
      "targetEntityType":"<Entity Type 2>",
      "relationship":"<Relationship Name>"
   }
}
```

The request body format uses the following parameters:

Source Column 1, 2

Source name of the input records.

Primary Key Column 1, 2

Primary key column value of the input records.

Entity Type 1, 2

Name of the business entity types to which the input records belong.

Number of Records

Optional. Maximum number of related records that you want to retrieve. Default is 500.

Number of Child Levels

Optional. Maximum number of child levels to include in the response. Default is 5.

Relationship Name

Name of the relationship that you want to create.

The following sample request body specifies the records that you want to relate:

```
{
   "input":{
      "startEntitySource":"CRM",
      "startEntityId":"10000003",
      "startEntityType":"Customer",
      "targetEntitySource":"SAP",
      "targetEntityId":"60000026",
      "targetEntityType":"Product",
      "relationship":"Owns"
   }
}
```

Sample Response

The following sample response shows the relationship created between two records:

```
{
  "input": {
    "relationship": "Duplicate",
    "targetEntitySource": "CRM",
    "startEntityType": "Customer",
    "startEntitySource": "CRM",
    "targetEntityId": "10000006",
    "targetEntityType": "Customer",
    "startEntityId": "10000000"
  },
  "debug": false,
  "resultCount": 0,
  "messages": {
    "INFO": "Relationship created successfully"
  }
}
```

Remove Relationship Web Service

The Remove Relationship web service remove the relationship between two records.

Request URL

Use the POST method to run the Remove Relationship web service.

To run the Remove Relationship web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
REMOVERELATIONSHIP
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/REMOVERELATIONSHIP
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Request Body

Use the request body to specify the input records and their relationship that you want to remove.

To specify the business entity type, use the following request body format:

```
{
   "input":{
      "startEntitySource":"<Source Column 1>",
      "startEntityId":"<Primary Key Column 1>",
      "startEntityType":"<Entity Type 1>",
      "targetEntitySource":"<Source Column 2>",
      "targetEntityId":"<Primary Key Column 2>",
      "targetEntityType":"<Entity Type 2>",
      "relationship":"<Relationship Name>"
   }
}
```

The request body format uses the following parameters:

Source Column 1, 2

Source name of the input records.

Primary Key Column 1, 2

Primary key column value of the input records.

Entity Type 1, 2

Name of the business entity types to which the input records belong.

Relationship Name

Name of the relationship that you want to remove.

The following sample request body specifies to remove the Household relationship between the input two records:

```
{
   "input":{
```

```
        "startEntitySource":"CRM",
        "startEntityId":"10000000",
        "startEntityType":"Customer",
        "targetEntitySource":"CRM",
        "targetEntityId":"10000006",
        "targetEntityType":"Customer",
        "relationship":"Duplicate"
    }
}
```

Sample Response

The following sample response shows the relationship that you removed:

```
{
  "input": {
    "relationship": "Duplicate",
    "targetEntitySource": "CRM",
    "startEntityType": "Customer",
    "startEntitySource": "CRM",
    "targetEntityId": "10000006",
    "targetEntityType": "Customer",
    "startEntityId": "10000000"
  },
  "debug": false,
  "resultCount": 0,
  "messages": {
    "INFO": "Relationship removed successfully"
  }
}
```

Viewing the Relationship Graph

After you create relationships between the business entity types, the relationships result in a relationship graph. You can search for a record and view all its related records in the relationship graph.

1. Open a browser, and enter the URL in the following format:

   ```
   https://<Host>:<Port>/<Relationship UI WAR Name>
   ```

   The URL format uses the following parameters:

   - `Host`. Name or IP address of the machine on which you deploy the relationship graph user interface WAR file.
   - `Port`. Port through which the host listens.
   - `Relationship UI WAR Name`. Name of the relationship graph user interface WAR file that you deploy on the Tomcat container. Default name is `bdrm-ui`.

   The following URL uses the default relationship user interface WAR file name:

   ```
   https://BDRMServer:8080/bdrm-ui
   ```

   The **Home** page appears and displays the business entity types for which you configured the `RESTEndPoint` parameter in the relationship configuration file.
2. Click the business entity type within which you want to search for records.
3. Specify the values for the searchable fields, and click **Search**.

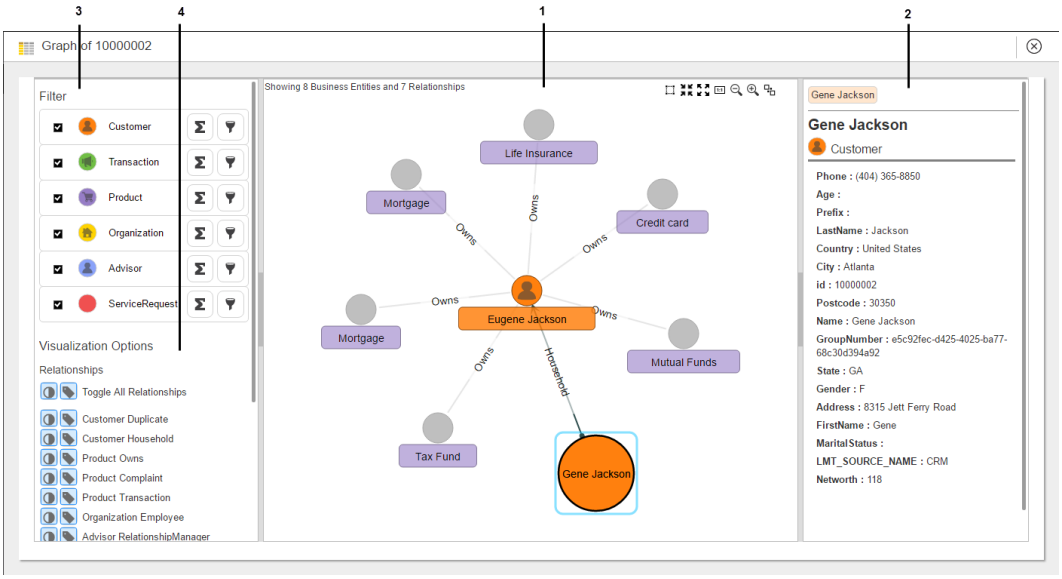
   The search results appear.
4. Click the primary key column value of a record for which you want to view the relationship graph.

   The relationship graph appears.

Relationship Graph User Interface

The relationship graph displays a graph, filters, and column values for the selected record. The graph displays all the related records of the selected record. You can filter the records in the graph based on the filters that are available for the business entity type.

The following image shows a sample relationship graph:



1. Graph panel
2. Fields panel
3. Filters panel
4. Visualization Options panel

A relationship graph includes the following panels:

Graph panel

Displays how the selected record is connected to other records. The graph changes based on the filters that you apply.

Fields panel

Displays the columns related to a node that you select in the graph. The columns that appear in the **Fields** panel depend on the parameters that you configure in the relationship configuration file.

Filters panel

Displays the filters that you can use to filter the records. You can filter the records based on the business entity types and their column values. You can also aggregate the records based on the columns that you specify.

Visualization Options

Allows you to highlight the relationships and business entity types. You can also hide or show the labels of the relationships and nodes.

Filtering the Records

You can filter the records based on the business entity types and their column values.

1. Open the relationship graph.

   By default, the graph displays the selected record and all its related records.
2. To remove all the records related to a business entity type from the graph, under **Filter**, clear the business entity type.

   The graph refreshes and removes the records related to the business entity type from the graph.
3. To include all the records related to a business entity type in the graph, under **Filter**, select the business entity type.
4. To filter the records based on the column values of a business entity type, perform the following tasks:

   a. Under **Filter**, click the filter icon  of a business entity type whose columns you want to use to filter the records.

      A list of columns appear. The columns that are available in the list are based on the parameters that you configure in the relationship configuration file.

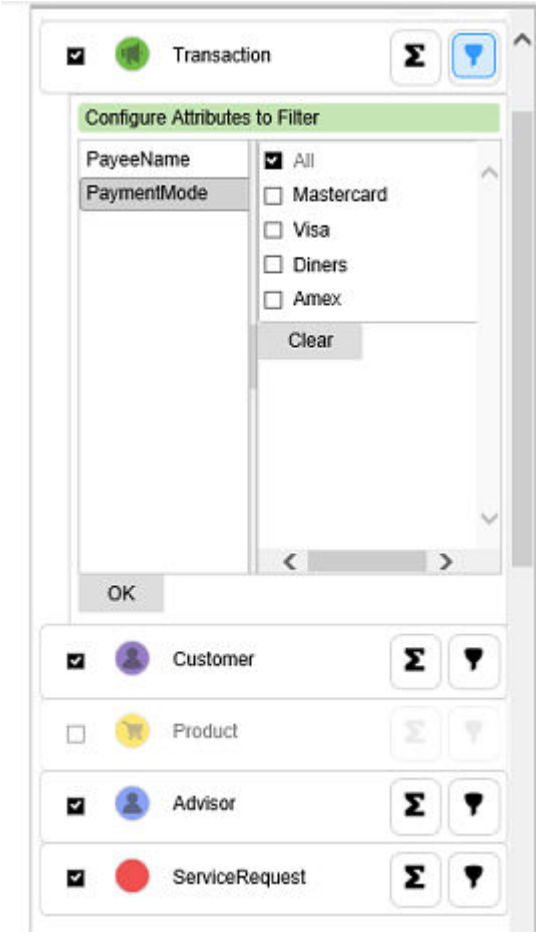
   b. Select a column.

      If you have configured the column as an exact filter, a text box appears. If you have configured the column as an enumeration filter, a list of enumeration values appear.

   c. For an exact filter, type the exact column value based on which you want to filter the records.

   d. For an enumeration filter, select the enumeration value based on which you want to filter the records.

      The following image shows the enumeration values configured for the `PaymentMode` column:



e. Click **OK**.

The graph refreshes to reflect the filtered records.

Aggregating the Records

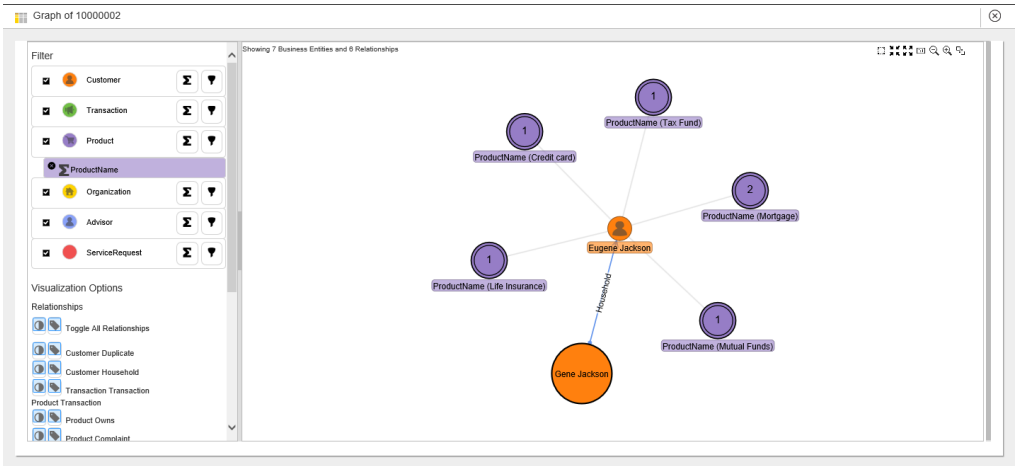
You can aggregate the records based on the selected columns of a business entity type.

1. Under **Filter**, click the aggregate icon Σ of a business entity type whose columns you want to use to aggregate the business entity types.

2. Select the column based on which you want to aggregate the records, and click **OK**.

   The graph refreshes to reflect the aggregated records.

   The following image shows Gene Jackson's related records that are aggregated based on the `ProductName` column:



Setting the Visualization Options

You can select to show or hide the labels of the relationships and nodes in the relationship graph. You can also turn the highlight effect on or off for the relationships and nodes.

In the **Visualizations Options** panel, click the following icons that you can find against each relationship and business entity:

Icon	Description
	Turns the highlight effect on or off.
	Shows or hides the labels of the relationships or nodes.

Modifying the Graph View

You can zoom in, zoom out, and perform other actions in the graph to modify the view of the graph.

In the **Graph** panel, click the following icons that you can find in the upper right corner of the panel:

Icon	Description
	Adjusts the graph to highlight how the selected node is related to the main node.
	Aligns the graph to the center of the panel.

Icon	Description
	Zooms in or zooms out the graph so that the entire graph fits the panel.
	Resets the graph to its original size.
	Zooms out to reduce the size of the nodes.
	Zooms in to increase the size of the nodes.
	Resets the graph to its default layout.

CHAPTER 6

Loading Linked and Consolidated Data into Hive

This chapter includes the following topics:

Loading Linked and Consolidated Data into Hive Overview

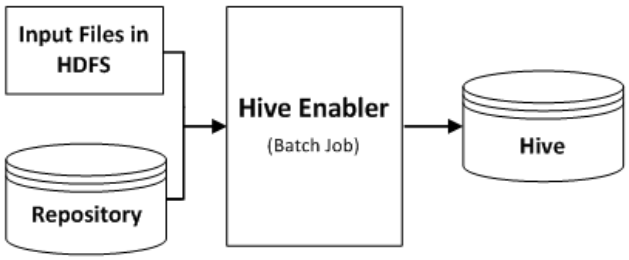
You can load the linked and consolidated data that you persist in the repository or in HDFS into Hive. You can perform analytics in Hive based on your requirement.

Loading Linked Data from the Repository

Use the Hive enabler job to load the linked data into Hive from the repository or to link a Hive table to the repository table.

If you load the linked data into Hive, to update the incremental linked data in Hive, you must run the Hive enabler job to drop the existing output table and re-create the output table. If you link a Hive table to the repository table, you do not have to run the Hive enabler job again because the linked data persists in the repository. You can use the Hive table to access the linked data in the repository.

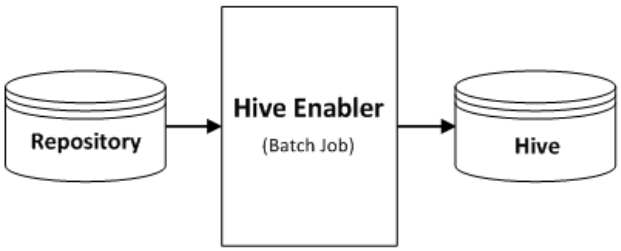
The following image shows how the Hive enabler job loads the linked data into Hive:



To load the linked data into Hive from the repository, run the Hive enabler job without the link option. The Hive enabler job performs the following tasks:

1. Joins the input data in HDFS and the primary key table that contains the cluster details of the input data in the repository.
2. Loads the joined data into Hive.

The following image shows how the Hive enabler job links the Hive table to the repository table:



To link a Hive table to the repository table, run the Hive enabler job with the link option. You can use the Hive table to access the linked data in the repository.

Note: In the configuration file, if you set `StoreAllFields` to false, the repository does not persist all the columns but persists only the columns that you use to index the input data. If you want to view all the columns in Hive, ensure that you set `StoreAllFields` to true in the configuration file when you link the input data.

Running the Hive Enabler Job

Run the Hive enabler job to load the linked data into Hive or to link a Hive table to a repository table.

To run the Hive enabler job, use the `run_hiveEnabler.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Loading Linked Data into Hive from the Repository

Use the following command to run the `run_hiveEnabler.sh` script without the link option:

