Reltio MDM Best Practices **NationalGrid**

**Table of Contents**

[Best Practices for Data Loads 3](#_Toc90986929)

[**Building Reliable Dataload Processes through Reltio REST APIs** 3](#_Toc90986930)

[**Optimized Data Model** 3](#_Toc90986931)

[**The Durability of Data Sources** 3](#_Toc90986932)

[**Loading Data into Reltio Through REST APIs** 3](#_Toc90986933)

[**Synchronous Requests** 3](#_Toc90986934)

[**Requests Sizes and Number of Simultaneous Requests** 3](#_Toc90986935)

[**Retries** 4](#_Toc90986936)

[**Recording Failed Records** 4](#_Toc90986937)

[**Monitoring and Logging** 4](#_Toc90986938)

[**Sample Code for Loading Data to Reltio** 4](#_Toc90986939)

[Configuration Best Practices 8](#_Toc90986940)

[Data Profiling and Cleansing 11](#_Toc90986941)

[**Using Statistically Representative Data Sample for Profiling** 12](#_Toc90986942)

[**Data Profiling Metrics for Source Attributes** 12](#_Toc90986943)

[**Typical Data Quality Issues and Cleansing Recommendations** 12](#_Toc90986944)

[**Optimizing Match Rules as per Data Profiling Results** 14](#_Toc90986945)

[**Optimizing Survivorship Rules as per Data Profiling Results** 14](#_Toc90986946)

[**Creating a Snowflake User** 15](#_Toc90986947)

[**Creating a Normal Account** 15](#_Toc90986948)

[**Creating a Default Account** 15](#_Toc90986949)

[**Configuring the Pipeline API** 15](#_Toc90986950)

[**Creating a Job** 15](#_Toc90986951)

[**Configuring a Scheduler Job** 15](#_Toc90986952)

[**Updating the Job Status** 16](#_Toc90986953)

[**Updating the Data Warehouse Size** 16](#_Toc90986954)

[Best Practices for User Management 17](#_Toc90986955)

[**Follow the Principle of Least Privilege** 17](#_Toc90986956)

[**Use IAM Services to Manage Access to Resources** 17](#_Toc90986957)

[**Use Group Policies** 17](#_Toc90986958)

[**Centralized Approach to Onboarding and Off-boarding Users** 18](#_Toc90986959)

[Best Practices for Reltio Data Science 18](#_Toc90986960)

[**Reltio Data Science Provisioning** 18](#_Toc90986961)

[**Default Limits** 18](#_Toc90986962)

[**Setting Cluster Size** 18](#_Toc90986963)

[Objects Count in import\_status.log 18](#_Toc90986964)

[**Sub-nested Analytical Attributes** 18](#_Toc90986965)

[Best Practices for Message Queue 18](#_Toc90986966)

[**Connection** 18](#_Toc90986967)

[**Security** 19](#_Toc90986968)

[**Messaging Destinations** 19](#_Toc90986969)

[**Message Consumer** 19](#_Toc90986970)

[**Message Delivery Considerations** 19](#_Toc90986971)

[**Improve Receive Throughput by Processing Concurrently** 19](#_Toc90986972)

[**Understanding Topic Subscriptions** 19](#_Toc90986973)

[**Optimize Network IO and AWS Cost by Batching** 19](#_Toc90986974)

[**Filter Out Events that are not needed** 20](#_Toc90986975)

[Best Practices for Match Strategies for the Most Common Attributes 20](#_Toc90986976)

[Person Related Attributes 20](#_Toc90986977)

[**First Name** 20](#_Toc90986978)

[**Fuzzy Matching on First Name** 20](#_Toc90986979)

[**Organization Related Attributes** 22](#_Toc90986980)

[**Other Attributes** 22](#_Toc90986981)

[**Using Reference Attributes in a Match Rule** 23](#_Toc90986982)

# Configuration Best Practices

To realize optimum performance from the Reltio platform, we must follow these best practices.

