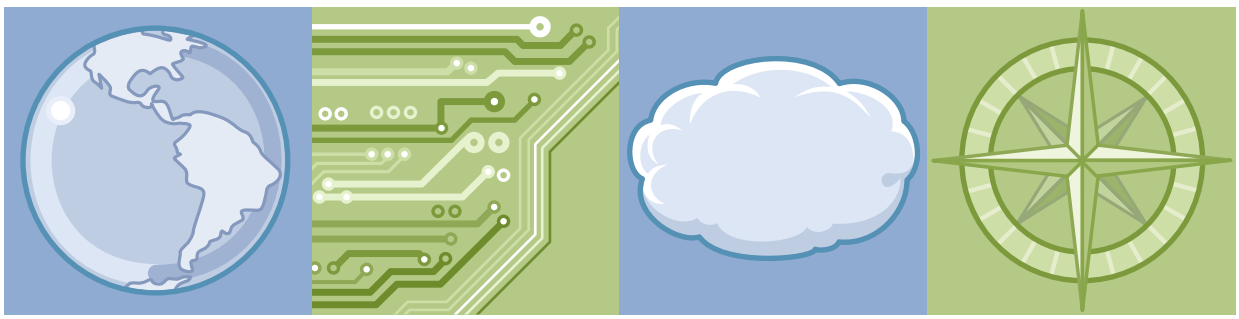
IBM Training

Student Notebook

InfoSphere MDM Algorithms

Course code ZZ780 ERC 1.0



Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

DataStage®	DB™	DB2®
InfoSphere®	Notes®	Rational®
Tivoli®	WebSphere®	

Pentium is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Lenovo and ThinkPad are trademarks or registered trademarks of Lenovo in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

VMware and the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the "Marks") of VMware, Inc. in the United States and/or other jurisdictions.

Other product and service names might be trademarks of IBM or other companies.

April 2014 edition

Contents

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

DataStage®	DB™	DB2®
InfoSphere®	Notes®	Rational®
Tivoli®	WebSphere®	

Pentium is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Lenovo and ThinkPad are trademarks or registered trademarks of Lenovo in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

VMware and the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the "Marks") of VMware, Inc. in the United States and/or other jurisdictions.

Other product and service names might be trademarks of IBM or other companies.

Unit 0. Course Introduction

What this unit is about

This unit introduces the course to the student.

What you should be able to do

After completing this unit, you should be able to:

- Understand the topics this course will present
- Understand the agenda for the course

Course objectives

After completing this course, you should be able to:

- Understand Matching and Linking
- Understand MDM configuration project and database tables used by the PME
- Understand PME Algorithms (Standardization, Bucketing and Comparison steps)
- Understand how to analyze Buckets in an algorithm
- Understand how to generate weights for a given algorithm
- Understand how to analyze weights that are generated using the workbench
- Understand how to deploy the PME configuration
- Understand how to create a custom converter for Physical MDM

Figure 0-1. Course objectives ZZ7801.0

Notes:

After completing this course, you should be able to:

- Understand Matching and Linking
- Understand MDM configuration project and database tables used by the PME
- Understand PME Algorithms (Standardization, Bucketing and Comparison steps)
- Understand how to analyze Buckets in an algorithm
- Understand how to generate weights for a given algorithm
- Understand how to analyze weights that are generated using the workbench
- Understand how to deploy the PME configuration
- Understand how to create a custom converter for Physical MDM

Agenda

Day 1

- Unit 1: PME and Virtual MDM Overview
- Unit 2: MDM Algorithms
  - Exercise 2: Creating a new Algorithm
- Unit 3: Virtual PME Data Model
  - Exercise 3: Loading Members and viewing Virtual data model

Day 2

- Unit 4: Bucket Analysis
  - Exercise 4: Bucket Analysis

Figure 0-2. Agenda ZZ7801.0

Notes:

Day 1

- Unit 1: PME and Virtual MDM Overview
- Unit 2: Virtual MDM Algorithms
  - Exercise 2: Creating a new Algorithm
- Unit 3: Virtual PME Data Model
  - Exercise 3: Loading Members and viewing Virtual data model

Day 2

- Unit 4: Bucket Analysis
  - Exercise 4: Bucket Analysis

Agenda cont …

Day 2 cont …

- Unit 5: Weights
  - Exercise 5: Generating Weights
- Unit 6: Thresholds
  - Exercise 6a: Bulk Cross Load
  - Exercise 6b: Pair Manager
  - Exercise 6c: Testing our Algorithms

Day 3

- Unit 7: PME and Physical MDM Overview
  - Exercise 7a: Resetting the MDM environment
  - Exercise 7b: Testing the default Physical PME algorithm

Figure 0-3. Agenda cont … ZZ7801.0

Notes:

Day 2 cont …

- Unit 5: Weights
  - Exercise 5: Generating Weights
- Unit 6: Thresholds
  - Exercise 6a: Bulk Cross Load
  - Exercise 6b: Pair Manager
  - Exercise 6c: Testing our Algorithms

Day 3

- Unit 7: PME and Physical MDM Overview
  - Exercise 7a: Resetting the MDM environment
  - Exercise 7b: Testing the default Physical PME algorithm

Agenda cont …

Day 3 cont …

- Unit 8: Physical PME Data Model and Mapping
  - Exercise 7a:Loading default Physical PME Configuration
- Unit 9: Physical MDM Algorithms
  - Exercise 9a: Exploring and Customizing the default algorithm
  - Exercise 9b: Analyzing our Buckets
  - Exercise 9c: Generating Weights
  - Exercise 9d: Deploying the Physical MDM PME Configuration
- Unit 10: Physical PME Adapters and Converters
  - Exercise 10 Create a Custom Converter

Figure 0-4. Agenda cont … ZZ7801.0

Notes:

Day 3 cont …

- Unit 8: Physical PME Data Model and Mapping
  - Exercise 7a:Loading default Physical PME Configuration
- Unit 9: Physical MDM Algorithms
  - Exercise 9a: Exploring and Customizing the default algorithm
  - Exercise 9b: Analyzing our Buckets
  - Exercise 9c: Generating Weights
  - Exercise 9d: Deploying the Physical MDM PME Configuration
- Unit 10: Physical PME Adapters and Converters
  - Exercise 10 Create a Custom Converter

Introductions

- Name
- Company
- Where you live
- Your job role
- Current experience with products and technologies in this course
- Do you meet the course prerequisites?
- Class expectations

Figure 0-5. Introductions ZZ7801.0

Notes:

Introductions

- Name
- Company
- Where you live
- Your job role
- Current experience with products and technologies in this course
- Do you meet the course prerequisites?
- Class expectations

Unit 1. PME and Virtual MDM Overview

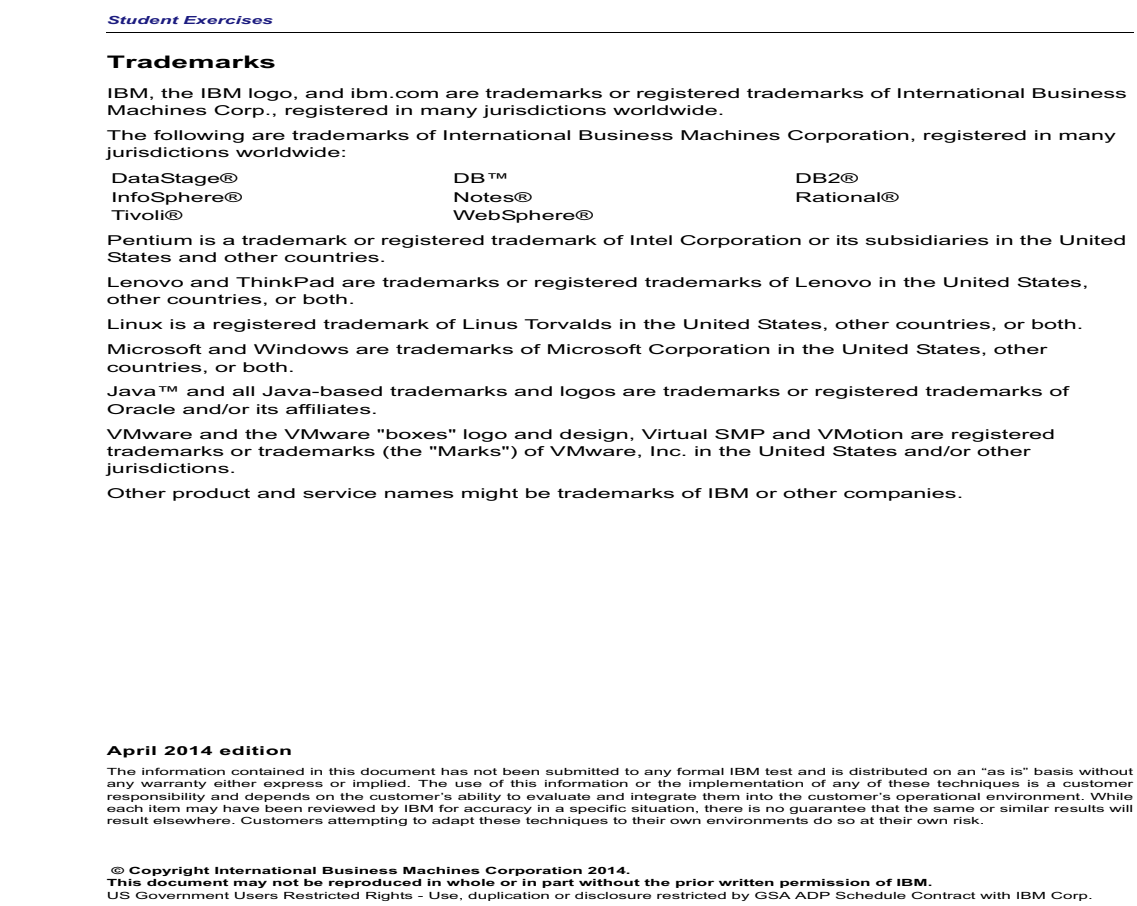
What this unit is about

This unit describes the key concepts behind the InfoSphere MDM Virtual module, including terminology and the basics of matching. This unit is an overview of the topics we will be discussing throughout the course.

What you should be able to do

After completing this unit, you should be able to:

- Describe key concepts of Virtual MDM
- Understand Virtual MDM terminology
- Understand how the PME is used by the Virtual MDM



Student Exercises

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

DataStage®	DB™	DB2®
InfoSphere®	Notes®	Rational®
Tivoli®	WebSphere®	

Pentium is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Lenovo and ThinkPad are trademarks or registered trademarks of Lenovo in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

VMware and the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the "Marks") of VMware, Inc. in the United States and/or other jurisdictions.

Other product and service names might be trademarks of IBM or other companies.

April 2014 edition

The information contained in this document has not been submitted to any formal IBM test and is distributed on an "as is" basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will result elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Figure 1-1. Unit objectives ZZ7801.0

Notes:

After completing this unit, you should be able to:

- Describe key concepts of Virtual MDM
- Understand Virtual MDM terminology
- Understand how the PME is used by the Virtual MDM

Contents

Figure 1-2. Virtual Implementation Overview ZZ7801.0

Notes:

The Virtual module behaves as a registry-style MDM implementation. Matching and linking member records from any number of sources is the main goal of the InfoSphere MDM Virtual module.

The biggest strength of a registry-style implementation is that we can accept data from as many sources that will be supplying data to the implementation. The Virtual module does not change any of the source data, it accepts the data that has been passed to it.

If Virtual MDM find other similar members in the implementation, we can link them together and create a unified, golden view of the member(s) to give you a 360 degree, current view of the member information.

If there is any question that the possibility of matching members together could still be in question, Virtual MDM creates a “task” that will need to be reviewed to help make the final determination if they are the same or not.

Figure 1-3. Virtual Implementation Overview ZZ7801.0

Notes:

With all of the member information in the Virtual modeul, we have the ability to meet your business goals of allowing you to search for members – or members who match and link and create a single golden view.

This information can be used to help your business in ways such as populating a data warehouse, or for business intelligence to make better decisions about your members.

Figure 1-4. Workbench ZZ7801.0

Notes:

Workbench is a graphic user interface that provides user management and configuration management tools for IBM InfoSphere MDM Virtual module. Simply put, it allows you to view and manage the configuration for Data or Relationships.

Using Workbench, a Virtual data model, algorithm, and thresholds can be easily adjusted to your requirements using a single toolset. Graphical analytics are available to correctly adjust the algorithms and thresholds to increase the accuracy and performance of a particular implementation. These features make it much easier for algorithms to be tuned for performance, and to improve the accuracy of matching based on analytics returned from Workbench.

Basic functionality

Workbench Configuration projects can be created and configured without a InfoSphere MDM instance or a data source. This is different from previous versions of configuration tools, like Identity hub Manager, where the engine and databases were written to directly. Instead, Workbench saves the configuration and uploads it to the engine allowing changes to be made “off line”.

Workbench allows users to perform many tasks that were once completed using scripts or multiple MDM software packages. Some of the key functionality includes:

- Creating, configuring, and editing member model dictionaries
- Creating, configuring, and editing algorithms
- Cleaning and de-duplicating data in the data extract
- Bucket analysis
- Threshold analysis
- Entity analysis

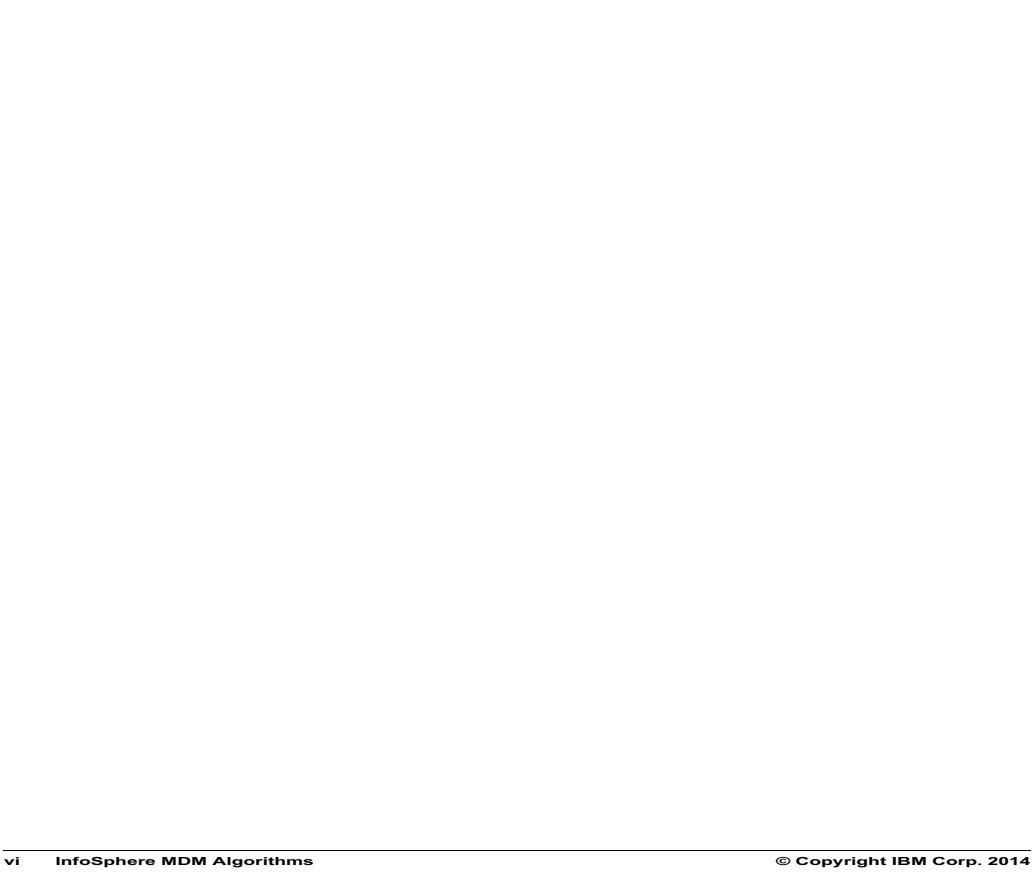


Figure 1-5. Inspector ZZ7801.0

Notes:

Inspector is a web-based interface used primarily by Data Stewards and Data Governors.

It allows the user to do searching for members and entities.

It also is the UI that Data Stewards use to resolve tasks by using drag and drop functionality.

You can also use Inspector for Relationships and Hierarchy management.

Data Governance can also be accomplished with Inspector when data manipulation is required.

Student Exercises

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

DataStage®	DB™	DB2®
InfoSphere®	Notes®	Rational®
Tivoli®	WebSphere®	

Pentium is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Lenovo and ThinkPad are trademarks or registered trademarks of Lenovo in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

VMware and the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the "Marks") of VMware, Inc. in the United States and/or other jurisdictions.

Other product and service names might be trademarks of IBM or other companies.

 Trademarks vii

Figure 1-6. Pair Manager ZZ7801.0

Notes:

Pair Manager is a stand-alone client that is used during the Threshold Analysis stage to allow end-users to look at a sample matched pair of members and determine if they are the same; not the same; or not enough information to determine same or not (basically determining that a tasks should be created for review).

It is used to help in determining what the Algorithm thresholds should be set to.

It allows the end user to look for and irregularities that may need to have the configuration adjusted to handle. Items such as False Positives.

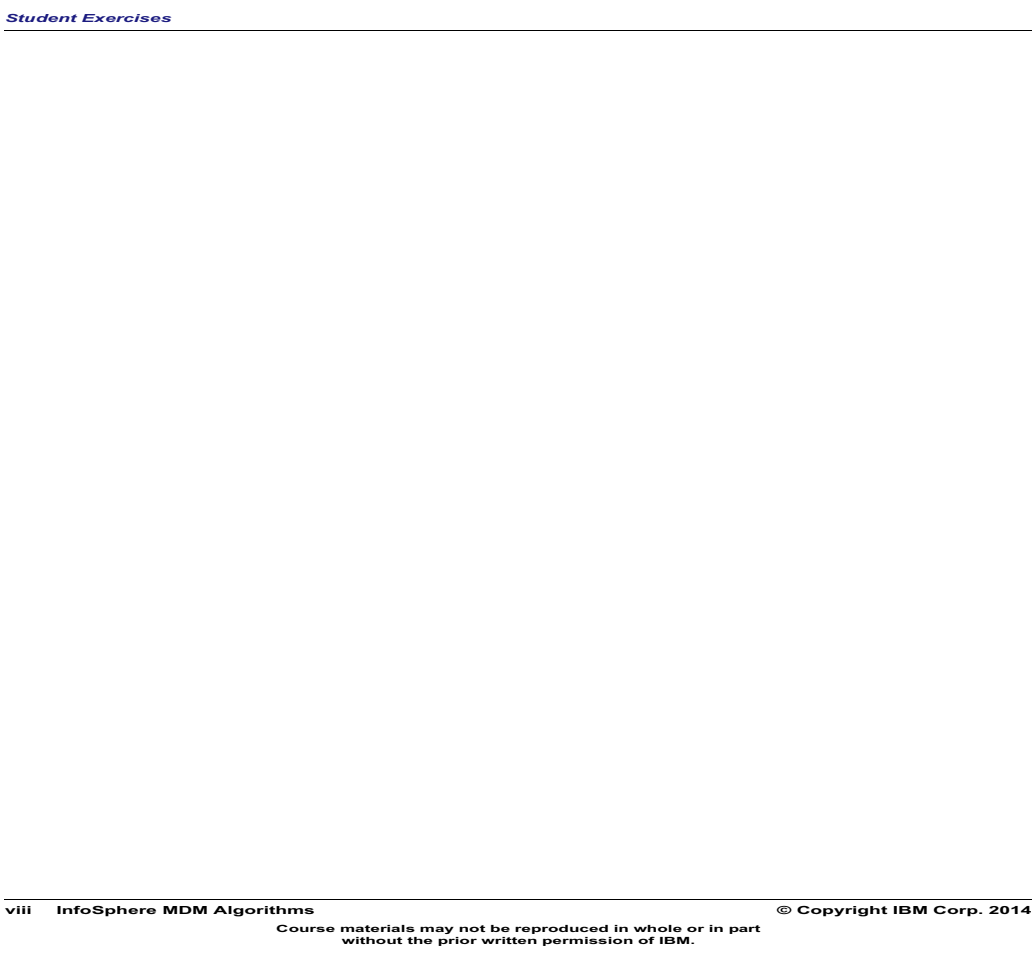
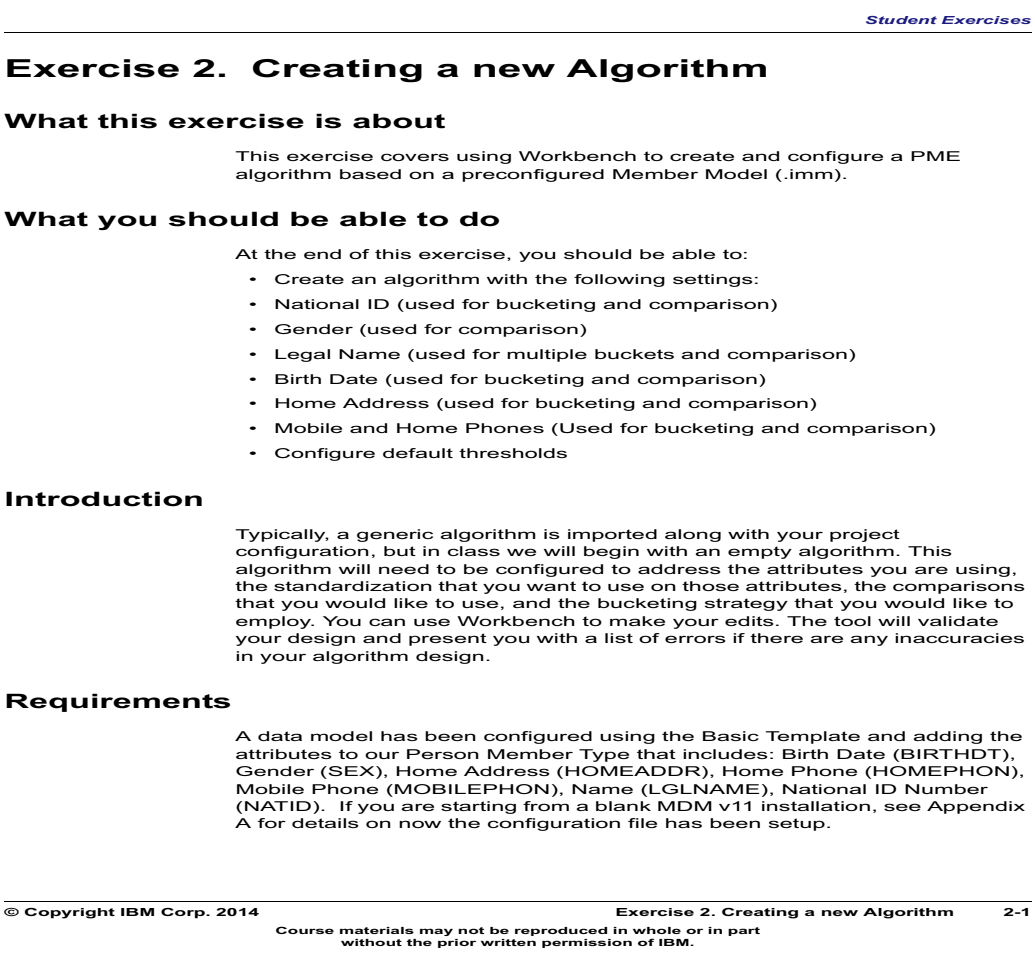


Figure 1-7. Terminology ZZ7801.0

Notes:

On the next slides, we will start discussing some of the terminology that is used when discussion the InfoSphere MDM Virtual module.

This list will cover the main items that we will be using.



Student Exercises

Exercise 2. Creating a new Algorithm

What this exercise is about

This exercise covers using Workbench to create and configure a PME algorithm based on a preconfigured Member Model (.imm).

What you should be able to do

At the end of this exercise, you should be able to:

- Create an algorithm with the following settings:
- National ID (used for bucketing and comparison)
- Gender (used for comparison)
- Legal Name (used for multiple buckets and comparison)
- Birth Date (used for bucketing and comparison)
- Home Address (used for bucketing and comparison)
- Mobile and Home Phones (Used for bucketing and comparison)
- Configure default thresholds

Introduction

Typically, a generic algorithm is imported along with your project configuration, but in class we will begin with an empty algorithm. This algorithm will need to be configured to address the attributes you are using, the standardization that you want to use on those attributes, the comparisons that you would like to use, and the bucketing strategy that you would like to employ. You can use Workbench to make your edits. The tool will validate your design and present you with a list of errors if there are any inaccuracies in your algorithm design.

Requirements

A data model has been configured using the Basic Template and adding the attributes to our Person Member Type that includes: Birth Date (BIRTHDT), Gender (SEX), Home Address (HOMEADDR), Home Phone (HOMEPHON), Mobile Phone (MOBILEPHON), Name (LGLNAME), National ID Number (NATID). If you are starting from a blank MDM v11 installation, see Appendix A for details on now the configuration file has been setup.

 Exercise 2. Creating a new Algorithm 2-1

Figure 1-8. Source ZZ7801.0

Notes:

Source: A source is any system that contributes member attributes (records) to the InfoSphere MDM.

Sources can add / update member data via any integration point such as the Inbound Message Broker, API, or periodic data feeds

A source will provide the InfoSphere MDM with 1 or more members.

For Example, the Sources here include: CRM, Sales, Web, and Data Warehouse

There are 2 different types of sources.

1. **Definitional source**: this is a source system that provide member records / attributes.
2. **Informational source**: a lookup or referral source that provides unique identifiers to a definitional source. For example, a Social Security Administration creates SSNs and the Department of Motor Vehicles create unique drivers license numbers.

Student Exercises

Exercise instructions

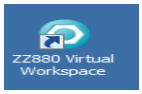
Part 1: Opening the PERSON algorithm

___ 1. Login to the VMWare image using the following credentials:

___ a. Username: **Administrator**

___ b. Password: **passw0rd**

___ 2. Once you are logged in, open the RAD environment by double clicking on the **ZZ880 Virtual Workspace** link on the desktop.



In the workspace, you'll find a PartyConfiguration project that was already created. (See Appendix A on how this was created). The PartyConfiguration was created using the basic template and the following Attributes:

- Birth Date
- Gender
- Home Address
- Home Phone
- Mobile Phone
- Name
- National ID Number

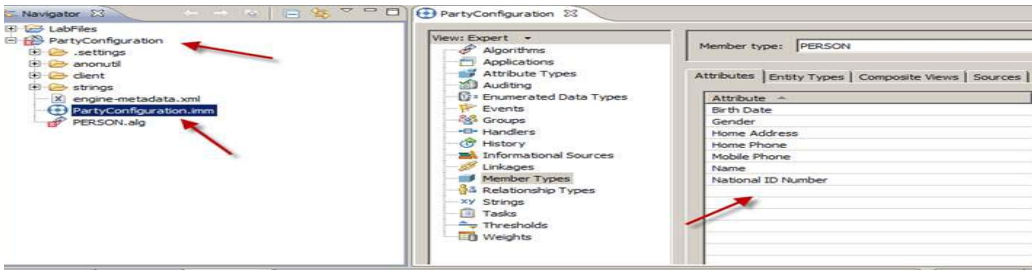
Figure 1-9. Entity ZZ7801.0

Notes:

Entity: A distinct person, organization, location, etc. that is represented by assigning the same Enterprise Identifier (EID) to one or more member records.

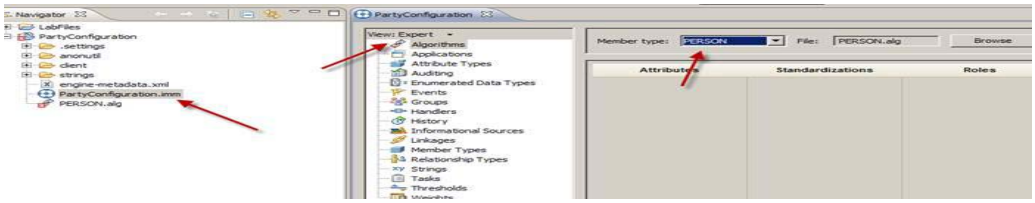
An Entity is essentially a link between one or more member records.

In this example Deb Becker-Smith is the Entity (EID 456) that links to her three Member records below.



The PartyConfiguration data model is what we will work with for our algorithm implementation. You'll may see errors in our workspace, this is because we have not configured the algorithm yet for this configuration project.

___ 3. Open the configuration file by double clicking on the **PartyConfiguration.imm**. In the Configuration perspective, select the Algorithms tab. You'll notice that the algorithm is currently empty for our only Member Type (PERSON).



Part 2: Configure the National ID Number attribute

We will start off by configuring the National ID Number algorithm piece. For the National ID Number we will create a bucket to allow the comparison of records with similar National ID Number. The image below is the configuration of our National ID that we would like to create:

Figure 1-10. Member ZZ7801.0

Notes:

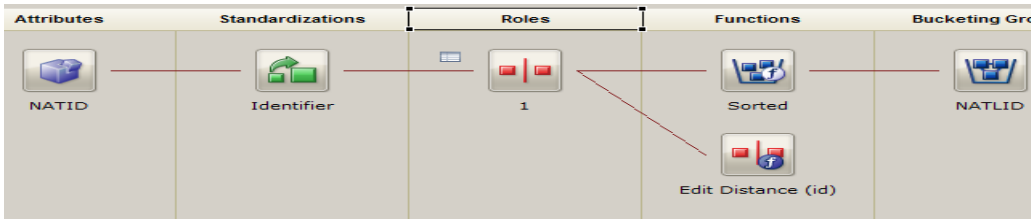
Member: A record that comes from a source which contains current and/or historical information
* Also referred to as a "Record"

A member is made up of 1 or more attributes.

In this example: member (record) 319883 is made up of attributes Name; Work Address; Home Phone; email.

Student Exercises

- Standardization: **Identifier** (special characters are filtered out, e.g. 675-asd-7008 is converted to 675ASD7008)
- Bucketing Function: **Sorted** (sorts the characters for buckets, e.g. ADS0056778)
- Comparison: **Edit Distance**



___ 1. We start off by adding the National ID Attribute to our algorithm. On the right hand side, expand the **Attributes** view in the palette. Select the **NATID (IDENT)** and place the attribute in the first column.

___ 2. Next, we will add the standardization function. On the right hand side of the algorithm, expand the **Standardizations** view in the palette and select the Identifier function. Place the Identifier standardization in the second column beside the NATID.

Figure 1-11. Attribute ZZ7801.0

Notes:

Attribute: A demographic data element made up of one or more fields that identifies the traits of a member record

Drilling down into Debbie Beckersmith's Member record, you can see her attributes: phone, name, DOB, gender, and address
Attributes can be comprised of one or more fields, for example:

- Dates are single-field attributes that are stored in string format
- Names are multi-field attributes (First-Name, Middle-Name, Last-Name, Title, etc)
- Other Attributes, such as Phone (whose Attribute Name is MEMPHONE), can contain multiple Attribute Types. (NEXT SLIDE)

___ 3. To connect the NATID to the Identifier, select the **Connection** icon in the palette. Once the Connection tool is selected, click the **NATID** icon in the Attributes column and then click the **Identifier** icon under the Standardization column.