```
run_hiveEnabler.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--outputtable=output_table_name
--hiveserver=Hive_server_host_name:port
```

```
--forceCopy
[--reducer=number_of_reducer_jobs]
[--hiveuser=user_name]
[--hivepassword=password]
[--hivedb=database_name]
[--outputpath=directory_for_output_files]
```

The following table describes the options and arguments that you can specify to run the `run_hiveEnabler.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the input files in HDFS.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. In a high availability-enabled cluster, prefix the absolute path with the logical URI of the cluster. Use the following format for the logical URI of a cluster: `hdfs://<nameservice ID>` The Hive enabler job uses the working directory to store the library files.
--outputtable	output_table_name	Unique name for the output table in Hive to which you want to load the linked data.
--hiveserver	Hive_server_host_name:port	Host name of the Hive server and the port number on which the Hive server listens. Use the following format to specify the `--hiveserver` parameter: `<Hive Server Host Name>:<Port Number>`
--hivedb	database_name	Optional. Name of the Hive database on which you want to create the output table. If you do not specify the name of the Hive database, the Hive enabler job creates the output table in the default database.
--hiveuser	user_name	Optional. Name of the user to access the Hive database. **Note:** Ensure that the user or the role to which the user belongs is granted the ALL privilege for the Hive database.
--hivepassword	password	Optional. Password for the user to access the Hive database.

Option	Argument	Description
--forceCopy		Copies the dependent library files to HDFS. Use this option only when you run the Hive enabler job for the first time.
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. In a high availability-enabled cluster, prefix the absolute path with the logical URI of the cluster. Use the following format for the logical URI of a cluster: `hdfs://<nameservice ID>` By default, the batch job loads the output files to the working directory in HDFS.

For example, the following command loads the linked data into Hive:

```
run_hiveEnabler.sh --config=/usr/local/config/Configuration.xml --input=/usr/hdfs/Source
--hdfsdir=hdfs://R360nameservice/usr/hdfs/Hive --outputtable=HiveOutput --
hiveserver=Analytics1:20000 --forceCopy
```

Linking a Hive Table to the Repository Table

Use the following command to run the `run_hiveEnabler.sh` script with the link option:

```
run_hiveEnabler.sh

--config=configuration_file_name

--linkHBase

--outputtable=output_table_name

--hiveserver=Hive_server_host_name:port

--forceCopy

[--reducer=number_of_reducer_jobs]

[--hiveuser=user_name]

[--hivepassword=password]

[--hivedb=database_name]
```

The following table describes the options and arguments that you can specify to run the `run_hiveEnabler.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run.
--outputtable	output_table_name	Unique name for the output table in Hive that you want to link to the repository table.

Option	Argument	Description
--hiveserver	Hive_server_host_name:port	Host name of the Hive server and the port number on which the Hive server listens. Use the following format to specify the `--hiveserver` parameter: `<Hive Server Host Name>:<Port Number>`
--hivedb	database_name	Optional. Name of the Hive database on which you want to create the output table. If you do not specify the name of the Hive database, the Hive enabler job creates the output table in the default database.
--hiveuser	user_name	Optional. Name of the user to access the Hive database. **Note:** Ensure that you grant all the privileges to the user or the role to which the user belongs for the Hive database.
--hivepassword	password	Optional. Password for the user to access the Hive database.
--linkHBase		Links the output table in Hive to the repository table that contains the linked data.
--forceCopy		Copies the dependent library files to HDFS. Use this option only when you run the Hive enabler job for the first time.

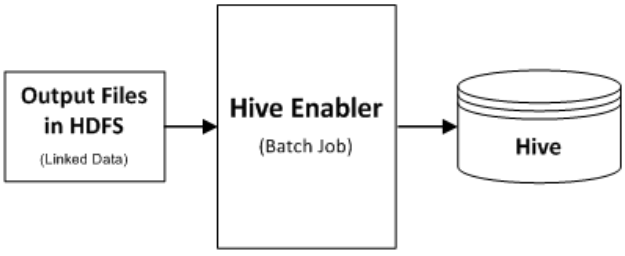
For example, the following command links the Hive table to the repository table:

```
run_hiveEnabler.sh --config=/usr/local/config/Configuration.xml --linkHBase --
outputtable=HiveOutput --hiveserver=Analytics1:20000 --forceCopy
```

Loading Linked Data from HDFS

Use the Hive enabler job to load the linked data that you persist in HDFS into Hive. After loading the initial linked data, you can also use the Hive enabler job to incrementally update the linked data in Hive.

The following image shows how the data is loaded into Hive if you persist the linked data in HDFS:



To load the linked data into Hive from HDFS, perform the following tasks:

1. Run the Hive enabler job in the initial mode.

   The Hive enabler job loads the linked data into Hive from HDFS.

2. To incrementally update the linked data in Hive, run the Hive enabler job in the incremental mode.

   You must run the Hive enabler job in the incremental mode whenever you want to update the linked data in Hive.

Running the Hive Enabler Job

Run the Hive enabler job in the initial mode to load the initial linked data into Hive. To incrementally update the linked data in Hive, run the Hive enabler job in the incremental mode to load the incremental data into Hive.

To run the Hive enabler job, use the `run_hiveEnabler.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Initial Mode

Use the following command to run the `run_hiveEnabler.sh` script in the initial mode:

```
run_hiveEnabler.sh
--config=configuration_file_name
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--outputtable=output_table_name
--hiveserver=Hive_server_host_name:port
--skipHBase
--forceCopy
[--reducer=number_of_reducer_jobs]
[--hiveuser=user_name]
[--hivepassword=password]
[--hivedb=database_name]
```

The following table describes the options and arguments that you can specify to run the `run_hiveEnabler.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the directory that contains linked data. If you run the initial clustering job without the `--outputpath` parameter, you can find the linked data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/output/dir/pass-join` If you run the initial clustering job with the `--outputpath` parameter, you can find the linked data in the following directory: `<Output Directory in HDFS>/batch-cluster/output/dir/pass-join`
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run.

Option	Argument	Description
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. In a high availability-enabled cluster, prefix the absolute path with the logical URI of the cluster. Use the following format for the logical URI of a cluster: `hdfs://<nameservice ID>` The Hive enabler job uses the working directory to store the library files.
--outputtable	output_table_name	Unique name for the output table in Hive to which you want to load the linked data.
--hiveserver	Hive_server_host_name:port	Host name of the Hive server and the port number on which the Hive server listens. Use the following format to specify the `--hiveserver` parameter: `<Hive Server Host Name>:<Port Number>`
--hivedb	database_name	Optional. Name of the Hive database on which you want to create the output table. If you do not specify the name of the Hive database, the Hive enabler job creates the output table in the default database.
--hiveuser	user_name	Optional. Name of the user to access the Hive database. **Note:** Ensure that the user or the role to which the user belongs is granted the ALL privilege for the Hive database.
--hivepassword	password	Optional. Password for the user to access the Hive database.
--skipHBase		Indicates that the linked data is in HDFS.
--forceCopy		Copies the dependent library files to HDFS. Use this option only when you run the Hive enabler job for the first time.
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. In a high availability-enabled cluster, prefix the absolute path with the logical URI of the cluster. Use the following format for the logical URI of a cluster: `hdfs://<nameservice ID>` By default, the batch job loads the output files to the working directory in HDFS.

For example, the following command loads the linked data into Hive from HDFS:

```
run_hiveEnabler.sh --config=/usr/local/config/Configuration.xml --input=/usr/hdfs/
workingdir/batch-cluster/MDMBDRM_931211654144593570/output/dir/pass-join --
hdfsdir=hdfs://r360nameservice/usr/hdfs/workingdir --outputtable=HiveOutput --
hiveserver=Analytics1:20000 --skipHBase --forceCopy
```

Incremental Mode

Use the following command to run the `run_hiveEnabler.sh` script in the incremental mode:

```
run_hiveEnabler.sh
```

```
--config=configuration_file_name

--input=input_file_in_HDFS

--hdfsdir=working_directory_in_HDFS

--outputtable=output_table_name

--hiveserver=Hive_server_host_name:port

--skipHBase

--incremental

[--hiveuser=user_name]

[--hivepassword=password]

[--hivedb=database_name]

[--reducer=number_of_reducer_jobs]

[--outputpath=directory_for_output_files]
```

The following table describes the options and arguments that you can specify to run the `run_hiveEnabler.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--input	input_file_in_HDFS	Absolute path to the directory that contains linked data. If you run the initial clustering job without the `--outputpath` parameter, you can find the linked data in the following directory: `<Working Directory in HDFS>/batch-cluster/<Job ID>/output/dir/pass-join` If you run the initial clustering job with the `--outputpath` parameter, you can find the linked data in the following directory: `<Output Directory in HDFS>/batch-cluster/output/dir/pass-join`
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS.In a high availability-enabled cluster, prefix the absolute path with the logical URI of the cluster. Use the following format for the logical URI of a cluster: `hdfs://<nameservice ID>` The Hive enabler job uses the working directory to store the library files.
--outputtable	output_table_name	Name of the table in Hive that contains the linked data.
--hiveserver	Hive_server_host_name:port	Host name of the Hive server and the port number on which the Hive server listens. Use the following format to specify the `--hiveserver` parameter: `<Hive Server Host Name>:<Port Number>`

Option	Argument	Description
--hivedb	database_name	Optional. Name of the Hive database that contains the output table. If you do not specify the name of the Hive database, the Hive enabler job uses the default database.
--hiveuser	user_name	Optional. Name of the user to access the Hive database. **Note:** Ensure that you grant all the privileges to the user or the role to which the user belongs for the Hive database.
--hivepassword	password	Optional. Password for the user to access the Hive database.
--skipHBase		Indicates that the linked data is in HDFS.
--incremental		Runs the Hive enabler job in the incremental mode. The Hive enabler job updates the output table with the incremental data .
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. In a high availability-enabled cluster, prefix the absolute path with the logical URI of the cluster. Use the following format for the logical URI of a cluster: `hdfs://<nameservice ID>` By default, the batch job loads the output files to the working directory in HDFS.

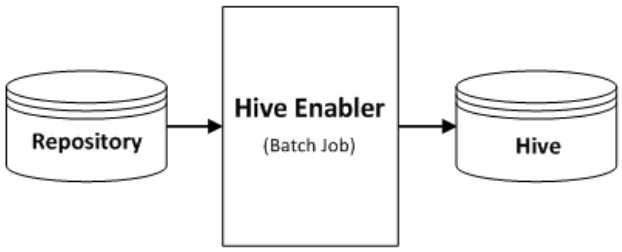
For example, the following command loads the incremental linked data into Hive from HDFS:

```
run_hiveEnabler.sh --config=/usr/local/config/Configuration.xml --input=/usr/hdfs/
workingdir/batch-cluster/MDMBDRM_931211654144593970/output/dir/pass-join --hdfsdir=/usr/
hdfs/workingdir --outputtable=HiveOutput --hiveserver=Analytics1:20000 --skipHBase --
incremental
```

Loading Consolidated Data from the Repository

Use the Hive enabler job to link a Hive table to the preferred records table in the repository. If you link a Hive table to the preferred records table in the repository, the data continues to persist in the repository, and the Hive table accesses data from the repository. You do not have to run the Hive enabler job again for the incremental data.

The following image shows how the Hive enabler job links the Hive table to the repository table:



Note: In the configuration file, if you set `StoreAllFields` to false, the repository does not persist all the columns but persists only the columns that you use to index the input data. If you want to view all the columns in Hive, ensure that you set `StoreAllFields` to true in the configuration file when you link the input data.

Running the Hive Enabler Job

Run the Hive enabler job to link a Hive table to the preferred records table in the repository.

To run the Hive enabler job, use the `run_hiveEnabler.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_hiveEnabler.sh` script:

```
run_hiveEnabler.sh
--config=configuration_file_name
--consolidate
--linkHBase
--outputtable=output_table_name
--hiveserver=Hive_server_host_name:port
--forceCopy
[--reducer=number_of_reducer_jobs]
[--hiveuser=user_name]
[--hivepassword=password]
[--hivedb=database_name]
```

The following table describes the options and arguments that you can specify to run the `run_hiveEnabler.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run.
--outputtable	output_table_name	Unique name for the output table in Hive that you want to link to the repository table.

Option	Argument	Description
--consolidate		Indicates to link the preferred records table in the repository that contains the consolidated data to the Hive table.
--linkHBase		Links the output table in Hive to the preferred records table in the repository.
--hiveserver	Hive_server_host_name:port	Host name of the Hive server and the port number on which the Hive server listens. Use the following format to specify the `--hiveserver` parameter: `<Hive Server Host Name>:<Port Number>`
--hivedb	database_name	Optional. Name of the Hive database on which you want to create the output table. If you do not specify the name of the Hive database, the Hive enabler job creates the output table in the default database.
--hiveuser	user_name	Optional. Name of the user to access the Hive database. **Note:** Ensure that the user or the role to which the user belongs is granted the ALL privilege for the Hive database.
--hivepassword	password	Optional. Password for the user to access the Hive database.
--forceCopy		Copies the dependent library files to HDFS. Use this option only when you run the Hive enabler job for the first time.

For example, the following command links the Hive table to the preferred records table:

```
run_hiveEnabler.sh --config=/usr/local/config/Configuration.xml --linkHBase --
consolidate --outputtable=HiveOutput --hiveserver=Analytics1:20000 --forceCopy
```

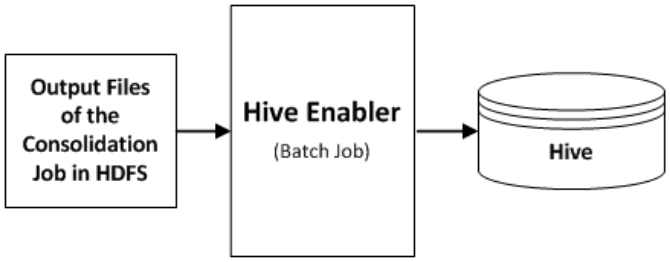
Loading Consolidated Data from HDFS

Use the Hive enabler job to load the consolidated data that you persist in HDFS into Hive. The Hive enabler job uses the output files of a consolidation job as input.

After you load the initial consolidated data into Hive, you cannot incrementally update the consolidated data. To update the consolidated data in Hive, you must reload the consolidated data of the entire dataset into Hive.

To create the consolidated data of the entire dataset, use the output files of an initial clustering job that you run in the incremental mode with the `--consolidate` option as the input for the consolidation job. You can then use the output files of the consolidation job as the input for the Hive enabler job.

The following image shows how the consolidated data in HDFS is loaded into Hive:



To load the consolidated data in HDFS into Hive, perform the following tasks:

1. Run the Hive enabler job.

   The Hive enabler job loads the consolidated data in HDFS into Hive.
2. To update the consolidated data in Hive, perform the following tasks:
   a. Run the initial clustering job in the incremental mode with the `--consolidate` option.
   b. Run the consolidation job that uses the output files of the initial clustering job.
   c. Drop the output table in Hive.
   d. If the `<Output table>_internal` table exists in Hive, drop it.
   e. Run the Hive enabler job that uses the output files of the consolidation job.

Running the Hive Enabler Job

Run the Hive enabler job to load the consolidated data in HDFS into Hive.

To run the Hive enabler job, use the `run_hiveEnabler.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_hiveEnabler.sh` script:

```
run_hiveEnabler.sh
--config=configuration_file_name
--consolidate
--skipHBase
--input=input_file_in_HDFS
--hdfsdir=working_directory_in_HDFS
--outputtable=output_table_name
--hiveserver=Hive_server_host_name:port
--forceCopy
[--reducer=number_of_reducer_jobs]
[--hiveuser=user_name]
[--hivepassword=password]
[--hivedb=database_name]
```

The following table describes the options and arguments that you can specify to run the `run_hiveEnabler.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--consolidate		Indicates to load the consolidated data into the Hive table.
--skipHBase		Indicates that the consolidated data is persisted in HDFS.
--input	input_file_in_HDFS	Absolute path to the directory that contains the consolidated data. If you run the consolidation job without the `--outputpath` parameter, you can find the preferred records in the following directory: `<Working Directory in HDFS>/batch-consolidation/<Job ID>/consolidation-output` If you run the consolidation job with the `--outputpath` parameter, you can find the preferred records in the following directory: `<Output Directory in HDFS>/batch-consolidation/<Job ID>/consolidation-output`
--reducer	number_of_reducer_jobs	Optional. Number of reducer jobs that you want to run.
--hdfsdir	working_directory_in_HDFS	Absolute path to a working directory in HDFS. In a high availability-enabled cluster, prefix the absolute path with the logical URI of the cluster. Use the following format for the logical URI of a cluster: `hdfs://<nameservice ID>` The Hive enabler job uses the working directory to store the library files.
--outputtable	output_table_name	Unique name for the output table in Hive to which you want to load the consolidated data.
--hiveserver	Hive_server_host_name:port	Host name of the Hive server and the port number on which the Hive server listens. Use the following format to specify the `--hiveserver` parameter: `<Hive Server Host Name>:<Port Number>`
--hivedb	database_name	Optional. Name of the Hive database on which you want to create the output table. If you do not specify the name of the Hive database, the Hive enabler job creates the output table in the default database.
--hiveuser	user_name	Optional. Name of the user to access the Hive database. **Note:** Ensure that the user or the role to which the user belongs is granted the ALL privilege for the Hive database.
--hivepassword	password	Optional. Password for the user to access the Hive database.

Option	Argument	Description
--forceCopy		Copies the dependent library files to HDFS. Use this option only when you run the Hive enabler job for the first time.
--outputpath	directory_for_output_files	Optional. Absolute path to a directory in HDFS to which the batch job loads the output files. Use a different directory when you rerun the batch job. If you want to use the same directory, delete all the files in the directory and rerun the job. In a high availability-enabled cluster, prefix the absolute path with the logical URI of the cluster. Use the following format for the logical URI of a cluster: `hdfs://<nameservice ID>` By default, the batch job loads the output files to the working directory in HDFS.

For example, the following command loads the consolidated data in HDFS into Hive:

```
run_hiveEnabler.sh --config=/usr/local/config/Configuration.xml --consolidate --
input=/usr/hdfs/workingdir/batch-consolidation/MDMBDRM_931211654144593570/consolidation-
output --hdfsdir=hdfs://r360nameservice/usr/hdfs/workingdir --outputtable=HiveOutput --
hiveserver=Analytics1:20000 --skipHBase --forceCopy
```

CHAPTER 7

Searching Data

This chapter includes the following topics:

Searching Data Overview

You can perform fuzzy or exact searches on the repository data based on the settings in the configuration file and the matching rules file. The fuzzy search can handle initials, aliases, common variations, prefixes, suffixes, transpositions, and word order. You can search for matching records or get a list of records in a specific cluster.

You can use the following methods to perform a search:

- RESTful web services
- Command line

Prerequisites

If you configure to perform distributed search, you must restart the repository server in all the region servers to activate the distributed search.

Searching Data by Using the RESTful Web Services

You can use the RESTful web services to search for matching records in the repository based on the matching criteria that you specify in the configuration file or the matching rules file. Use the JSON format to

specify the input parameters for the web services, and the web services return the results in the JSON format. Ensure that you have deployed the RESTful web services before you use them.

1. If you have secured the web services, run the Authenticate web service to generate the authentication token.
2. Run the Get Multisearch Layout web service.

   The Get Multisearch Layout web service gets the required fields based on the matching rules file. You must specify the required fields as the input parameters in the JSON format when you search for the matching records.
3. Run the Multisearch web service.

   The Multisearch web service searches for the matching records based on the searching and matching criteria that you specify in the matching rules file. Use the search layout that the Get Multisearch Layout web service returns to specify the input parameters for the Multisearch web service.
4. To get a list of records that are part of a cluster, perform the following tasks:
   a. Run the Get Cluster Layout web service to get the required fields based on the configuration file.

      You must specify the required fields as the input parameters in the JSON format to get the list of records that are part of a cluster.
   b. Run the Get Cluster web service to get the list of other records that are part of the same cluster to which the specified record belongs.
5. To get the preferred record of a cluster, perform the following tasks:
   a. Run the Get Preferred Record Layout web service. The web service returns the layout that you can use to specify the input parameters for the Get Preferred Record web service.
   b. Run the Get Preferred Record web service. The web service returns the preferred record for the specified cluster or generates the preferred record for the specified cluster in the run time.
6. To search for the matching records and get the preferred records for the matching records, perform the following tasks:
   a. Run the Get Multisearch Layout web service. The web service returns the layout that you can use to specify the input parameters for the Preferred Record Search web service.
   b. Run the Preferred Record Search web service. The web service gets the matching records for the input data and uses the cluster numbers of the matching records to get the preferred records of the clusters.

Authenticate Web Service

The Authenticate web service authenticates the user credentials that you specify in the web service request. After a successful authentication, the web service request returns an encoded authentication token that you can specify in the header of the subsequent web service requests. A web service request returns appropriate response only if the request contains a valid token or user credentials.

Note: Use the Authenticate web service only if you have secured the RESTful web services.

Request URL

Use the GET method to run the Authenticate web service.

To run the Authenticate web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
Authenticate
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/Authenticate
```

Request Header

Use the request header to specify the following headers:

Authorization

Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Sample Response

The following sample response shows the token generated after a successful authentication:

```
{
   "timeTaken":0.134,
   "token":"AQIC5wM2LY4Sfczqyo9VO8tpm3qq7vDAsf3BB9nzatkh_S8.*AAJTSQACMDEAAlNLABI4MzAzMj
   EwNzMzODg0ODAxNTkAAlMxAAA.*"
}
```

Note: An authentication token is valid for the web service requests that use the same WAR file and host name as that of the Authenticate web service request.

Get Multisearch Layout Web Service

The Get Multisearch Layout web service gets the search layout for the matching rules file that you use to generate the WAR file. The layout contains the required fields that you can specify as input parameters in the JSON format to run the MULTISEARCH web service.

Note: If you do not use a matching rules file to generate the WAR file, the Get Multisearch Layout web service does not return any search layout.

Request URL

Use the POST method to run the Get Multisearch Layout web service.

To run the Get Multisearch Layout web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Identifier for Configuration and
Matching Rules Files>/GetMultisearchLayout
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/GetMultisearchLayout
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:
`Basic <Encoded User Credentials>`

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Sample Response

The following sample response shows the search layout for a matching rules file:

```
{
   "sortField":"",
   "searchType":"Football",
   "resultlimit":0,
   "scoreThreshold":0,
   "ignoreMatch":false,
   "searchinput":{

   },
   "searchlayout":{
      "<Field 1>":"mandatory",
      "<Field 2>":"mandatory",
      "<Field 3>":"mandatory",
      "<Field 4>":"optional"
   },
   "searchResults":[

   ],
   "searchToken":-1,
   "pageLimit":10,
   "pageOffset":0,
   "totalCount":0,,
   "debug":false,
   "resultCount":0,
   "messages":{

   }
}
```

A search layout includes the following parameters:

sortField

Optional. Column name based on which you want to sort the search results lexicographically.

You can use one of the columns that you define in the `PZMAP` section of the configuration file to sort the search results. You can also use the parameter value as `score` to sort the results in the descending order of the search result scores. For example, `"sortField":"score"`

searchType

Name for the search.

resultlimit

Optional. Maximum number of records that you want to return. Default is 0, which indicates to return a maximum of 200 records.

scoreThreshold

Optional. Minimum matching score of the records that you want to include in the results. Default is 0, which indicates to include all the records in the results.

ignoreMatch

Optional. Indicates whether to perform matching on the records retrieved based on the key ranges. Set to true to ignore matching, and set to false to perform matching. Default is false.

searchlayout

List of required search fields that you must specify.

Use the following format to specify a search field:

```
"<Field 1>": "<Field Value>"
```

searchToken

Optional. Indicates whether to enable pagination for the search results when you perform a search for the first time. If you enable pagination, the search request returns a token with the search results. You can use the token in the subsequent requests to get the search results from cache to avoid performing the search again.

When you run a search request for the first time, set `searchToken=0` to enable pagination. In the subsequent requests, you can use the token that the search request returns. For example, `searchToken=1994262671816343815`. Set `searchToken=-1` to disable pagination. By default, pagination is disabled.

pageLimit

Optional. Maximum number of search results to return. The `pageLimit` parameter is applicable if you enable pagination. For example, `pageLimit=20` returns 20 search results. Default is 10.

pageOffset

Optional. Number of search results to skip. The `pageOffset` parameter is applicable if you enable pagination. For example, `pageOffset=40` indicates to skip the first 40 search results and return search results starting from 41. Default is 0.

Multisearch Web Service

The Multisearch web service generates key ranges, compares the key ranges with the repository data, and gets the matching records based on the searching and matching rules that you configure in the matching rules file.

Before you run the Multisearch web service, run the Get Multisearch Layout web service to get the search layout in the JSON format for the matching rules file. Based on the search layout, you can configure the input parameters for the Multisearch web service.

Note: If you do not use a matching rules file to generate the WAR file, the Multisearch web service does not return any results.

You can run the Multisearch web service in the debug mode. The debug mode returns performance metrics for the Multisearch web service. You can use the debug mode for troubleshooting purposes. To enable the debug mode, you must add the `Debug` parameter to the `MDMBDRMEnablerOptions` section within the `MDMBDRMMatchRulesSet` section of the matching rules file and set it to true.

Request URL

Use the POST method to run the Multisearch web service.

To run the Multisearch web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Identifier for Configuration and Matching Rules Files>/Multisearch
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/Multisearch
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Content-Type

Format of the request body. The supported format is JSON. Specify `application/json` as the header value.

Request Body

Use the request body to specify the input parameters for the Multisearch web service based on the search layout that the Get Multisearch Layout web service returns.

Sample Response

The following sample response shows the matching records based on the input parameters:

```
{
   "searchType":"Football",
   "resultlimit":200,
   "scoreThreshold":0,
   "ignoreMatch":false,
   "searchinput":{
      "NAME":"Smith"
   },
   "searchlayout":{

   },
   "searchResults":[
      {
         "COMPANY":"INFA",
         "MatchRuleID":"match.rule",
         "NAME":"John Smith",
         "AGE":"30",
         "ADDRESS":"Bagmane Tech Park",
         "score":"094",
         "LMT_MATCHED_SCORE":"100",
```

```
            "DOJ":"2012",
            "LMT_MATCHED_RECORD_SOURCE":"SAP",
            "match_decision":"ACCEPT",
            "PHONE":"08040201001",
            "CLUSTERNUMBER":"00df1d63-44d2-4a73-9658-e5a5e47ed99c",
            "SOURCE":"Baan",
            "SALARY":"10000",
            "ID":"14",
            "PINCODE":"560101",
            "LMT_MATCHED_PK":"1",
            "CITY":"Bangalore",
            "MatchedIndex":"00",
            "LMT_SOURCE_NAME":"Baan"
         },
         {
            "COMPANY":"INFA",
            "MatchRuleID":"match.rule",
            "NAME":"John Smith",
            "AGE":"30",
            "ADDRESS":"Bagmane Tech Park",
            "score":"094",
            "LMT_MATCHED_SCORE":"100",
            "DOJ":"2013",
            "LMT_MATCHED_RECORD_SOURCE":"SAP",
            "match_decision":"ACCEPT",
            "PHONE":"08040201001",
            "CLUSTERNUMBER":"00df1d63-44d2-4a73-9658-e5a5e47ed99c",
            "SOURCE":"Baan",
            "SALARY":"10000",
            "ID":"18",
            "PINCODE":"560101",
            "LMT_MATCHED_PK":"1",
            "CITY":"Bangalore",
            "MatchedIndex":"00",
            "LMT_SOURCE_NAME":"Baan"
         }
      ],
      "searchToken":1994262671816343800,
      "pageLimit":10,
      "pageOffset":0,
      "totalCount":2,
      "debug":false,
      "resultCount":2,
      "messages":{
         "Message.0": "Fresh Token generated",
         "Response Time":"111 ms"
      }
   }
```


A search result includes the following parameters:

match_decision

Indicates whether the match is accepted or rejected based on the match level and the search level that you configure.

Fields

Column values of all the index fields, the match fields, and the partition field, if defined.

score

Matching score for the record.

LMT_MATCHED_PK

Primary key column value of a record in the repository that matches with the matched record. The value 0 indicates that the matched record does not match with another record in the repository.

For example, a search returns record A as the matching record. If record A matches with record B, `LMT_MATCHED_PK` displays the primary key column value of record B.

LMT_MATCHED_SCORE

Matching score between the matched record and the record that matches with the matched record.

For example, a search returns record A as the matching record. If record A matches with record B, `LMT_MATCHED_SCORE` displays the matching score between record A and record B.

LMT_SOURCE_NAME

Source name of the matched record.

GROUPNO

Cluster identifier of the matched record.

<Primary Key Column Name>

Primary key column value of the matched record.

searchToken

Token that you can use in the subsequent search requests to retrieve the results from cache to avoid performing the search again. The token expires after the validity time that you configure in the `SearchTokenValidity` parameter of the configuration file. By default, the token is valid for 600 seconds. A search token value of -1 indicates that the pagination is disabled.

Note: If you use search tokens to retrieve search results from cache, the other parameters in the search layout do not affect the search results.

totalCount

Number of matching search results.

resultCount

Number of search results that the search request returns. The number of results depends on the value of the `pageLimit` parameter.

Get Cluster Layout Web Service

The Get Cluster Layout web service gets the cluster layout for the configuration file based on which you generate the WAR file. The cluster layout contains the required fields that you can specify as input parameters in the JSON format for the GETCLUSTER web service.

Request URL

Use the POST method to run the Get Cluster Layout web service.

To run the Get Cluster Layout web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
GetClusterLayout
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/GetClusterLayout
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:
`Basic <Encoded User Credentials>`

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Sample Response

The following sample response shows the cluster layout for a configuration file:

```
{
    "clusterinput": {
        "LMT_SOURCE_NAME": "mandatory",
        "<Primary Key Column Name>": "mandatory"
    },
    "clusterResults": { },
    "resultCount": 0,
    "messages": { }
}
```

A cluster layout includes the following parameters:
LMT_SOURCE_NAME

Source name of the record.

<Primary Key Column Name>

Column name that you set as primary key in the configuration file.

Get Cluster Web Service

The Get Cluster web service gets a list of records in a cluster based on the primary key column value of a record that is part of the cluster.

Before you run the Get Cluster web service, run the Get Cluster Layout web service to get the cluster layout in the JSON format for the configuration file. Based on the cluster layout, you can configure the input parameters for the Get Cluster web service.

Request URL

Use the POST method to run the Get Cluster web service.

To run the Get Cluster web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
GetCluster
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/GetCluster
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Content-Type

Format of the request body. The supported format is JSON. Specify `application/json` as the header value.