Reltio platform provides the following configuration best practices across a range of topics and object types. Reltio encourages our customers to work closely with their Customer Success Architect and/or Reltio Professional services to gain clarity on these topics and optimize the configuration of their Reltio platform.

|  |  |
| --- | --- |
| **Entity Configuration** | |
| Max number of OV rulesets for an entity type: | 10 |
| Max number of match rules for an entity type: | 10 |
| Max number of Reference attributes in a entity: | 20 |
| Max number of Reference attributes from other entities to a single entity record (Not related to Nested Attributes): | 100 |
| Max number of populated attributes in an entity: | 200 |
| Max number of defined attributes in an entity type: | 500 |
| **Relationship Configuration** | |
| Max number of relationships between any two entity types: | 50 |
| Max number of attributes defined in a relationship type: | 50 |
| Maximum number of relationship types defined between any two entity types: | 50 |
| Relationship attributes must not contain reference attributes. |  |
| **Crosswalk Configuration** | |
| Max number of crosswalks in an entity or relationship: | 200 |
| If a surrogate has been defined, it must be the same for all sources. |  |
| **Match Rules Configuration** | |
| Use Negative match rules sparingly as they are resource intensive. |  |
| When creating a match rule on an entity type where a surrogate crosswalk is defined, ensure that the match rules account for all attributes in the surrogate crosswalk to avoid matching issues. |  |
| **Entity or Relationship Configuration** | |
| Max percent of an object’s attributes that must be nested: | 25% |
| **Attributes Configuration** | |
| Max number of values present in an attribute: | 4 |
| **Nested Attributes Configuration** | |
| Max number of sub-attributes in a nest: | 50 |
| Nested attributes must not contain reference attributes. (While the configuration will technically support this, it is not a supported use of the information model). |  |
| **Location Configuration** | |
| When using a relationship to link a party (for example) to a location entity type, (for example use of the HasAddress relationship type), make sure that all attributes like the ADDR TYPE attribute are held in the relationship type and not in the location entity. Otherwise you will just keep disconnecting the old address and connecting to the new address rather than adding one for each address type. |  |
| The cleanse configuration must contain the declaration, "ovOnly": "true" which ensure that only the OV of the address values will be sent for cleansing. |  |
| The configuration for address cleansing must have a AVC Code for Status Mapping. |  |
| At least one Address match rule must be defined. |  |
| All address match rules must include a match on the verification status ="Verified". |  |
| A surrogate key must be defined. |  |
| The list of attributes defined in a surrogate crosswalk must be the same that are configured in the match rule. |  |
| All the attributes that are used in the match rule must be defined in the surrogate crosswalk, even if the source does not provide the information. Even if the match rule is lenient, with few attributes, the same set of attributes must be used in the surrogate crosswalk. |  |
| The survivorship strategy must be defined only for the lowest granular attribute (addressline1), all other attributes must be referring to the source of lowest granular attribute (other crosswalk winner). |  |
| Reltio cleanser crosswalk must have the highest source priority. |  |
| The cleanse button must be disabled in all UI configurations. |  |
| **RDM Configuration** | |
| Max number of lookups created in one POST to an RDM tenant: | 100 |
| Max number of lookups created in one POST to an RDM tenant: | 100 |
| Maximum number of lookup values for all types: | 20,000 |
| Maximum number of lookup values for all types: | 20,000 |
| Delta Detection: Implementations MUST code delta detection and only POST updates and inserts. |  |
| If any attributes in the configuration use RDM, then all must use RDM. Otherwise any attributes not using RDM will be not be able to utilize a drop-down list in the UI. |  |
| Every RDM Value set must include a source mapping for source "Reltio". |  |
| The "rdmConfig": {...} setting in the L3 must be properly configured in order to link the MDM tenant to the RDM tenant. |  |
| **LCA Configuration** | |
| Carefully evaluate whether LCAs must be used or are appropriate. LCAs must only be used for short-running operations because they will impact performance. |  |
| Ensure that the LCA is not performing data manipulation or calculation of derived data based on a specific source. Such logic must be designed and implemented at the source integration layer. |  |
| LCAs must be disabled for the high volume loads if possible. (both initial and incremental). |  |
| LCA must not invoke Reltio APIs as this can cause performance degradation and also possibly cause circular references. Reltio Engineering approval must be obtained for any exceptions. |  |
| Ensure that the LCA code does not contain memory leaks, exploits, or attempts to work with file system/network and any other slow devices. |  |
| Ensure that the LCA code does not contain Reltio classes defined in the LCA framework. |  |
| Ensure that the LCA code does not contain any hard coded credentials. |  |
| Ensure that the LCA code does not contain any output to a system. All outputs must be written using the logging service. |  |
| All LCA Code must be submitted to engineering at least 3 weeks in advance of approval so it can be thoroughly tested via a test harness available on the Reltio repository for all the test cases. Submission to Reltio engineering is done by filing a support ticket with a link to a Bitbucket or git repository that holds the code. The ticket must fully explain the goal and strategy of the LCA. |  |
| Submit test-run logs that ensure that the LCA code is executed with the minimum throughput of 1000tps. |  |
| Submit test-run logs that ensure the latency due to the LCA call is no more than 10ms. |  |
| Submit test-run logs that ensure the LCA code can be run in the multi-threaded environment. |  |
| Ensure that the LCA code is well tested using the testing framework available on the Reltio maven repository. |  |
| Ensure that the JAR file(s) is compiled and signed with the Life Cycle Handler implementation. |  |
| Ensure that necessary filter condition is configured for every LCA (if applicable) in the L3 which will prevent unnecessary calls to the LCA module. |  |
| We recommend that the attribute name must never be repeated across the L3 configuration. |  |
| **Streaming Configuration** | |
| Ensure the streaming is disabled in the tenant if the applications it supports do not need streaming. |  |
| It is recommended to configure only Entity and Reltio URI in the JMSEventsFilteringFields (physical configuration) for better performance of the tenant. |  |
| Disable the Streaming API before a task affecting large volume of data in the tenant such as Reindex the tenant. |  |
| Ensure the streaming is disabled in the tenant if the applications it supports do not need streaming. |  |
| **UI Related Configuration** | |
| Maximum Dashboard facets: | 20 |
| Maximum Pivot facets: | 20 |
| The number of tags displayed on the dashboard cannot exceed the Maximum: | 20 |
| Max number of entities rendered in relationship facet: | 500 |
| Maximum facets on a page: | No Max |
| Maximum Pivot attributes: | No Max |
| Each entity has its own perspective defined (that is, no default perspective). |  |
| Each facet has proper labels defined. |  |
| All UI extensions have a unique ID (No ID used twice). |  |
| **Data Loading Configuration** | |
| Design data loads so that you do not update the same entity in a parallel thread in a single load (try to load each entity only once). |  |
| Load data in appropriately sized batches and threads. |  |