___ 4. Press the **ESC** key or select the **Select** icon in the palette to exit the Connection tool.

___ 5. Next we need to customize the function's properties. Select the **Identifier** icon in the Standardizations column. In RAD, you will find a Properties window (bottom left hand corner). Select the **Properties** window tab.

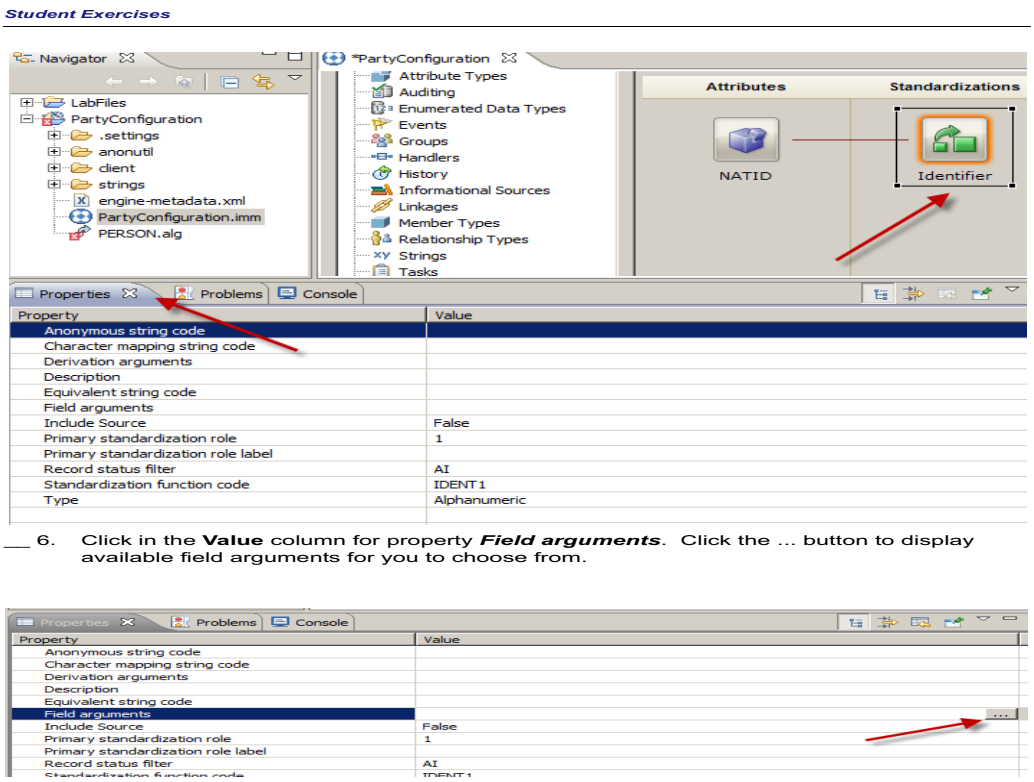
Figure 1-12. Attribute Type ZZ7801.0

Notes:

Attribute Type: The database segment (or table) in which similar data elements are stored

This slide shows that we have 3 different segments (attribute type)

- MEMADDR is used to store address information about members
- MEMPHONE is used to store information about member phone numbers
- MEMNAME is used to store information about member names



___ 6. Click in the **Value** column for property ***Field arguments***. Click the ... button to display available field arguments for you to choose from.

___ 7. Since there is only one field from an IDENT attribute type and the Identifier standardization function accepts one field, the **idnumber** will already be selected. Click the **OK** button.

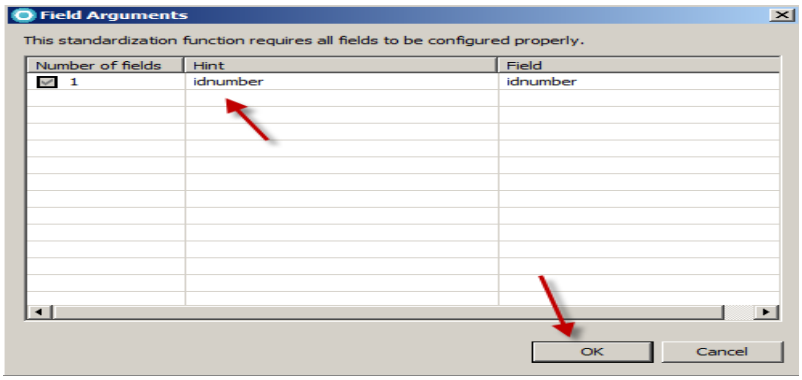
Figure 1-13. Matching ZZ7801.0

Notes:

The InfoSphere MDM has two methods for matching records:

1. **Deterministic Matching**: A rules-based process to determine an “exact match” between two records
   - Works best for simple, easily-defined matches
   - Skeptical form of matching – Yes or No -> The correlation between the two records must pass the test to be trusted.
   - InfoSphere MDM uses deterministic matching for relationships and hierarchies
     - e.g. 2 records with exact same address should be linked as one household
     - e.g. Employee-to-Supervisor relationships also use deterministic matching
2. **Probabilistic Matching:** A process of using statistical analysis to determine the overall likelihood that two records match
   - Preferred method for matching large data sets or when a large number of attributes are involved in the matching process

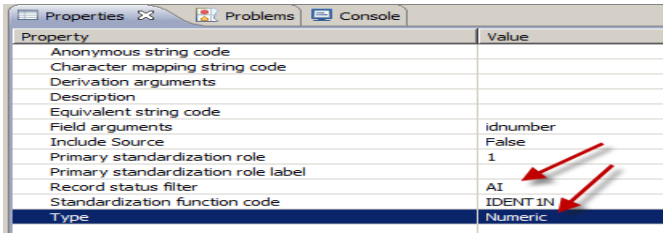
- How much Confidence, based contextually on your data, do we have that two records match
- Optimistic form of matching – Maybe! -> Looks for any level of correlation between the two records. So for the two mismatched records, the system would say “hey, look at that, you have almost nothing in common, but you both live on a Street and have the middle initial “g”. The score is slightly credited with that correlation, even though it might still be very negative overall.
- e.g. When you use a Search Engine to find something on the web, the results will return with an x% that the results match your search criteria.



___ 8. Under the **Properties** window, change the following values:

___ a. Record Status Filter: **AI**.

___ b. Type: **Numeric**.



___ 9. Once the value has been standardized, all the values get placed in a Comparison Role (this feeds the bucketing and comparison functions). Back in the Algorithms window, select the **Comparison Role** icon in the palette and place it in the 3rd column beside the Identifier standardization.

Figure 1-14. Algorithm ZZ7801.0

Notes:

The Algorithm is a series of processes (Standardization, Bucketing, and Comparison) used to probabilistically match records that results in a comparison score.

An algorithm uses the following steps

1. **Standardization** converts data into its simplest form for easier use during the matching process. In other words, standardization is used to clean up the data by reformatting it into consistent chunks.
2. **Bucketing** organizes records that share common values for faster search retrieval. Think of bucketing this way: is it easier to find a needle in a hay stack or in a jar labeled “needles”?
3. **Comparison** uses the probabilistic method to compare pairs of records and then assign a score based on the similarities and differences between the two records. In other words, the comparison function is a means for finding similarity or difference between two records and their attributes.

For Example:

Using the Phone Number (512) 634-5144, the standardization process removes the non-numeric data [()-] and shaves the numbers down to the last 7 digits (6345144)

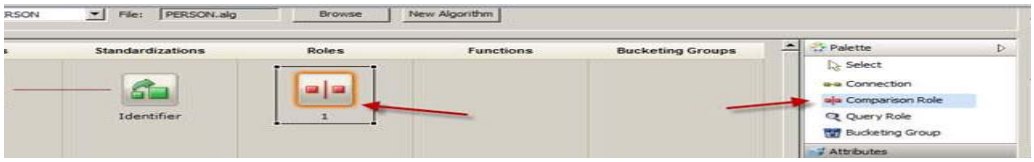
The Bucketing function (6345144) would be sorted in Ascending order (1344456) and placed in the “1344456” bucket

Comparison: phone numbers 6345144 and 6345414 have an edit distance of one (a simple transposition of the “14” versus a “41” which has a high correlation

So would likely score around 3.9 points

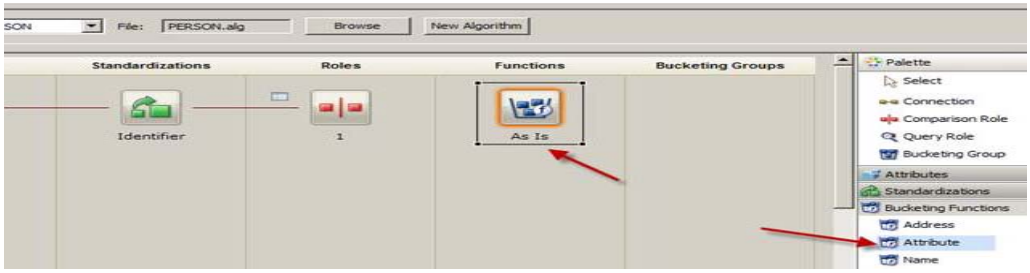
While we have only reviewed this process for Phone Numbers, it is important to note that during the Algorithm’s analysis of the data, it runs parallel analyses of each of the attributes in the records.

Student Exercises



___ 10. Connect the **Identifier** Icon to the **Comparison Role 1** using the Connection tool from the palette.

___ 11. Next we will add our bucketing, this will be based on a sorted value of our Identifier, meaning it will be compared to other ID with the same character values (but possibly in a different order. Expand the **Bucketing Functions** view in the palette and select the **Attribute** icon. Place the Attribute bucketing function in the fourth row of the algorithm.



___ 12. Connect the **Comparison Role 1** to the **As Is** bucketing function using the Connection tool from the palette.

Figure 1-15. Standardization: Are these addresses the same? ZZ7801.0

Notes:

The first step of the algorithm is to standardize the data to be used by the algorithm.

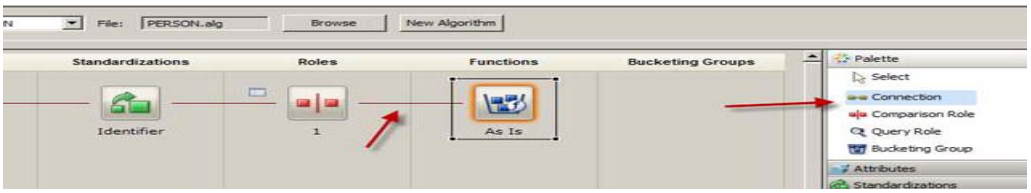
This slide is an example of one of the existing standardization functions that is used against address data.

For all standardization functions, 2 things happen – 1st convert all the data to UPPER CASE and 2nd remove any punctuation to cleanse the data.

** note that this is only happening to the data that will be used to create the derived data used by the algorithm – the source data stays intact **

Then finally, using the standardized data – and depending on the standardization function you choose – parse the data elements into individual fields (tokens) that can be used for comparison.

The bottom of the slide is showing you how each of the 4 address provided would look once standardized.

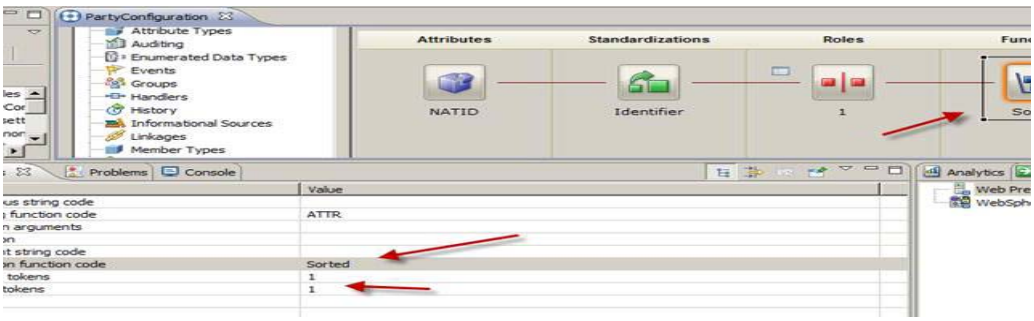


___ 13. Select the **As Is** bucketing function icon in our algorithm and open the **Properties** window in RAD. Change the following values in the Properties window:

___ a. Generation Function Code: **Sorted**.

___ b. Max Tokens: **1**.

___ c. Min Tokens: **1**.



___ 14. Now that the bucketing function is set up, we need the actual bucket group in place. Select the **Bucketing Group** from the palette and place the Bucketing Group in the 5th column.

Figure 1-16. Bucketing: What do these people have in common? ZZ7801.0

Notes:

The second step of the algorithm is to create the searchable buckets of the standardized data.

Look at the following five people. What are the things they have in common?

Create buckets based off of the records shared criteria, then note which people would be in the bucket (e.g., people with black hair).

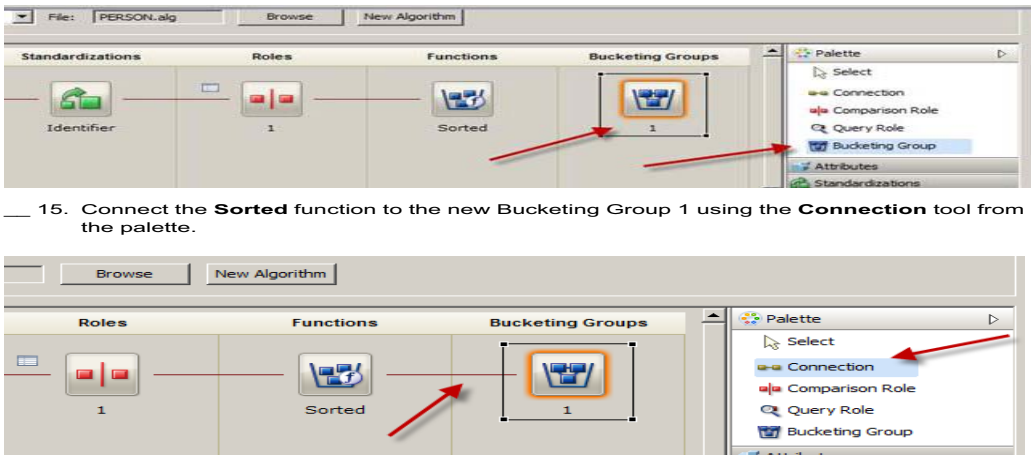


Figure 1-17. Comparison Functions: Which one would you use? ZZ7801.0

Notes:

The third step of the algorithm is to do the comparison of the standardized data.

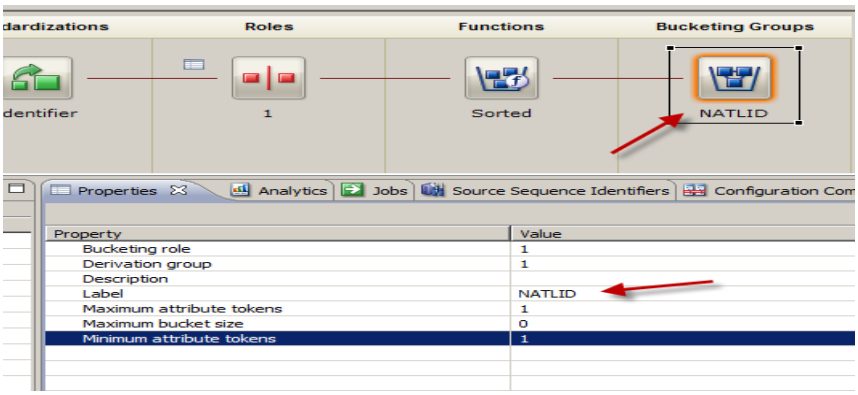
There are several forms of comparison that exists within the engine / algorithm.

It is not strict like a deterministic algorithm where “If first name = JOHN and middle name = G and birth date = 10/23/1960

The probabilistic algorithm allows you to take into consideration things like:

- Exact match
- Starts with
- Edit distance – how many changes need to happen to make the 2 values the same
- Phonetics – Stacie vs Stacy
- Equivalency – Rob = Bob = Robert
- (date of birth) YEAR – does the year match between these to members?

You have the ability to choose from the list of comparison functions when configuring your algorithm to best meet the needs of your customer for this project.



___ 17. Not only will we compare members that have a similar national id, we will also use the edit distance between the national ids to contribute to the overall comparison score. To add a comparison function, expand the **Comparison Functions** view in the palette, select the **Edit Distance** and place the function in the 4th column under the **Sorted** bucketing function.

Figure 1-18. Weights ZZ7801.0

Notes:

Weights are used to define what the statistical frequency is of this piece of data for this attribute for the dataset.

When you are comparing members, you compare as much data as provided for the 2 members.

You can control how the comparison is done using the different comparison functions so specific weights are used depending on the comparison function. The weight generation process will create weights for those attributes that are determined to be used the most throughout the dataset.

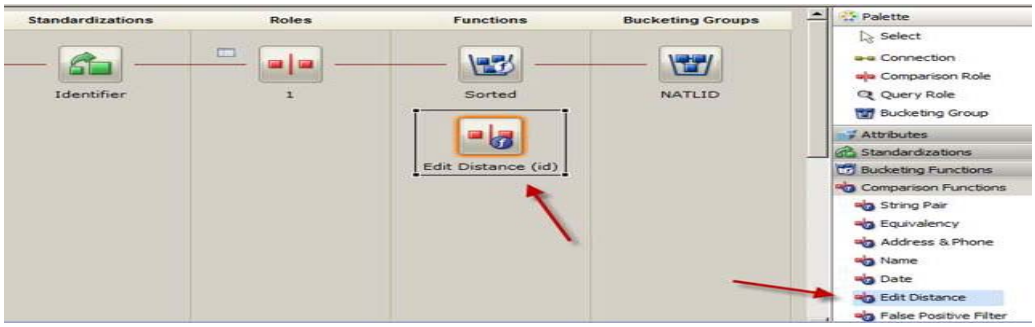
Weights can be positive or negative values.

Here are examples of the 3 weight types that get created.

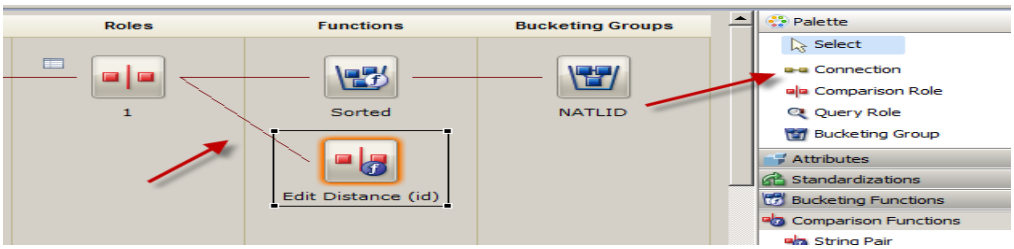
Frequency-based Weights: provide a score based on how often a value appears within the overall data population. Common values (like John) have a low score, rare values (like Chitsumungo) have a high score.

Edit Distance Weights: are a measure the similarity between two values. For example, “Gordon” vs. “Gorton” has a distance of 1 edit. Exact match has the highest score, but each edit lowers the score by a certain degree.

Parameterized (PARM) Weights: These weights control maximum caps on scores, extra credit points, and penalties for variance. For example, there is a maximum weight for Full Name that ensures that the name does not generate a disproportionate score.



___ 18. Using the Connection tool from the palette, connect the **Comparison Role 1** to the **Edit Distance (id)** comparison function.



___ 19. Press the **ESC** key to exit from the Connection tool.

___ 20. Select the **Edit Distance (id)** icon in the Functions column and open the **Properties** tab in RAD. Change the following values for our comparison function:

- ___ a. Comparison Spec Code: **NATLID**.
- ___ b. Enable Review ID Task: **Yes**.
- ___ c. Kind: **1 Role, 1 Dimension.**
- ___ d. Type: **Quick**.

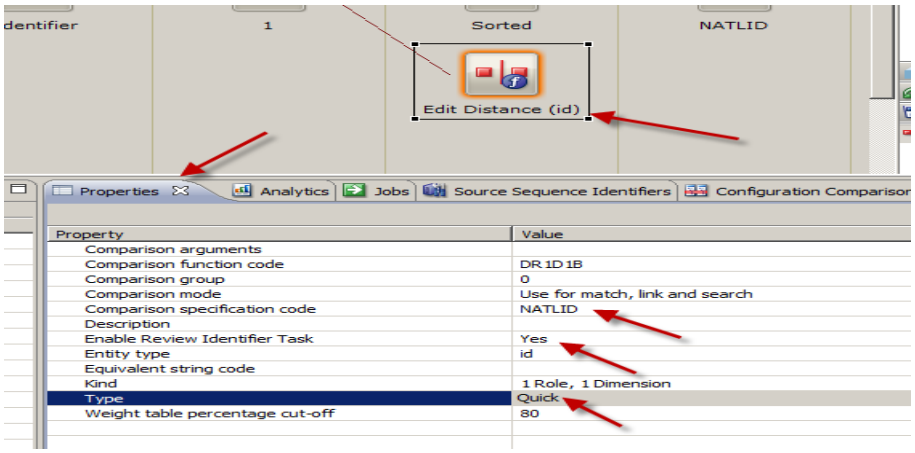
Figure 1-19. Comparison Score ZZ7801.0

Notes:

When 2 members are being compared, each attribute comparison will produce a weight determined by which comparison function you have chosen to use for the attribute.

The comparison score is the accumulation of these weights that are determined by the algorithm for each attribute comparison that you have configured.

Then use the comparison score to determine where on the scale these 2 members compare.



___ 21. Save the project **(CTRL+S)**

Part 3: Configure the sex attribute

The sex (or gender) attribute will not be used for bucketing (since they would create huge buckets) but we would like to give a small value to the comparison score if the gender matches. Since we've walked you through creating the algorithm in the first part, in this part you will attempt to create the Sex algorithm without the steps. (if you get stuck we provided the steps in Appendix B).

___ 1. Create the following algorithm pieces (and connections) for the SEX attribute with the following values:

- ___ a. Attribute: **SEX (ATTR)**
- ___ b. Standardization: **Attribute**
  - i. Field arguments: **1 - attrval**
  - ii. *Record Status Filter*: **AI**.
  - iii. *Type*: **Alphanumeric**.
- ___ c. *Comparison Role*
- ___ d. *Comparison Function: **Equivalency***

Figure 1-20. Linkages ZZ7801.0

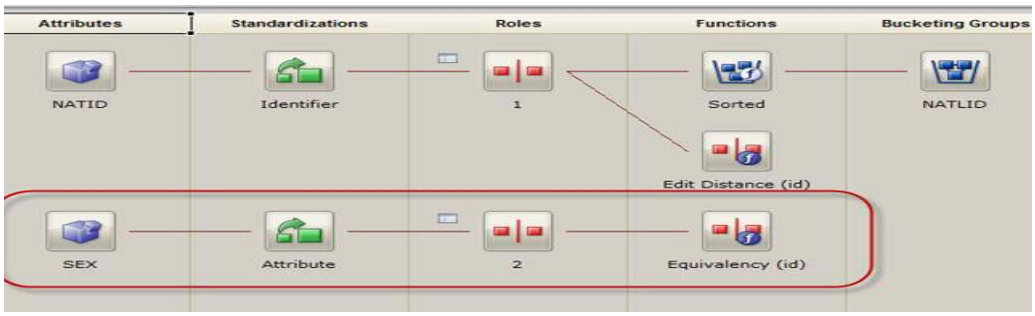
Notes:

Link: (v) The act of assigning two Member records to the same Entity either manually (using Inspector) or automatically (if the comparison score crosses the AL Threshold)

Going back to the Probabilistic Matching "maybe continuum," let's look at when Member records are automatically or manually assigned to an Entity.

- If the two Member records' comparison score falls at or above the AL Threshold, the records will be automatically linked into the same entity.
- If the two Member records' comparison score fall above the CR Threshold, but below the AL Threshold, a Task will be created.
- If the two Member records' comparison score is below the CR Threshold, these 2 members are not similar at all and would be put into their own entity.

iv. Comparison Spec Code: SEX.
v. Enable Review ID Task: No.
vi. Type: Alphanumeric.



Part 4: Configure the legal name attribute

For the Person's Name, we will create 2 Bucketing functions, one that includes the Name + Birth Date and another that includes only the Last Name and Zip code. The reason we don't want to create a bucket for just the Name is that the buckets might become very large and impact performance, so we will compare people that have similar sounding names (phonetic) and have the same Birth Data or Zip Code.

For the comparison of the Name, we will use the Name comparison function. The Name comparison function will compare each token in the Names, providing equivalency names, bonuses for position and frequency based scores (i.e. common names score lower than unique names)

Legal name plus Birth Date

We will start off creating the Bucketing for the Legal Name and Birth Date. The detailed steps to this algorithm piece are also found in **Appendix A** if you require additional information.

___ 1. Create the following algorithm pieces (and connections) for the **LGLNAME** attribute with the following values:

___ a. Attribute: **LGLNAME**

___ b. Standardization: **Name**

vii. Field arguments: **onmlast, onmfirst, onmmiddle, onmprefix, onsuffix.**

viii. Primary standardization role label: **Name Tokens**.

Figure 1-21. Entity Management ZZ7801.0

Notes:

Entity Management is the process that runs in the background to do the member comparison and determine if 2 members should be in the same entity, create a task, or be in their own entity.

Entity Management can be part of the running engine process or as a stand-alone process.

ix. Record Status Filter: **AI**.
x. Type: **Person**.
xi. Anonymous string code: **ANONNAME**.

__ c. Comparison Role

Hint

For the Field argument of the Standardization Function, you will need to check each of the checkbox 1 to 5 and select the values in the order above.

__ d. *Bucketing Function:* ***Name***

i. *Description*: **Full Name**.
ii. *Generation function code*: **Equivalence & Phonetic**.
iii. *Derivation arguments*: **NORMPHONE**
iv. *Equivalent string code*: **ALTNAME**.
v. *Maximum tokens*: **1**.
vi. *Minimum tokens*: **1**.
vii. *Type*: **Person**.

Note

By setting the Maximum token and minimum token to 1, this means the buckets that are generated will only have one token from the Legal Name in order to create the bucket. This is important because we want our buckets to use one token from the Name and the other from the Birth Date.

__ e. Bucketing Group

i. *Description:* **Use 1 name token + dob to build.**
ii. *Label:* **1 Name token + DOB**.
iii. *Maximum attribute tokens:* **2**.
iv. *Maximum bucket size:* **0**.
v. *Minimum attribute tokens:* **2.**

Figure 1-22. Thresholds ZZ7801.0

Notes:

There are 2 threshold that exist within the InfoSphere MDM – Autolink and Clerical Review.

The **Autolink** threshold is the upper limit / score where when 2 members are being compared and their comparison score is equal to or greater than the AL Threshold, these 2 members are considered to be the same and will be assigned the same entity Id (EID).

The **Clerical Review** threshold is the lower limit / score where when 2 members are being compared and their comparison score is less than the CR Threshold, these 2 members do not have enough in common that they are 2 different members. They would then be assigned their own entity ID (EID).

If the comparison score is equal to or greater than the CR Threshold but less than the AL Threshold, a task will be created for these members.

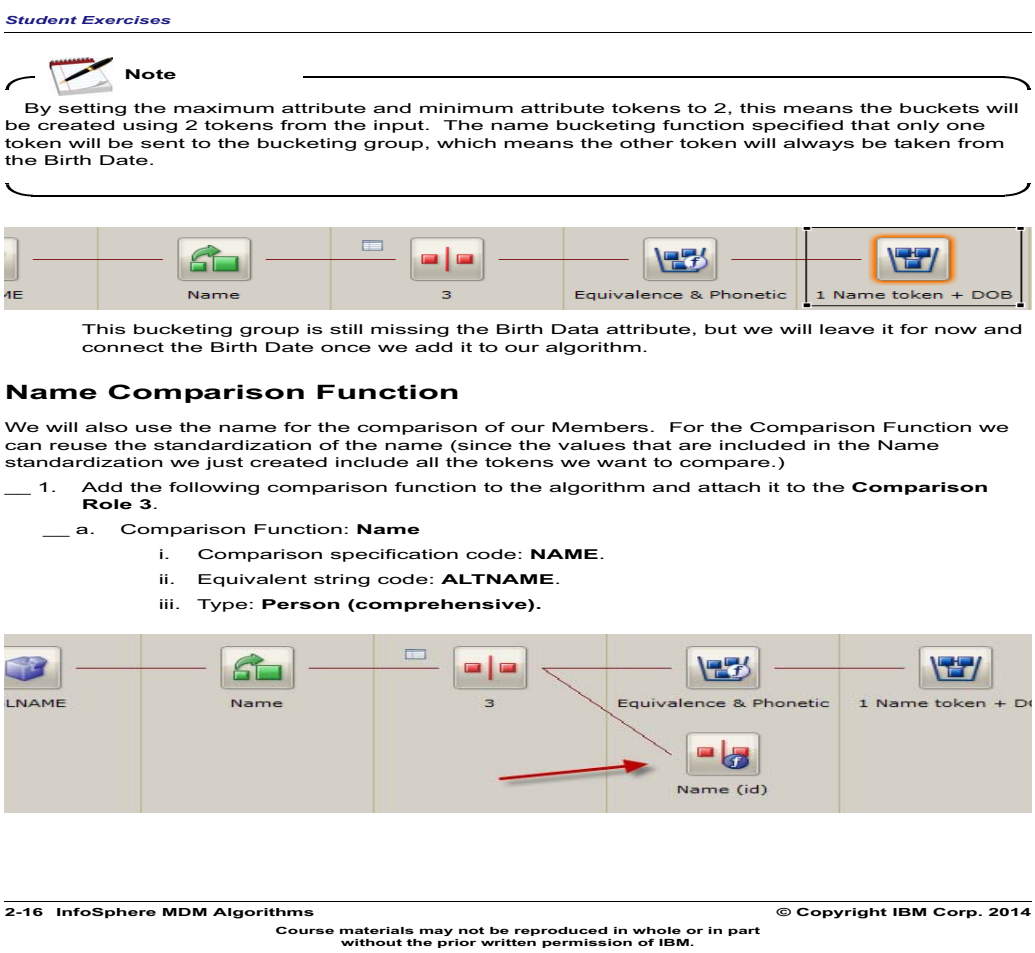


Figure 1-23. False Positives and False Negatives ZZ7801.0

Notes:

There is the potential where the algorithm is not smart enough or configured correctly to handle every possible scenario.

Certain situations could create a False Positive or a False Negative.

A **False Positive** occurs when 2 members are linked together when they are actually 2 different members.