Request Body

Use the request body to specify the input parameters for the Get Cluster web service based on the layout that the Get Cluster Layout web service returns.

Sample Response

The following sample response shows the list of records in the cluster to which the specified input record belongs:


```
{
   "clusterinput":{
      "SOURCE":"SAP",
      "ID":"31"
   },
   "clusterResults":{
      "16bf81d3-e4f5-4f25-a352-3c2a4ff15fb3":{
         "SAP#31":{
            "key":"31",
            "source":"SAP"
         },
         "Salesforce#33":{
            "key":"33",
            "source":"Salesforce"
         },
         "Baan#32":{
            "key":"32",
            "source":"Baan"
         }
      }
   },
   "debug":false,
   "resultCount":3,
   "messages":{

   }
}
```

Get Preferred Record Layout Web Service

The Get Preferred Record Layout web service gets the layout details that you can use to specify the input parameters in the JSON format for the Get Preferred Record web service. The layout contains parameters that you can use to retrieve the preferred record of a cluster or to define consolidation rules and generate the preferred record for a cluster during run time.

Request URL

Use the POST method to run the Get Preferred Record Layout web service.

To run the Get Preferred Record Layout web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
GetPreferredRecordLayout
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/GetPreferredRecordLayout
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Sample Response

The following sample response shows the layout details that you can use to specify the input parameters for the Get Preferred Record web service:

```
{
   "keyData":{
      "<Cluster Column Name>":"mandatory",
      "LMT_SOURCE_NAME":"mandatory",
      "<Primary Key Column Name>":"mandatory"
   },
   "record":{

   },
   "rowRules":{
      "row":[
         {
            "target":{
               "value":"",
               "name":""
            },
            "source":[
```

```
            {
               "name":"",
               "weight":1
            }
         ],
         "name":"",
         "rule":"MODAL_EXACT",
         "order":1
      }
   ]
},
"columnRules":{
   "column":[
      {
         "source":[
            {
               "name":"",
               "weight":1
            }
         ],
         "target":{
            "value":"",
            "name":""
         },
         "name":"",
         "rule":"MODAL_EXACT",
         "order":1,
         "columnName":""
      },{
         "source":[
            {
               "name":"",
               "weight":1
            }
         ],
         "target":{
            "value":"",
            "name":""
         },
         "name":"",
         "rule":"MODAL_EXACT",
         "order":1,
         "columnName":""
      }
   ],
   "columnGroup":[
      {
         "source":[
            {
               "name":"",
               "weight":1
            }
         ],
         "target":{
            "value":"",
            "name":""
         },
         "column":[
            {
               "name":""
            },
            {
               "name":""
            }
         ],
         "name":"",
         "rule":"MODAL_EXACT"
      }
   ]
},
"defaultRules":{
```

```
    },
    "debug":false,
    "resultCount":0,
    "messages":{

    }
}
```

The layout details include the following parameters:

LMT_SOURCE_NAME

Source name of the record.

Primary Key Column Name

Column name that you set as primary key in the configuration file.

Cluster Column Name

Name of the column on which you store the cluster identifiers.

rowRules

Optional. List of row rules. If you want to generate the preferred record for the specified cluster in the run time, specify the rules.

Under `rowRules`, specify the parameters based on the rules that you plan to use.

For more information about the row rules, see the *Informatica MDM - Relate 360 Installation and Configuration Guide*.

columnRules

Optional. List of column rules. If you want to generate the preferred record for the specified cluster in the run time, specify the rules.

Under `columnRules`, specify the parameters based on the rules that you plan to use.

For more information about the column rules, see the *Informatica MDM - Relate 360 Installation and Configuration Guide*.

columnGroup

Optional. Group of columns to which you want to apply the column rules.

Under `columnGroup`, specify the parameters based on the rule that you plan to use.

For more information about the column groups, see the *Informatica MDM - Relate 360 Installation and Configuration Guide*.

Get Preferred Record Web Service

The Get Preferred Record web service gets the preferred record for the specified cluster. The web service can also generate the preferred record for the cluster in the run time based on the specified consolidation rules.

Before you run the Get Preferred Record web service, run the Get Preferred Record Layout web service to get the layout details for the preferred record. Based on the layout details, you can configure the input parameters for the Get Preferred Record web service.

Request URL

Use the POST method to run the Get Preferred Record web service.

To run the Get Preferred Record web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
GetPreferredRecord
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/GetPreferredRecord
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Content-Type

Format of the request body. The supported format is JSON. Specify `application/json` as the header value.

Request Body

Use the request body to specify the input parameters for the Get Preferred Record web service based on the layout details that the Get Preferred Record Layout web service returns.

Sample Response

The following sample response shows the preferred record for the specified cluster:


```
{
   "keyData":{
      "CLUSTERNUMBER":"4d8f6161-78be-4868-a2de-1876827d8de1",
      "SOURCE":"",
      "ID":""
   },
   "record":{
      "COMPANY":"AB Associates",
      "NAME":"John Smith",
      "AGE":"30",
      "ROW_RULE":"MODAL_EXACT",
      "COLUMN_RULE":"NONE",
      "ADDRESS":"Freeway Lane",
      "LMT_DIRTY_IND":"0",
      "DOJ":"2012",
      "PHONE":"08040201001",
      "CLUSTERNUMBER":"4d8f6161-78be-4868-a2de-1876827d8de1    ",
      "SALARY":"10000",
      "SOURCE":"Baan",
      "ID":"        26",
      "PINCODE":"560103",
```

```
            "CITY":"Chicago",
            "LMT_CREATE_DATE":"20160908063700"
        },
        "rowRules":{
            "row":[

            ]
        },
        "columnRules":{
            "column":[

            ],
            "columnGroup":[

            ]
        },
        "debug":false,
        "resultCount":1,
        "messages":{
            "Response Time":"35 ms"
        }
    }
```

Preferred Record Search Web Service

The Preferred Record Search web service searches the primary key table in the repository and retrieves the records that match the input data based on the rules in the matching rules file. The Preferred Record Search web service then uses the cluster numbers of the matching records to return the corresponding preferred records from the preferred record table.

Before you run the Preferred Record Search web service, run the Get Multisearch Layout web service to get the search layout details in the JSON format for the matching rules file. Based on the search layout details, you can configure the input parameters for the Preferred Record Search web service.

Note: Ensure that you consolidate the linked data before you run the Preferred Record Search web service.

Request URL

Use the POST method to run the Preferred Record Search web service.

To run the Preferred Record Search web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Identifier for Configuration and Matching Rules Files>/PreferredRecordSearch
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/PreferredRecordSearch
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Content-Type

Format of the request body. The supported format is JSON. Specify `application/json` as the header value.

Request Body

Use the request body to specify the input parameters for the Preferred Record Search web service based on the search layout details that the Get Multisearch Layout web service returns.

Sample Response

The following sample response shows the preferred record for the input record:


```
{
   "searchType":"Football",
   "resultlimit":200,
   "scoreThreshold":0,
   "ignoreMatch":false,
   "searchinput":{
      "NAME":"Wayne"
   },
   "searchlayout":{

   },
   "searchResults":[
      {
         "COMPANY":"Manchester Unit",
         "NAME":"Wayne Rooney",
         "AGE":"31",
         "ROW_RULE":"MAX_PK",
         "COLUMN_RULE":"CITY:MOST_DATA",
         "ADDRESS":"Old Trafford",
         "LMT_DIRTY_IND":"0",
         "DOJ":"2011",
         "PHONE":"08040201000",
         "CLUSTERNUMBER":"097e692d-7e9d-4262-b1d8-25272efbbefa    ",
         "SALARY":"10000",
         "SOURCE":"SAP",
         "ID":"        52",
         "PINCODE":"560103",
         "CITY":"Manchester",
         "LMT_CREATE_DATE":"20160914101437"
      }
   ],
   "searchToken":-1,
   "pageLimit":10,
   "pageOffset":0,
   "totalCount":0,
   "debug":false,
   "resultCount":1,
   "messages":{
      "Response Time":"117 ms"
   }
}
```

Get Strategies Web Service

The Get Strategies web service gets a list of consolidation rules that you can use to consolidate the linked data.

Request URL

Use the POST method to run the Get Strategies web service.

To run the Get Strategies web service, use the following URL format:

```
http://<Host>:<Port>/<WAR File Name>/<Version Number>/<Configuration File Identifier>/
GetStrategies
```

For example:

```
http://localhost:8080/MDMBDRMSYS/v4.0/SYS/GetStrategies
```

Request Header

Use the request header to specify the following headers:

Token

Required if you secure the RESTful web services. Authentication token that the Authenticate web service returns. If the token is not valid or you do not specify the `Token` header, the web service request uses the `Authorization` header.

Authorization

Required if you have secured the RESTful web services, but optional if you specify the `Token` header. Type of authentication and the user credentials in the Base64 format. The RESTful web services use the HTTP basic authentication. Use the following format to specify the value for the `Authorization` header:

```
Basic <Encoded User Credentials>
```

`Basic` indicates the HTTP basic authentication, and `Encoded User Credentials` indicates the Base64 format of the user name and password separated by a colon. For example, `Authorization: Basic dGVzdDpUZXN0`

Accept

Format of the response. The supported response format is JSON. Specify `application/json` as the header value.

Sample Response

The following sample response shows a list of rules that the Get Strategies web service returns:

```
{
   "rowStrategies":[
      "RANK",
      "MOST_DATA",
      "MODAL_EXACT",
      "MIN_COLUMN",
      "MAX_COLUMN",
      "MOST_FILLED",
      "EQUALS_COLUMN"
   ],
   "columnStrategies":[
      "RANK",
      "MOST_DATA",
      "MODAL_EXACT",
      "MIN_COLUMN",
      "MAX_COLUMN",
      "FROM_MASTER",
      "EQUALS_OTHER_COLUMN",
      "MIN_OTHER_COLUMN",
      "MAX_OTHER_COLUMN"
   ],
```

```
   "defaultStrategies":[
      "LATEST_TIMESTAMP",
      "MAX_PK"
   ],
   "debug":false,
   "resultCount":0,
   "messages":{

   }
}
```

Searching Data by Using the Command-Line Commands

You can use the command-line commands to search for matching records in the repository based on the matching criteria that you specify in the configuration file or the matching rules file. Use the JSON format to specify the input parameters, and the command-line commands return the results in the JSON format.

1. Perform the Get Multisearch Layout operation.

   The Get Multisearch Layout operation gets the required fields based on the matching rules file. You must specify the required fields as the input parameters in the JSON format when you search for the matching records.

2. Perform the Multisearch operation.

   The Multisearch operation searches for the matching records based on the searching and matching criteria that you specify in the matching rules file. Use the search layout that the Get Multisearch Layout operation returns to specify the input parameters for the Multisearch operation.

3. To get a list of records that are part of a cluster, perform the following tasks:
   a. Perform the Get Cluster Layout operation to get the required fields based on the configuration file.

      You must specify the required fields as the input parameters in the JSON format to get the list of records that are part of a cluster.

   b. Perform the Get Cluster operation to get the list of other records that are part of the same cluster to which the specified record belongs.

4. To get the preferred record of a cluster, perform the following tasks:
   a. Run the Get Preferred Record Layout operation. The operation returns the layout details that you can use to specify the input parameters for the Get Preferred Record operation.
   b. Run the Get Preferred Record operation. The operation returns the preferred record for the specified cluster or generates the preferred record for the specified cluster in the run time.

5. To search for the matching records and get the preferred records for the matching records, perform the following tasks:
   a. Run the Get Multisearch Layout operation. The operation returns the layout details that you can use to specify the input parameters for the Preferred Record Search operation.
   b. Run the Preferred Record Search operation. The operation gets the matching records for the input data and uses the cluster numbers of the matching records to get the preferred records of the clusters.