# Data Profiling and Cleansing

Data cleansing is the process of applying the findings of data profiling to standardize the data and remove anomalous patterns. Whereas, data profiling is the process of examining your source data.

It is crucial to profile and analyze the data before bringing it into any data management repository, including Reltio. Data profiling helps with many aspects of design and the following are some of those aspects:

* Determining the quality, the range of values, consistency, and completeness of data within a source and across all sources
* Identifying the source attributes that qualify as good elements for matching purposes
* Identifying the source attributes that must never be used in the matching process. These attributes may negatively impact the performance or result of the matching.
* Identifying the reference data, consistency, and commonality of the referenced data across sources
* Identifying the attributes that can be used for faceted search
* Data mapping from customer data sources to the target model within the Reltio Connected Cloud

Some recommendations for data profiling and cleanse that we can consider.

## **Using Statistically Representative Data Sample for Profiling**

For large data sets, use a statistically representative sample of data. Here are some guidelines around selecting such samples:

* The sample must be large enough. Typically, the sample must have at least 10% or > 500,000 records, whichever is greater.
* The sample must be composed of records randomly selected throughout the dataset and not just from the beginning of the file or table. An easy way to do this is to select every 10th record. This process is fairly random and gets you 10% data.

**Note:** Selecting all data from a particular state or zip code may not create representative data from the entire source. For example, CA has a large number of HCPs. However, HCPs in Northern Florida have multiple state licenses as they practice in multiple states.

## **Data Profiling Metrics for Source Attributes**

Data profiling metrics are captured for every source attribute. Following is the list of data profiling metrics that must be captured:

* Number of unique values
* Number of null values
* Type of the attribute (Number, Character, Boolean, Date, and so on)
* Maximum and minimum length of each attribute
* The average length of each attribute
* Modal (most frequent) values

## **Typical Data Quality Issues and Cleansing Recommendations**

This section provides information about typical data quality issues and cleansing recommendations.