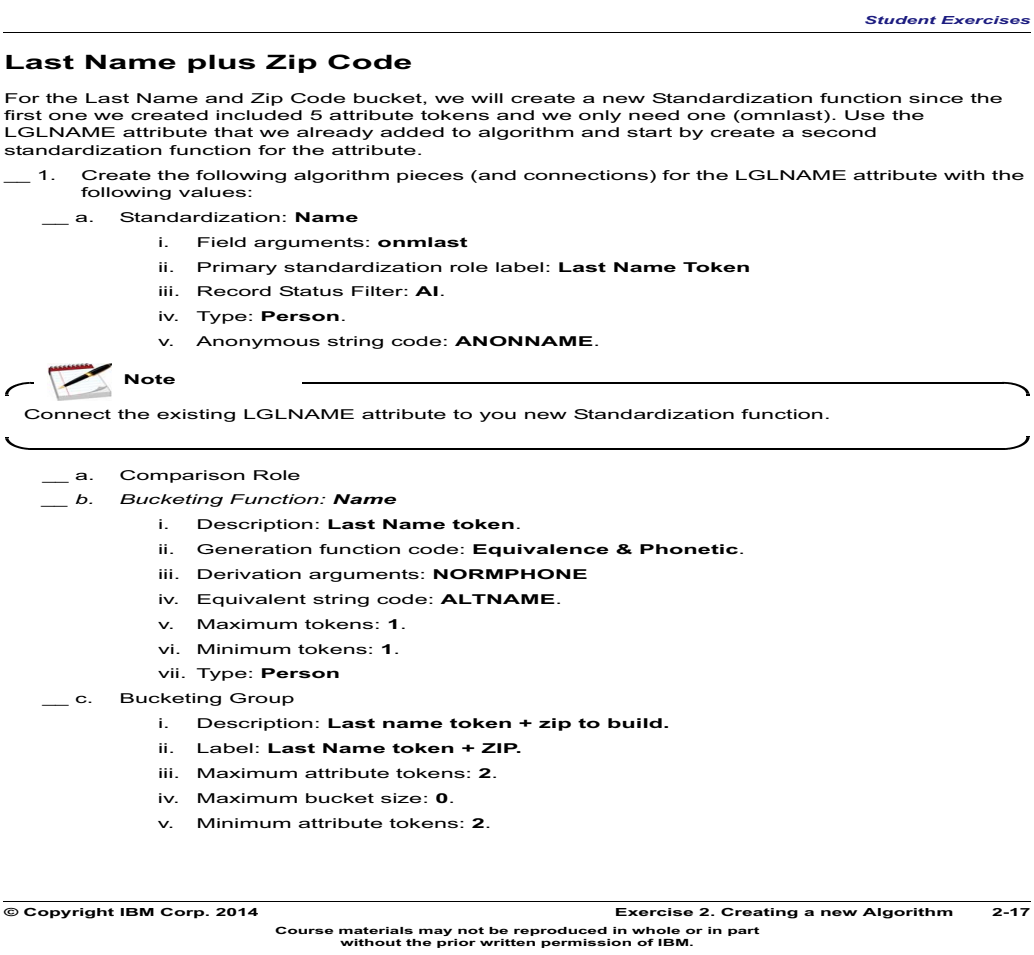
This is usually the case in situations such as multiple births where same last name, same address, same home phone, etc..

A **False Negative** occurs when 2 members are not linked together when they are actually the same member.

This usually happens when there is thin data being provided by the sources and the algorithm does not have enough information to compare.

There are special comparison functions that exist to help in handling the False Positive since this can cause bigger issues than a False Negative.

This special comparison function is called the False Positive Filter (FPF).



Student Exercises

Last Name plus Zip Code

For the Last Name and Zip Code bucket, we will create a new Standardization function since the first one we created included 5 attribute tokens and we only need one (omnlast). Use the LGLNAME attribute that we already added to algorithm and start by create a second standardization function for the attribute.

___ 1. Create the following algorithm pieces (and connections) for the LGLNAME attribute with the following values:

___ a. Standardization: **Name**

i. Field arguments: **onmlast**
ii. Primary standardization role label: **Last Name Token**
iii. Record Status Filter: **AI**.
iv. Type: **Person**.
v. Anonymous string code: **ANONNAME**.

Note

Connect the existing LGLNAME attribute to you new Standardization function.

___ a. Comparison Role

___ *b.* *Bucketing Function:* ***Name***

i. Description: **Last Name token**.
ii. Generation function code: **Equivalence & Phonetic**.
iii. Derivation arguments: **NORMPHONE**
iv. Equivalent string code: **ALTNAME**.
v. Maximum tokens: **1**.
vi. Minimum tokens: **1**.
vii. Type: **Person**

___ c. Bucketing Group

i. Description: **Last name token + zip to build.**
ii. Label: **Last Name token + ZIP.**
iii. Maximum attribute tokens: **2**.
iv. Maximum bucket size: **0**.
v. Minimum attribute tokens: **2**.

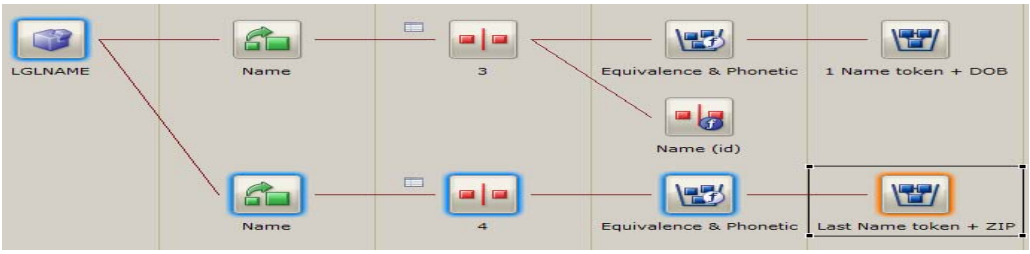
Figure 1-24. Tasks ZZ7801.0

Notes:

Task: (n) A record, or set of records, that requires human intervention to make a decision about two joining records that fall between the CR & AL thresholds

1. Potential Overlay occurs when a record is updated with information that is radically different than the data that was already there
   - Example: (customer support)
     Example: (married)
2. Potential Duplicate occurs when two records are in the same source and appear to represent the same person or organization.
   - Example: (same person – same source)
   - Example: (baby or twins – same source)
3. Potential Linkage occurs when two records in different sources and appear to represent the same person or organization.
   - Example: (same person – 2 sources)
   - Example: (baby or twins – 2 sources)

4. Review Identifier occurs when two records from the same source seem to be using the same identifier (SSN, Passport, Insurance, or Credit Card Number)
   - Example: (identity theft)
   - Example: (household credit card)



Part 5: Configure the birth date attribute

For the Birth Date, we won't create a bucket by itself (buckets would be too large), but will attach the attribute to our Name plus DOB bucket for the second token in that bucket. We will also create a date comparison function for the Birth Date that will provide a score when the day, month or year match.

___ 1. Create the following algorithm pieces (and connections) for the BIRTHDT attribute with the following values:

- ___ a. Attribute: **BIRTHDT**
- ___ b. Standardization: **Date**
  - i. Field Arguments: **dateval**
  - ii. Anonymous string code: **ADATE**.
  - iii. Primary Standardization role label: **Birth Date.**
  - iv. Record status filter: **AI**.
  - v. Type: **Standard**.
- ___ c. Comparison Role
- ___ *d. Bucketing Function: Date*
  - i. Description: **Birthdate**.
  - ii. Generation function code: **YYYYMM**.
  - iii. Maximum tokens: **1**.
  - iv. Minimum tokens: **1**.
  - v. Type: **Normal**

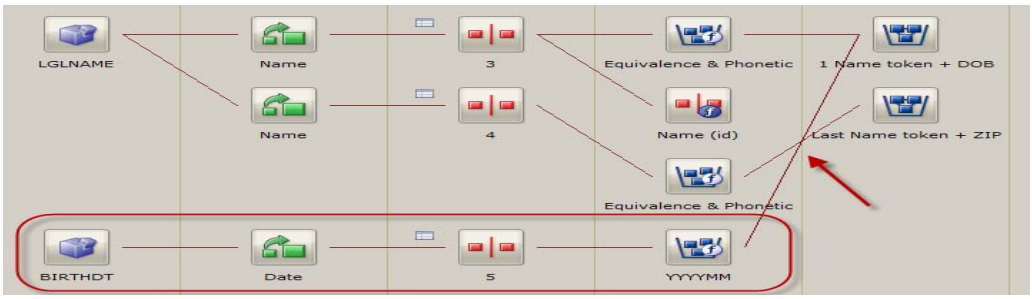
Figure 1-25. Implementation architecture (for course) ZZ7801.0

Notes:

Here is the basic setup of our exercises for our course

- We will have data from our 2 (definitional) source.
- We will be creating a new algorithm for the PME that operates in the MDM Operation Server
- We will use a DB2 database that has been created for us.

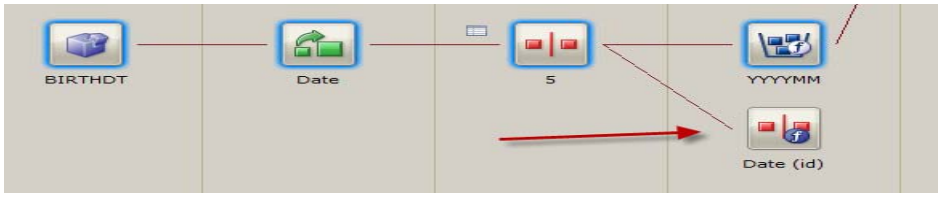
___ e. Bucketing Group: **Connect the Bucketing Function to the 1 Name token + DBO Bucketing Group.**



___ 2. For the Comparison of our Birth Dates, add the following Comparison Function attaching it to the Comparison Role 5

___ a. Comparison specification code: **DOB**.

___ b. Type: **Date**.



The Comparison function will use values specified for the various combinations of birth dates. To generate these values, we will populate the frequency of each date. To generate a frequency for each date regardless if there is only one value we need to modify the minimum frequency value.

___ 3. Click the connection between the **Date** Standardization icon and the **Comparison Role 5**.

Figure 1-26. Attribute definitions ZZ7801.0

Notes:

Attribute definitions

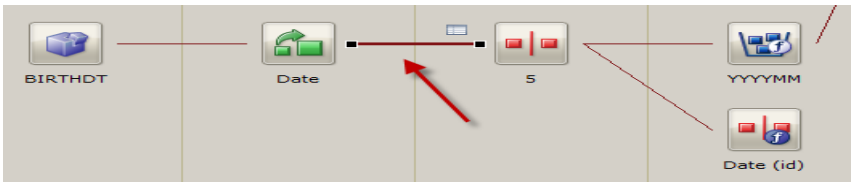
The image that we will work with already has 7 attributes to store the data that is being provided by the (definitional) sources.

Attribute specific definitions also include Number of Active Attributes (Number Active), and Number of Historical attributes (Number Exists).

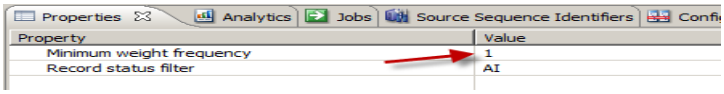
- **Number of Active Attributes (Number Active)** - Usually the most current attribute value is setup as the “Active” value for an attribute. In this case, the field would be set to 1. Should you have a case where you need more than one active value for the same attribute, increase the number active value.
  - e.g. Marital status - At any given point in time, a person should only have one active status, such as Married.
- **Number of Historical attributes (Number Exists)** - Over a period of time, an attribute value may change. The number exists value determines how many historical or previous values along with the current active value(s) should be stored in the database.

- e.g. if number exists is set to 3 and number active is set to 1 for name and Mary Jones gets married to Jonathan Smith, her name entries may look like the following (oldest to most current):
  - Mary Jonas - Status=Inactive
  - Mary Jones - Status=Inactive
  - Mary Smith - Status=Active
- If Mary gets married again, then the very first name or the oldest value would get purged.
- In the example above, Mary Jonas is removed and the resulting entries look like the following:
  - Mary Jones - Status=Inactive
  - Mary Smith - Status=Inactive
  - Mary Johnson -Status=Active

Student Exercises



___ 4. Under the **Properties** window, change the Minimum weight frequency to **1**.



___ 5. Click the **Save** button to save your project.

Part 6: Configure the home address attribute

To configure the home address, we will break it into 2 steps. The first step is to separate the Zip Code (or Postal Code) from the address so that we can use it in the Last Name token + Zip code bucket. The second step will be to create a comparison algorithm that will use the entire address (and phone) for the comparison. We will not create a bucket for the address alone, because even with multiple tokens in the buckets (e.g. New York, St, NY), you could have too many members. If we were storing business addresses however, it might work to create a bucket for a complete match of the address. By using a 2 dimensional comparison with Address and Phone, we can give someone bonus points for have both matches (Phone and Address), and adjust the weights as they get further away on both attributes.

Postal Code (Zip Code for Last Name bucket)

___ 1. Create the following algorithm pieces (and connections) for the **HOMEADDR(ADDR)** attribute with the following values:

- ___ a. Attribute: **HOMEADDR(ADDR)**
- ___ b. Standardization: **Postal Code**
  - i. Field arguments: ***zipcode***
  - ii. *Locale*: **United States**.
  - iii. *Record status filter*: **AI**.
- ___ c. *Comparison Role*

Figure 1-27. Unit summary ZZ7801.0

Notes:

Having completed this unit, you should be able to:

- Describe key concepts of Virtual MDM
- Understand Virtual MDM terminology

Unit 2. Virtual MDM Algorithms

What this unit is about

This unit describes the algorithms used in Matching and Searching in InfoSphere MDM. We will look at the various components of an Algorithm including Standardization, Bucketing and Comparison.

What you should be able to do

After completing this unit, you should be able to:

- Understand the components of a PME algorithm
- Understand the role of standardization and options in an algorithm
- Understand the role of bucketing and options designing our buckets
- Understand the role of the comparison functions and our comparison options
- Create a new Algorithm from scratch

Unit objectives

After completing this unit, you should be able to:

- Understand the components of a PME algorithm
- Understand the role of standardization and options in an algorithm
- Understand the role of bucketing and options designing our buckets
- Understand the role of the comparison functions and our comparison options
- Create a new Algorithm from scratch

Figure 2-1. Unit objectives ZZ7801.0

Notes:

After completing this unit, you should be able to:

- Understand the components of a PME algorithm
- Understand the role of standardization and options in an algorithm
- Understand the role of bucketing and options designing our buckets
- Understand the role of the comparison functions and our comparison options
- Create a new Algorithm from scratch

Thinking about algorithms

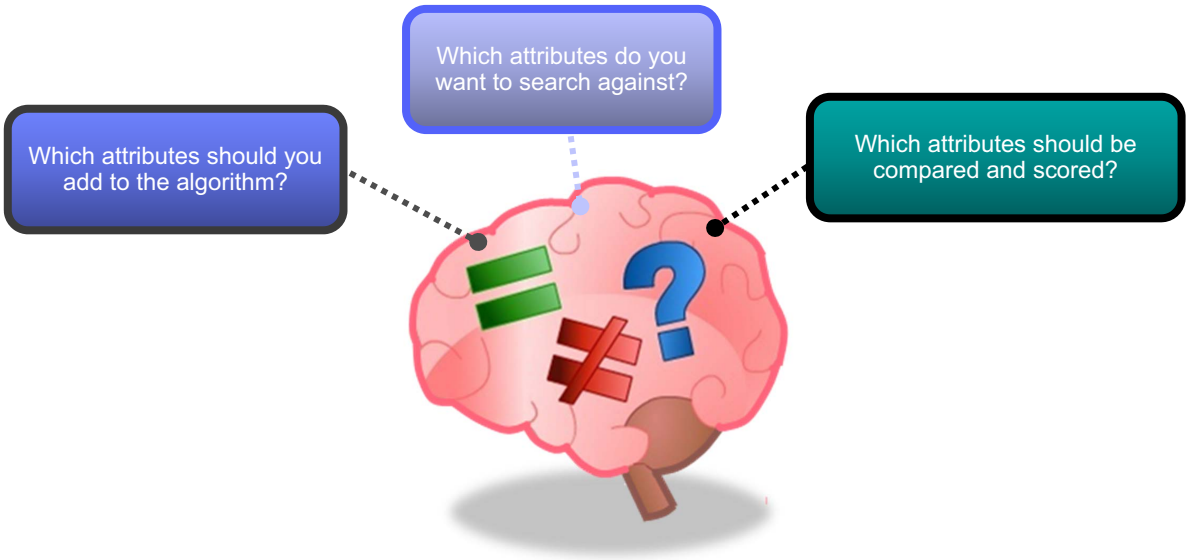


Figure 2-2. Thinking about algorithms ZZ7801.0

Notes:

So let's start off Thinking about our algorithm.

Hopefully you were able to have adequate time during Customer Requirement Gathering to touch on all of the information required for configuring the

Member Model and as well discussing how your end user will be allowed to search for members in the InfoSphere MDM and what attributes you should be using for comparing members together to see if they are the same or not.

Thinking about algorithms: Attributes

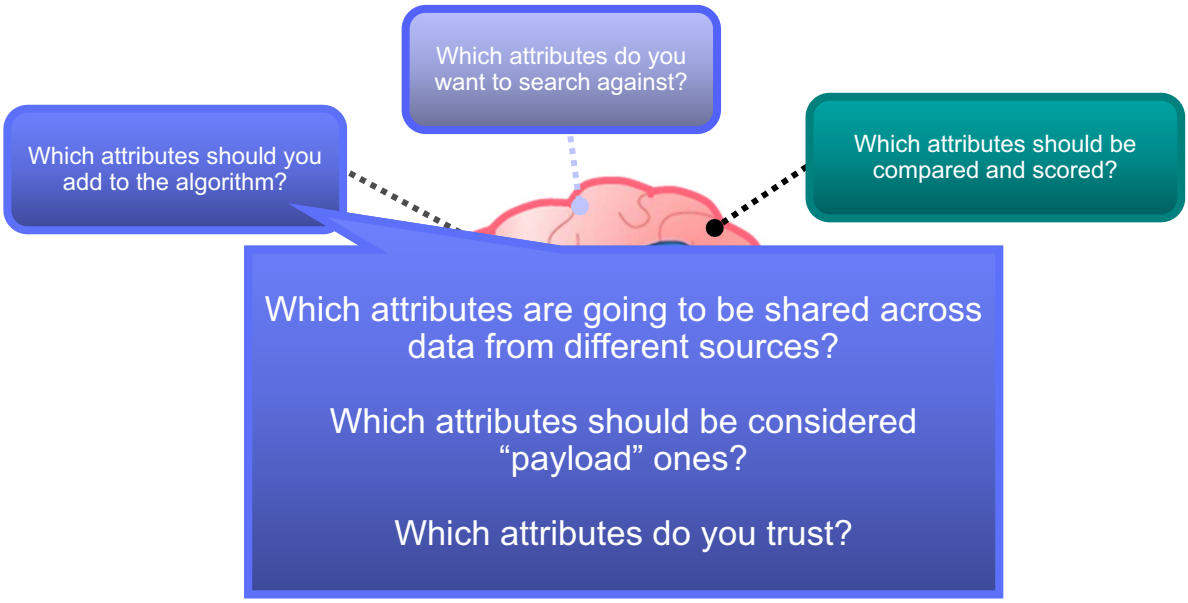


Figure 2-3. Thinking about algorithms: Attributes ZZ7801.0

Notes:

The first thing we need to know is "Which attributes should you add to / use in the algorithm?"

The shared attributes between source systems are the most likely to help provide a match. If a particular attribute is only known to one source it will not be very useful in the matching process, but you may still want to use it in a search process.

A payload attribute is one that is simply "along for the ride". It might be vital to display to an end user, but not relevant to searching or matching. Some examples might be: Maiden name, Last Order Date, Total $ Value Sold, Recent Test Results (Cholesterol Score), etc.

Attributes that you trust more than others can also make a difference in how you design your algorithm. Trustworthy data should be used in the algorithm, while attributes that you question the validity of might be left out of the key matching and linking processes.

Again, knowing what attributes you will be provided by the sources should have been discussed in the Customer Requirements Gathering.

Thinking about algorithms: Search

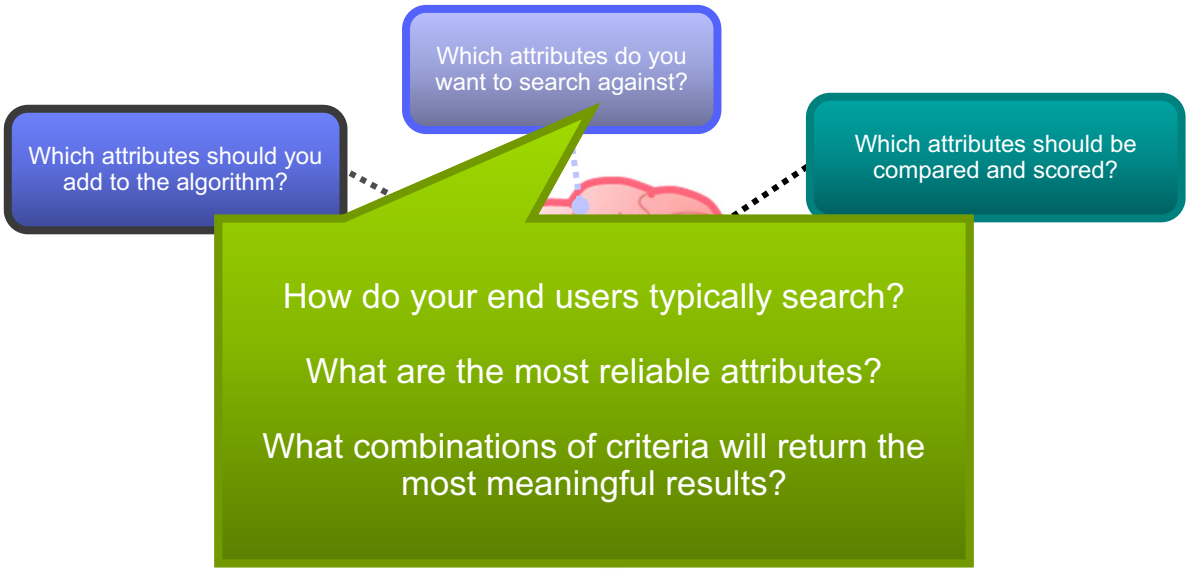


Figure 2-4. Thinking about algorithms: Search ZZ7801.0

Notes:

Next we should be thinking about which attributes will you allow end users to be able to use for searching.

Understanding how a customer looks for their data helps us design bucketing strategies that can mimic current processes or help us educate customers on search paradigm changes.

Reliable data for bucketing is not always about traditional searching. Bucketing selects which records are compared when data is updated or added to the InfoSphere MDM in addition to supporting the traditional search. For example, there are attributes like “Customer Account Number” that a customer might not know when calling the help desk, but multiple records from different sources all share it. So, bucketing on it is still valuable because it will help member records find each other during the matching process.

Sometimes combining criteria will help you get at records quicker. For example, Last Name only might produce too many “Smiths”. First and Last Name together might work for most names, but there could still be hundreds of “James Smiths”. But, if you use First Name, Last Name, and Birth Date the odds are that very few people named James Smith were born on Jan. 3, 1967. These techniques can help you design searches that are fast and meaningful.

Thinking about algorithms: Match

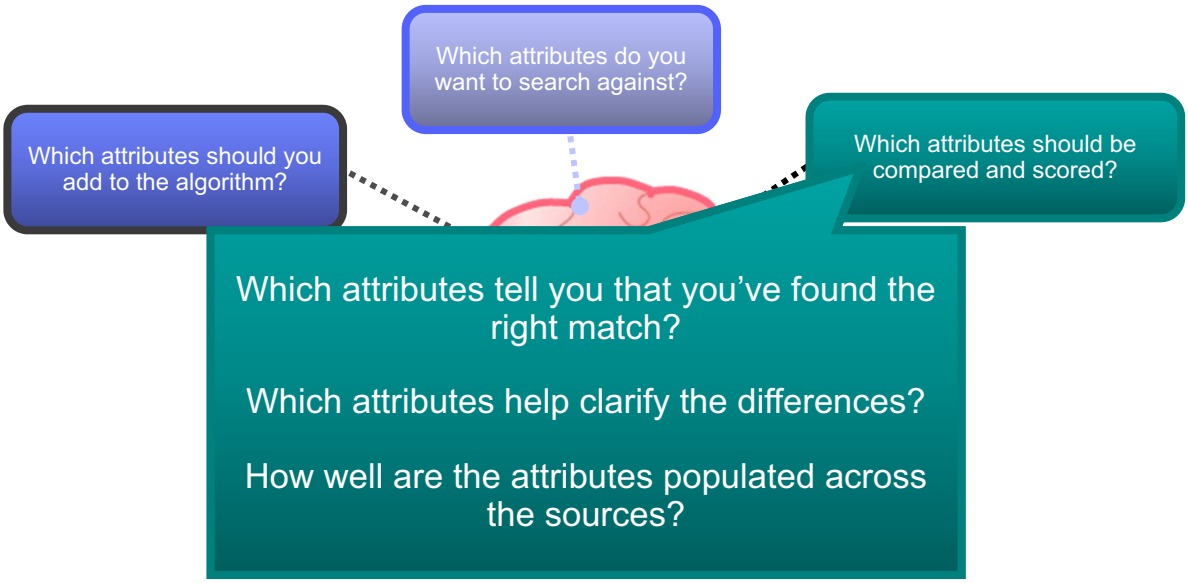


Figure 2-5. Thinking about algorithms: Match ZZ7801.0

Notes:

We should also be thinking about once we have 2 members that have something in common, which attributes should be used for the comparison and scoring of these members.

Some attributes tell you a lot about how well the records being compared match. These attributes tend to have fairly unique values, like SSN, National ID, Drivers License, Email Address, Phone Number, and Account/Policy Numbers. When you encounter these attributes, the deterministic thinking we've been taught over the years kicks in. We tend to think that these key matching fields hold the answer to matching data. But, the probabilistic method takes a more holistic approach by factoring in the overall match for the member records.

Other attributes do a better job of telling you that member records don't match. Attributes like Birth Date and Gender tell us more when they don't match. They help us take people with similar names and make them clearly different. Often the score for matching is not all that high, but the penalty for being different is quite steep.

If you find that an attribute is only known to one source, or if an attribute is poorly populated across sources, you might choose not to include it in your comparisons. Data needs to be common across sources to really get a good measurement of the degree of match between records. Sparsely populated data or attributes that are unique to one system are not going to contribute meaningfully

to the scores. Does that mean that they should not be included in your implementation? Sometimes those attributes are important to a downstream system or process, so they still qualify as Master Data. These attributes might also be used for candidate selection (bucketed) where they help select the records to be compared.

Algorithm configuration

- Algorithm defines the core logic for matching and searching
- There are three steps to algorithm:

Standardization: Re-structures data for consistency (does not alter data displayed)

Bucketing: Organizes records with similar data into an index that speeds searching

Comparison: Defines the method for comparing data to assess the level of match

Figure 2-6. Algorithm configuration ZZ7801.0

Notes:

The algorithm is the “secret sauce” / “black box” of the InfoSphere MDM.

The algorithm does 3 things: Standardization; Bucketing; Comparison.

The first step is to standardize the data.

Standardization transforms similar data received in various formats to a common format.

The algorithm uses standardization routines tailored to meet your configuration needs. The Standardization functions are designed to cleanse a specific attribute’s data before passing that data on to be used to create a searchable bucket and what will be used for comparison. This process includes standardizing all alphabetic characters, setting all alphabetic characters to uppercase, removal of punctuation, anonymous value checks, data ordering, etc..

For example, Source A sends address information for John Public with a street name of “West Easy St,” while Source B sends it as “W. Easy Street.” One of the standardization functions may be configured to take all street names with the value of “West” and

format it as “W” and “Street” to “ST”.

By standardizing data, searchable buckets are easier to find and comparison accuracy is enhanced because the algorithm is looking at the same value for that attribute.

The algorithm takes the standardized data and creates the configured searchable buckets.

Your **bucketing** configuration, a part of your derived data specification, determines which members are candidates for comparison.

Any searchable bucket can have one or more attributes involved. For example, buckets can be defined for name (first, last, middle), birth date + last name, address, and National ID number or other identifiers.

We use an 'OR' condition, instead of an 'AND' condition, during candidate selection (searching) to maximize the number of relevant candidates returned and hence reducing the likelihood of leaving out a genuine candidate. We optimize your bucketing configuration based on your data volumes, profile, and business objectives.

The algorithm takes the standardized data and creates the configured comparison information.

Comparison data consists of all the comparable attributes (or data chunks) used to identify a member grouped into a single delimited string.

We use comparison routines tailored to meet your data needs. Comparison is comprised of a function that indicates which data elements are compared along with additional data validation checks.

During implementation, the project team identifies the attributes and the specific algorithm routines (functions) to employ.

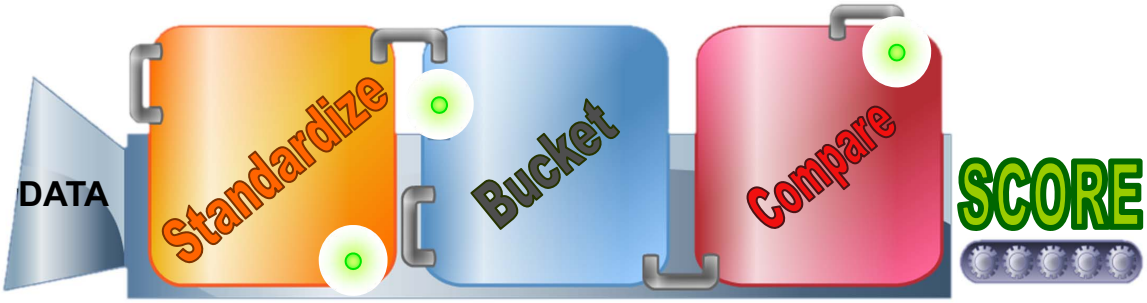
Standardization, Bucketing, and Comparison functions exist for you to use when configuring your algorithm.

You do not have the ability to create new or modify any portion of the algorithm functions.

You can only request to have algorithm functions created or modified.

Algorithms: the secret sauce / black box

The algorithm is the brain behind the searching and matching processes:



Converts data to simplest form for easier use during matching process.

Organizes records that share common values for faster search retrieval.

Compares pairs of records using the probabilistic method to calculate a score.

Jim Smith ➔ SMITH:JIM ➔ JN + SNT ➔ **Jim Smith vs. Jim Smith = 6.1**
➔ **Jim Smith vs. James Smith = 5.9**
➔ **Jim Smith vs. John Smith = 4.2**

Figure 2-7. Algorithms: the secret sauce / black box ZZ7801.0

Notes:

Algorithm: A series of processes (Standardization, Bucketing, and Comparison) used to probabilistically match records that results in a comparison score.

Data: is ingested by InfoSphere MDM and goes through the following steps….

Standardization: converts data into its simplest form for easier use during the searching and comparison processes. In other words, standardization is used to clean up the data by reformatting it into consistent chunks to be used by the algorithm.