Get Multisearch Layout Operation

The Get Multisearch Layout operation gets the search layout for your matching rules file. The search layout contains the required fields that you can specify in the input JSON file to perform the MULTISEARCH operation.

Run the `run_client.sh` script located in the following directory to perform the Get Multisearch Layout operation: `/usr/local/mdmbdrm-<Version Number>`

To run the `run_client.sh` script, use the following command format:

```
run_client.sh

--config=configuration_file_name

--operation=GETMULTISEARCHLAYOUT

[--outputfile=output_file_name]

--rule=matching_rules_file_name
```

The following table describes the options and arguments that you can specify to run the `run_client.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--operation	GETMULTISEARCHLAYOUT	Type of operation that you want to perform. Specify GETMULTISEARCHLAYOUT.
--outputfile	output_file_name	Optional. Absolute path and name of the output JSON file to which you want to load the search layout.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.

For example:

```
run_client.sh --config=/usr/local/tree/configuration.xml --
operation=GETMULTISEARCHLAYOUT --outputfile=/usr/local/tree/output.json --rule=/usr/
local/conf/matching_rules.xml
```

The following sample response shows the search layout for a matching rules file:

```
{
   "sortField":"",
   "searchType":"Football",
   "resultlimit":0,
   "scoreThreshold":0,
   "ignoreMatch":false,
   "searchinput":{

   },
   "searchlayout":{
      "<Field 1>":"mandatory",
      "<Field 2>":"mandatory",
      "<Field 3>":"mandatory",
      "<Field 4>":"optional"
   },
   "searchResults":[

   ],
   "searchToken":-1,
   "pageLimit":10,
   "pageOffset":0,
```

```
    "totalCount":0,,
    "debug":false,
    "resultCount":0,
    "messages":{

    }
}
```

A search layout includes the following parameters:

sortField

Optional. Column name based on which you want to sort the search results lexicographically.

You can use one of the columns that you define in the `PZMAP` section of the configuration file to sort the search results. You can also use the parameter value as `score` to sort the results in the descending order of the search result scores. For example, `"sortField":"score"`

searchType

Name for the search.

resultlimit

Optional. Maximum number of records that you want to return. Default is 0, which indicates to return a maximum of 200 records.

scoreThreshold

Optional. Minimum matching score of the records that you want to include in the results. Default is 0, which indicates to include all the records in the results.

ignoreMatch

Optional. Indicates whether to perform matching on the records retrieved based on the key ranges. Set to true to ignore matching, and set to false to perform matching. Default is false.

searchlayout

List of required search fields that you must specify.

Use the following format to specify a search field:

```
"<Field 1>": "<Field Value>"
```

searchToken

Optional. Indicates whether to enable pagination for the search results when you perform a search for the first time. If you enable pagination, the search request returns a token with the search results. You can use the token in the subsequent requests to get the search results from cache to avoid performing the search again.

When you run a search request for the first time, set `searchToken=0` to enable pagination. In the subsequent requests, you can use the token that the search request returns. For example, `searchToken=1994262671816343815`. Set `searchToken=-1` to disable pagination. By default, pagination is disabled.

pageLimit

Optional. Maximum number of search results to return. The `pageLimit` parameter is applicable if you enable pagination. For example, `pageLimit=20` returns 20 search results. Default is 10.

pageOffset

Optional. Number of search results to skip. The `pageOffset` parameter is applicable if you enable pagination. For example, `pageOffset=40` indicates to skip the first 40 search results and return search results starting from 41. Default is 0.

Multisearch Operation

The Multisearch operation generates key ranges, compares the key ranges with the repository data, and gets the matching records based on the searching and matching criteria that you specify in the matching rules file. You can perform the Multisearch operation to identify the matching records for a single record or multiple records.

To perform the Multisearch operation, run the `run_client.sh` script located in the following directory: `/usr/local/mdmbdrm-<Version Number>`

You can perform the Multisearch operation in the debug mode. The debug mode returns performance metrics for the Multisearch operation. You can use the debug mode for troubleshooting purposes. To enable the debug mode, you must add the `Debug` parameter to the `MDMBDRMEnablerOptions` section within the `MDMBDRMMatchRulesSet` section of the matching rules file and set it to true.

Single Record Search

Before you run the `run_client.sh` script, create an input JSON file and configure the parameters based on the search layout of the matching rules file. To get the search layout for the matching rules file in the JSON format, perform the Get Multisearch Layout operation.

To identify matching records for a single search record, use the following command to run the `run_client.sh` script:

```
run_client.sh

--config=configuration_file_name

--rule=matching_rules_file_name

--operation=MULTISEARCH

--input=input_file_name

[--outputfile=output_file_name]
```

The following table describes the options and arguments that you can specify to run the `run_client.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.
--operation	MULTISEARCH	Type of operation that you want to perform. Specify MULTISEARCH as the value.
--input	input_file_name	Absolute path and name of the input JSON file that you configure based on the search layout.
--outputfile	output_file_name	Optional. Absolute path and name of the output JSON file to which you want to load the search results.

For example, the following command identifies the matching records for a single search record:

```
run_client.sh --config=/usr/local/tree/configuration.xml --rule=/usr/local/conf/
matching_rules.xml --operation=MULTISEARCH --input=/usr/local/tree/input.json --
outputfile=/usr/local/tree/output.json
```

The following sample response shows the matching records based on the input parameters:


```
{
   "searchType":"Football",
   "resultlimit":200,
   "scoreThreshold":0,
   "ignoreMatch":false,
   "searchinput":{
      "NAME":"Smith"
   },
   "searchlayout":{

   },
   "searchResults":[
      {
         "COMPANY":"INFA",
         "MatchRuleID":"match.rule",
         "NAME":"John Smith",
         "AGE":"30",
         "ADDRESS":"Bagmane Tech Park",
         "score":"094",
         "LMT_MATCHED_SCORE":"100",
         "DOJ":"2012",
         "LMT_MATCHED_RECORD_SOURCE":"SAP",
         "match_decision":"ACCEPT",
         "PHONE":"08040201001",
         "CLUSTERNUMBER":"00df1d63-44d2-4a73-9658-e5a5e47ed99c",
         "SOURCE":"Baan",
         "SALARY":"10000",
         "ID":"14",
         "PINCODE":"560101",
         "LMT_MATCHED_PK":"1",
         "CITY":"Bangalore",
         "MatchedIndex":"00",
         "LMT_SOURCE_NAME":"Baan"
      },
      {
         "COMPANY":"INFA",
         "MatchRuleID":"match.rule",
         "NAME":"John Smith",
         "AGE":"30",
         "ADDRESS":"Bagmane Tech Park",
         "score":"094",
         "LMT_MATCHED_SCORE":"100",
         "DOJ":"2013",
         "LMT_MATCHED_RECORD_SOURCE":"SAP",
         "match_decision":"ACCEPT",
         "PHONE":"08040201001",
         "CLUSTERNUMBER":"00df1d63-44d2-4a73-9658-e5a5e47ed99c",
         "SOURCE":"Baan",
         "SALARY":"10000",
         "ID":"18",
         "PINCODE":"560101",
         "LMT_MATCHED_PK":"1",
         "CITY":"Bangalore",
         "MatchedIndex":"00",
         "LMT_SOURCE_NAME":"Baan"
      }
   ],
   "searchToken":1994262671816343800,
   "pageLimit":10,
   "pageOffset":0,
   "totalCount":2,
   "debug":false,
   "resultCount":2,
   "messages":{
      "Message.0": "Fresh Token generated",
      "Response Time":"111 ms"
   }
}
```


A search result includes the following parameters:

match_decision

Indicates whether the match is accepted or rejected based on the match level and the search level that you configure.

Fields

Column values of all the index fields, the match fields, and the partition field, if defined.

score

Matching score for the record.

LMT_MATCHED_PK

Primary key column value of a record in the repository that matches with the matched record. The value 0 indicates that the matched record does not match with another record in the repository.

For example, a search returns record A as the matching record. If record A matches with record B, `LMT_MATCHED_PK` displays the primary key column value of record B.

LMT_MATCHED_SCORE

Matching score between the matched record and the record that matches with the matched record.

For example, a search returns record A as the matching record. If record A matches with record B, `LMT_MATCHED_SCORE` displays the matching score between record A and record B.

LMT_SOURCE_NAME

Source name of the matched record.

GROUPNO

Cluster identifier of the matched record.

<Primary Key Column Name>

Primary key column value of the matched record.

searchToken

Token that you can use in the subsequent search requests to retrieve the results from cache to avoid performing the search again. The token expires after the validity time that you configure in the `SearchTokenValidity` parameter of the configuration file. By default, the token is valid for 600 seconds. A search token value of -1 indicates that the pagination is disabled.

Note: If you use search tokens to retrieve search results from cache, the other parameters in the search layout do not affect the search results.

totalCount

Number of matching search results.

resultCount

Number of search results that the search request returns. The number of results depends on the value of the `pageLimit` parameter.

Multiple Records Search

Before you run the `run_client.sh` script, create a flat file and add the search records in the same format as that of the input data in HDFS.

To identify matching records for multiple search records, use the following command to run the `run_client.sh` script:

```
run_client.sh
--config=configuration_file_name
--rule=matching_rules_file_name
--operation=MULTISEARCH
--batchfile=batch_file_name
[--outputdir=output_directory_name]
```

The following table describes the options and arguments that you can specify to run the `run_client.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.
--operation	MULTISEARCH	Type of operation that you want to perform. Specify MULTISEARCH.
--batchfile	batch_file_name	Absolute path and name of the flat file that contains the search records. The format of the records in the batch file must match the format of the input data in HDFS. Specify the `--batchfile` parameter if you use multiple search records.
--outputdir	output_directory_name	Optional. Absolute path of a directory to which you want to load the search results. Ensure that you have the write permission to the directory. By default, the Multisearch operation loads the search results to the following directory: `<Working Directory>/relate_output`

For example, the following command identifies the matching records for multiple search records:

```
run_client.sh --config=/usr/local/tree/configuration.xml --rule=/usr/local/conf/
matching_rules.xml --operation=MULTISEARCH --batchfile=/usr/local/tree/batch.in --
outputdir=/usr/local/tree
```

The following sample response shows the matching records based on the input parameters:

```
{
   "searchType":"Football",
   "resultlimit":200,
   "scoreThreshold":0,
   "ignoreMatch":false,
   "searchinput":{
      "NAME":"Smith"
   },
   "searchlayout":{

   },
   "searchResults":[
      {
```

```
            "COMPANY":"INFA",
            "MatchRuleID":"match.rule",
            "NAME":"John Smith",
            "AGE":"30",
            "ADDRESS":"Bagmane Tech Park",
            "score":"094",
            "LMT_MATCHED_SCORE":"100",
            "DOJ":"2012",
            "LMT_MATCHED_RECORD_SOURCE":"SAP",
            "match_decision":"ACCEPT",
            "PHONE":"08040201001",
            "CLUSTERNUMBER":"00df1d63-44d2-4a73-9658-e5a5e47ed99c",
            "SOURCE":"Baan",
            "SALARY":"10000",
            "ID":"14",
            "PINCODE":"560101",
            "LMT_MATCHED_PK":"1",
            "CITY":"Bangalore",
            "MatchedIndex":"00",
            "LMT_SOURCE_NAME":"Baan"
         },
         {
            "COMPANY":"INFA",
            "MatchRuleID":"match.rule",
            "NAME":"John Smith",
            "AGE":"30",
            "ADDRESS":"Bagmane Tech Park",
            "score":"094",
            "LMT_MATCHED_SCORE":"100",
            "DOJ":"2013",
            "LMT_MATCHED_RECORD_SOURCE":"SAP",
            "match_decision":"ACCEPT",
            "PHONE":"08040201001",
            "CLUSTERNUMBER":"00df1d63-44d2-4a73-9658-e5a5e47ed99c",
            "SOURCE":"Baan",
            "SALARY":"10000",
            "ID":"18",
            "PINCODE":"560101",
            "LMT_MATCHED_PK":"1",
            "CITY":"Bangalore",
            "MatchedIndex":"00",
            "LMT_SOURCE_NAME":"Baan"
         }
      ],
      "searchToken":1994262671816343800,
      "pageLimit":10,
      "pageOffset":0,
      "totalCount":2,
      "debug":false,
      "resultCount":2,
      "messages":{
         "Message.0": "Fresh Token generated",
         "Response Time":"111 ms"
      }
   }
```


A search result includes the following parameters:

match_decision

Indicates whether the match is accepted or rejected based on the match level and the search level that you configure.