* Generic issues
  + Alpha characters in numeric attributes must be stripped.
  + Non-printable characters must be stripped.
  + Special characters in alphabetic attributes (for example, Name) must be stripped.
* Phone numbers
  + Phone numbers must be formatted consistently; especially international phone numbers.
  + Check for the following type of numbers. If necessary, remove them.
    - High-frequency phone numbers to see if they are default numbers. For example, 999-999-9999.
    - Specific rules of thumb numbers that are entered during data entry. Such as the sales person’s phone number or the main phone number of the department or company.
* Organization Names - May have noise words such as CORP, LTD, ASSOCIATION, and so on. These words must either be standardized or removed. For example, CORP must be standardized to CORPORATION or removed entirely to make the name attribute more consistent.
  + The preferred way to do this is in the ETL layer, as every source has its own noise words and patterns.
  + The less preferred method is to configure match rules to ignore noise words based on a customized dictionary.
* Person Names
  + Person names sometimes include Title (for example, Mr., Dr., and so on) and Suffix (for example, Jr, Sr, and so on). These words must be removed and moved under separate attributes.
  + First names commonly include nicknames (For example, Bill Smith for William Smith). Reltio's match engine uses a dictionary to substitute full names for nicknames in flight, without changing the original data. The NameDictionary cleanser must be configured to use this.
* Email
  + Basic verification of email format is to check if the email address conforms to the pattern, email@domain.com. The preferred way to do this is in the ETL layer, where source specific cleansing rules can be applied.
  + This does not guarantee that the email address is valid. But informs that it is in the expected format.
* Address
  + Reltio’s built-in address cleansing engine can interpret and standardize poor quality source addresses. However, if the data contains anomalous patterns that are source-specific (for example, uncommon abbreviations or keywords), the engine may not be able to parse the address correctly.
  + The recommended approach to improve address parsing and quality is to apply pre-processing quality rules:
    - Noise words such as UNKNOWN, NONE, or N/A must be stripped to improve the quality of data and to prevent bad matches. If these values are found in key attributes such as Address Line 1, City or State, then Address Line 1 is considered the minimum mandate field. The value of this data is questionable. It must be filtered out and not loaded into Reltio.
    - Address Line 1 must contain premise and street information. Additional information such as Contact Name, Department, and so on, must be removed from Address Line 1.
    - Fix basic transposition errors. For example, Street address in Address line 2 and Suite in Address Line 1.
    - The country code is key to cleansing international addresses and must always be provided.
    - It is suggested to provide Zip5 for better responses.
* Dates
  + Check to make sure all dates follow the same format or adjust them to be consistent.
  + Check the high-frequency dates to determine if they are default values such as 1/1/1900. These default values are not meant for record matching. These must be removed or noted to be excluded from the match rules.
* Reference Data
  + Harmonization of reference data such as Specialty Codes, SIC codes, and so on, across sources can be a difficult exercise. This must be started as early in the project as possible.
  + Reltio provides RDM (Reference Data Management) capabilities that can ease the work of defining reference data and mappings. The RDM capabilities also help by applying them automatically when loading data into Reltio.

## **Optimizing Match Rules as per Data Profiling Results**

This section provides information about the ways to optimize match rules based on the data profiling results obtained.

* High Cardinality (high degree of uniqueness) attributes such as identifiers are the best candidates for match attributes.
* Medium cardinality attributes such as Name and Address are the next best.
* Low cardinality attributes such as Gender and State Code must only be used in conjunction with a high or medium-cardinality attribute. This helps to partition or filter the match population.
* Sparse attributes (those with a high percentage of nulls) must be treated as low cardinality attributes.
* Data profiling can highlight the medium cardinality attributes that contain frequently repeating values. This results in the following issues and must be avoided:
  + Creates hot spots (that is, data with subsets that are highly matchy)
  + Poor match performance and high numbers of potential matches

## **Optimizing Survivorship Rules as per Data Profiling Results**

You can optimize survivorship rules based on the data profiling results obtained. Business users usually have a good idea of the quality and priority of their sources. Having data profiling metrics for each source can help augment that knowledge and improved the design of survivorship rules.

**Best Practices for Reltio Connected Data for Snowflake**

These recommendations enable you to accomplish Snowflake related tasks in the most efficient way.

The following are the recommendations on how to use Reltio Connected Data for Snowflake effectively:

## **Creating a Snowflake User**

To create a Snowflake user, the Public key must be in a single line string format.

## **Creating a Normal Account**

To create a Normal Account using the Create Account API, the Private key parameter passed in the request body must be in a single line string.