Bucketing: organizes records that share common values for faster search retrieval. Think of bucketing this way: is it easier to find a needle in a hay stack or in a jar labeled “needles”?

Comparison: uses the probabilistic method to compare pairs of records and then assign a score based on the similarities and differences between the two records. In other words, the comparison function is a means for finding similarity or difference between two records and their attributes.

The output from the algorithm is the **Score** which is the accumulation of the weights that are determined during comparison.
For Example:

Using the name Jim Smith, the **Standardization** process (depending on the function chosen) simply removes punctuation and capitalizes the name (plus whatever additional the function you choose does) and produces cleansed outputs that would be a colon between the fields.

The **Bucketing** function in this example is using Equivalence & Phonetic, so it converts the nickname/alternate name/equivalent name of Jim to the formal James and then converts James to the key sound markers JN. Smith has no nickname, so it is just converted to the key sound markers SNT. These two are combined and placed in the “JN + SNT” bucket, which requires that both tokens are required for a search.

Comparison will occur if there is more than one record in the “JN + SNT” bucket. Here we see a Jim Smith compared to a Jim Smith. The exact match scores 6.1 points.

Next we see a Jim Smith compared to a James Smith. Jim is a nickname for James, so they score almost as high as if they were an exact match, in this case 5.9 points.

Also in this bucket is John Smith (John is also converted to JN). When John and Jim are compared, they still match phonetically… but the phonetic match is not scored as high as the Nickname or Exact Match. So the score is 4.2, which shows us that John Smith is not very far off.

While we have only reviewed this process for a Name, it is important to note that during the Algorithm’s analysis of the data, it runs parallel analyses of each of the attributes in the records for those attributes that have data available to use.

When are algorithms invoked?

Most key processes in InfoSphere MDM use the algorithm:

- **Search:** Buckets help locate records, compare and sorts the results via API or Inspector
- **Adds/Updates:** Records are derived & compared via Message Brokers, API, or Inspector
- **Bulk Cross Match:** Bulk compare of data using binary files
- **Incremental Cross Match:** Comparison of data where sources are loaded in increments
- **Weight Generation:** Calculation of the numbers used to score the comparisons

Figure 2-8. When are algorithms invoked? ZZ7801.0

Notes:

So just when is the algorithm actually invoked? The easiest answer is ALWAYS.

Every time you enter in criteria to **search** for members, the criteria goes through the algorithm to get cleansed to create the searchable bucket data and also build a comparison string. The cleansed bucket criteria is used to search for members that has something in common.

Whenever you **add** data, we need to create these searchable buckets and comparison strings so the newly added or **updated** data goes through the algorithm.

We can force the algorithm to be invoked if you manually start a process such as the **bulk cross match (BXM)** since this will compare members against other members in bulk.

There is a similar process called the **incremental cross match** which will compare a grouping of members in bulk.

The **weight generation** process will also trigger the algorithm since it is necessary to know what attributes are part of the algorithm and what functions are used on those attributes.

As you can see, the algorithm is invoked just about any time data is being manipulated.

What can standardization do?

- Standardization provides a wide variety of functions:
  - **Case Conversion:** Converts Karen Jones to JONES:KAREN
  - **Truncation of Values:** Converts (312) 832-1212 to 8321212
  - **Anonymous Values:** Removes Phones with (000) 000-0000
  - **Validating Data Length:** Rejects NATLIDs that do not have 9 digits
  - **Validating Data Format:** Rejects e-mail addresses without an @ sign
  - **Nickname/Equivalency Translation:** Converts values like Apartment to APT, West to W

Figure 2-9. What can standardization do? ZZ7801.0

Notes:

The standardization process “cleans” or transforms member data into a format that makes attribute comparison by InfoSphere MDM easier.

Normally, standardization of data means capitalization of all alphabetic characters, removal of punctuation, anonymous value checks, and data ordering and validation. After being standardized, the data is stored as the comparison components of the derived data and used in the generation of the bucketing data. The standardized data is stored in the **mpi_memcmpd** table only. It is never overwritten in the member files, so the original member data is always preserved. For example, a phone number might be entered into a source as 232-123-4567. The standardization routine might strip the dashes and the area code and format the number as 1234567. That number is stored in mpi_memcmpd while the original number stored in the mpi_memphone table remains 232-123-4567.

What does standardized data look like?

- Stored in the **mpi_memcmpd** table:
  - Attributes delimited by carats “^”
  - Tokens within attributes delimited by colons “:”
  - ‘OR’ indicated by tilde “~”
  - Strings must be less than 256 characters in length
  - If there are more than 256 characters, a second line is created

```
PINCHON:ARMAND:N^228073164^M^85022^32537_REPUBULIC^2621259~2022647^BROWN
HAYWOOD:DALE:Y:JR^240251125^M^85217^2835_PERRIER^1090013~1217253^BROWN
SCAGLIONE:STEPHANIE:A^275311207^F^85032^20085_TOULOUSE^2611037~2032074^BLUE
TEW:WENDY:H^242611929^F^85219^1769_MARKET^1998719~1172916^BROWN
GRIMM:JEANICE:I^617634723^F^85217^22960_CANAL^3122086~2172304^GREEN
WEIDLER:MARK:I:III^288915381^M^85007^1848_EXPOSITION^9248645~9007819^BROWN
```


Figure 2-10. What does standardized data look like? ZZ7801.0

Notes:

The derived data comparison string created by the standardization process is stored in the cmpval field in **mpi_memcmpd** and has the following properties:

- Attributes are delimited by carats (^).
- Tokens within the attributes (for example, last name, first name) are delimited by colons (:).
- Periods indicate the final element within the standardized value. Periods are most often seen in values standardized by the PXNM and ADDR2 functions.
- Strings are less than 256 characters in length. If a string is greater than 256 characters, a second line in the mpi_memcmpd table is created.

The slide has several examples of what the comparison string (mpi_memcmpd.cmpval) looks like after data has passed through

Standardization functions.

Additional examples of data standardization

Original Data	Standardized Data
Maria R. Fontana	FONTANA:MARIA:R::.
391-20-1923	391201923
(832) 812-1193	8121193
(832) 811-2915	8112915
mfont91@us.ibm.com	MFONT91USIBM
1973-09-21	19730921

The standardized data is stored in the MEMCMPD table:

FONTANA:MARIA:R::.^391201923^8121193~8112915^MFONT91USIBM^19730921

Figure 2-11. Additional examples of data standardization ZZ7801.0

Notes:

Here are some more data examples of passing though standardization functions.

The first row is a name. Names are converted to upper case, and placed into a colon delimited format, usually Last Name comes first.

The second row is an identifier. The special characters are simply removed.

The third and forth rows are phone numbers. Special characters and spaces are removed, then the number is truncated to the last 7digits. From there Home and Mobile numbers are added together with a '~' tilde to indicate and "OR" condition. The tilde is also used when historical data has been collected.

The fifth row is an email: The alphabetic characters are capitalized, special characters (like the @), and the ".com" is removed. For non-".com" extensions it is based off of the period, not the word 'com.' If there is no @ present in the email, or there is no dot after the @, the attribute is treated as ANON.

The last row is a date. Dates are always stored in the YYYYMMDD format in Standard Edition. We either ask customers to reformat before sending dates, or we will reformat during the insert process (Message Brokers, API – Handlers, or ETL). When standardized, we simply remove any delimiters.

Once the standardization functions have been applied to the data, the comparison string (mpi_memcmpd.cmpval) is created.

Standardization functions

- Address
- Attribute
- Biometric
- Date
- Geographic
- Identifier
- Name
- Phone
- Miscellaneous
- Passthrough

Most functions contain additional variations:

Name: Company, Person, or Provider

Date: Standard, Partial, Fixed, or Age

Address: Canada, International, United States, etc.

Hair Color, Eye Color, Race, Height, etc.

Identifier: Numeric, Alphabetic, or Alphanumeric

Figure 2-12. Standardization functions ZZ7801.0

Notes:

Standardization functions can be grouped by the following categories:

- Address
- Attribute
- Biometric
- Date
- Geographic
- Identifier
- Name
- Phone
- Miscellaneous
- Passthrough (do nothing)

The Workbench Users Guide (http://www-01.ibm.com/support/knowledgecenter/SSWSR9_11.0.0/com.ibm.mdshs.wbuser.doc/topics/r_wbuser_standardization_functions.html?lang=en) goes into much greater detail on each of the individual standardization functions, including variations on the standardization, as you can see on the slide.

Standardization: Are these addresses the same?

Address 1	Address 2	Address 3	Address 4
208 West Chicago Apt. #3 Oak Park, IL 60302	208 W. Chicago Ave. Apartment 3 Oak Park, IL 60302	208 Wst. Chicago, Apt. 3 Oak Park, IL 60302	#3 – 208 W Chicago Avenue Oak Park, IL 60302

The Expanded Address function converts strings to UPPER CASE, isolates each distinct String and Number, and identifies them with an “S-“ or “N-“. These elements are delimited by a colon (:) and common words like Street, Apartment, and North are abbreviated to “ST”, “APT”, and “N”.

Ex: 123 West Main Street ➔ N-123:S-W:S-MAIN:S-ST:S-HOMETOWN:S-USA:N-12345
Hometown, USA 12345

	Standardized Addresses
Address 1	N-208:S-W:S-CHICAGO:S-APT:N-3:S-OAK:S-PARK:S-IL:N-60302
Address 2	N-208:S-W:S-CHICAGO:S-AVE:S-APT:N-3:S-OAK:S-PARK:S-IL:N-60302
Address 3	N-208:S-W:S-CHICAGO:S-APT:N-3:S-OAK:S-PARK:S-IL:N-60302
Address 4	N-3:N-208:S-W:S-CHICAGO:S-AVE:S-OAK:S-PARK:S-IL:N-60302

Figure 2-13. Standardization: Are these addresses the same? ZZ7801.0

Notes:

Let’s take a look at the 4 addresses on this slide and see how the standardization ADDR function (and variation) derive / cleanse the data for the comparison string.

Notice that the pre-configured standardization functions parse the address data in a fashion that attempts to make the data look as consistent as possible so it will be easier to use for comparison.

As with the majority of the standardization functions, the first thing that happens is set all to upper case and then also remove punctuation.

Then the rest depends on which variation of the function you choose.

Common abbreviations are brought into the addresses to make the standardized version of the data more alike, instead of more disparate.

Another thing that the function does is to create individual tokens of each portion of the address data that it can.

By tokenizing the data in the addresses we can assess the match better because there is cleansed data in several different tokens to allow the algorithm to compare each and every possible permutation of the tokens to find the best possible match.

So in the example from the slide, it does not matter that for one address that the apartment number is the first token, last token, or any other token.

The algorithm will look at every combination.

What is bucketing?

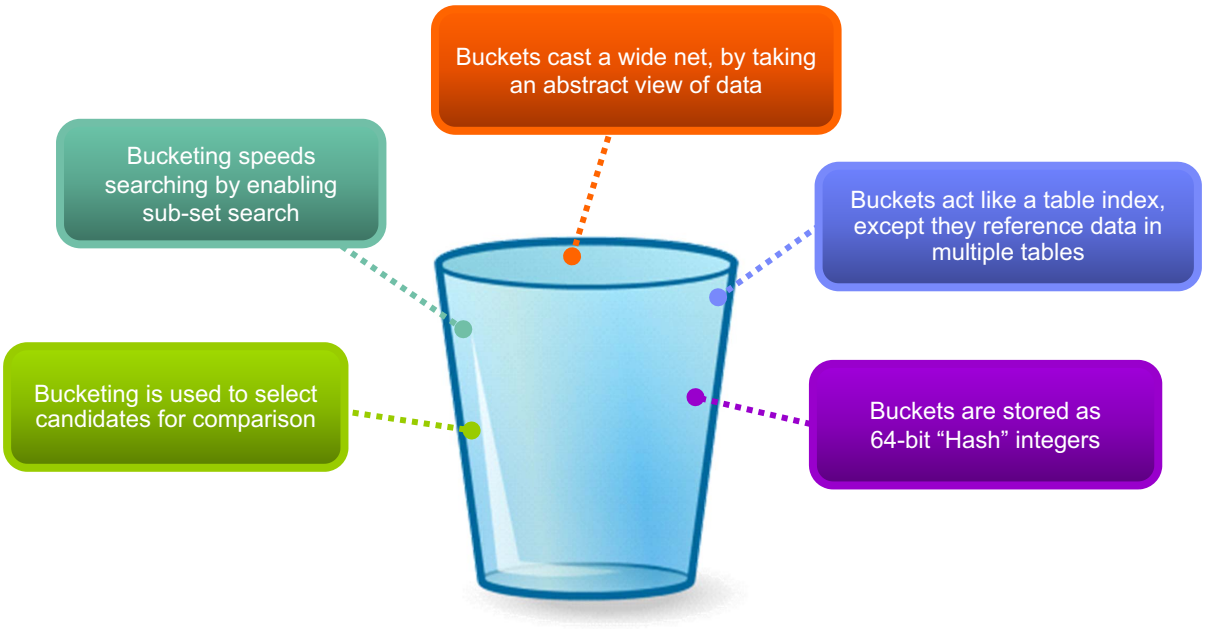


Figure 2-14. What is bucketing? ZZ7801.0

Notes:

Bucketing selects the candidate records that will be compared when you search and when you update a record in InfoSphere MDM. In the case of searching, only the buckets that correspond to the criteria entered are searched. In the case of inserts and updates to the data, the buckets that correspond to each of the values in the record are scanned for possible matches.

Buckets speed up searching because instead of scanning through an entire table, like a traditional SQL query does, the buckets enable sub-set searches. Only the records in the database that have something in common with your search criteria are checked.

Buckets cast a wide search net by looking at data through abstract methods. For example, instead of looking at the exact spelling of a name, the bucket is built phonetically – on key sound markers in the word. This allows for subtle variations in spelling, yet still locates the right records for comparison. Likewise, another technique is called “Sorted” bucketing. It takes numerical digits and re-orders them sequentially. So instead of the bucket containing only people with the ID number 820-84-1882, it has all people that have one 0, one 1, two 2s, a 4 and four 8s (012248888). This abstraction of the actual data accounts for typographical errors, an all too human problem.

The buckets serve a similar purpose to Database Table Indexes. The key difference is that a Table Index only catalogs the data within one individual table, where buckets are one master index to the

data across all of the tables. Instead of searching across multiple tables, the mpi_membktd table acts as a master search index.

Buckets are actually stored in the **mpi_membktd** table as 64-bit hashed integers. That means that the value is converted from a string into an integer that is 19 digits long. This further speeds up the searching time because all hashes are the same length and integers can be scanned in a database faster than strings. The hashes cannot be “un-hashed” but Workbench does contain analytical reports that will display the values that the hashes represent.

Organizing buckets

Bucketing defines how a record is organized, each record may have several buckets

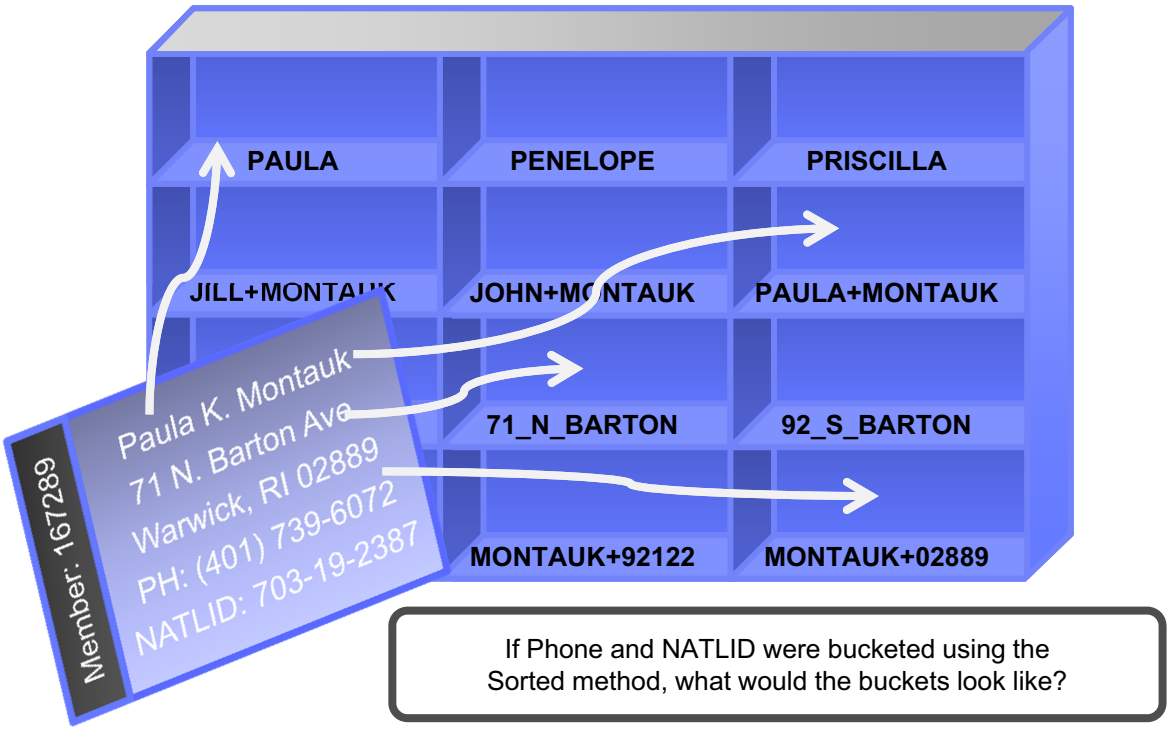


Figure 2-15. Organizing buckets ZZ7801.0

Notes:

Here we see how this record is organized in several ways. The First Name alone is searchable (though this is usually discouraged for performance reasons), the First + Last Names are searchable, the Street Line 1 field is searchable, and the Last Name + Zip Code is searchable. A Phone or NATLID bucket may also exist, but in this example are not shown.

Organizing buckets

Bucketing defines how a record is organized, each record may have several buckets

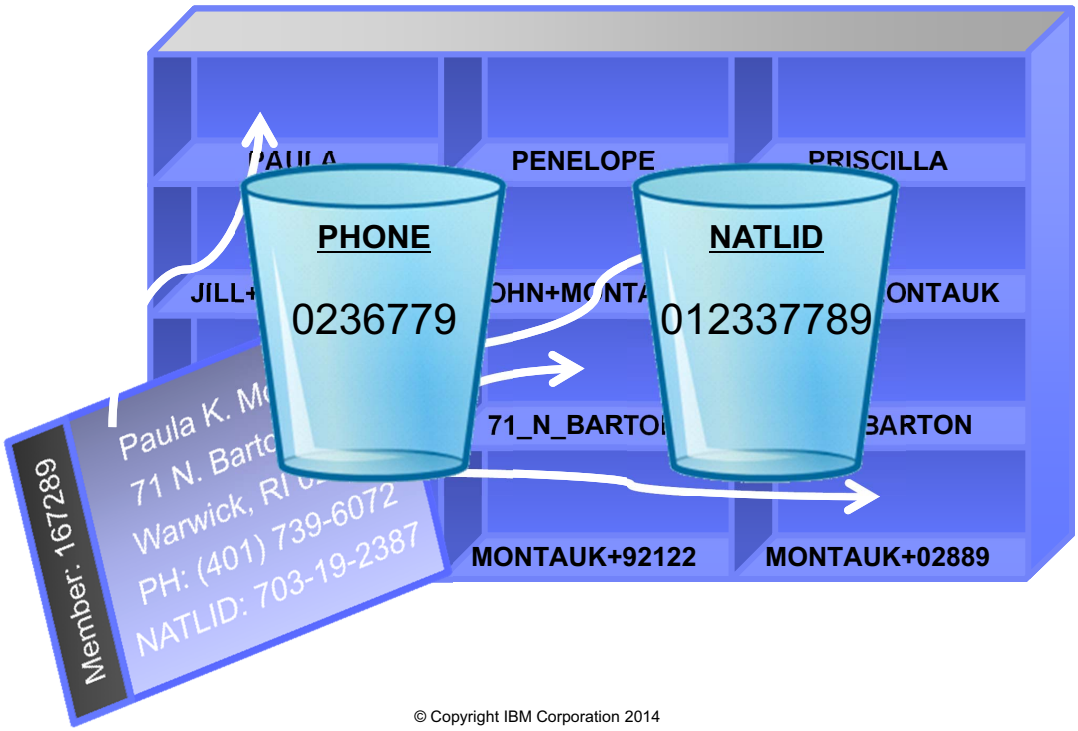


Figure 2-16. Organizing buckets ZZ7801.0

Notes:

If using a “Sorted” bucket design for Phone and NATLID, what would the buckets be?

We would take the bucketing function output and right before creating the hash value, the Bucket Generation Type allows you to manipulate the data that will be hashed.

In the case of the Phone, the value that would be hashed is 0236779.

In the case of the NATLID, the value that would be hashed is 012337789.

Reviewing bucket hash values

NAME: Alec Speegle **NATLID:**780-54-7291 **DOB:**1982-05-02 **ZIP:**85038 **PH:**(312) 271-6291

Bucket Hash	Bucket Value	What Happened
4540265678601088809	012457789	NATLID of 780547291 appears in sorted order as 012457789
-8809730092881002640	1122679	Phone of 2716291 appears in sorted order as 1122679
6085846524090996483	198205+ALK	DOB of May 2, 1982 is added to a phonetic form of Alec
6394223249508127562	198205+ALKSNT	DOB is added to a phonetic form of Alexander (the formal name for Alec)
6312547120776979299	198205+SPKL	DOB is added to phonetic form of Speegle
5256576921179425119	ALK+SPKL	Phonetic form of Alec is added to a phonetic form of Speegle
6590837625867207706	ALKSNT+SPKL	Phonetic form of Alexander (the formal name for Alec) is added to a phonetic form of Speegle
8128495530969973259	85038+SPKL	A sorted Zip Code (85038 > 03588) is added to phonetic form of Speegle

Figure 2-17. Reviewing bucket hash values ZZ7801.0

Notes:

Again, remember that the data we will be using to create the bucket hash is the data we received from the source (or search criteria), passes through the chosen standardization function, passes through the chosen bucketing function, then looks at the bucket generation type to create the hash value that will be stored.

Let's take a look at a few more bucket hash values.

For our NATLID value, we decided that the best way to handle the data and not miss possible data entry issues is to sort the value prior to creating the hash.

Same goes for the phone number.

If we determined that we will allow search criteria to be to use last name and first name, we want to be able to not miss possible spelling variations so we decide to use the phonetic of the value.

Designing multi-token buckets

- Buckets become multi-token when either:
  - Attributes have more than one field (Name, Address, etc.)
  - Multiple attributes sent to same bucket group (like Name + DOB)
- With Multiple tokens you must define:
  - **Maximum Tokens:** Total number of tokens used simultaneously
  - **Minimum Tokens:** Minimum number of tokens required to gen bucket
- System dynamically creates each permutation of tokens
- Max/Min settings exist at Bucket Function and Bucket Group levels

Figure 2-18. Designing multi-token buckets ZZ7801.0

Notes:

When we talk about buckets, we are really talking about the searching.

During Customer Requirements Gathering, an important topic that needs to be covered is how end-users will be allowed to search – what is the acceptable search criteria.

Although you may want to allow end-users to search on any possible single attribute, that may not always be the best search criteria design strategy.

You have the ability to create searchable buckets that are made up of 1 or more fields of one or more attributes.

When you create multi-token buckets, it is that combination of the attribute fields that will be hashed and stored.

Not only can you create multi-token buckets, you also have the ability to configure how many tokens are required from each attribute that you are using to build the bucket. You control this by setting the minimum and maximum tokens properties.

Designing multi-token buckets example

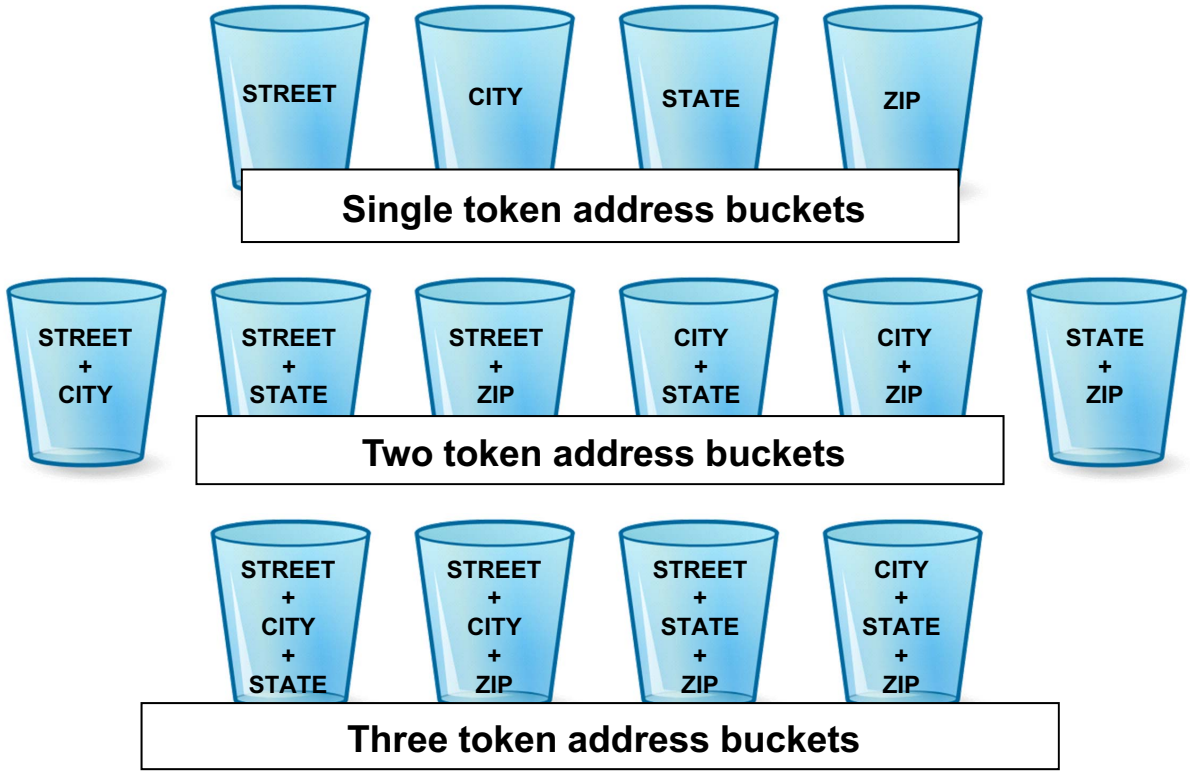


Figure 2-19. Designing multi-token buckets example ZZ7801.0

Notes:

Let's use an address as an example to build multi-token buckets.

If we were to receive as part of member address data the street, city, state, and zip/postal code, we can create single token buckets.

1 for the street field; one for the city field; one for the state field; and one for the zipcode field.

(as long as we receive data for those fields)

Maybe we determined that the each of those fields alone is not enough information to search by.

We decided that we need 2 pieces of information to search by.

We could create 2 token buckets by combining each combination of the data that we receive.

So we would have all of the following possible searchable buckets combinations:

(as long as data supplied)

- Street + city
- Street + state

- Street + zip
- City + state
- City + zip
- State + zip

We would now have 6 searchable buckets from the 4 pieces of information.

There is no limit on the number of attribute fields that you can create a bucket from.

Notice the 4 possible searchable buckets if we were to create 3 token buckets.

Designing multi-token buckets example cont…

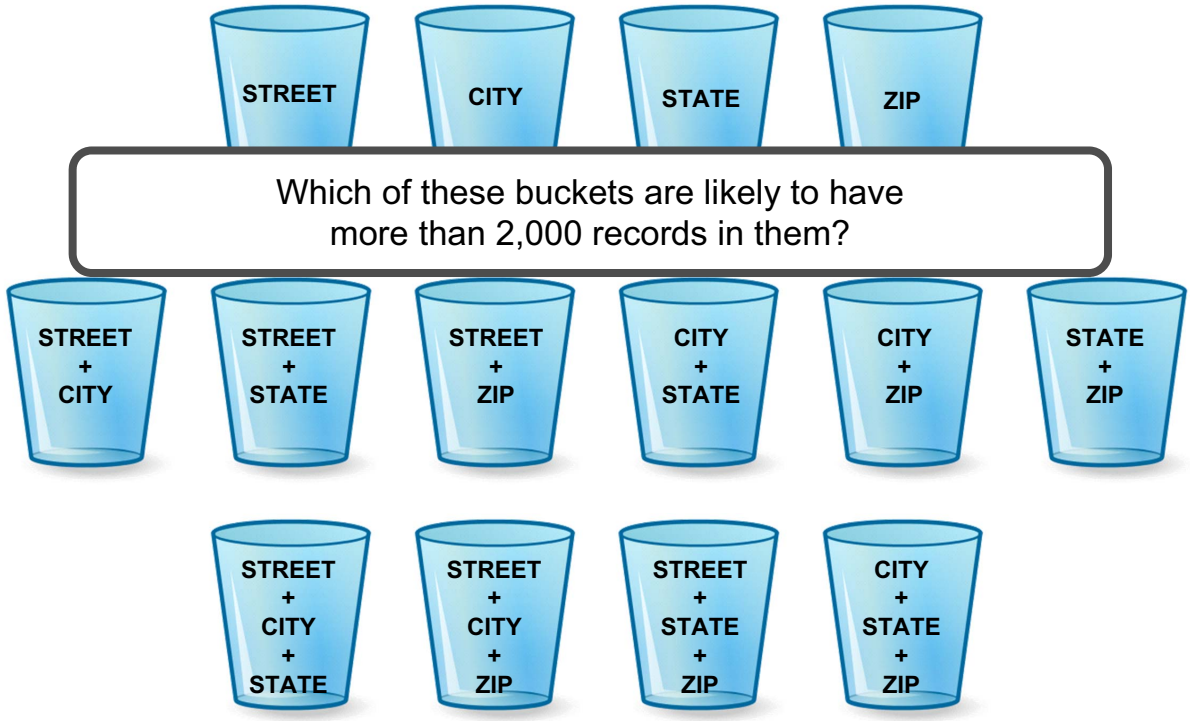


Figure 2-20. Designing multi-token buckets example cont… ZZ7801.0

Notes:

When designing your buckets, one thing to take into consideration is that there is a chance that a searchable bucket could have too many members that have the same value and render that piece of information as unusable.

If you notice that this is becoming an issue, you have the ability to set a property that not use that portion of the search criteria if too many members have the same bucket.

Setting the maximum bucket size will control this for you.

A good rule-of-thumb is to use 2,000 as this maximum bucket size – but this will depend on your customer's environment.

Designing multi-token buckets example cont…

Figure 2-21. Designing multi-token buckets example cont… ZZ7801.0

Notes:

If we were to set the maximum bucket size to 2,000 for our address searchable buckets, which buckets do you think have the potential to be greater that 2,000?

There is a chance that more than 2,000 members in InfoSphere could have the same street address.

There will most likely be more than 2,000 members who live in the same city and state.