Fields

Column values of all the index fields, the match fields, and the partition field, if defined.

score

Matching score for the record.

LMT_MATCHED_PK

Primary key column value of a record in the repository that matches with the matched record. The value 0 indicates that the matched record does not match with another record in the repository.

For example, a search returns record A as the matching record. If record A matches with record B, `LMT_MATCHED_PK` displays the primary key column value of record B.

LMT_MATCHED_SCORE

Matching score between the matched record and the record that matches with the matched record.

For example, a search returns record A as the matching record. If record A matches with record B, `LMT_MATCHED_SCORE` displays the matching score between record A and record B.

LMT_SOURCE_NAME

Source name of the matched record.

GROUPNO

Cluster identifier of the matched record.

<Primary Key Column Name>

Primary key column value of the matched record.

searchToken

Token that you can use in the subsequent search requests to retrieve the results from cache to avoid performing the search again. The token expires after the validity time that you configure in the `SearchTokenValidity` parameter of the configuration file. By default, the token is valid for 600 seconds. A search token value of -1 indicates that the pagination is disabled.

Note: If you use search tokens to retrieve search results from cache, the other parameters in the search layout do not affect the search results.

totalCount

Number of matching search results.

resultCount

Number of search results that the search request returns. The number of results depends on the value of the `pageLimit` parameter.

Get Cluster Layout Operation

The Get Cluster Layout operation gets the cluster layout for your configuration file. The cluster layout contains the required fields that you can specify in the input JSON file to perform the Get Cluster operation.

Run the `run_client.sh` script located in the following directory to perform the Get Cluster Layout operation: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_client.sh` script:

```
run_client.sh

--config=configuration_file_name

--operation=GETCLUSTERLAYOUT

[--outputfile=output_file_name]
```

The following table describes the options and arguments that you can specify to run the `run_client.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--operation	GETCLUSTERLAYOUT	Type of operation that you want to perform. Specify GETCLUSTERLAYOUT.
--outputfile	output_file_name	Optional. Absolute path and name of the output JSON file to which you want to load the cluster layout.

For example:

```
run_client.sh --config=/usr/local/tree/configuration.xml --operation=GETCLUSTERLAYOUT --
outputfile=/usr/local/tree/output.json
```

The following sample response shows the cluster layout for a configuration file:

```
{
    "clusterinput": {
        "LMT_SOURCE_NAME": "mandatory",
        "<Primary Key Column Name>": "mandatory"
    },
    "clusterResults": { },
    "resultCount": 0,
    "messages": { }
}
```

A cluster layout includes the following parameters:

LMT_SOURCE_NAME

Source name of the record.

<Primary Key Column Name>

Column name that you set as primary key in the configuration file.

Get Cluster Operation

The Get Cluster operation gets a list of records in a cluster based on the primary key column value of a record that is part of the cluster.

Before you perform the Get Cluster operation, perform the Get Cluster Layout operation to get the cluster layout in the JSON format for the configuration file. Based on the cluster layout, you can configure the input parameters for the Get Cluster operation.

Run the `run_client.sh` script located in the following directory to perform the Get Cluster operation: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_client.sh` script:

```
run_client.sh

--config=configuration_file_name

--operation=GETCLUSTER

--input=input_file_name

[--outputfile=output_file_name]
```

The following table describes the options and arguments that you can specify to run the `run_client.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--operation	GETCLUSTER	Type of operation that you want to perform. Specify GETCLUSTER.
--input	input_file_name	Absolute path and name of the input JSON file that contains the cluster layout for the configuration file.
--outputfile	output_file_name	Optional. Absolute path and name of the output JSON file to which you want to load the list of records in the cluster.

For example:

```
run_client.sh --config=/usr/local/tree/configuration.xml --operation=GETCLUSTER --
input=/usr/local/tree/input.json --outputfile=/usr/local/tree/output.json
```

The following sample response shows the list of records in the cluster to which the specified input record belongs:

```
{
   "clusterinput":{
      "SOURCE":"SAP",
      "ID":"31"
   },
   "clusterResults":{
      "16bf81d3-e4f5-4f25-a352-3c2a4ff15fb3":{
         "SAP#31":{
            "key":"31",
            "source":"SAP"
         },
         "Salesforce#33":{
            "key":"33",
            "source":"Salesforce"
         },
         "Baan#32":{
            "key":"32",
            "source":"Baan"
         }
      }
   },
   "debug":false,
   "resultCount":3,
   "messages":{

   }
}
```

Get Preferred Record Layout Operation

The Get Preferred Record Layout operation gets the layout that you can use to specify the input parameters in the JSON format for the Get Preferred Record operation. The layout contains parameters that you can use to retrieve the preferred record of a cluster or to define consolidation rules and generate the preferred record for a cluster in the run time.

Run the `run_client.sh` script located in the following directory to perform the Get Preferred Record Layout operation: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_client.sh` script:

```
run_client.sh

--config=configuration_file_name

--operation=GETPREFERREDRECORDLAYOUT

[--outputfile=output_file_name]
```

The following table describes the options and arguments that you can specify to run the `run_client.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--operation	GETPREFERREDRECORDLAYOUT	Type of operation that you want to perform. Specify GETPREFERREDRECORDLAYOUT.
--outputfile	output_file_name	Optional. Absolute path and name of the output JSON file to which you want to load the layout.

For example:

```
run_client.sh --config=/usr/local/tree/configuration.xml --
operation=GETPREFERREDRECORDLAYOUT --outputfile=/usr/local/tree/output.json
```

The following sample response shows the layout that you can use to specify the input parameters for the GETPREFERREDRECORD operation:

```
{
   "keyData":{
      "<Cluster Column Name>":"mandatory",
      "LMT_SOURCE_NAME":"mandatory",
      "<Primary Key Column Name>":"mandatory"
   },
   "record":{

   },
   "rowRules":{
      "row":[
         {
            "target":{
               "value":"",
               "name":""
            },
            "source":[
               {
                  "name":"",
                  "weight":1
               }
            ],
            "name":"",
            "rule":"MODAL_EXACT",
            "order":1
         }
      ]
   },
   "columnRules":{
      "column":[
         {
            "source":[
               {
                  "name":"",
                  "weight":1
               }
```

```
            ],
            "target":{
               "value":"",
               "name":""
            },
            "name":"",
            "rule":"MODAL_EXACT",
            "order":1,
            "columnName":""
         },{
            "source":[
               {
                  "name":"",
                  "weight":1
               }
            ],
            "target":{
               "value":"",
               "name":""
            },
            "name":"",
            "rule":"MODAL_EXACT",
            "order":1,
            "columnName":""
         }
      ],
      "columnGroup":[
         {
            "source":[
               {
                  "name":"",
                  "weight":1
               }
            ],
            "target":{
               "value":"",
               "name":""
            },
            "column":[
               {
                  "name":""
               },
               {
                  "name":""
               }
            ],
            "name":"",
            "rule":"MODAL_EXACT"
         }
      ]
   },
   "defaultRules":{

   },
   "debug":false,
   "resultCount":0,
   "messages":{

   }
}
```


A cluster layout includes the following parameters:

LMT_SOURCE_NAME

Source name of the record.

Primary Key Column Name

Column name that you set as primary key in the configuration file.

Cluster Column Name

Name of the column on which you store the cluster identifiers.

rowRules

Optional. List of row rules. If you want to generate the preferred record for the specified cluster in the run time, specify the rules.

Under `rowRules`, specify the parameters based on the rules that you plan to use.

For more information about the row rules, see the *Informatica MDM - Relate 360 Installation and Configuration Guide*.

columnRules

Optional. List of column rules. If you want to generate the preferred record for the specified cluster in the run time, specify the rules.

Under `columnRules`, specify the parameters based on the rules that you plan to use.

For more information about the column rules, see the *Informatica MDM - Relate 360 Installation and Configuration Guide*.

columnGroup

Optional. Group of columns to which you want to apply the column rules.

Under `columnGroup`, specify the parameters based on the rule that you plan to use.

For more information about the column groups, see the *Informatica MDM - Relate 360 Installation and Configuration Guide*.

Get Preferred Record Operation

The Get Preferred Record operation gets the preferred record for the specified cluster. The operation can also generate the preferred record for the cluster in the run time based on the specified consolidation rules.

Before you perform the Get Preferred Record operation, perform the Get Preferred Record Layout operation to get the layout for the preferred record. Based on the layout, you can configure the input parameters for the Get Preferred Record operation.

Run the `run_client.sh` script located in the following directory to perform the Get Preferred Record operation: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_client.sh` script:

```
run_client.sh

--config=configuration_file_name

--operation=GETPREFERREDRECORD

--input=input_file_name

[--outputfile=output_file_name]
```

The following table describes the options and arguments that you can specify to run the `run_client.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--operation	GETPREFERREDRECORD	Type of operation that you want to perform. Specify GETPREFERREDRECORD.
--input	input_file_name	Absolute path and name of the input JSON file that contains the layout for the preferred record.
--outputfile	output_file_name	Optional. Absolute path and name of the output JSON file to which you want to load the preferred record of the cluster.

For example:

```
run_client.sh --config=/usr/local/tree/configuration.xml --operation=GETPREFERREDRECORD
--input=/usr/local/tree/input.json --outputfile=/usr/local/tree/output.json
```

The following sample response shows the preferred record for the specified cluster:


```
{
   "keyData":{
      "CLUSTERNUMBER":"4d8f6161-78be-4868-a2de-1876827d8de1",
      "SOURCE":"",
      "ID":""
   },
   "record":{
      "COMPANY":"AB Associates",
      "NAME":"John Smith",
      "AGE":"30",
      "ROW_RULE":"MODAL_EXACT",
      "COLUMN_RULE":"NONE",
      "ADDRESS":"Freeway Lane",
      "LMT_DIRTY_IND":"0",
      "DOJ":"2012",
      "PHONE":"08040201001",
      "CLUSTERNUMBER":"4d8f6161-78be-4868-a2de-1876827d8de1    ",
      "SALARY":"10000",
      "SOURCE":"Baan",
      "ID":"        26",
      "PINCODE":"560103",
      "CITY":"Chicago",
      "LMT_CREATE_DATE":"20160908063700"
   },
   "rowRules":{
      "row":[

      ]
   },
   "columnRules":{
      "column":[

      ],
      "columnGroup":[

      ]
   },
   "debug":false,
   "resultCount":1,
   "messages":{
      "Response Time":"35 ms"
   }
}
```

Preferred Record Search Operation

The Preferred Record Search operation searches the primary key table in the repository and retrieves the records that match the input data based on the rules in the matching rules file. The Preferred Record Search operation then uses the cluster numbers of the matching records to return the corresponding preferred records from the preferred record table.