## **Creating a Default Account**

To create a Default Account using the Create Account API, the Private key parameter passed in the request body must be in a single line string. It is also recommended that:

* The defaultForTenant parameter passed in the request body must be set to true
* The defaultDatabase, defaultWarehouse, defaultSchema parameter values must be passed in the request body

## **Configuring the Pipeline API**

The pipelines must be configured for the specific entity types, relation types, or interaction types.

**Note:** To prevent the creation of a number of empty tables or views in Snowflake, avoid using default pipelines.

To avoid overriding data, use different database schemas for different tenants and pipelines.

Finally, creating multiple pipelines for the same entity types, relation types, and interaction types causes the same data to get exported via multiple pipelines to the Snowflake database. This is not recommended.

## **Creating a Job**

No simultaneous job executions must be done for the same Pipeline. It is also recommended that you create separate pipelines for entities, relations, and interactions if the tenant has a large amount of data. You can start the job for each pipeline in parallel.

## **Configuring a Scheduler Job**

The **ScheduleConfig** type must be configured only for a SCHEDULER job config type. The values can be BATCH or INCREMENTAL\_BATCH. It is recommended that:

* The startTime parameter is mandatory and cannot be a null value. If the value is a negative number, 0, or a positive number, then the current system time is considered as the start time.
* The interval parameter contains the number of days and the hours. It must be in the following format:
* "any number from 0 to 999""case-insensitive : D"" any number from 0 to 23 "" case-insensitive : H".
* For example, if the interval parameter is 0d1h, it means that the incremental job runs every hour.
* The value of the endTime parameter must be later than the value of the startTime parameter.
* Incremental jobs must be run every 24 hours. Running the jobs earlier can cause data inconsistency in the Snowflake database, due to incremental jobs running concurrently for the same pipeline.

## **Updating the Job Status**

The following valid Job state transitions are updated through the API:

* **INPROGRESS** > **CANCELLING**
* **SCHEDULED** > **CANCELLING**
* **INPROGRESS** > **COMPLETED**

## **Updating the Data Warehouse Size**

As the connector needs a pre-configured Data Warehouse from Snowflake to run queries, ingestion, and transformation jobs, it is recommended that you use the following warehouse sizes for different work loads so as to avoid long-running jobs and job failures. Based on our performance tests, these recommendations are applicable for 80-90% of the use cases:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Job Type** | **Avg. Record Count** | **Avg. Attribute Size** | **Avg. Run Time**  **(Ingestion + Transformation)** | **Recommended Warehouse size** |
| Batch | <1M | <20 | 3 minutes | X-Small |
|  |  | 20-50 | 5 minutes | X-Small |
|  |  | >50 | 10 minutes | X-Small |
|  | 1M-10M | <20 | 10 minutes | Small |
|  |  | 20-50 | 12 minutes | Small |
|  |  | >50 | 15 minutes | Medium |
|  | 10M-50M | <20 | 20 minutes | Medium |
|  |  | 20-50 | 20 minutes | Large |
|  |  | >50 | 25 minutes | X-Large |
|  | >50M | <20 | 30 minutes | Large |
|  |  | 20-50 |  | X-Large |
|  |  | >50 |  | X-Large |

For Incremental Jobs, these recommendations are applicable for ingestion and transformation jobs. If the incremental update volume is significantly less than the initial batch jobs, downgrade to a lower warehouse than the warehouse used for batch ingestion.

For example: If a tenant has 10M profiles and 1M incremental updates are ingested every week, then you can use a Medium warehouse for batch jobs and use a Small warehouse for incremental jobs.

# Best Practices for Data Loads

## **Building Reliable Dataload Processes through Reltio REST APIs**

Cloud-based distributed systems require different techniques from traditional programming models to meet the higher expectations of users on commodity cloud-based hardware. Building resilient systems requires designing applications to cope with and recover from inevitable system failures without any data loss.

In order to have the best performance and reliability during dataloads, it’s critical to make sure that the customer’s ETL follows the guidelines described below.

## **Optimized Data Model**

In order to achieve the best performance of loading data into Reltio it is critical to follow Reltio’s best practices for data modeling - e.g. make sure immutable reference attributes, good-performing match rules.