Organize these people into buckets

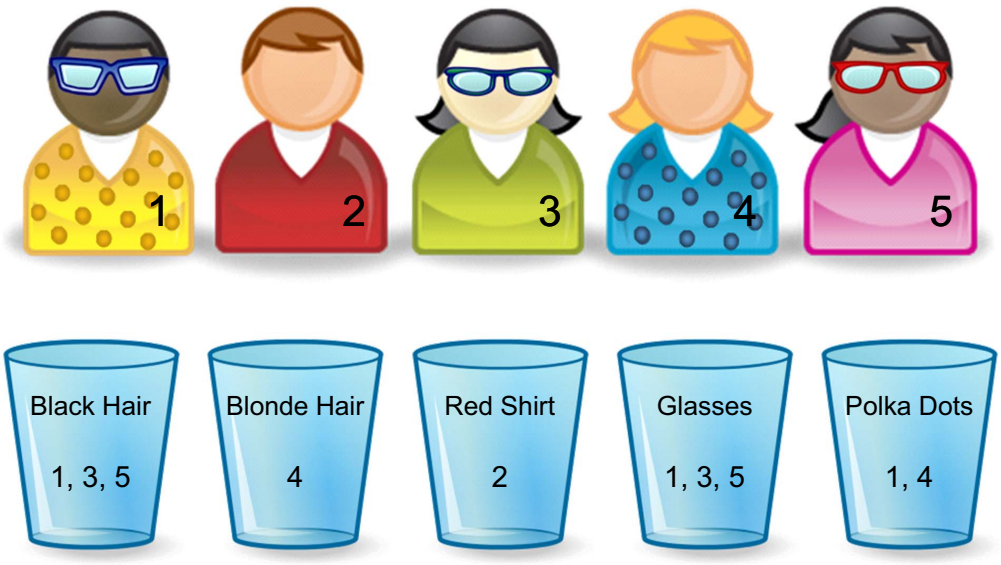


Figure 2-22. Organize these people into buckets ZZ7801.0

Notes:

Let's look at some other examples of buckets.

Which members have black hair?

Which members have Blonde hair?

Which members wear glasses?

As you can see, you have complete control of how you configure searchable buckets.

Available bucket functions

- **Attribute**
  - Used with most attributes
  - Only generates a single token
- **Date**
  - (Normal) Used with most date attributes
  - (Circa) Used when standardization is also Circa (groups into ages)
- **Name**
  - (Person) Up to 6 person name tokens
  - (Company) Up to 6 company name tokens
  - (Provider) Up to 6 provider/small business name tokens
- **Address**
  - Only used with “expanded” or ADDR2 standardized attributes,
  - Provides up to 6 address tokens

Figure 2-23. Available bucket functions ZZ7801.0

Notes:

Here are the available bucketing functions.

Notice that the list is very short.

These 4 functions handle all the data that you will use in your implementation.

Attribute

- Attribute can be used with most attributes, but only generates a single token

Date

- Normal can be used with most date attributes
- Circa can only be used when standardization is also set to Circa (groups into ages)

Name

- Person generates a set of up to 6 person name tokens
- Company generates a set of up to 6 company name tokens
- Provider generates a set of up to 6 provider/small business name tokens

Address

- Address can only be used with “expanded” or ADDR2 standardized attributes, provides up to 6 address tokens

The Workbench Users Guide goes into much greater detail on each of the individual bucketing functions and their variations.

http://www-01.ibm.com/support/knowledgecenter/SSWSR9_11.0.0/com.ibm.mdshs.wbuser.doc/topics/r_wbuser_bucketing_functions.html?lang=en

What is the role of bucket generation types?

bucket generation types

- **Control how buckets are conceived**
  - Format data for optimized searching
  - Make buckets more accepting of discrepancies
  - Leverage the standardized version of the data
- **Why use bucket generation types?**
  - Helps work around typos & misspellings
  - Makes searches less reliant on exact matches
  - Casts wide net, includes nicknames & key sounds
  - Makes searches fuzzier

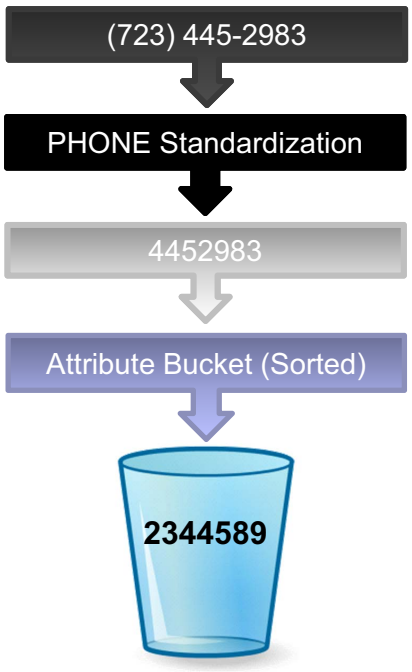


Figure 2-24. What is the role of bucket generation types? ZZ7801.0

Notes:

Once you have chosen the bucketing function to use on your data, you have the ability to control the data even further by selecting a bucket generation type.

Think of bucket generation types as giving you the ability to manipulate the data prior to creating your hash bucket.

Bucket generation types exist to help take into consideration things such as typos and misspellings, sort the numbers to make sure that transpositions can be handled, allow you to take into consideration nicknames/alternative names/equivalent names.

Using this functionality allows you to cast a much larger net when selecting members who could possibly share the same bucket.

This also gives you the ability to do searching taking into consideration “fuzzy” logic.

Think of doing a search in google – how exact do you need to be to find the results you are looking for?

This also helps in eliminating exact matches on your search criteria.

In the example above a phone number is run through the algorithm

- Standardization shaves the phone to 7 digits
- Sorted generation type rearranges the 7 digits
- Bucket holds all phone numbers containing 2-3-44-5-8-9
- Bucketing generation type makes 4452983 comparable to 4452938 (Is 83 vs. 38 a typo?)

Available bucket generation types

Generation Type	What Happens
As Is	Leaves each token “as is.” For example, the zip code 77392 is put into 77392 bucket.
Phonetic	**Metaphone**: Converts tokens to key sound markers. Industry standard phonetic methodology.
	First Three Characters: Outputs first three characters of token. Acts like a starts with wildcard.
	Initiate: Uses Initiate’s proprietary phonetic algorithm, works best with Western languages.
	Arabic Name: Applies phonetics to the English translation of Arabic names.
	Improved Arabic Name: Works with Arabic data (Unicode) to bucket on common homonyms.
Equivalence	Applies a String Equivalency (Nickname or Abbreviation) from lookup table (Jim becomes James)
Equivalence & Phonetic	Blends the phonetic and equivalence conversions against tokens. (ED ➔ EDWARD ➔ ETWRT)
Sorted	Sorts contents of numeric tokens. 3014201324 becomes 0011223344 & 52431 becomes 12345
Numeric Range	Puts numeric data into ranges. Commonly used with height, weight. (Weight: 150-159 lbs ➔ 150)
String Range	Applies a range to a string. Usually used with Full Dates and Years if they are bucketed together.
YYYYMMDD	Buckets records according to a Full Date, so there is no transformation.
YYYYMM	Uses portion of date token to group dates by Month & Year. (19280912 and 19280930 ➔ 192809)
MMDD	Uses MMDD portion of the date token to group dates by anniversary (19280912 becomes 0912)

Figure 2-25. Available bucket generation types ZZ7801.0

Notes:

Here are some examples of the Bucket Generation Types available to you.

A good starting point for bucket generation types is to use Equivalence & Phonetic on names; sorted on values that are numeric.

The Workbench Users Guide goes into much greater detail on each of the individual bucket generation types.

http://www-01.ibm.com/support/knowledgecenter/SSWSR9_11.0.0/com.ibm.mdshs.wbuser.doc/topics/r_wbuser_bucketing_functions.html?lang=en

Comparison methodologies

- Comparisons:
  - **Exact Match:** Do two values match, Yes or No?
  - **Starts With:** Do two values start with same initial or digits?
  - **Edit Distance:** How many edits does it take to make values match?
  - **Phonetics:** Do values have same key sound markers?
  - **Equivalency:** Could one value be nickname or alias?

- Set of functions that blend approaches
  - **Name (comprehensive):** Uses all five methods, best match wins
  - **Address & Phone:** Uses all methods for address, but edit distance for phone
  - **Date:** Approach blends exact match and edit distance

Figure 2-26. Comparison methodologies ZZ7801.0

Notes:

The last piece to the algorithm to discuss is the comparison.

There are several forms of comparisons that are available for you to use.

We can do an exact match (e.g. are the 2 values the same – yes or no)

We can also compare based on looking at how the values start.

Another way to do comparisons is to use edit distance – this looks at the data that is being compared and figure how out how many edits need to take place to make the 2 values the same.

We also have the ability to compare using phonetics – looking at the same key sound markers.

Using equivalency is another option that allows you to look at nicknames / alternate names .

These comparison methodologies are bundled in the functions so you can choose the best approach for your comparisons.

What is the best way to compare?

- Wide variety of choices for comparing data
  - Our data scientists have crafted comparison functions that blend the 5 approaches
  - Comparison functions are generally named according to the type of data analyzed
  - Many functions have variations that can be accessed through the “type’ property

Figure 2-27. What is the best way to compare? ZZ7801.0

Notes:

You may wonder what is the best way to do comparisons.

Well, there is no best way since you have many choices to compare based on which function you choose.

Just as in standardization and bucketing functions, there are also variations to the comparison functions that you can choose by setting properties for the function.

Comparison functions

- There are several comparison function categories:
  - String Pair
  - Address & Phone
  - Name
  - Date
  - Edit Distance
  - Equivalency
  - False Positive Filter
  - Geocode
  - Height & Weight
  - US Zipcode

Figure 2-28. Comparison functions ZZ7801.0

Notes:

Comparison functions can be grouped by the following categories (we will look at a few of these in the next slides):

- String pair
- Address & phone
- Name
- Date
- Edit distance
- Equivalency
- False Positive filter
- Geocode
- Height & weight
- Zip/postal code

The Workbench Users Guide goes into much greater detail on each of the individual comparison functions, including variations available.

http://www-01.ibm.com/support/knowledgecenter/SSWSR9_11.0.0/com.ibm.mdshs.wbuser.doc/topics/r_wbuser_comparison_functions.html?lang=en

Edit distance

- Measures similarity between two strings (# of edits to make strings match)
- Four kinds of edits:
  - **Transposition:** Two adjacent characters are swapped (STEIN vs. STIEN)
  - **Insertion:** Add another character (MARGRET vs. MARGARET)
  - **Deletion:** Remove an extra character (66271 vs. 66271)
  - **Replacement:** Swap a character for another (GORDON vs. GORTON)

Distance	Meaning
0	Exact Match
1	One edit off
2	Two edits off
>12	Rarely calculated beyond 12

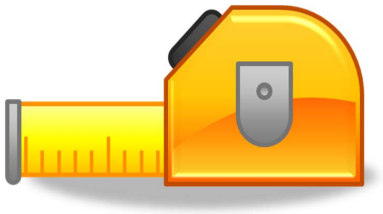


Figure 2-29. Edit distance ZZ7801.0

Notes:

Edit distance functions compare two strings and determine the number of insertions, deletions, replacements, or transpositions it would take to make the two strings the same.

You can have one-dimensional, two-dimensional, three-dimensional, or four-dimensional comparisons. Each dimension corresponds to a wgtdim table (mpi_wgt1dim, mpi_wgt2dim, mpi_wgt3dim, and mpi_wgt4dim).

There is a common set of numbers used in edit distance results:

- A 0 (zero) edit distance means that one string is missing from the comparison input.
- 1 means an edit distance of 0 and both strings match exactly.
- 2 means that there is an edit distance of 1, as in one edit (insertion, deletion, replacement, or transposition) must be made to make the strings match.
- 3 means that two edits must be made for a match, and so on.
- >10 to 12 means that you have a mismatch.

What's the distance?

- Calculate the distance between these strings:

String 1	String 2	Distance
2171_W_KLINE	1217_N_KLINE	
MARY	GARY	
FELICIA	FELIX	
818121463	818211436	
NGUYEN_9617852	NUGYEN_712814	
ROBERT	NORBERT	
839769198362	177172824194	
KLEINFELTER	LINGENFELTER	

Figure 2-30. What's the distance? ZZ7801.0

Notes:

Let's take a look at some example comparisons and apply an edit distance to them.

Take a look at the first line – What do you think the edit distance is?

What about the edit distance for the second line?

How about the third line?

How about the forth line?

Now jump down a few lines to the second to last line.

What do you think the edit distance would be for this comparison?

Ok – stop counting.

If you haven't been able to figure out what the edit distance is by now, we should look at the way we are doing edit distance calculation.

What's the distance? Cont…

- Calculate the distance between these strings:

String 1	String 2	Distance
2171_W_KLINE	1217_N_KLINE	3
MARY	GARY	1
FELICIA	FELIX	3
818121463	818211436	2
NGUYEN_9617852	NUGYEN_712814	7
ROBERT	NORBERT	2
839769198362	177172824194	12
KLEINFELTER	LINGENFELTER	5

Figure 2-31. What's the distance? Cont… ZZ7801.0

Notes:

Here is the actual edit distance for each of the above.

When you start looking at your data and how you are comparing, you need to determine if you really want top use edit distance in the first place and if you do, you need to make sure that you are choosing the right options for your edit distance.

Choosing edit distance functions

- Special properties:
  - **Role:** # of standardized attributes connected to edit distance
  - **Dimension:** # of fields considered from each attribute (usually 1)
  - **Type:** Level of detail that distance calculator focuses on
    - **Simple:** Assess exact match or different
    - **Quick:** Approximates the edit distance for faster analysis
    - **Real:** Calculates true edit distance, uses more computation
  - **Include Source:** Concatenates 'Issuer' data to value at time of compare

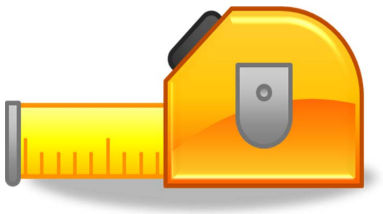


Figure 2-32. Choosing edit distance functions ZZ7801.0

Notes:

Most of the time, you can deduce the edit distance functionality by understanding how they are named.

Names are formatted as DR<**r**>D<**d**><**t**> where:

- **D** means that this is an edit distance function.
- **R<r>** means edit Distance Role where the <**r**> indicates the number of roles (or standardized attributes) that are being compared. This is typically a number between 1 and 4.
  - If you are comparing phone number, you would have one role.
  - If you are comparing ZIP + address + phone, you would have three roles.
- **D<d>** represents Dimensions with <**d**> being the number of dimensions (or tokens/fields that make up an attribute) that are being compared in each role.
- <**t**> is a letter representing the type of comparison being performed. Options are:
  - A = simple edit distance comparison resulting in match or no match, so each dimension of the weight table has exactly two entries.

- B = quick edit distance returns an integer representing the edit distance. The returned edit distance can sometimes be higher than the true edit distance, but it is much more efficient. This comparison is suggested for long strings.
- C = a real or true edit distance. This is the most accurate comparison, but can be very system resource intensive. This comparison is suggested for short strings or in instances where absolute accuracy is vital.

If you are doing an edit distance function against an attribute you defined as attribute type “Identifier”, you have the ability to include the information source provider along with the identifier itself.

An example of this could be:

Do an edit distance compare with “12345” against “12345”.

What do you think the edit distance will be?

If you said edit distance of 0 (zero) – you would be right.

But what if I told you that these are drivers license numbers - AND I want to include the issuing agency of the drivers license number along with the actual value?

Then the edit distance compare would be “MN:12345” against “AZ:12345”.

What would you think the edit distance is now?

Since we are including the source along with the attribute value, these two values look different.

It is your choice to include the source or not, but this will be determined by what works for this customer for this environment.

You also have the ability to use the source.

Since this appears to be an easy comparison function to understand, many try to use the edit distance functionality all the time. If you have performance concerns, you might want to avoid using an edit distance calculation for too many attributes in your algorithm.

To compare or not to compare?

- When searching, can choose which functions yield scores:
  - Comparison Mode sets behavior
    - **Search only:** only applies to search results, not to match compare
    - **Match and Link only:** only applies to match compare, not to search
    - **Search, Match, and Link:** uses the same scoring for all processes
  - **Search only** is good for source-specific attributes that are not shared across sources
  - **Match and Link only** most often used for False Positive Filter to ensure that searches return a wide variety of results for the user to pick from
  - **Search, Match, and Link** is the default and works for most processes

Figure 2-33. To compare or not to compare? ZZ7801.0

Notes:

There is some additional comparison functionality / configuration that you should be aware of to help you control when you should and should not use the comparison score.

The Comparison mode property enables you to set the behavior of when to apply a score.

The Algorithm usage types are:

- use for searching, matching and linking (default). This uses the comparison score when searching and for matching and linking. This is the default setting that is used the majority of the time.
- use for matching and linking only. Only apply the comparison score during matching and linking, but not during searching.
- use for searching only. Only apply the comparison score during searching, but not during matching and linking.

If you know that you will have certain sources providing some information, you may want to set this to “Search only”.

The “Match and Link only” setting comes in handy when you are configuring your Algorithm to take advantage of the False Positive Filter (FPF).

This functionality allows your searches to return a wider variety (cast a larger net) to find potential candidate members to compare against, then using the score during the matching and linking to verify when that should happen.

False positive filters are used to filter information, rather than function as normal comparison functions.

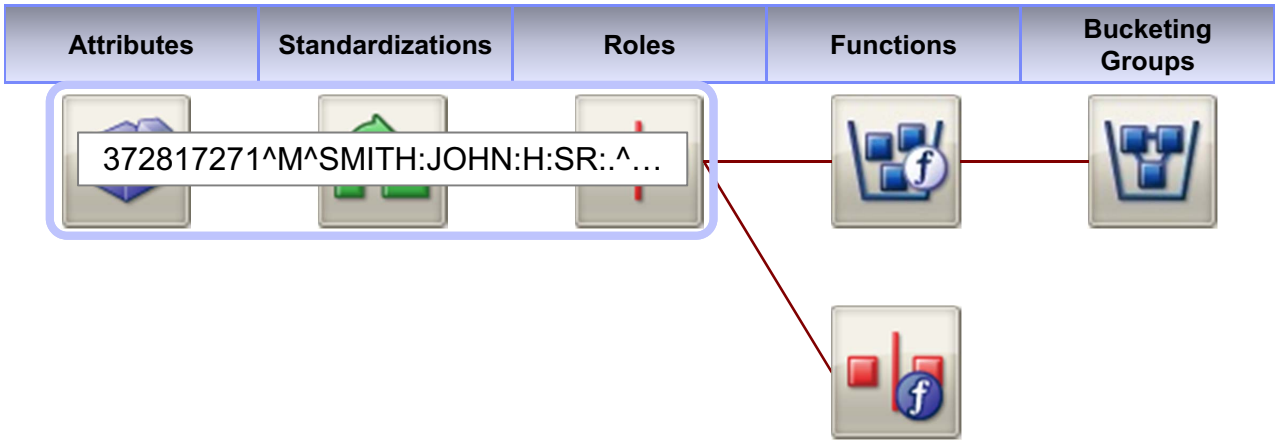
False positive filters are used to ascertain whether the records do indeed represent the same person rather than members belonging to the same family.

There are 2 versions to the FPF.

- **FPF** - comparison function uses the member Name, Date of birth, and Gender for the comparison. This catches situations where a multi-birth happens and not all information is available for each of the births.
- **FPF2** - comparison function uses the member Name, birth year differences, birth date edit distance, gender, along with the included ID edit distance comparison. This catches situations involving father and son that have same last name, live at same address, one having DOB of 10/01/1960 and the other having DOB of 10/01/1980. And since the members being compared are not a birth, hopefully you will have

Generally, FPF2 is a better option than FPF since it provides one more piece of information to compare on.

Algorithm Flow - Standardization



Standardization:

Data is pulled from the attribute, formatted for more efficient comparison, and stored in the MEMCMPD table according to the role number.

Figure 2-34. Algorithm Flow - Standardization ZZ7801.0

Notes:

Reading an algorithm diagram can be challenging, but each algorithm uses the same basic flow.

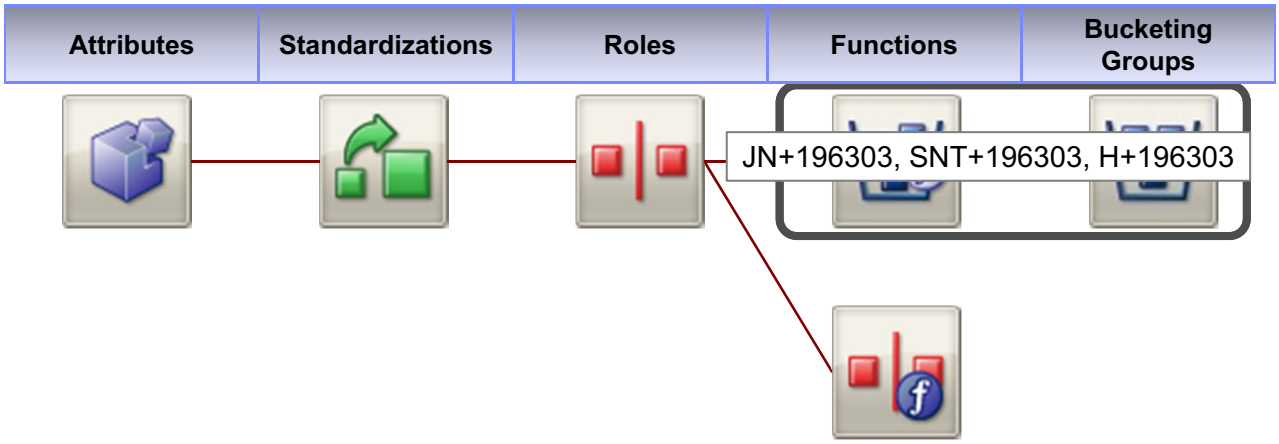
Knowing that the algorithm does 3 things – Standardization, Bucketing, Comparison – every piece of data that passes through the system is touched by the algorithm.

So first step is to take the source data that was passed in and cleanse it based on the Standardization function you choose.

Remember, Standardization simply cleanses and formats the source data for more efficient use by the engine.

This standardized data is passed on to be used by bucketing and comparison.

Algorithm Flow - Bucketing



Bucketing

Bucket function pulls data from MEMCMPD table, transforms it according to the bucketing function design (As Is, Sorted, YYYYMM, Equivalence & Phonetic), then sends data to the bucketing group, which creates actual buckets and stores them in MEMBKTD table.

Figure 2-35. Algorithm Flow - Bucketing ZZ7801.0

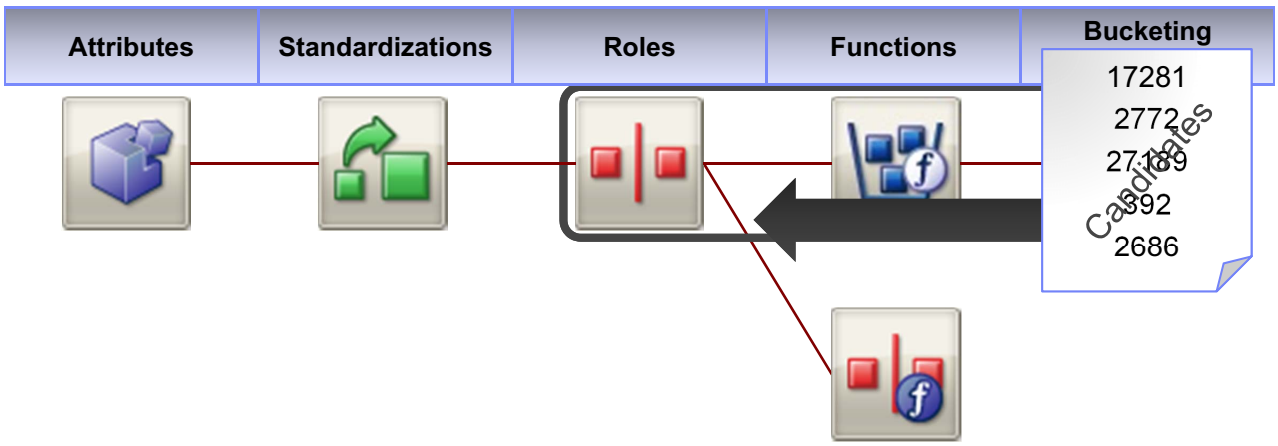
Notes:

Bucketing takes the standardized data as input and depending on the Bucket Generation Type that you chose, create the searchable 64-bit hash values.

These bucket hash values are stored in the mpi_membktd table.

Every member should (hopefully) have at least one row in the bucketing table (if not, that record cannot be searched)

Algorithm Flow - Comparison



Comparison:

Comparison function queries through MEMBKTD to locate a list of candidates to consider for matching.

Figure 2-36. Algorithm Flow - Comparison ZZ7801.0

Notes:

Comparison takes the standardized data as input and uses it to build the comparison string.

These comparison strings are stored in the mpi_memcmpd table.

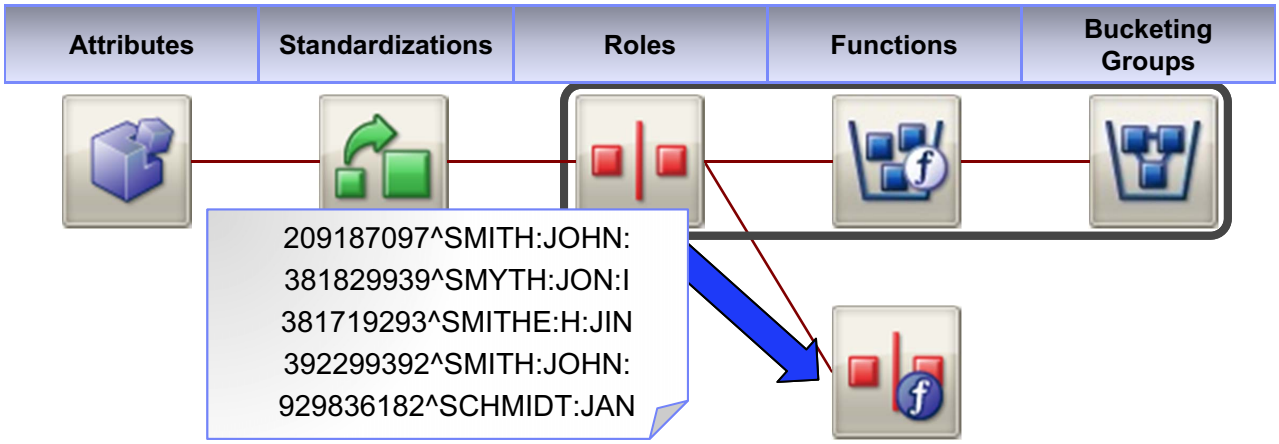
Every member should (hopefully) have at least one attribute that exists to help build the comparison string that they can be compared against others.

So what happens during a search is that the search criteria entered by the user goes through the algorithm.

The data passed in goes through a Standardization function to cleanse the source data and build the hash values that will be used to search for members who have at least one searchable bucket hash in common. This “candidate selection” step builds a list of member record numbers (the internal primary key for the hub is named “memrecno”) that will be passed on to the comparison step.

There has to be at least one common searchable bucket between members (or a member and the search criteria) before they are compared.

Algorithm Flow – Comparison cont…



Comparison

The MEMCMPD entries for those candidates are compared and a score is issued for each record. The scores are sent to the entity manager to decided the next step or they are sent to the search results to be displayed in descending order.

Figure 2-37. Algorithm Flow – Comparison cont… ZZ7801.0

Notes:

The algorithm builds a comparison string from the standardized the search criteria.

Once we have the candidate selection list of memrecnos from the searchable buckets, the comparison strings are retrieved from the database for all the memrecnos from candidate selection.

The comparison string of the search criteria is then used to compared against each of the retrieved comparison strings.

Algorithm Flow - Scores

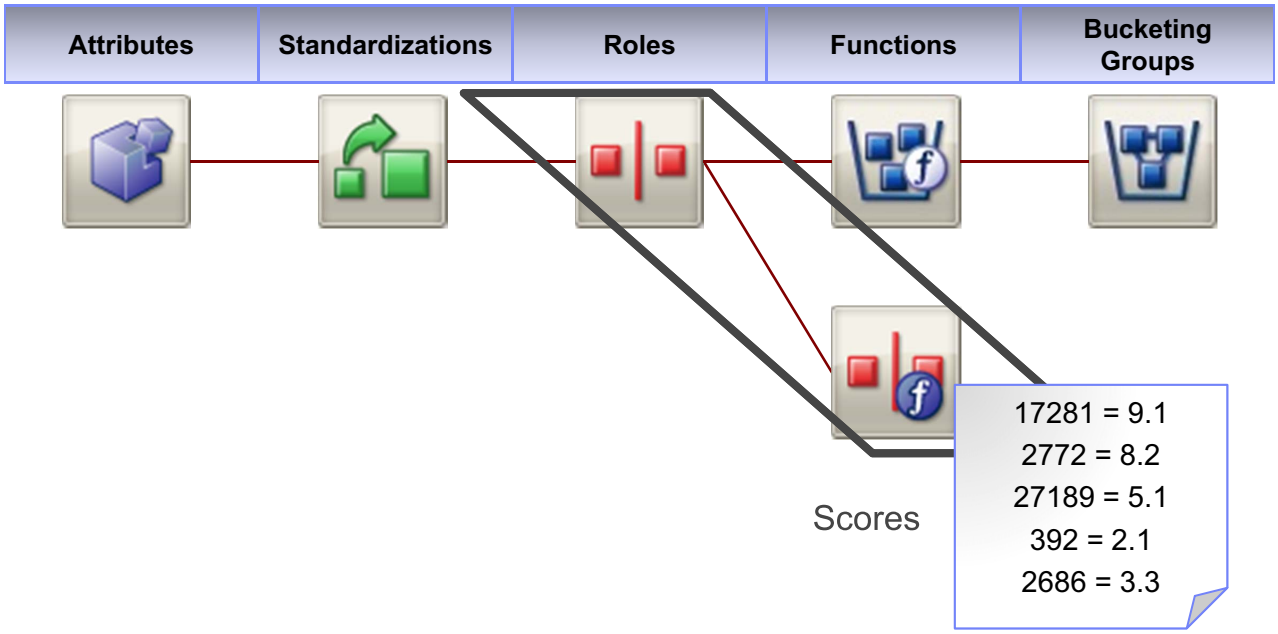


Figure 2-38. Algorithm Flow - Scores ZZ7801.0

Notes:

The weights are accumulated to create the comparison score and then passed on to the entity management process to decide if these 2 members should be autolinked, put into their own entity, or if neither of those then create a task that will need to be worked by a data steward.