Before you run the Preferred Record Search operation, run the Get Multisearch Layout operation to get the search layout details in the JSON format for the matching rules file. Based on the search layout details, you can configure the input parameters for the Preferred Record Search operation.

Note: Ensure that you consolidate the linked data before you run the Preferred Record Search operation.

Run the `run_client.sh` script located in the following directory to perform the Preferred Record Search operation: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_client.sh` script:

```
run_client.sh

--config=configuration_file_name

--rule=matching_rules_file_name

--operation=PREFERREDRECORDSEARCH

--input=input_file_name

[--outputfile=output_file_name]
```

The following table describes the options and arguments that you can specify to run the `run_client.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--rule	matching_rules_file_name	Absolute path and file name of the matching rules file that you create.
--operation	PREFERREDRECORDSEARCH	Type of operation that you want to perform. Specify PREFERREDRECORDSEARCH as the value.
--input	input_file_name	Absolute path and name of the input JSON file that you configure based on the search layout details.
--outputfile	output_file_name	Optional. Absolute path and name of the output JSON file to which you want to load the search results.

For example, the following command identifies the matching records for a single search record:

```
run_client.sh --config=/usr/local/tree/configuration.xml --rule=/usr/local/conf/
matching_rules.xml --operation=PREFERREDRECORDSEARCH --input=/usr/local/tree/input.json
--outputfile=/usr/local/tree/output.json
```

The following sample response shows the preferred record for the input record:

```
{
   "searchType":"Football",
   "resultlimit":200,
   "scoreThreshold":0,
   "ignoreMatch":false,
   "searchinput":{
      "NAME":"Wayne"
   },
```

```
   "searchlayout":{

   },
   "searchResults":[
      {
         "COMPANY":"Manchester Unit",
         "NAME":"Wayne Rooney",
         "AGE":"31",
         "ROW_RULE":"MAX_PK",
         "COLUMN_RULE":"CITY:MOST_DATA",
         "ADDRESS":"Old Trafford",
         "LMT_DIRTY_IND":"0",
         "DOJ":"2011",
         "PHONE":"08040201000",
         "CLUSTERNUMBER":"097e692d-7e9d-4262-b1d8-25272efbbefa    ",
         "SALARY":"10000",
         "SOURCE":"SAP",
         "ID":"        52",
         "PINCODE":"560103",
         "CITY":"Manchester",
         "LMT_CREATE_DATE":"20160914101437"
      }
   ],
   "searchToken":-1,
   "pageLimit":10,
   "pageOffset":0,
   "totalCount":0,
   "debug":false,
   "resultCount":1,
   "messages":{
      "Response Time":"117 ms"
   }
}
```


Get Strategies Operation

The Get Strategies operation gets a list of consolidation rules that you can use to consolidate the linked data.

Run the `run_client.sh` script located in the following directory to perform the Get Strategies operation: `/usr/local/mdmbdrm-<Version Number>`

Use the following command to run the `run_client.sh` script:

```
run_client.sh

--config=configuration_file_name

--operation=GETSTRATEGIES

[--outputfile=output_file_name]
```

The following table describes the options and arguments that you can specify to run the `run_client.sh` script:

Option	Argument	Description
--config	configuration_file_name	Absolute path and file name of the configuration file that you create.
--operation	GETSTRATEGIES	Type of operation that you want to perform. Specify GETSTRATEGIES.
--outputfile	output_file_name	Optional. Absolute path and name of the output JSON file to which you want to load the list of rules.

For example:

```
run_client.sh --config=/usr/local/tree/configuration.xml --operation=GETSTRATEGIES --
outputfile=/usr/local/tree/output.json
```

The following sample response shows a list of rules that the Get Strategies operation returns:

```
{
   "rowStrategies":[
      "RANK",
      "MOST_DATA",
      "MODAL_EXACT",
      "MIN_COLUMN",
      "MAX_COLUMN",
      "MOST_FILLED",
      "EQUALS_COLUMN"
   ],
   "columnStrategies":[
      "RANK",
      "MOST_DATA",
      "MODAL_EXACT",
      "MIN_COLUMN",
      "MAX_COLUMN",
      "FROM_MASTER",
      "EQUALS_OTHER_COLUMN",
      "MIN_OTHER_COLUMN",
      "MAX_OTHER_COLUMN"
   ],
   "defaultStrategies":[
      "LATEST_TIMESTAMP",
      "MAX_PK"
   ],
   "debug":false,
   "resultCount":0,
   "messages":{

   }
}
```

CHAPTER 8

Monitoring the Batch Jobs

This chapter includes the following topics:

Monitoring Overview

Use the Hadoop ResourceManager or `mapred` commands to get a list of batch jobs and to get the status of a batch job that you run. You can also stop a batch job that you already started.

The Hadoop ResourceManager is a daemon within Hadoop that you can use to monitor the batch jobs.

Starting the Hadoop ResourceManager

Use the Hadoop ResourceManager to track the status of the batch jobs and to list the batch jobs that you run.

In a browser, type the following URL to start the Hadoop ResourceManager:

```
http://<ResourceManager Host Name>:<Port>
```

Specify the following parameters in the URL:

ResourceManager Host Name

Host machine that runs the Hadoop ResourceManager.

Port

Port number on which the Hadoop ResourceManager listens. Default is 8088.

Running the Mapred Command

Run the `mapred` commands to get a list of batch jobs and to get the status of a batch job that you run. You can also stop a batch job that you started.

You can run the following commands from the command line:

mapred job -list

Gets a list of batch jobs that you run.

mapred job -status <Job Identifier>

Gets the status of the batch job that you specify. You can get the job identifier when you list the batch jobs.

For example, `mapred job -status job_1416639675736_0228`

mapred job -kill <Job Identifier>

Stops the batch job that you specify. You can get the job identifier when you list the batch jobs.

For example, `mapred job -kill job_1416639675736_0228`

CHAPTER 9

Troubleshooting

This chapter includes the following topics:

Troubleshooting Batch Jobs

If you encounter any issues with the batch jobs, use the following information to troubleshoot.

The region splitter job takes a longer time to run.

The region splitter job uses random sampling to analyze the input data. The random sampling is based on the default values of the following parameters:

--sortmaxsamples

Maximum number of samples that the job uses to analyze the input data. Default is 100,000.

--sortmaxsplitssampled

Maximum number of splits that the job uses to extract the sample data. Default is 20.

--sortsampleprobability

Frequency to sample the input data in each split. Specify a value between 0.0 and 1.0. A higher value results in dense sampling of each split and a lower value results in sparse sampling of each split. Default is 1.0.

If the input data is uniformly distributed, you can use a small sample size, few splits, and a higher sampling frequency to reduce the running time of the job. If the input data is skewed, you can use a large sample size, more number of splits, and a lower sampling frequency to reduce the running time of the job.

For example, the following command runs the region splitter job with the additional parameters:

```
run_hbase_region_analysis.sh --config=/usr/local/conf/config_big.xml --input=/usr/hdfs/
workingdir/MDMBDRMInitialBatch/MDMBDE0063_1602999447744334391/output/dir/pass-join --
hdfsdir=/usr/hdfs/workingdir --rule=/usr/local/conf/matching_rules.xml --regions=14 --
sortmaxsamples=200000 --sortmaxsplitssampled=30  --sortsampleprobability=0.5
```

When you rerun the Hive enabler job with the same Hive-related options, the job fails.

When you run the Hive enabler job for the first time, the job creates an output table and an internal table in Hive. When you rerun the Hive enabler job with the same Hive-related options, you get an error in the following format:

```
AlreadyExistsException(message:Table <Table Name>|<Table Name>_internal already exists)
```

where `Table Name` indicates the output table, and `<Table Name>_internal` indicates the internal table. For example, `mdmbdrm002_emp` indicates the output table and `mdmbdrm002_emp_internal` indicates the internal table.

To rerun the Hive enabler job with the same Hive-related options, perform the following tasks:

1. If you ran the Hive enabler job without the `--linkHBase` parameter, drop the output table as a view.
2. If you ran the Hive enabler job with the `--linkHBase` parameter, drop the output table.
3. If the `<Table Name>_internal` table exists, drop it.
4. Rerun the Hive enabler job.

In an encrypted environment, when you run the Hive enabler job, the job fails.

In an encrypted environment, when you run the Hive enabler job, you get the following error:

```
ERROR transport.TSaslTransport: SASL negotiation failure
javax.security.sasl.SaslException: No common protection layer between client and server
```

When you do not specify the authentication type that your environment uses in the configuration file, you get this error.

To fix this issue, in the `HiveConfiguration` section of the configuration file, specify the `sasl.qop` parameter in the `JDBCUrl` parameter.

For more information about the `JDBCUrl` parameter, see the *Informatica MDM - Relate 360 Installation and Configuration Guide*.

When you run the load clustering job, the job fails.

If the repository configuration in the configuration file is not in sync with the `hbase-site.xml` file, the job fails. You can find the `hbase-site.xml` file in the following directory: `${HBASE_HOME}/conf/hbase-site.xml`

Ensure that the values that you specify in the `HBASEConfiguration` section of the configuration file are in sync with the values in the `hbase-site.xml` file.

For more information about the repository configuration, see the *Informatica MDM - Relate 360 Installation and Configuration Guide*.

Troubleshooting Spark

If you encounter any issues with Spark, use the following information to troubleshoot.

You get a ClassNotFoundException error in the Spark driver log file.

If you get a ClassNotFoundException error in the Spark driver log file, view the Spark executor log file for more information about the error.

Troubleshooting Search Requests

If you encounter any issues when you run search requests, use the following information to troubleshoot.

The distributed search does not return expected results.

When you configure to perform distributed search and the search requests return unexpected results, you can disable the distributed search.

To disable the distributed search, perform the following tasks:

1. Remove or comment the following parameters in the configuration file:
   - CoprocessorPath
   - CoprocessorClass
2. Remove the coprocessor JAR file that you deployed.
3. Run the load clustering job to reload the linked or tokenized data from HDFS into the repository.

Troubleshooting Relationship Graph

If you encounter any issues when you use the relationship graph user interface, use the following information to troubleshoot.

When you aggregate the related records, the relationship graph does not show all the related records.

The maximum number of nodes that you see in the relationship graph depends on the `graph_maxnode` parameter that you configure in the properties file of the relationship graph user interface. Default is 500.

The relationship graph can display only the configured number of nodes even if a record returns more nodes than the configured number of nodes.

To increase the maximum number of nodes that you want to view in the relationship graph, perform the following tasks:

1. In the properties file of the relationship graph user interface, increase the `graph_maxnode` parameter value.
2. Regenerate the relationship graph user interface WAR file.
3. Deploy the generated WAR file on the Tomcat container.

APPENDIX A

Glossary

Hadoop Distributed File System (HDFS)

A distributed file storage system that Hadoop applications use.

HBase

A nonrelational database that runs on top of HDFS.

Hive

A data warehouse infrastructure built on top of Hadoop. Hive supports an SQL-like language called HiveQL for data summarization, query, and analysis.

Kafka

A distributed messaging system that manages input data stream.

Linking

A process of grouping related records into clusters based on the matching rules.

Matching

A process of comparing two records to identity whether they match based on the matching rules.

Population file

File that contains rules specific to the particular population of data. The rules define how to build keys and how the search and match strategies function for the specified population.

Searching

A process of comparing the input data with the repository data to identify matching records.

Spark

A distributed realtime computation system that processes the streaming data.

SSA-NAME3

A component of Relate 360 that builds keys, generates an array of search key ranges, and uses the key ranges to identify records for matching.

Storm

A distributed realtime computation system that processes the streaming data.

Tokenization

A process of adding a fuzzy token, which is an encoded key, to each input record

INDEX