## **The Durability of Data Sources**

In order to make sure that there is no loss of data before loading it into Reltio, it is critical that all objects that need to be loaded into Reltio are stored in durable storage: disk, AWS S3, GCP GCS, Azure BLOB Storage, stream-processing platforms. An object can be marked as successfully loaded into Reltio (e.g. acknowledge an event in a stream-processing platform) only after getting a response from Reltio with 200 HTTP status code.

## **Loading Data into Reltio Through REST APIs**

Reltio offers various REST APIs to load data to Reltio. All of the REST APIs are synchronous and may include multiple objects within the payload of a POST request. REST API requests can be executed in parallel with the recommended number of parallel simultaneous requests to have the optimal performance when loading data to Reltio.

## **Synchronous Requests**

Reltio REST APIs are synchronous. Synchronous calls must keep the network connection open until a response is returned from the service, unlike asynchronous calls which close the connection as soon as the request is submitted.

## **Requests Sizes and Number of Simultaneous Requests**

Each tenant and dataset loaded will exhibit different behavior with respect to performance. Below is a chart with suggested numbers of objects per post and thread counts based on the size of the message body and attribute count.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Size** | **Object Size** | **Approx. Attributes Per Object** | **Records per POST Request** | **Threads** |
| Small | 0Kb - 15Kb | 0 - 300 | 50 - 100 | 15 - 20 |
| Medium | 15Kb - 70Kb | 300+ | 30-60 | 10-15 |
| Large | 70Kb+ | 300+ | 10-30 | 5-10 |

**Note**: If the limit of the number of simultaneous requests for a tenant is reached, Reltio may return a response with a 503 or a 429 HTTP error code, which indicates the client must slow down requests using an exponential backoff algorithm

## **Retries**

A request retry can be done only after receiving a not successful response from Reltio (not 200 HTTP status code). A retry should be performed:

* Within the same pool of requests within the recommended number of parallel requests. It is strongly recommended not to create new connections to Reltio for failed records as it will increase the number of simultaneous requests.
* Use exponential backoff:  
  https://en.wikipedia.org/wiki/Exponential\_backoff.
* A retry should be done only for failed records and not for all the initial records in a source. For example, for daily dataloads, if 10 records that had issues to load it is not recommended to fail the whole data load reload all data for that day.

## **Recording Failed Records**

If a record is failed to be loaded into Reltio after certain retries, it is critical to save those records in a file or a dead letter queue to make sure that those records can be investigated and re-loaded again.

It is always a great idea to record a reason for failure with the original response details that were got from the platform.

## **Monitoring and Logging**

It is critical to make sure that the integration code has the proper level of logging and monitoring. It will be useful to have a real-time status for the progress of the current Dataload:

- how many records have been loaded

- how many records have been failed

- current operations per second

## **Sample Code for Loading Data to Reltio**

The following is a sample code for handling HTTP response codes where it is recommended that the client application retry the request. The code is illustrative only - it does not include the full code required to perform HTTP requests.