Working with the algorithm editor

- To add a component to the algorithm, you click to pick up, then click to put down

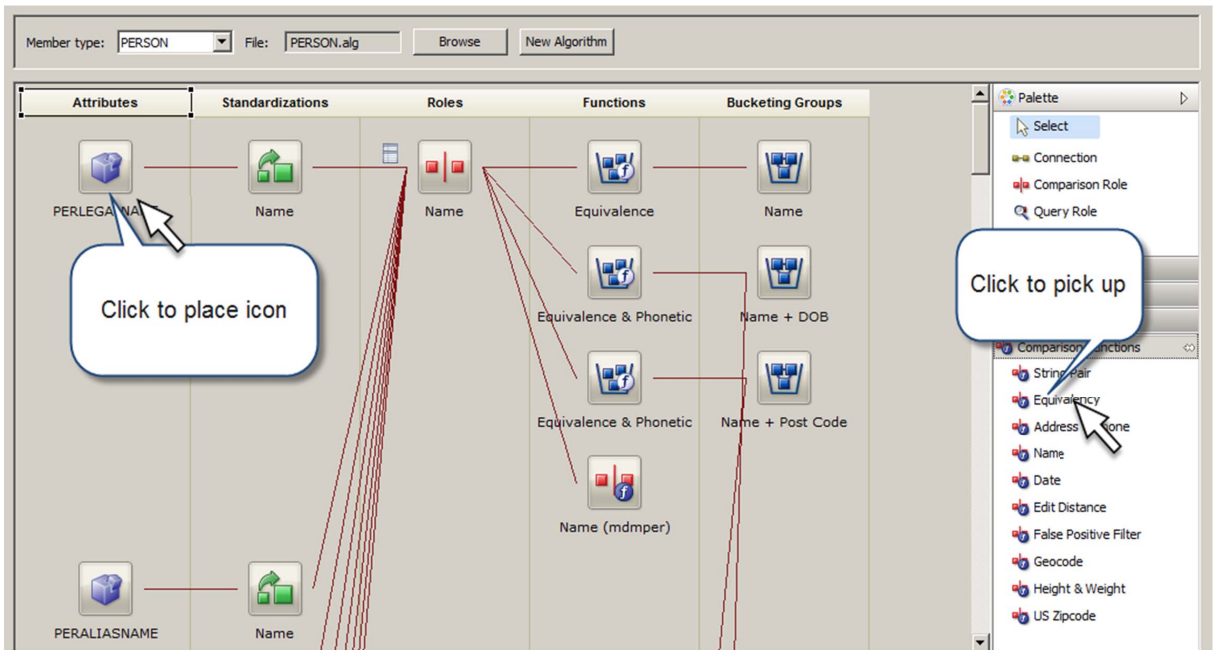


Figure 2-39. Working with the algorithm editor ZZ7801.0

Notes:

Just a few comments on using the algorithm editor.

The columns are called swim lanes.

These swim lanes serve specific purposes.

The Attributes lane is where you will drop the attributes that you want to use in your algorithm.

You can only place an attribute on the palette once , but can use it as much as needed.

The second swim lane is where you will place the standardization function that you want to use on this attribute.

The third swim lane is where you place the what role in the comparison string this standardized data wil llive.

A better way to describe it is “in what position of the comparison string does this standardized data live?”

The forth swim lane is where you place your bucketing functions and your comparison functions.

The fifth and last swim lane is where you define your bucketing group.

This is the where you define the searchable buckets.

When you want top add an icon to the palette on the algorithm editor, all you need to do is first click the icon you want, then move to where you want to drop the icon and click again.

You do not need to click and hold to drag and drop.

Also, to make sure that you see all of the relevant information, make sure that you connect the icons so workbench can check the configuration and inform you of any errors, warnings, or other information.

Algorithm add-ons

Anonymous values

- placeholders or simply fake values used to bypass a required input
  - **Anonymous:** BABYBOY, BABYGIRL, UNKNOWN, (999)999-9999, 01/01/1901, etc.
  - These values are treated as “NULL” by the algorithm logic to avoid bad matching

Equivalent strings

- Synonymous values used map common nicknames and standard abbreviations
  - **Nicknames:** Ed ➔ Edward, Edgar, and Edwin
  - **Abbreviations:** West ➔ W, Apartment ➔ APT, or Pediatrician ➔ PED

Phonetic conversion

- Converts strings to key sound markers or truncates to handle spelling/typo errors
  - Choose from a variety of options, Initiate, Metaphone, Arabic, or First 3 Characters
  - **Initiate:** Balasubramanian and Balusubrimaniam ➔ BLSBRN

Figure 2-40. Algorithm add-ons ZZ7801.0

Notes:

There are some additional functionality that you can incorporate into your algorithm by setting certain properties.

One is to eliminate certain values that you would consider as “dummy” data.

We call these **Anonymous** values.

You can create lists of Anonymous values and tie them to each specific attribute that you are using on your algorithm.

You can also create lists of **Equivalent** strings.

These can also be called nicknames or alternative names / values.

These also can include data abbreviations.

You also have the ability to configure your algorithm to use **Phonetic** conversions.

This helps in bringing the data together on the bucket searching and comparison strings to use key sound markers when potential spelling is different.

The Virtual Module includes nickname translations that we developed over time based on our algorithm and matching experience.

Some implementation data sets may require additional nickname translations. Your installation uses our standard person nicknames.

Take a look at the examples of each on the slide.

Exercise Algorithm synopsis

7 attributes:
NATLID, SEX, LGLNAME, BIRTHDT, HOMEADDR, HOMEPHON, & MOBILEPHON

9 standardizations:
Name & Address have 2

8 comparison roles (MEMCMPD):
Phone numbers are collapsed

6 bucketing functions:
SEX does not have a bucket

4 bucketing groups (MEMBKTD):
NATLID, 1 Name + DOB, Last + Zip, & Phone

5 comparisons:
NATLID, SEX, NAME, DOB, AXP

Figure 2-41. Exercise Algorithm synopsis ZZ7801.0

Notes:

This slide shows you the Algorithm synopsis.

We will be configuring our algorithms using 7 attributes.

From those attributes, we will be doing the following:

1. standardizing 9 different ways;
2. Have 8 different positions (comparison role) in the comparison string;
3. Use 6 different bucketing functions to create 4 actual searchable buckets;
4. Compare 5 different ways.

Exercise introduction

- In this exercise, you will:
  - Create a new algorithm for the PME using the following attributes:
    - National ID
    - Gender
    - Legal Name
    - Birth Date
    - Home Address
    - Home Phone
    - Mobile Phone

Exercise: Creating a new Algorithm

Figure 2-42. Exercise introduction ZZ7801.0

Notes:

In this exercise, you will:

- Create a new algorithm for the PME using the following attributes:
  - National ID
  - Gender
  - Legal Name
  - Birth Date
  - Home Address
  - Home Phone
  - Mobile Phone

Exercise: Creating a new Algorithm

Unit summary

Having completed this unit, you should be able to:

- Understand the components of a PME algorithm
- Understand the role of standardization and options in an algorithm
- Understand the role of bucketing and options designing our buckets
- Understand the role of the comparison functions and our comparison options
- Create a new Algorithm from scratch

Figure 2-43. Unit summary ZZ7801.0

Notes:

Having completed this unit, you should be able to:

- Understand the components of a PME algorithm
- Understand the role of standardization and options in an algorithm
- Understand the role of bucketing and options designing our buckets
- Understand the role of the comparison functions and our comparison options
- Create a new Algorithm from scratch

Unit 3. Virtual PME Data Model

What this unit is about

This unit describes the data model using the Virtual MDM for the PME operations, including the where the algorithm and derived data is stored.

What you should be able to do

After completing this unit, you should be able to:

- Review basic Member model
- Understand the Algorithm Data Model
- Understand the Member Derived Data tables
- Understand the Bucketing data model
- Deploy the PME Configuration

Unit objectives

After completing this unit, you should be able to:

- Review basic Member model
- Understand the Algorithm Data Model
- Understand the Member Derived Data tables
- Understand the Bucketing data model
- Deploy the PME Configuration

Figure 3-1. Unit objectives ZZ7801.0

Notes:

After completing this unit, you should be able to:

- Review basic Member model
- Understand the Algorithm Data Model
- Understand the Member Derived Data tables
- Understand the Bucketing data model
- Deploy the PME Configuration

Member Model vs. Data Model

- **Data Model**
  - Physical database tables and fields:
  - Data model is highly configurable, can add new tables
- **Member Model (IMM)**
  - Metadata layer
  - Establishes which data elements are used in a Member Record
  - Defines how Attributes are stored
  - Defines Entity and Relationship types

Figure 3-2. Member Model vs. Data Model ZZ7801.0

Notes:

The Virtual data model is the physical way the InfoSphere MDM stores data. The Member model (.imm) is a metadata layer that classifies and organizes the components used to track and match member data elements.

The database architecture has been designed to anticipate varying needs by offering an extensible table structure with configurable data modeling. Since each project brings a unique set of data, containing different fields as well as different processing needs, the database has been designed with inherent flexibility.

Although the data model is highly flexible, it also recognize that there are similarities between many data sets. Therefore, the InfoSphere MDM comes with a set of core tables that can hold and process a large variety of key identifying data.

The key to the Virtual Modules model is its database design. The database design allows for the ability to create complex relationships and search requests that cannot be satisfied under normal SQL relational terms.

To these ends, the database has been carefully designed to have:

- Simple relationships

- Highly efficient, proprietary indexing strategies
- Traces of member and entity access and alteration
- A highly configurable entity algorithm scheme
- Compound attribute types that match common, real-world entities
- Highly flexible and configurable workflow types and statuses

Important table naming conventions

Virtual tables

- **mpi_:** Every Virtual module table starts with mpi_
  - mpi_memname: stores names of member records
- **head:** top level tables that assign internal keys
  - mpi_memhead stores the member record number (memrecno) for each record
- **x:** x in the middle of table name means ‘cross’ or ‘by’
  - mpi_srcxsrc stores the thresholds that data between sources are linked with
- **eia:** Enterprise ID Assignment or “linkage”
  - mpi_entxeia_id holds the linkage history for identity entities
- **_id:** Entity tables have suffix indicating the type of entity, in class we will see _id
  - mpi_entlink_id holds the data about which records currently belong to which entities

Figure 3-3. Important table naming conventions ZZ7801.0

Notes:

There are several common naming conventions used in the InfoSphere MDM Virtual module database’s table structure. Below are some of the more common items you will run across in the data model:

- **mpi** – The mpi prefix stands for master person index.
- **head** – If the table name has *head* in it, then it is a core table which will map and control other tables with a similar prefix (for example, the mpi_memhead table controls the other mpi_mem**** tables).
- **x** – When you see an *x* in the name of your table, that usually means *cross* or *by* (for example, mpi_srcxsrc hold source by source thresholds).
- **eia** – The acronym *eia* stands for Enterprise Id Assignment and typically means that the table helps maintain relationships between members and the entities that they are a part of.
- **_id** – The suffix on your entity tables (the ones that begin with *mpi_ent*) indicates which type of entity the table is referencing. You can have more than one type of entity, like Identity and Householding.

Other common naming conventions

Name	Meaning
anon	Anonymous values are treated as NULL values
attr	Attributes are specific demographics
aud	Audit tables track user and record level activity
bkt	Buckets organize data for searching
cmp	Comparisons check for similarity
dvd	Derivation definition tables hold the actual algorithm settings
ent	Entities are distinct individuals or organizations
equi	Equivalent strings that can be interchangeable (like, nicknames)
func	Functions are components used in the algorithm
mem	Members are individual records from a source
rel	Relationships are links between entities, not records
seg	Segments are tables that contain similar data
src	Sources are systems that contribute data to the Hub
std	Standardization changes data to consistent formats
str	Text strings are reference lists used by the algorithm
tsk	Tasks are items that require human review
wgt	Weights are lookup tables used for scoring

Figure 3-4. Other common naming conventions ZZ7801.0

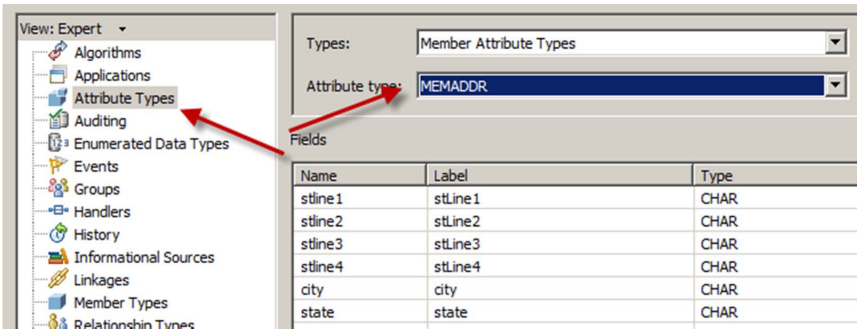
Notes:

Other common suffixes:

- **anon:** Anonymous values are treated as NULL values
- **attr:** Attributes are specific demographics
- **aud:** Audit tables track user and record level activity
- **bkt:** Buckets organize data for searching
- **cmp:** Comparisons check for similarity
- **dvd:** Derived data has been processed by the algorithm
- **ent:** Entities are distinct individuals or organizations
- **equi:** Equivalent strings that can be interchangeable
- **func:** Functions are components used in the algorithm
- **mem:** Members are individual records from a source
- **rel:** Relationships are links between entities, not records

- **seg:** Segments are tables that contain similar data
- **src:** Sources are systems where the hub data came from
- **std:** Standardization changes data to consistent formats
- **str:** Text strings are reference lists used by the algorithm
- **tsk:** Tasks are items that require human review
- **wgt:** Weights are lookup tables used for scoring

Data Dictionary Tables – Attribute Types



mpi_seghead

SEGCODE	SEGNAME	…
MEMADDR	Mpi_memaddr	
MEMPHONE	Mpi_mdmphone	
…		

mpi_segxfield

SEGCODE	FLDNAME	…
MEMADDR	stline1	
MEMADDR	stline2	
MEMADDR	city	
…		

Figure 3-5. Data Dictionary Tables – Attribute Types ZZ7801.0

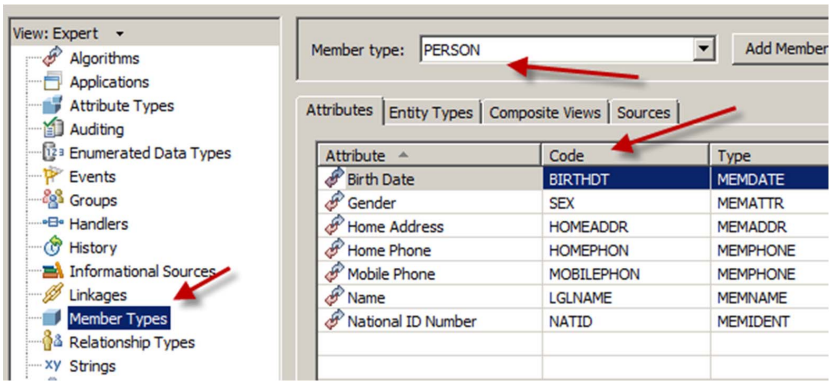
Notes:

As we have mentioned, the data model for InfoSphere MDM Virtual is highly configurable. Before we start looking at the member tables (where the master data is stored), let's take a look at the data dictionary (used to defined the tables and meta data). The data dictionary is configurable using the workbench. When the configuration is deploy, the data dictionary information is persisted to the database and tables that need be created are created.

One of the first pieces of the data dictionary is the Segments . The segments define the attribute types that can used when we are defining member attributes. The mpi_seghead table stores the various segments (e.g. MEMADDR, MEMPHONE, etc) and where they are persisted(e.g. mpi_memaddr table).

Not only do we defined the attribute type, but also the fields that belong to this attribute type (and are shared across all attribute that are of this type). This field information is stored in the mpi_segxfield table. For example, the MEMADDR contains fields stline1, stline2, city, etc and therefore any attribute that is defined as a MEMADDR (e.g. Home Address, Business Address, etc) will have those fields defined.

Data Dictionary Tables – Attributes



mpi_memtype

MEMTYPENO	MEMTYPE	…
1	PERSON	
8	ORG	
…		

mpi_segattr

MEMTYPENO	SEGCODE	ATTRCODE	…
1	MEMDATE	BIRTHDT	
1	MEMIDENT	NATID	
1	MEMNAME	LGLNAME	
…			

Figure 3-6. Data Dictionary Tables – Attributes ZZ7801.0

Notes:

Once we've defined out attributes types (tables and fields that store our master data), we can define out attribute for our member types. The Member types are defined in the mpi_memtype tables. The default basic template will create a PERSON and ORG member type. In our exercises we will only work with the PERSON Member type.

The mpi_segattr table will store the attributes that belong to each member (e.g. Home Address, Business Address, etc…) and what type of attribute they are (e.g. MEMADDR).

These member types and attributes can be defined in the workbench and deployed to the InfoSphere MDM.

There are many other tables used in the Data Dictionary, including Sources, Entity Types, Composite Views but these tables will give us enough to understand the algorithm tables. See the Product Documentation for information on other pieces of the Data Dictionary. http://www-01.ibm.com/support/knowledgecenter/SSWSR9_11.0.0/KC_ditamaps/mdm11.0_welcome.html

Master Data Tables - Members

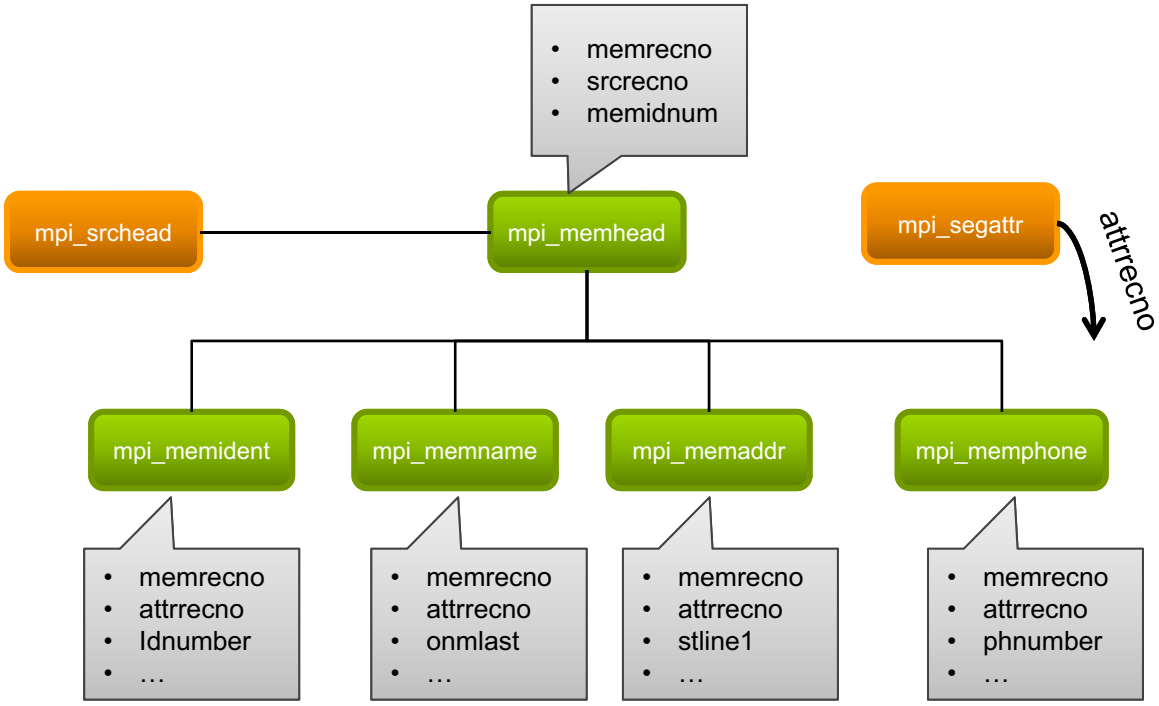


Figure 3-7. Master Data Tables - Members ZZ7801.0

Notes:

Now that we know how to define our Member attributes and types, we can look at how our Master Data is stored. The tables in the diagram above (e.g. mpi_memident, mpi_memname, mpi_memaddr) exist for the basic template. These table were defined under the Attribute Types in the workbench..

The core table for the Member is the mpi_memhead, which defines the member being stored in InfoSphere MDM. The mpi_memhead table will store the source reference (srcrecno), the id of the member as it is defined on the source (memidnum) and a unique id in InfoSphere MDM (memrecno). All member tables will be connected through the memrecno field.

The member tables include the following.

- **mpi_memhead**: Member header information (including original source IDs).
- **mpi_memaddr:** Address information from the source record (home address, shipping address, etc.)
- **mpi_memattr:** Miscellaneous attributes from the source record (gender, e-mail, etc.)
- **mpi_memdate:** Dates from the source record (birth date, account date, last activity date, etc.)

- **mpi_memident:** Identifier information from the source record (SSN, Passport, Driver License, etc.)
- **mpi_memname:** Name information from the source record (Legal Name, Organization Name, etc.)
- **mpi_memphone:** Phone information from the source record (Home Phone, Mobile Phone, Fax, etc.)

Algorithm Metadata – Segment Attribute

Attributes	Standardizations	Roles	Functions	Bucketing Groups

mpi_segattr

MEMTYPENO	ATTRCODE	SEGCODE	…
PERSON	BIRTHDT	MEMDATE	
PERSON	NATID	MEMIDENT	
PERSON	LGLNAME	MEMNAME	
PERSON	HOMEPHON	MEMPHONE	
…			

Figure 3-8. Algorithm Metadata – Segment Attribute ZZ7801.0

Notes:

Now that we have a basic understanding of the Member tables, let's take a look at what we did when we defined a new Algorithm. The first step in the algorithm was to place an Attribute in the Attribute column.

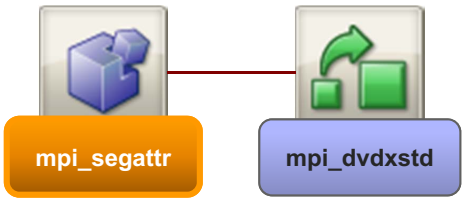
The Attributes that are placed in the algorithm are not defined in the Algorithm, but defined in the Member Type tab of the configuration file. When you place the Attribute in the algorithm you are referencing it to connect it to a standardization function which will see in the next slide.

As we saw previously, the attributes are defined in the data dictionary table mpi_segattr. The following key columns are defined in the table:

- **MemTypeNo**: connects to the member type, in our implementation it is either Person or Organization
- **AttrCode**: defines the unique attribute code for the member that is referenced in other tables
- **SegCode**: defines the type of attribute, which is used to group the attributes into physical tables.
- **AttrName:** The readable name of the attribute

Algorithm Metadata – Standardization Function

Attributes	Standardizations	Roles	Functions	Bucketing Groups



mpi_dvdxstd

ATTRCODE	STDFUNCCODE	STDROLE1	FLDARGS	…
BIRTHDT	DATE1	1	dateval	
NATID	IDENT1N	2	idnum	
LGLNAME	PXNM	3	onmlast, onmfirst, …	
…				

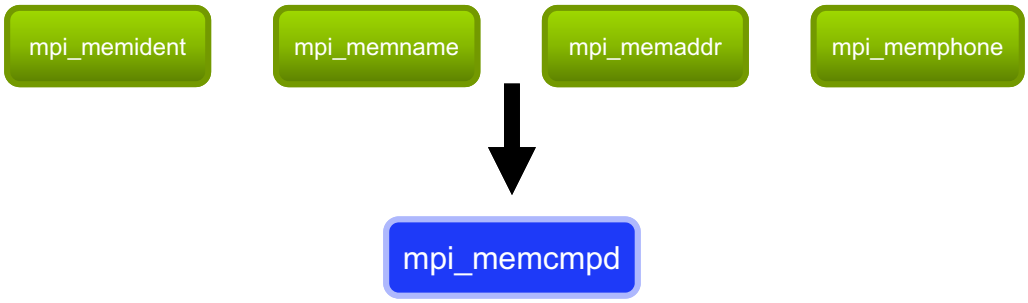
Figure 3-9. Algorithm Metadata – Standardization Function ZZ7801.0

Notes:

The second part of the algorithm is the Standardization function that is connected to the Attribute. The standardization function configuration is defined in the mpi_dvdxstd table and contains the following key fields:

- **AttrCode** – The attribute that is used for the standardization. Connects to the mpi_segattr.
- **StdFuncCode** – the function that will be executed for the standardization (e.g. DATE1, IDENT1N, etc)
- **StdRole** – the unique standardization id within the algorithm
- **FldArgs** – the fields from the attribute that will get fed into the standardization function
- **AnonStrCode** – anonymous strings, used to filter out values that are considered empty

Master Data Tables – Derived Tables



mpi_memcmpd

MEMRECNO	SRCRECNO	CMPVAL	…
1	1	BROSE:PERRY:O:.^85217^N-5520:S-BUCHANAN:S-ST:.S-APACHE:S-JUNCTION:S-AZ:.N-85217:.	
2	1	TEEPLE:DANIAL:E:.^85283^N-30829:S-SIXTEENTH:S-ST:.S-TEMPE:S-AZ:.N-85283:.	
3	1	HOWSE:ADELAIDE:W:.^^N-14553:S-TRIANON:S-PLZ:.S-PHOENIX:S-AZ:..^312212702^3986314	
…			

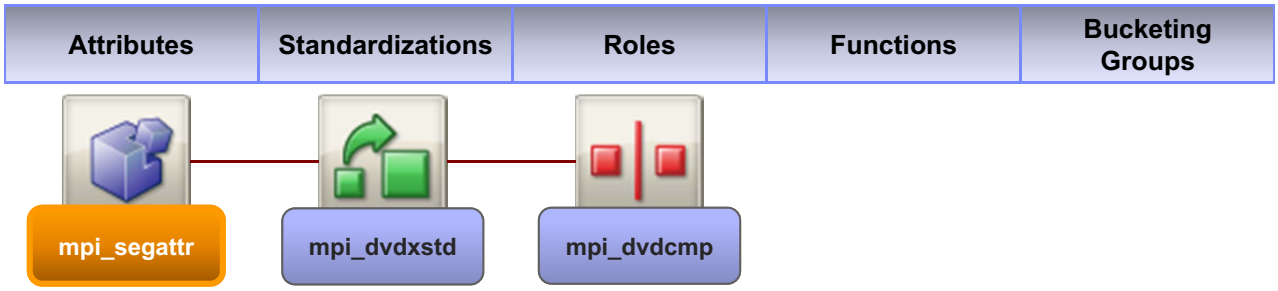
Figure 3-10. Master Data Tables – Derived Tables ZZ7801.0

Notes:

When we add Master Data to the InfoSphere MDM, the data will go through the Algorithm and create the derived data. After the standardization function the data is stored in the mpi_memcmpd table. Some of the key fields of the mpi_mdmcmpd table include:

- **MemRecNo**: The unique identifier of the record
- **SrcRecNo**: The source of the member record
- **CmpVal**: The derived data produced after the standardization function

Algorithm Metadata – Comparison Role



mpi_dvdcmp

STDROLE1	CMPROLE	CMPROLELABEL	…
1	1	Name	
2	1	Name	
3	2	Address	
…			

Figure 3-11. Algorithm Metadata – Comparison Role ZZ7801.0

Notes:

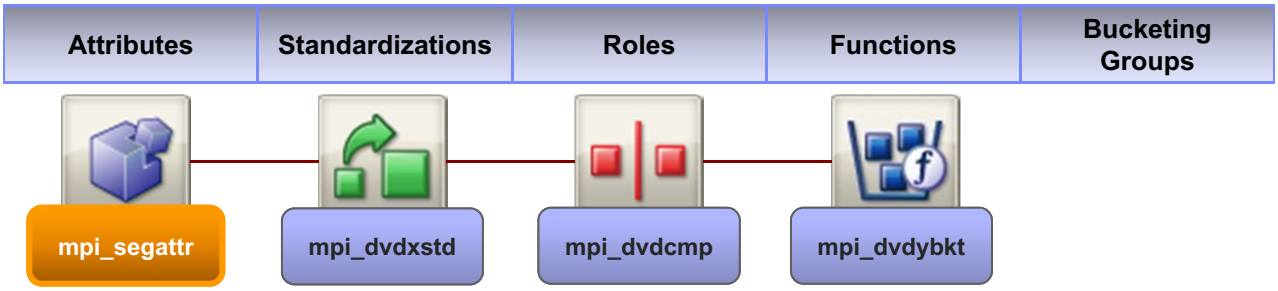
The Comparison Role in the Algorithm is an intermediate table that connects the standardized data (StdRole1) to the Bucketing function and/or the Comparison Functions (CmpRole).

You can think of the Comparison Role as a reference to the standardized data.

Some of the key fields of the mpi_dvdcmp table include:

- **stdrole** – The standardization role
- **cmprole** – The unique identifier for the comparison role
- **cmprolelabel** – The label for the comparison role in the algorithm
- **minWgtFreq** – used to create frequency weight values for values above this threshold

Algorithm Metadata – Bucketing Function



mpi_dvdybkt

CMPROLE	BKTFUNCCODE	GENFUNCCODE	…
1	ATTR	Sorted	
2	ATTR	NGRAM	
3	DATE	YYYYMM	
…			

Figure 3-12. Algorithm Metadata – Bucketing Function ZZ7801.0

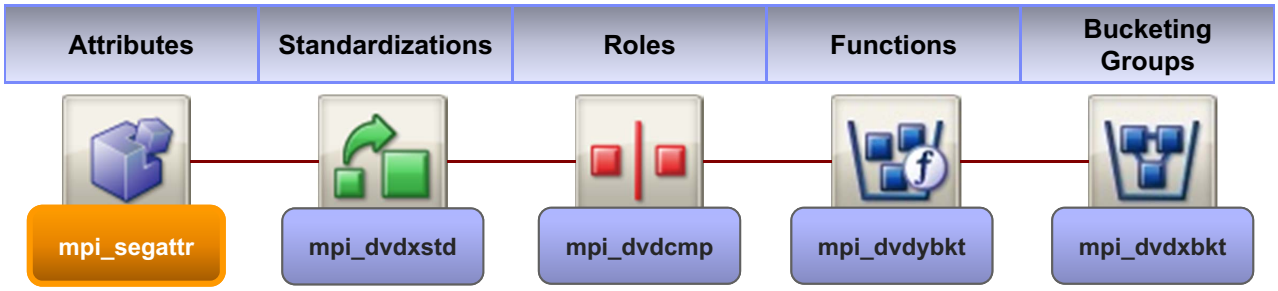
Notes:

The bucketing function in the algorithm is stored in the mpi_dvdybkt table.

The bucketing function table includes the following fields:

- **CmpRole**: Connects the Bucketing Function to the Comparison Role
- **BktFuncCode**: Defined one of the four bucketing functions (Address, Attribtue, Name, Date)
- **GenFuncCode**: Further defines the type of bucketing function (e.g. Sorted, YYYYMM, Equivalency and Phonetic)'
- **DvdGrp:** Connects the Bucketing Function to the Bucketing group

Algorithm Metadata – Bucketing Group



mpi_dvdxbkt

BKTROLE	MINNWAY	MAXNWAY	DVDGRP	
1	2	2	1	
2	1	1	2	
3	2	3	3	
…				

Figure 3-13. Algorithm Metadata – Bucketing Group ZZ7801.0

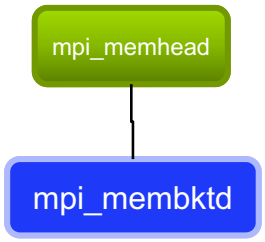
Notes:

The Bucketing Group is defines how the buckets will be created and is stored in the mpi_dvdxbkt table.