|  |
| --- |
| import java.util.HashMap;  /\*\*  \* Try to send a HTTP request, retrying if get a response  \* back to slow down due to a retry response code.  \*  \* This is illustrative code only - it compiles and runs but  \* does not do real HTTP API calls.  \*/  class ExponentialRetry {  /\*\*  \* Map HTTP status response codes with number of retry attempts.  \*/  static private final HashMap<Integer,Integer> retries  = new HashMap<>();  static {  retries.put(401, 12); // Retry limit for 401  retries.put(429, 12); // Retry limit for 429  retries.put(502, 10); // Retry limit for 502  retries.put(503, 12); // Retry limit for 503  retries.put(504, 5); // Retry limit for 504  }  /\*\*  \* Delays in seconds to use for backoff durations.  \* First array value is delay for first retry,  \* second value second retry, etc.  \*/  static private final int[] retryDelay = {  1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047, 4095  };  /\*\*  \* The method to do the API call, implementing an  \* exponential backoff algorithm.  \*/  public void doExponentialRetryCall()  {  boolean debugging = true;  int attempt = 0;  int totalDuration = 0;  boolean succeeded = false;  while (!succeeded) {  try {  // Insert the HTTP request code here.  doHttpRequest();  succeeded = true;  } catch (HttpException e) {  // Fetch the HTTP status code.  int statusCode = e.getStatusCode();  // Look up the HTTP status code to get  // the retry count (if defined).  Integer retryLimit = retries.get(statusCode);  if (retryLimit == null) {  // Not a retryable error code.  // Abort with the exception.  throw e;  }  // If too many retries, give up.  if (attempt >= retryLimit) {  throw new RuntimeException(  "HTTP request failed after " + attempt  + " retries over "  + totalDuration + " seconds", e);  }  try {  // Look up the desired retry elapsed time  int seconds = retryDelay[attempt++];  totalDuration += seconds;  // Log progress.  if (debugging) {  System.out.println(  "Retry " + attempt + ", sleeping "  + seconds + " seconds...");  }  // Sleep for specified duration (could add  // Math.random() \* 1000 to stagger delays  // slightly)  Thread.sleep(seconds \* 1000);  } catch (InterruptedException e1) {  // Ignore  }  }  }  }  /\*\*  \* Dummy class to simulate a HTTP transport exception.  \*/  public class HttpException extends RuntimeException  {  private int statusCode;  public HttpException(int statusCode) {  this.statusCode = statusCode;  }  public int getStatusCode() {  return statusCode;  }  }  /\*\*  \* Dummy method to simulate doing a HTTP API call which  \* throws an exception.  \*/  private void doHttpRequest()  {  throw new HttpException(429);  }  /\*\*  \* Dummy method to implement multi threading  \*/  public static void main(String args[]){  try {  ThreadPoolExecutor executorService = (ThreadPoolExecutor)  Executors.newFixedThreadPool(<<threadsCount>>);  //read file etc.  boolean eof = false;  while (!eof) {  for (int threadNum = futures.size(); threadNum <  dataloaderInput.getThreadCount(); threadNum++) {  inputRecords.clear();  for (int k = 0; k < payloadSize; k++) {  //Read records as per getGroupsCount (eg. 50)  //and add to inputRecords  ...  }  if (inputRecords.size() > 0) {  futures.add(executorService.submit(new  Callable<Long>() {  @Override  public Long call() {  long requestExecutionTime = 0L;  long startTime =  System.currentTimeMillis();  try {  //Call the HTTP request, handle  //retries and error logging  //Refer httpPost sample method above  response =  reltioAPIService.httpPost(srcUrl,  stringToSend);  //Log progress time for load  } catch (GenericException e)  } catch (ReltioAPICallFailureException e)  } catch (IOException e)  return requestExecutionTime;  }  }));  }  }  }  }  }  /\*\*  \* Dummy class to simulate handling of HTTP transport exceptions.  \*/  private String httpPost() throws GenericException,  //Define RETRY\_LIMIT  try {  //Call the Reltio HTTP request and get the Response Code  responseCode = APIService.Execute(request, Headers);  } catch (APICallFailureException e) {  if (responseCode == 401 && retryCount <= RETRY\_LIMIT) {  //Retrying with a new token and increase retry count  } else if (responseCode == 400 &&INVALID\_REFRESH\_TOKEN\_ERROR)){  //Retrying with a new token and increase retry count  } else if (responseCode == 502 && retryCount <= RETRY\_LIMIT) {  try {  //Sleep and Try after backoff time period  sleepTime = (long)  Math.min(AuthenticationProperties.  MAXIMUM\_BACKOFF\_TIME\_MILLI\_SEC,  (Math.pow(2,retryCount-1)\*1000 +  Math.random() \* 1000 + 1));  Thread.sleep(sleepTime);  //Increase retryCount  } catch (InterruptedException ex) {  //Log error  }  } else if (responseCode == 503 && retryCount <= RETRY\_LIMIT) {  try {  //Sleep and Try after backoff time period  //Increase retryCount  } catch (InterruptedException ex) {  //Log error  }  } else if (responseCode == 504 && retryCount <= RETRY\_LIMIT) {  //Sleep and Try after backoff time period  //Increase retryCount  } catch (InterruptedException ex) {  //Log error  }  } else {  //Log error in the file  }  } catch (GenericException e) {  if (retryCount < RETRY\_LIMIT) {  // Add Reltio Authorization Token in the header  //Retry and increase retryCount  } else {  //Log Unexpected error in the file  }  }  return responseStr;  }  /\*\*  \* Dummy main program for testing.  \*/  public static void main(String[] args)  {  ExponentialRetry er = new ExponentialRetry();  er.doExponentialRetryCall();  }  } |

# Best Practices for Match Strategies for the Most Common Attributes

You can use the strategies outlined below to match commonly used attributes.

The following sections explain the match strategy for the most commonly used attribtues.

## Person Related Attributes

## **First Name**

There are various tactics that can be used on the First Name.

Exact matching of First Name (and/or LastName)

* Recommended Comparison Operator: Exact
* Recommended Comparator class: BasicStringComparator
* Recommended Token Generator class: ExactMatchToken

## **Fuzzy Matching on First Name**

When thinking about fuzzy matching for the First Name attribute, there are at least two tactics that can be employed which you should consider. First, if your objective is to successfully find and match cases of first names or last names that have misspellings, you could use the DamerauLevenshteinDistance or the DynamicDamerauLevenshteinDistance comparator, coupled with the FuzzyTextAndNumberMatchToken. Another tactic would be to use the DoubleMetaphoneComparator and DoubleMetaphoneMatchToken which uses a phonetic approach AND takes common misspellings into account automatically. Lastly if you wish to employ either of the previous suggestions but also match across common synonyms, then you can add the use of the Name Dictionary Cleanser to your rule.

**Full Name**

This is a good tactic if you want to avoid matching on first and last names independently. You can use the profile-level cleanser to form the Full Name as a concatenation of the First and Last names if the full name is not available to you directly.

Since the combined name will naturally increase the statistical amount of variation into the match process, you should use the Fuzzy comparison operator and choose fuzzy comparator and token classes as you see fit.

**Phone Number**

The following table lists the recommended comparator and token generator classes:

| **Recommended** | **Class** |
| --- | --- |
| Recommended comparator class | PhoneNumberComparator |
| Recommended token generator class | PhoneNumberMatchToken |

**U.S. Social Security Number and other similar Identifier numbers**

The following table lists the recommended comparator and token generator classes:

| **Recommended** | **Class** |
| --- | --- |
| Recommended comparator class | BasicStringComparator or DynamicDamerauLevenshteinDistance |
| Recommended token generator class | ExactMatchToken |
| Best ractice guidance | Regex can be used to remove special characters from IDs before comparison and tokenization. |

**Gender**

The following table lists the recommended comparison operator, comparator class, and token generator class:

| **Recommended** | **Class** |
| --- | --- |
| Recommended comparison operator | Exact |
| Recommended comparator class | BasicStringComparator |
| Recommended token generator class | (none, use ignoreIntoken to suppress this) |
| Best practice guidance | If the population of data is not extremely good, then consider using ExactOrNull that allows for one or both gender attributes to be <null>. |

**Suffix**

The following table lists the recommended comparison operator, comparator class, and token generator class:

| **Recommended** | **Class** |
| --- | --- |
| Recommended comparison operator | Exact |
| Recommended comparator class | (Recommended using ignoreIntoken to suppress this) |
| Recommended token generator class | ExactMatchToken |
| Best practice guidance | If the population of data is not extremely good, then consider using ExactOrNull that allows for one or both gender attributes to be <null>. Be sure to clean and standardize values like Jr, Jr., Junior, to a common value like jr. |

## **Organization Related Attributes**

**Organization Name**

The following table lists the recommended comparison operator, comparator class, and token generator class:

| **Recommended** | **Class** |
| --- | --- |
| Recommended comparison operator | Fuzzy |
| Recommended comparator class | OrganizationNamesComparator or DamerauLevenshteinDistance |
| Recommended token generator class | OrganizationNameMatchToken |

**Tax ID**

Similar to Social Security Number (SSN) or other similar identifiers.  
  
**Other Attributes**

**Address**

The following table lists the recommended comparison operator, comparator class and token generator class:

| **Recommended** | **Class** |
| --- | --- |
| Recommended comparison operator | Fuzzy |
| Recommended comparator class | AddressLineComparator |
| Recommended token generator class | AddressLineMatchToken |

**Email**

The following table lists the recommended comparison operator, comparator class and token generator class:

| **Recommended** | **Class** |
| --- | --- |
| Recommended comparison operator | Fuzzy or Exact |
| Recommended comparator class | BasicStringComparator or DamerauLevenshteinDistance |
| Recommended token generator class | ExactNumberMatchToken |

## **Using Reference Attributes in a Match Rule**

We should avoid the use of reference attributes in match rules as much as possible. Significant use of reference attributes in match rules increase performance overhead for the platform. Whenever possible, denormalization of attributes within an entity is better for performance.