The bucketing function table includes the following fields:

- **BktRole**: the bucketing role that is used to identify the bucketing group
- **MinNWay**: the minimum number of tokens used to create a bucket
- **MaxNWay**: the maximum number of tokens used to create a bucket
- **DVDGrp**: The bucketing group id, this is connection between the bucketing function and group

Master Data Tables – Bucketing Table



mpi_membktd

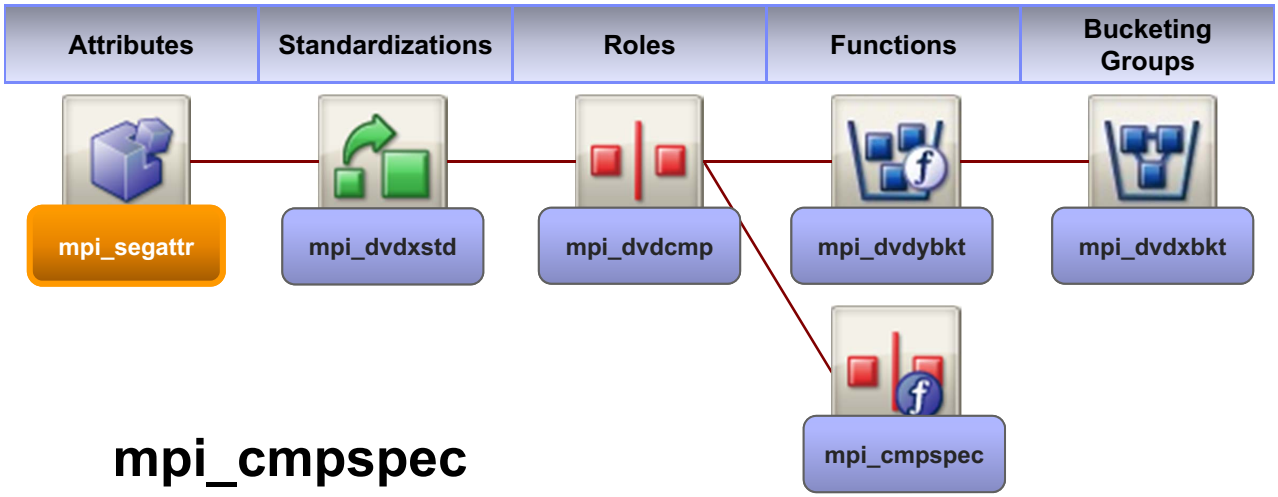
MEMRECNO	SRCRECNO	BKTHASH	…
1	1	9059273197451191198	
1	1	2351273197451191198	
2	1	7252340299695367517	
…			

Figure 3-14. Master Data Tables – Bucketing Table ZZ7801.0

Notes:

Once the standardized data is run through the bucketing process, the members are linked to the buckets that they are associated with. The mpi_membktd table stores the connection from the Member to the Bucket. The Bucket is identified by a hash code that represents the values, this improves the performance when retrieving members from a bucket. Each Hash Code represents a different bucket value (e.g. the bucket Hash 9059273197451191198 might be James Smith).

Algorithm Metadata – Comparison Function



mpi_cmpspec

CMPFUNCCODE	CMPSPECCODE	CMPROLE1	…
DR1D1C	NATID	1	
QXNM	NAME	2	
…			

Figure 3-15. Algorithm Metadata – Comparison Function ZZ7801.0

Notes:

The last piece of the algorithm is the Comparison Function which is stored in the mpi_cmpspec table.

The following key fields exist in the mpi_cmpspec table:

- **cmpcode**: the comparison role link
- **cmpfunccode**: the function that will be run for the comparison
- **cmpspeccode**: the unique name of the comparison function

Other algorithm related tables

Name	Meaning
mpi_srcxsrc	Stores clerical review and auto-link thresholds for each source to source combo
mpi_stranon	Stores list of anonymous values that weed out filler data
mpi_strequi	Contains equivalencies, like nicknames, used in data processing
mpi_wgt1dim	Stores the weight scores for 1 dimensional and basic edit distance comparisons
mpi_wgt2dim	Stores two dimensional, edit distance weights, like Address & Phone
mpi_wgtsval	Lists the scores for strings like names and address elements
mpi_wgtnval	Holds the weight scores for exact match on dates

Figure 3-16. Other algorithm related tables ZZ7801.0

Notes:

A few other tables are used for the weights and thresholds that we will look at in the next unit

- **mpi_srcxsrc:** Stores clerical review and auto-link thresholds for each source to source combo
- **mpi_stranon:** Stores list of anonymous values that weed out filler data
- **mpi_strequi:** Contains equivalencies, like nicknames, used in data processing
- **mpi_wgt1dim:** Stores the weight scores for 1 dimensional and basic edit distance comparisons
- **mpi_wgt2dim:** Stores two dimensional, edit distance weights, like Address & Phone
- **mpi_wgtsval:** Lists the scores for strings like names and address elements
- **mpi_wgtnval:** Holds the weight scores for exact match on dates

Exercise introduction

- In this exercise, you will:
  - Deploy the InfoSphere MDM configuration
  - Load data into the InfoSphere MDM

Exercise: Loading Members and viewing Virtual data model

Figure 3-17. Exercise introduction ZZ7801.0

Notes:

In this exercise, you will:

- Deploy the InfoSphere MDM configuration
- Load data into the InfoSphere MDM

Exercise: Loading Members and viewing Virtual data model

Unit summary

Having completed this unit, you should be able to:

- Understand the Algorithm Data Model
- Understand the Member Derived Data tables
- Understand the Bucketing data model
- Deploy the PME Configuration

Figure 3-18. Unit summary ZZ7801.0

Notes:

Having completed this unit, you should be able to:

- Understand the Algorithm Data Model
- Understand the Member Derived Data tables
- Understand the Bucketing data model
- Deploy the PME Configuration

Unit 4. Bucket Analysis

What this unit is about

This unit describes the options for analyzing buckets that were created in the algorithm.

What you should be able to do

After completing this unit, you should be able to:

- Understand options for evaluating our buckets
- Running Bucket Analysis
- Updating Buckets based on Analysis

Unit objectives

After completing this unit, you should be able to:

- Understand options for evaluating our buckets
- Running Attribute Completeness
- Running Bucket Analysis
- Updating Buckets based on Analysis

Figure 4-1. Unit objectives ZZ7801.0

Notes:

After completing this unit, you should be able to:

- Understand options for evaluating our buckets
- Running Attribute Completeness
- Running Bucket Analysis
- Updating Buckets based on Analysis

Bucket analysis is the key to performance

- Beyond hardware and memory, bucket design has greatest impact on performance
- There are simple checks to ensure that buckets are not causing issues:
  - **Check bucket size**
    - Bucket Analysis Overview Query
  - **Find unbucketed members:**
    - Bucket Analysis Overview Query. Without buckets, record cannot be searched or compared
  - **Possible anonymous values:**
    - Largest buckets, often can see values that may be anonomous
  - **Note bucket count:**
    - Lot of buckets can require more memory to hold in cached memory for faster retrieval

Figure 4-2. Bucket analysis is the key to performance ZZ7801.0

Notes:

Bucket analysis is an important performance steps as buckets have the greatest impact on performance beyond hardware and memory.

Organization’s searching requirements, performance goals, and population size are key when designing bucketing algorithms. Bucketing technologies and techniques should be selected after a careful review of the organization's needs and source data characteristics. Improperly applied bucketing methods can lead to excessively large buckets with will impact the performance goals.

***Large* buckets: 2,000+**

Large buckets are generally defined as those with 2,000 members or more. Simply using FBB(frequency based bucketing) to eliminate the largest buckets is not sufficient. Check for cases where there are 100's or 1000's of buckets with 1000-2000 members. Buckets in that range would just slip under what was targeted with FBB. If a large quantity of medium to large-sized buckets are left after applying FBB, then the algorithm might need to be tuned or the FBB maxbucketsize setting might need to be lowered. Having a large number of medium to large-sized buckets can just as adversely affect performance as having a few huge buckets.

The other possibility for large bucket is anonymous value. You should check the largest buckets and look for any anonymous value may actually cause the large bucket.

Members not in a bucket

When a member is not contained in any bucket, it means the only way to retrieve it is by unique identifier and it would never be returned by a standard attribute search. Usually this happens with very sparsely populated members or members comprised mainly of junk or anonymous values.

It is important to confirm that members with no rows in mpi_membktd are in fact made up of missing or anonymous values. Verify that the missing rows are a product of correct behavior. If a large number of legitimate members are missing from mpi_membktd - perhaps it is because you are bucketing using fields that are missing from those members. For example, if they only have name and address information while you are bucketing on name+phone or name+DOB this would cause no buckets to be created since DOB or phone data is missing. If this is not acceptable the algorithm should be updated so that buckets are created for the affected members.

Finally you should pay attention to **bucket count:** because the buckets act like an index to the database when searching, if there are a lot of buckets it can require more memory to hold those bucket in cached memory for faster retrieval

Bucket optimization

- Performance directly linked to Bucket size
  - Large buckets take longer to search
  - Should generally have < 2,000 members

- Bucket optimization can be conducted a few ways
  - Tweaking bucket design by combining multiple tokens
  - Disabling features like Sorted and Phonetic bucketing
  - Running Frequency Based Bucketing to disable large buckets

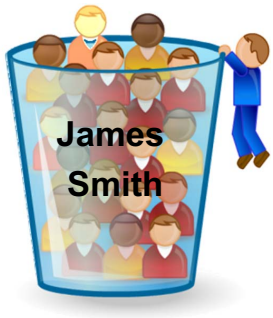


Figure 4-3. Bucket optimization ZZ7801.0

Notes:

Performance is directly linked to Bucket size: Large buckets take longer to search; Buckets should generally have <2,000 members

Bucket optimization can be conducted in a few ways: Tweaking bucket design by combining multiple tokens; Disabling features like Sorted and Phonetic bucketing; Running Frequency Based Bucketing to disable large buckets.

Frequency Based Bucketing is a feature which can be enabled at bucketing group level. To enable frequency-based bucketing on a particular bucketing group, select the bucketing group in the algorithm editor, then set the “Maximum bucket size” property in the Properties view to a value greater than 0, then re-deploy the algorithm to the Master Data Engine.

Frequency-Based Bucketing (FBB) feature let you count the number of members in each bucket, and then determine the buckets that exceed the

Maximum Bucket Frequency number; Frequency-based bucketing *cuts-off* searching against these large buckets cause the amount of members in that hash becomes too high to provide meaningful and/or fast enough results. If there are10,000 members in the system whose derived data includes

a particularly common bucket, then whenever your search includes that bucket, the system can take a LONG time to retrieve the attribute data for all those members.

On this slide, we can see if we have a 2 token bucket on name. Potentially we can end up with some very large bucket. For instance, common name like James Smith can constitute very larger buckets. In this case, we need tweak the design of bucket by combining multiple tokens.

Bucket optimization

- Performance directly linked to Bucket size
  - Large buckets take longer to search
  - Should generally have < 2,000 members

- Bucket optimization can be conducted a few ways
  - Tweaking bucket design by combining multiple tokens
  - Disabling features like Sorted and Phonetic bucketing
  - Running Frequency Based Bucketing to disable large buckets

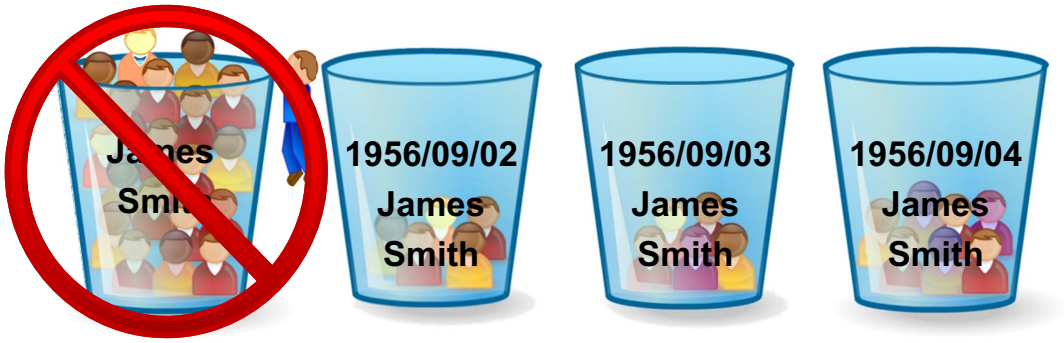


Figure 4-4. Bucket optimization ZZ7801.0

Notes:

Continue on the previous case, if we change the bucket design by combing name and date of birth, potentially we can avoid the previous problem we had encountered.

Bucket Analytics Options

- **Bucket Analysis Overview:** general stats, 10 largest buckets, and unbucketed
- **Bucket Composition:** shows individual members inside bucket
- **Bucket Size Distribution:** charts buckets by size of buckets and number of buckets
- **Buckets By Size:** lists buckets by size
- **Member Bucket Frequency:** shows number of buckets for members
- **Member Bucket Values:** shows specific buckets for a member
- **Member Comparison Distribution:** average number of records compared
- **Members By Bucket Count:** shows how many members in each bucket

Figure 4-5. Bucket Analytics Options ZZ7801.0

Notes:

There are multiple bucket analysis queries can be run:

- **Bucket Analysis Overview:** shows general stats, 10 largest buckets, and unbucketed
- **Bucket Composition:** shows individual members inside a bucket
- **Bucket Size Distribution:** charts buckets by size of buckets and number of buckets
- **Buckets By Size:** lists buckets by size
- **Member Bucket Frequency:** shows number of buckets for members
- **Member Bucket Values:** shows specific buckets for a member
- **Member Comparison Distribution:** average number of records compared at search
- **Members By Bucket Count:** shows how many members in each bucket

Bucket Analytics Concerns

- When you run Bucket Analysis, look for:
  - Are there any “Unbucketed Members”?
  - Are there any Buckets with more than 2,000 members?
  - Are there any large buckets which seem to be caused by “fake data”?

Figure 4-6. Bucket Analytics Concerns ZZ7801.0

Notes:

We will do the bucket analysis lab next. After you finish the lab, please find answers to the following questions:

- Are there any “Unbucketed Members”?
- Are there any Buckets with more than 2,000 members?
- Are there any large buckets which seem to be caused by “fake data”?

Exercise introduction

- In this exercise, you will:
  - Run Attribute Completeness
  - Run Bucket Analysis
  - Update Algorithm to improve buckets

Exercise: Bucket Analysis

Figure 4-7. Exercise introduction ZZ7801.0

Notes:

In this exercise, you will:

- Run Attribute Completeness
- Run Bucket Analysis
- Update Algorithm to improve buckets

Exercise: Bucket Analysis

What did you find in your analysis?

- Were there any buckets that were larger than 2,000 records?
- Were there any anonymous values that you discovered?
- Were there any unbucketed members?
- Are you happy with the current options for searching?

Figure 4-8. What did you find in your analysis? ZZ7801.0

Notes:

Now you have finished the lab, what did you find in your analysis?

- Were there any buckets that were larger than 2,000 records?
- Were there any anonymous values that you discovered?
- Were there any unbucketed members?
- Are you happy with the current options for searching?

What did you find in your analysis?

- Were there any buckets that were larger than 2,000 records?
  - 99999999 phone number bucket
- Were there any anonymous values that you discovered?
  - 99999999 phone number
- Were there any unbucketed members?
  - 589 unbucketed members (no phone, SSN, or DOB)
  - These unbucketed members all have names
- Are you happy with the current options for searching?
  - Currently you cannot search by First & Last Name

Figure 4-9. What did you find in your analysis? ZZ7801.0

Notes:

Here are the answers to the previous slide.

How can you fix the problems?

- Fixing the 9999999 phone bucket:
  - Create a new anonymous Phone string, attach phone.txt, and plug into algorithm
- Anonymizing the date 1901-01-01:
  - Create a new anonymous Date string, attach date.txt, and plug into the algorithm
- Finding buckets for your unbucketed members:
  - Change the Name + DOB bucket so that the Name bucket (Equivalence & Phonetic) has a maximum token setting of 2 (allows two names, like First & Last, to go to the bucket)
  - Change the Date bucket (YYYYMM) so that it has a minimum tokens of 0 (makes date optional)

Figure 4-10. How can you fix the problems? ZZ7801.0

Notes:

So how do we fix the 9999999 phone bucket: we could create a new anonymous Phone string, attach it to phone.txt, and plug into algorithm

We already added the 1901-01-01 anonymous date of birth in our data set by creating a new anonymous Date string, attached the anondate.txt, and plugged into the algorithm

Finally, let's fix the unbucketed members problem. We can change the Name + DOB bucket so that the Name bucket (Equivalence & Phonetic) has a maximum token setting of 2 (allows two names, like First & Last, to go to the bucket)

We could change the name token + dob search bucket by setting the minimum tokens to 2 and then maximum tokens to 3 and then set the Date bucket (YYYYMM) function so that it has a minimum tokens of 0 (makes date optional) and maximum of 1 token.

Unit summary

Having completed this unit, you should be able to:

- Understand options for evaluating our buckets
- Running Attribute Completeness
- Running Bucket Analysis
- Updating Buckets based on Analysis

Figure 4-11. Unit summary ZZ7801.0

Notes:

Having completed this unit, you should be able to:

- Understand options for evaluating our buckets
- Running Attribute Completeness
- Running Bucket Analysis
- Updating Buckets based on Analysis

Unit 5. Weights

What this unit is about

This unit describes the process of generating weights for the comparison function. We will first discuss the process and then go into details on how InfoSphere MDM generates weight for various functions.

What you should be able to do

After completing this unit, you should be able to:

- Understand how weights are generated
- Understand how to evaluate weights
- Understand various types of weights
  - Frequency based weights
  - Edit Distance weights
- Parameterized weight

Unit objectives

After completing this unit, you should be able to:

- Understand how weights are generated
- Understand how to evaluate weights
- Understand various types of weights
  - Frequency based weights
  - Edit Distance weights
  - Parameterized weights

Figure 5-1. Unit objectives ZZ7801.0

Notes:

After completing this unit, you should be able to:

- Understand how weights are generated
- Understand how to evaluate weights
- Understand various types of weights
  - Frequency based weights
  - Edit Distance weights
  - Parameterized weights

Weight generation

- Weights
  - Scores that represent confidence that a value uniquely identifies a record
  - Calculated for each comparison, based on data and algorithm
  - Stored in lookup tables – when data is compared, the scores are pulled

- Weight generation is very resource intensive
  - Recommended weight generation run when low server traffic
  - Large datasets (>10 million) may take few days to process
  - In class, weight generation on dataset will take 30-45 minutes

- To save time we will kick off the weight generation job first, then discuss weights

Figure 5-2. Weight generation ZZ7801.0

Notes:

As indicated earlier, weight generation is a process within the InfoSphere MDM that enables the generation and assignment of weight values to attributes. Weight assignments enable determination of a match or non-match between members. The weight generation process is executed from InfoSphere MDM Workbench. Weights are calculated for each entity type, based on their data and algorithms. Weights are stored in lookup tables. Weight generation is very resource intensive. Weight generation for large datasets (>10 million) may take a few days to process. In our class, weight generation will take 30-45 minutes.

Exercise introduction

- In this exercise, you will:
  - Generate Wights (Up to Step 5)

Exercise: Generating Weights

Figure 5-3. Exercise introduction ZZ7801.0

Notes:

In this exercise, run the steps in the Generating Weights exercise up to the point where the Weight Generation job is started.

Exercise: Generating Weights

Weights overview

- 3 kinds of weights:
  - **Frequency-based Weights:**
    - Sore based on how often value appears within overall data
    - Common values (e.g. John) have low score, rare values (e.g Chitsumungo) have high score.
  - **Edit Distance Weights:**
    - Measures similarity between two values.
    - E.g. “Gordon” vs. “Gorton” has a distance of 1 edit.

  - **Parameterized (PARM) Weights:**
    - Control max caps on scores, extra credit points, and penalties for variance.
    - E.g. max weight for Full Name ensures that name does not generate a disproportionate score.

Figure 5-4. Weights overview ZZ7801.0

Notes:

There are three kinds of weights:

- **Frequency-based Weights:** provide a score based on how often a value appears within the overall data population. Common values (like John) have a low score, rare values (like Chitsumungo) have a high score.
- **Edit Distance Weights:** are a measure the similarity between two values. For example, “Gordon” vs. “Gorton” has a distance of 1 edit. Exact match has the highest score, but each edit lowers the score by a certain degree.
- **Parameterized (PARM) Weights:** These weights control maximum caps on scores, extra credit points, and penalties for variance. For example, there is a maximum weight for Full Name that ensures that the name does not generate a disproportionate score.

Weights operate on the 80/20 rule. When InfoSphere MDM calculates weights, it typically only generates specific weight values for the most common values in the population up to the point where 80% of the population is accounted for. The remaining 20% of the population have the rarest values and therefore can be given the same “default” weight score which is the highest possible weight for that attribute/token. Remember, the more rare the value, the higher its weight. Therefore,

the values that are most rare all receive the highest possible weight score. The beauty in this approach is that when a distinctly new value is added to InfoSphere MDM, it can still be assigned a weight value, even though the system has never seen that value before.

What makes you think these records match?

	Member A	Member B
Name	Robert L. Kognoski	Bob L. Kognosky
NATLID	813-12-1147	813-12-1174
Gender	Male	Male

- The fact that Bob is a nickname for Robert?
- That they share the Middle Initial of L.?
- That Kognoski and Kognosky sound alike?
- The fact that the SSN values just have a simple typo?
- The fact that they both are Male?
- Could it be a combination of all of these elements?

Figure 5-5. What makes you think these records match? ZZ7801.0

Notes:

Weight are a measure or calculation of our confidence that a particular set of attributes are a match or a non-match based on the results of a comparison. Let us look at a sample pair.

What makes you confident that both of these members are the same person?

- Is it the fact that Bob is a nickname for Robert?
- Is it that they share the Middle Initial of L.?
- Is it that Kognoski and Kognosky sound alike?
- Is it the fact that the SSN values just have a simple typo?
- Is it the fact that they both are Male?
- Could it be a combination of all of these elements?

Rate the following matches

- Rate the following scenarios on a scale of 1-10 where 10 = “definite match”:

Scenario	Rating
If you see two Female records does that mean that they are the same woman?	
If you see two people who both have the same SSN are they the same person?	
If you see one record named Sue Chaudray-Patel and another named Susan C. Patel are they the same person?	
If you see one record for John Smith born on June 5, 1938 and another record for a John Smith with no birth date are they the same person?	
If you see one record with a DOB of 1962-04-23 and a SSN of 718-12-0921 and another record with a DOB of 1962-04-13 and a SSN of 718-21-0921 are they the same person?	

Figure 5-6. Rate the following matches ZZ7801.0

Notes:

Look at the following questions and rate your answer on a scale of 1 to 10 (10 being very confident) that you have found a match:

- If you see two Female records does that mean that they are the same woman?
- If you see two people who both have the same SSN, are they the same person?
- If you see one record named Sue Chaudray-Patel and another named Susan C. Patel, are they the same person?
- If you see one record for John Smith born on June 5, 1938 and another record for a John Smith with no birth date, are they the same person?
- If you see one record with a DOB of 1962-04-23 and a SSN of 718-12-0921 and another record with a DOB of 1962-04-13 and a
- SSN of 718-21-0921, are they the same person?

Rate the following matches

- Rate the following scenarios on a scale of 1-10 where 10 = “definite match”:

Scenario	Rating
If you see two Female records does that m	
If you see two people who both have the same SSN are they the s person?	
If you see one record named Sue Chaudray-Patel and another named Susan C. Patel are they the same person?	
If you see one record for John Smith born on June 5, 1938 and another record for a John Smith with no birth date are they the same person?	
If you see one record with a DOB of 1962-04-23 and a SSN of 718-12-0921 and another record with a DOB of 1962-04-13 and a SSN of 718-21-0921 are they the same person?	

Why does Gender mean so much less than SSN?

Figure 5-7. Rate the following matches ZZ7801.0

Notes:

Do you see how different elements such as Gender versus SSN would give you different levels of confidence? This is because the more frequently a value appears, the less weight it has. The rarer a value is, the more weight it has. -- Platinum and Aluminum are both metals, but the rarity of Platinum gives it more value.

Rate the following matches

- Rate the following scenarios on a scale of 1-10 where 10 = “definite match”:

Scenario	Rating
If you see two Female records does that mean that they are the same woman?	
If you see two people who both have the same SSN are they the same person?	
If you see one record named Sue Chaudray-Patel and another named Susan C. Patel are they the same person?	
If y … other record for …	
If you see one record with a DOB of 1962-04-23 and a SSN of 718-12-0921 and another record with a DOB of 1962-04-13 and a SSN of 718-21-0921 are they the same person?	

What if I know that Sue and Susan are the same person, how can data be that different?

Figure 5-8. Rate the following matches ZZ7801.0

Notes:

Names can be spelled differently, especially some names can have different nicknames. Last name like Chaudray-Patel is really rare compared to Patel. That’s why Chardray-Patel will definitely have a higher weight.

Rate the following matches

- Rate the following scenarios on a scale of 1-10 where 10 = “definite match”:

Scenario	Rating
If you see two Female records does that mean that they are the same woman?	
If you see two people who both have the same SSN are they the same person?	
If you see one record named Sue Chaudray-Patel and another named Susan C. Patel are they the same person?	
If you see one record for John Sr for a John Smith with no birth dat	
If you see one record with a DOB and another record with a DOB o they the same person?	

Why did you rank records with an exact match (John Smith) lower than two records that had different DOBs and SSNs?

Figure 5-9. Rate the following matches ZZ7801.0

Notes:

Common name like John Smith carries a lot less weight. That’s why even though we have an exact match on name, like John Smith, we can still potentially rank these records lower than two records that had different DOBs and SSNs.

Thinking more about weights

- When your brain compares data, you make snap judgments
  - Weights help MDM make snap decisions
  - Weights are calculated through extensive process that measures your real data
  - Weights are lookup tables, at run-time there's less analysis and more tabulation
- Your brain is inherently metaphorical, so you can easily say data is 'like' other data
  - Weights (particularly Edit Distance and PARM weights) have built in measurements for dealing with data that is not exactly the same
  - Scores for edit distance, phonetics, and equivalents are lower than exact matches
- When you don't have all the facts you go with what you know
  - Weights can't see into the unknown, so they only focus on one element at a time
  - If some data matches (like, Address and Phone) but other data does not (like, Name and Birth Date) some attributes will have high scores, while others have low
  - It's the cumulative score, the sum of all the individual weights, that really matters

Figure 5-10. Thinking more about weights ZZ7801.0

Notes:

When your brain goes through the cognitive process of comparing two records, you can make the judgment that these two members are the same person because you have ample information and you know that some data elements are more important than others. That is similar to how weights work.

But, what if your information is less than complete? For example, if you had two cell phone numbers that were exactly the same, you would reasonably expect them to belong to the same person. But if you saw that the Gender, Birth Date, and Name of the members were different, your confidence would drop. That is also how weights work.

Identifying the weight tables

Table	Description
mpi_wgthead	Core definitions of weights
mpi_wgt1dim	1 dimensional weight values that have single comparison attribute (such as SSN)
mpi_wgt2dim	2 dimensional weight values that have two attributes(such as Eye Color + Hair Color).
mpi_wgt3dim	3 dimensional weight values that have 3 attributes (such as Zip Code + Address + Phone Number). **Not used in class.**
mpi_wgt4dim	Weight values for the False Positive Filter, which uses four separate attributes to controls situations like Twins or Jr's & Sr's who are mistakenly linked. **Not used in class.**
mpi_wgtsval	Common string (text) weight values based purely on frequency.
mpi_wgtnval	Common numeric weight values based purely on frequency.

Figure 5-11. Identifying the weight tables ZZ7801.0

Notes:

There are several tables that come into play when dealing with weights. The dimensional weight tables read the algorithm directly to see how many values are compared together in the same comparison function. If two attributes like Last Name and Zip are compared together, then the weights of Last Name + Zip will appear in the mpi_wgt2dim table.

- **mpi_wgthead:** Holds core definitions of the weights, including the comparison specification string and the weight type (such as 1dim or sval).
- **mpi_wgt1dim**: Holds the weight values for the appropriate comparison functions that have a single comparison attribute (such as SSN and Date of Birth).
- **mpi_wgt2dim**: Holds the weight values for the appropriate comparison functions that use two attributes (such as Eye Color + Hair Color).
- **mpi_wgt3dim**: Holds the weight values for the appropriate comparison functions that use three attributes (such as Zip Code + Address + Phone Number).
- **mpi_wgt4dim** : Holds the weight values for the False Positive Filter, which uses four separate attributes to controls situations like Twins or Jr's & Sr's who are mistakenly linked.

- **mpi_wgtsval**: Holds common string (text) weight values based purely on frequency. This is where you will see the weights for people's names and attributes that use a simple *match or do not match* like gender.
- **mpi_wgtnval:** Holds common numeric weight values based purely on frequency. You will typically see date information like birth year weighed here.

One dimensional weights – wgt1dim

- In class, this table will store weights for SSN, DOB, and AXP (address & phone)

Comparison Type	Index	Weight
CMPID-SSN-DIST	0	0
CMPID-SSN-DIST	1	556
CMPID-SSN-DIST	2	403
CMPID-SSN-DIST	3	315
CMPID-SSN-DIST	4	177
CMPID-SSN-DIST	5	31
CMPID-SSN-DIST	6	-91
CMPID-SSN-DIST	7	-205
CMPID-SSN-DIST	8	-291
CMPID-SSN-DIST	9	-299
CMPID-SSN-DIST	10	-311

Comparison Type	Index	Weight
CMPID-DOB-DIST	0	0
CMPID-DOB-DIST	1	0
CMPID-DOB-DIST	2	44
CMPID-DOB-DIST	3	-176
CMPID-DOB-DIST	4	-285

Note: All weight scores are divided by 100 in the display layer (556 = 5.56 points)

Comparison Type	Index	Weight
CMPID-AXP-1DIM	0	0
CMPID-AXP-1DIM	1	419
CMPID-AXP-1DIM	2	504
CMPID-AXP-1DIM	3	596
CMPID-AXP-1DIM	4	723
CMPID-AXP-1DIM	5	860
CMPID-AXP-1DIM	6	997
CMPID-AXP-1DIM	7	1126
CMPID-AXP-1DIM	8	1273

Figure 5-12. One dimensional weights – wgt1dim ZZ7801.0

Notes:

On this slide, you will see some 1 dimensional weight tables of the SSN, DOB and AXP (Address and Phone)

One dimensional weights – wgt1dim

- In class, this table will store weights for SSN, DOB, and AXP (address & phone)

Comparison Type	Index	Weight
CMPID-SSN-DIST	0	0
CMPID-SSN-DIST	1	556
CMPID-SSN-DIST	2	403
CMPID-SSN-DIST	3	315
CMPID-SSN-DIST	4	177
CMPID-SSN-DIST	5	31
CMPID-SSN-DIST	6	-91
CMPID-SSN-DIST	7	-205
CMPID-SSN-DIST	8	-291
CMPID-SSN-DIST	9	-299
CMPID-SSN-DIST	10	-311

Comparison Type	Index	Weight
CMPID-DOB-DIST	0	0
CMPID-DOB-DIST	1	0
CMPID-DOB-DIST	2	44
CMPID-DOB-DIST	3	-176
CMPID-DOB-DIST	4	-285

Comparison Type	Index	Weight
CMPID-AXP-1DIM	0	0
CMPID-AXP-1DIM	1	419
CMPID-AXP-1DIM	2	504
CMPID-AXP-1DIM	3	596
CMPID-AXP-1DIM	4	723
CMPID-AXP-1DIM	5	860
CMPID-AXP-1DIM	6	997

Edit distance weights, like for SSN, use the index as a placeholder for the edits. 0=missing, 1=exact match, 2=one edit, 3=two edits, etc. The numbers decrease from index 1 through 10, penalizing more as data becomes different.

Figure 5-13. One dimensional weights – wgt1dim ZZ7801.0

Notes:

On this slide, the left most weight table is for SSN. It uses edit distance weight. The index is used as a placeholder for the edits. 0=missing, 1=exact match, 2=one edit, 3=two edits, etc. The numbers decrease from index 1 through 10, penalizing more as data becomes different.

One dimensional weights – wgt1dim

- In class, this table will store weights for SSN, DOB, and AXP (address & phone)

Comparison Type	Index	Weight
CMPID-SSN-DIST	0	0
CMPID-SSN-DIST	1	556
CMPID-SSN-DIST	2	403
CMPID-SSN-DIST	3	315
CMPID-SSN-DIST	4	177
CMPID-SSN-DIST	5	31
CMPID-SSN-DIST		
CMPID-SSN-DIST		
CMPID-SSN-DIST		
CMPID-SSN-DIST		
CMPID-SSN-DIST		

Comparison Type	Index	Weight
CMPID-DOB-DIST	0	0
CMPID-DOB-DIST	1	0
CMPID-DOB-DIST	2	44
CMPID-DOB-DIST	3	-176
CMPID-DOB-DIST	4	-285

Comparison Type	Index	Weight
CMPID-AXP-1DIM	0	0
CMPID-AXP-1DIM	1	419
CMPID-AXP-1DIM	2	504
CMPID-AXP-1DIM	3	596
CMPID-AXP-1DIM	4	723
CMPID-AXP-1DIM	5	860
PID-AXP-1DIM	6	997
PID-AXP-1DIM	7	1126
PID-AXP-1DIM	8	1273

The Date comparison function stores weights across two tables. 0=missing, 1=exact match (scores are in wgtnval), 2=one edit, 3=two edits, etc. Scores decrease with each edit.

Figure 5-14. One dimensional weights – wgt1dim ZZ7801.0

Notes:

On this slide, the middle weight table is for birth date.

The Date comparison function stores weights across two tables. 0=missing, 1=exact match (scores are in wgtnval), 2=one edit, 3=two edits, etc. Scores decrease with each edit.

One dimensional weights – wgt1dim

- In class, this table will store weights for SSN, DOB, and AXP (address & phone)

Comparison Type	Index	Weight
CMPID-SSN-DIST	0	0
CMPID-SSN-DIST	1	556
CMPID-SSN-DIST	2	403
CMPID-SSN-DIST	3	315

Comparison Type	Index	Weight
CMPID-DOB-DIST	0	0
CMPID-DOB-DIST	1	0
CMPID-DOB-DIST	2	44
CMPID-DOB-DIST	3	-176

Comparison Type	Index	Weight
CMPID-AXP-1DIM	0	0
CMPID-AXP-1DIM	1	419
CMPID-AXP-1DIM	2	504
CMPID-AXP-1DIM	3	596
	4	723
	5	860
		997
	7	1126
	8	1273

Dimension weights, like for AXP, use the index to indicate the digit count in numerical address elements. 0=missing, 1=one digit, 2=two digits, 3=three digits, etc. Scores are applied when two addresses contain the exact same number. These scores increase - one-digit numbers are very common (low score), but eight-digit numbers are more rare (high score).

Figure 5-15. One dimensional weights – wgt1dim ZZ7801.0

Notes:

Dimension weights, like for address and phone will use multiple weight tables. On this slide, you will see one 1-dimensional weight table used. The index is used to indicate the digit count in numerical address elements. 0=missing, 1=one digit, 2=two digits, 3=three digits, etc. Scores are applied when two addresses contain the exact same number. These scores increase - one-digit numbers are very common (low score), but eight-digit numbers are more rare (high score).

Two dimensional weights – wgt2dim

- In class, this table will store the final weights for AXP (address & phone)

Comparison Type	Index	Ph. Missing	Ph. Exact	Ph. 1 ED	Ph. 2 ED	Ph. 3 ED	Ph. 4 ED	Ph. 5 ED	Ph. 6 ED
CMPID-AXP-2DIM	0	0	299	217	60	-89	-127	-174	-285
CMPID-AXP-2DIM	1	300	615	560	420	300	250	220	200
CMPID-AXP-2DIM	2	263	584	530	395	283	205	159	132
CMPID-AXP-2DIM	3	226	553	500	370	266	160	98	64
CMPID-AXP-2DIM	4	189	522	470	345	249	115	37	-4
CMPID-AXP-2DIM	5	152	491	440	320	232	70	-24	-72
CMPID-AXP-2DIM	6	115	460	410	295	215	25	-85	-140
CMPID-AXP-2DIM	7	78	429	380	270	198	-20	-146	-208
CMPID-AXP-2DIM	8	41	398	350	245	181	-65	-207	-276
CMPID-AXP-2DIM	9	4	367	320	220	164	-110	-268	-344
CMPID-AXP-2DIM	10	-33	336	290	195	147	-155	-329	-412
CMPID-AXP-2DIM	11	-70	305	260	170	130	-200	-390	-480
CMPID-AXP-2DIM	12	-107	274	230	145	113	-245	-451	-548
CMPID-AXP-2DIM	13	-144	243	200	120	96	-290	-512	-616
CMPID-AXP-2DIM	14	-181	212	170	95	79	-335	-573	-684
CMPID-AXP-2DIM	15	-218	181	140	70	62	-380	-634	-752

Figure 5-16. Two dimensional weights – wgt2dim ZZ7801.0

Notes:

On this slide, you will see an example of 2 dimensional weight table.

We are looking at what a 2-dim table could possibly look like if we were using the AXP (address by phone) comparison function.

Two dimensional weights – wgt2dim

- In class, this table will store the final weights for AXP (address & phone)

Comparison Type	Index	Ph. Missing	Ph. Exact	Ph. 1 ED	Ph. 2 ED	Ph. 3 ED	Ph. 4 ED	Ph. 5 ED	Ph. 6 ED
CMPID-AXP-2DIM	0	0	299	217	60	-89	-127	-174	-285
CMPID-AXP-2DIM	1	300	615	560	420	300	250	220	200
CMPID-AXP-2DIM	2	263	584	530	395	283	205	159	132
CMPID-AXP-2DIM	3	226	553	500	370	266	160	98	64
CMPID-AXP-2DIM	4								4
CMPID-AXP-2D									72
CMPID-AXP-2D									40
CMPID-AXP-2D									08
CMPID-AXP-2D									76
CMPID-AXP-2D									44
CMPID-AXP-2D									12
CMPID-AXP-2DIM	11	-70	305	260	170	130	-200	-390	-480
CMPID-AXP-2DIM	12	-107	274	230	145	113	-245	-451	-548
CMPID-AXP-2DIM	13	-144	243	200	120	96	-290	-512	-616
CMPID-AXP-2DIM	14	-181	212	170	95	79	-335	-573	-684
CMPID-AXP-2DIM	15	-218	181	140	70	62	-380	-634	-752

Two-dimension weights for AXP use the columns to represent phone edit distance and the rows to represent address 'calculated' edit distance. The Index column for address works like the wgt1dim (0=missing, 1=exact, 2=one edit, etc.).

Figure 5-17. Two dimensional weights – wgt2dim ZZ7801.0

Notes:

This 2-dimensional weight table is used for address and phone. The columns are used to represent phone edit distance and the rows to represent address 'calculated' edit distance. The Index column for address works like the wgt1dim (0=missing, 1=exact, 2=one edit, etc.).

Two dimensional weights – wgt2dim

- In class, this table will store the final weights for AXP

Comparison Type	Index	Ph. Missing	Ph. Exact	Ph. 1 ED	Ph. 2 ED	Ph. 3 ED	Ph. 4 ED	Ph. 5 ED	Ph. 6 ED
CMPID-AXP-2DIM	0	0	299	217	60	-89	-127	-174	-285
CMPID-AXP-2DIM	1	300	615	560	420	300	250	220	200
CMPID-AXP-2DIM	2	263	5				05	159	132
CMPID-AXP-2DIM	3	226	5				60	98	64
CMPID-AXP-2DIM	4	189	5				15	37	-4
CMPID-AXP-2DIM	5	152	4				70	-24	-72
CMPID-AXP-2DIM	6	115	460	410	295	215	25	-85	-140
CMPID-AXP-2DIM	7	78	429	380	270	198	-20	-146	-208
CMPID-AXP-2DIM	8	41	398	350	245	181	-65	-207	-276
CMPID-AXP-2DIM	9	4	367	320	220	164	-110	-268	-344
CMPID-AXP-2DIM	10	-33	336	290	195	147	-155	-329	-412
CMPID-AXP-2DIM	11	-70	305	260	17				0
CMPID-AXP-2DIM	12	-107	274	230	14				8
CMPID-AXP-2DIM	13	-144	243	200	12				6
CMPID-AXP-2DIM	14	-181	212	170	95	79	-335		-664
CMPID-AXP-2DIM	15	-218	181	140	70	62	-380	-634	-752

Score when both phone and address are exact match

Score when both phone and address are totally different

Figure 5-18. Two dimensional weights – wgt2dim ZZ7801.0

Notes:

You can see the highest score where both phone and address are exact match. And you will also see score when both phone and address are totally different.

Two dimensional weights – wgt2dim

- In class, this table will store the final weights for AXP

Comparison Type	Index	Ph. Missing	Ph. Exact	Ph. 1 ED	Ph. 2 ED	Ph. 3 ED	Ph. 4 ED	Ph. 5 ED	Ph. 6 ED
CMPID-AXP-2DIM	0	0	299	217	60	-89	-127	-174	-285
CMPID-AXP-2DIM	1	300	615	560	420	300	250	220	200
CMPID-AXP-2DIM	2	263	584	530	395	283	205	159	132
CMPID-AXP-2DIM	3	226	553	500	370	266	160	98	64
CMPID-AXP-2DIM	4	189	522	470	345	249	115	37	-4
CMPID-AXP-2DIM	5	152	491	440	320	232	70	-24	-72
CMPID-AXP-2DIM	6	115	460	410	295	215	25	-85	-140
CMPID-AXP-2DIM	7	78	429	380	270	198	-20	-146	-208
CMPID-AXP-2DIM	8	41	398					07	-276
CMPID-AXP-2DIM	9	4	367					68	-344
CMPID-AXP-2DIM	10	-33	336					29	-412
CMPID-AXP-2DIM	11	-70	305					390	-480
CMPID-AXP-2DIM	12	-107	274	230	145	113	-245	-451	-548
CMPID-AXP-2DIM	13	-144	243	200	120	96	-290	-512	-616
CMPID-AXP-2DIM	14	-181	212	170	95	79	-335	-573	-684
CMPID-AXP-2DIM	15	-218	181	140	70	62	-380	-634	-752

Numbers should continually decrease in all directions from the highest score

Figure 5-19. Two dimensional weights – wgt2dim ZZ7801.0

Notes:

Numbers should continually decrease in all directions from the highest score

The flow here is that as you move down and to the right, the values should become smaller.

(and possibly move into negative)

String based weights – wgtsval

- In class, this table will store weights for SEX, NAME, and AXP (address & phone)

Comparison Type	Index	Weight
CMPID-NAME-XACT	R	115
CMPID-NAME-XACT	L	133
CMPID-NAME-XACT	N	133
CMPID-NAME-XACT	X	267
CMPID-NAME-XACT	CHRIS	316
CMPID-NAME-XACT	JOHN	318
CMPID-NAME-XACT	JENNIFER	333
CMPID-NAME-XACT	BRITTANY	339
CMPID-NAME-XACT	LUPE	352
CMPID-NAME-XACT	DARIUS	354
CMPID-NAME-XACT	a	396
CMPID-NAME-XACT	d	-100
CMPID-NAME-PARM	__FULLNAME_MAXWGT	594
CMPID-NAME-PARM	__NORM_MCCIDX_EQUAL	20

Comparison Type	Index	Weight
CMPID-AXP-XACT	AZ	145
CMPID-AXP-XACT	ST	181
CMPID-AXP-XACT	AVE	360
CMPID-AXP-XACT	RD	465
CMPID-AXP-XACT	JUNCTION	471
CMPID-AXP-XACT	CREEK	557
CMPID-AXP-XACT	BLVD	566
CMPID-AXP-XACT	DR	566
CMPID-AXP-XACT	HILLS	597
CMPID-AXP-XACT	WEST	598
CMPID-AXP-XACT	AFB	617
CMPID-AXP-XACT	PARK	633
CMPID-AXP-XACT	a	883
CMPID-AXP-PARM	__ADDR_STREET_MAXWGT	3000

Figure 5-20. String based weights – wgtsval ZZ7801.0

Notes:

These are examples for string based weights that were created during weight generation that are stored in the mpi_wgtsval table.

String based weights – wgtsval

- In class, this table will store weights for SEX, NAME, and AXP (address & phone)

Comparison Type	Index	Weight
CMPID-NAME-XACT	R	115
CMPID-NAME-XACT	L	133
CMPID-NAME-XACT	N	133
CMPID-NAME-XACT	X	267
CMPID-NAME-XACT	CHRIS	316
CMPID-NAME-XACT	JOHN	318
CMPID-NAME-XACT	JENNIFER	333
CMPID-NAME-XACT	BRITTANY	339
CMPID-NAME-XACT	LUPE	352
CMPID-NAME-XACT	DARIUS	354
CMPID-NAME-XACT	a	396
CMPID-		33
CMPID-		33
CMPID-		00

Comparison Type	Index	Weight
CMPID-AXP-XACT	AZ	145
CMPID-AXP-XACT	ST	181
CMPID-AXP-XACT	AVE	360
CMPID-AXP-XACT	RD	465
CMPID-AXP-XACT	JUNCTION	471
CMPID-AXP-XACT	CREEK	557
CMPID-AXP-XACT	BLVD	566
CMPID-AXP-XACT	DR	566
CMPID-AXP-XACT	HILLS	597
CMPID-AXP-XACT	WEST	598
CMPID-AXP-XACT	AFB	617

XACT weights are frequency-based, so the more common a value is, the lower the score. The more rare a value is, the higher the score.

Figure 5-21. String based weights – wgtsval ZZ7801.0

Notes:

XACT weights are frequency-based, so the more common a value is, the lower the score. The more rare a value is, the higher the score.

String based weights – wgtsval

- In class, this table will store weights for SEX, NAME, and AXP (address & phone)

Comparison Type	Index	Weight
CMPID-NAME-XACT	R	115
CMPID-NAME-XACT	L	133
CMPID-NAME-XACT	N	133
CMPID-NAME-XACT		
CMPID-NAME-XACT		
CMPID-NAME-XACT		
CMPID-NAME-XACT		
CMPID-NAME-XACT		
CMPID-NAME-XACT	DARIUS	354
CMPID-NAME-XACT	a	396
CMPID-NAME-XACT	d	-100
CMPID-NAME-PARM	__FULLNAME_MAXWGT	594
CMPID-NAME-PARM	__NORM_MCCIDX_EQUAL	20

Comparison Type	Index	Weight
CMPID-AXP-XACT	AZ	145
CMPID-AXP-XACT	ST	181
CMPID-AXP-XACT	AVE	360
		465
	ON	471
	K	557
		566
		566
	LS	597
CMPID-AXP-XACT	WEST	598
CMPID-AXP-XACT	AFB	617
CMPID-AXP-XACT	PARK	633
CMPID-AXP-XACT	a	883
CMPID-AXP-PARM	__ADDR_STREET_MAXWGT	3000

Default agree weights are used when a value is encountered that does not have a preset weight score. The ‘a’ weight is the highest score, because the value is very rare.

Figure 5-22. String based weights – wgtsval ZZ7801.0

Notes:

Default agree weights are used when a value is encountered that does not have a preset weight score. The ‘a’ weight is the highest score, because the value is very rare.

Your weights operate on the 80/20 rule. When the hub calculates weights, it typically only generates specific weight values for the most common values in the population up to the point where 80% of the population is accounted for. The remaining 20% of the population have the rarest values and therefore can be given the same “default” weight score which is the highest possible weight for that attribute/token.

Remember, the more rare the value, the higher its weight. Therefore, the values that are most rare all receive the highest possible weight score. The beauty in this approach is that when a distinctly new value is added to the hub, it can still be assigned a weight value, even though the system has never seen that value before.

String based weights – wgtsval

- In class, this table will store weights for SEX, NAME, and AXP (address & phone)

Comparison Type	Index	Weight
CMPID-NAME-XACT	R	115
CMPID-NAME-XACT	L	133
CMPID-NAME-XACT	N	133
CMPID-NAME-XACT	X	267
CMPID-NAME-XACT	CHRIS	316
CMPID-NAME-XACT	JOHN	318
CMPID-NA		
CMPID-NA		
CMPID-NA		
CMPID-NA		
CMPID-NAME-XACT	a	396
CMPID-NAME-XACT	d	-100
CMPID-NAME-PARM	__FULLNAME_MAXWGT	594
CMPID-NAME-PARM	__NORM_MCCIDX_EQUAL	20

Comparison Type	Index	Weight
CMPID-AXP-XACT	AZ	145
CMPID-AXP-XACT	ST	181
CMPID-AXP-XACT	AVE	360
CMPID-AXP-XACT	RD	465
CMPID-AXP-XACT	JUNCTION	471
CMPID-AXP-XACT	CREEK	557
	VD	566
	R	566
	LS	597
	ST	598
CMPID-AXP-XACT	AFB	617
CMPID-AXP-XACT	PARK	633
CMPID-AXP-XACT	a	883
CMPID-AXP-PARM	__ADDR_STREET_MAXWGT	3000

PARM weights issue bonus points, put caps on maximum scores, and indicate the penalties to subtract for edit distance, nicknames, and phonetics.

Figure 5-23. String based weights – wgtsval ZZ7801.0

Notes:

PARM weights issue bonus points, put caps on maximum scores, and indicate the penalties to subtract for edit distance, nicknames, and phonetics

Numeric based weights – wgtnval

- In class, this table will store exact match weights for DOB:

Comparison Type	Index	Weight
CMPID-DOB-YEAR	1900	342
CMPID-DOB-YEAR	1899	386
CMPID-DOB-YEAR	1898	398
CMPID-DOB-YEAR	1897	419
CMPID-DOB-YEAR	1901	423
CMPID-DOB-YEAR	1896	425
CMPID-DOB-YEAR	1895	445
CMPID-DOB-YEAR	1894	463
CMPID-DOB-YEAR	2001	463
CMPID-DOB-YEAR	1954	473
CMPID-DOB-YEAR	1955	473
CMPID-DOB-YEAR	1977	475
CMPID-DOB-YEAR	2003	475
CMPID-DOB-YEAR	-1	477

Figure 5-24. Numeric based weights – wgtnval ZZ7801.0

Notes:

Now let's take a look of an example of numeric based weights that are stored in the mpi_wgtnval table.

Numeric based weights – wgtnval

- In class, this table will store exact match weights for DOB:

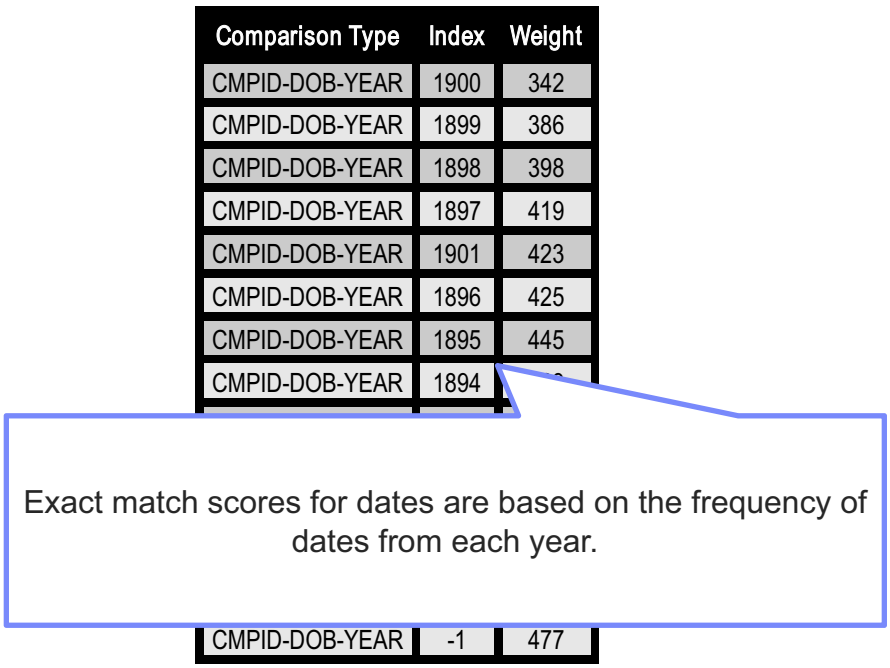


Figure 5-25. Numeric based weights – wgtnval ZZ7801.0

Notes:

Notice this weight table will store exact match weights for DOB. Exact match scores for dates are based on the frequency of dates from each year.

Numeric based weights – wgtnval

- In class, this table will store exact match weights for DOB:

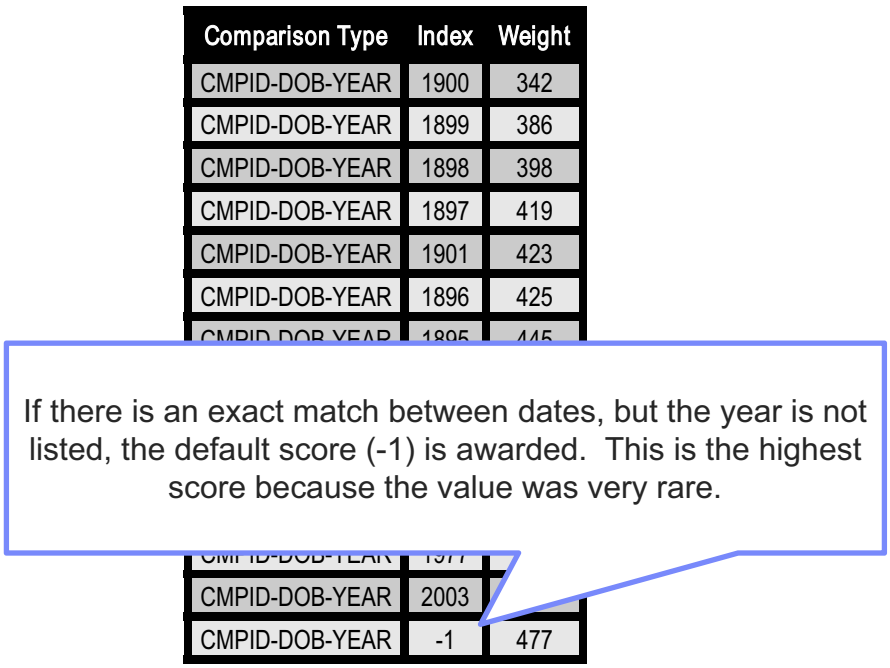


Figure 5-26. Numeric based weights – wgtnval ZZ7801.0

Notes:

If there is an exact match between dates, but the year is not listed, the default score (-1) is awarded. This is the highest score because the value was very rare.

The 80/20 weight rule

- 80% of values (most common) get weighed, 20% (rarest) are not
  - Common values get own weight scores,
  - Rarest values use a default score
  - Gives illusion of a learning weighting strategy, but weights are static
  - If we had score for every distinct value, list would be unwieldy and slow to query
  - The 80/20 rule is applied to frequency-based weights

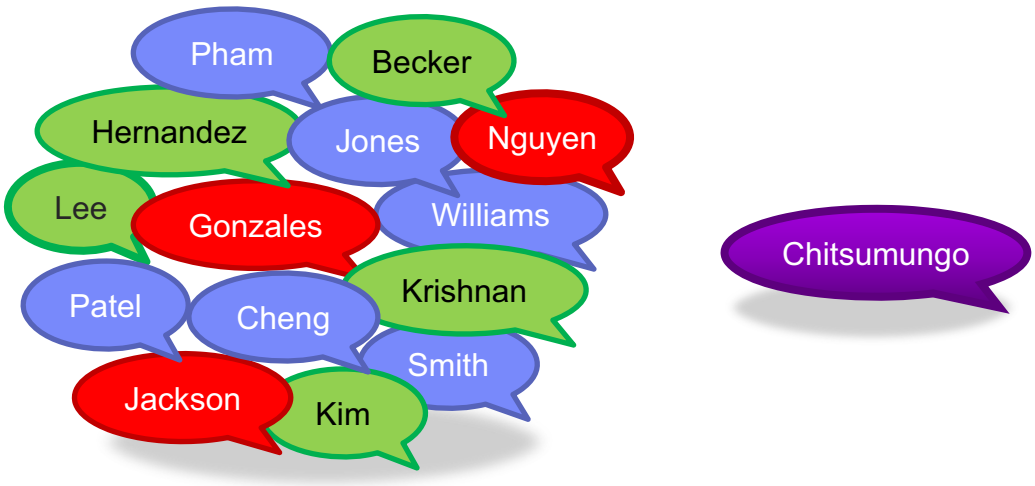


Figure 5-27. The 80/20 weight rule ZZ7801.0

Notes:

Your weights operate on the 80/20 rule. When the InfoSphere MDM calculates weights, it typically only generates specific weight values for the most common values in the population up to the point where 80% of the population is accounted for. The remaining 20% of the population have the rarest values and therefore can be given the same “default” weight score which is the highest possible weight for that attribute/token.

Remember, the more rare the value, the higher its weight. Therefore, the values that are most rare all receive the highest possible weight score. The beauty in this approach is that when a distinctly new value is added to the InfoSphere MDM, it can still be assigned a weight value, even though the system has never seen that value before.

When should you recalculate weights?

- No hard rule, but guidelines:
  - Added or changed comparison function or comparison code in Algorithm
  - Population has grown by more than 20% of original population (remember the 80/20 rule… now you have different 20%)
  - 2 years since last calculated weights
  - Added new source
  - Upgrading to new version of InfoSphere MDM

Figure 5-28. When should you recalculate weights? ZZ7801.0

Notes:

There is no set rule that dictates when you need to recalculate your weights, but there are a few scenarios that merit running weights again. That said, you might find that the change in weights was so negligible that you can continue using your old weights.

Reasons to check for updated weights:

- If you have added or changed comparison function or comparison code in your Algorithm.
- When your population has grown by more than 20% of the original population (remember the 80/20 rule… now you have a different 20%).
- It has been 2 years since you last calculated weights.
- When you have added a new source, especially one that comes from a different geographical area (the East Coast has different name, phone and address distributions than the West Coast).
- When you are upgrading to a new version of the InfoSphere MDM

The magical weight formula

- Based on mathematical & statistical theory
- If R is the result of a comparison function, then the weight for R is:
  Log10(Probability(R in matched pairs)/Probability(R in unmatched pairs))
- 3 things needed calculate formula:
  - **Matched Pairs:** pairs of records are brought together because of shared key criteria
  - **Unmatched Pairs:** pairs of record are selected at random, using no logic
  - **Population Frequency:** simple number of occurrences divided by total population

	Member A	Member B
Name	Ernest L. Johns	Ernest L. Johns
DOB	1932-03-26	1932-03-26
SSN	810-78-1206	810-78-1206
Phone	312.445.2343	708.293.7093
Gender	Male	Male

	Member A	Member B
Name	John P. O'Sullivan	Paula K. Ophelio
DOB	1972-10-05	1962-07-12
SSN	182-12-0727	399-62-8913
Phone	312.729.8166	773.901.1481
Gender	Male	Female

Figure 5-29. The magical weight formula ZZ7801.0

Notes:

The process of calculating weights is based on solid mathematical and statistical research. Our data scientists have leveraged empirical research and theory to create the formulas that measure and calculate weight. The basic weight generation formula is shown below.

If R is the result of a comparison function, then the weight for R is:

Log10(Probability(R in matched pairs)/Probability(R in unmatched Pairs))

Matched Pairs are sets of records that likely represent the same person. The process is not concerned with duplicates or linkages at this time and matched pairs do not need to find an exact match on all attributes since it uses the probabilistic algorithm. Notice we have a sample matched pair on the bottom left.

Unmatched Pairs are sets of records that are pulled randomly from the database. Because they are random, it is most likely that they will not match, although it is possible they will. Each attribute will be compared for agreement and frequency.

Notice we have a sample pair on the bottom right.

Gathering and assessing matched pairs

- **Gathering:** Attribute pairs might be pulled by using the other attributes that match
  - Pairs not gathered because the attribute matches, the attribute is ignored during selection
  - E.g. Name is used, selection is done using DOB, Gender, Address and SSN
- **Agreement:** Once pairs are gathered, simple question asked: **Does the Attribute agree?**
  - Tally: how many matched pairs 'agreed' on attribute
  - Actual values not important (in this step), simply whether or not pair agrees
- **Usage:** Matched pair statistics used as confidence indicator in weight formula
  - Not all pairs end up matching, because of incomplete, out-of-date, or inconsistent data
  - Attributes that are well populated and relatively accurate will have >90% agreement
  - Error-filled or sparsely populated attributes have lowered weight scores

Figure 5-30. Gathering and assessing matched pairs ZZ7801.0

Notes:

Matched pairs generate in multiple sets, focusing on a single attribute, like Name:

- **Gathering:** Name pairs might be pulled by DOB, SSN, Address, and Gender match
  - Pairs are not gathered because the names match, names are ignored during selection
  - If measuring a different attribute, like Gender, other attributes are used to gather pairs
- **Agreement:** Once pairs are gathered, a simple question is asked: **Do the names agree?**
  - A tally of how many matched pairs ended up 'agreeing' on name is kept
  - The actual values are not important (in this step) simply whether or not the pair agrees
- **Usage:** Matched pair statistics are used as a confidence indicator in the weight formula
  - Not all pairs end up matching, because of incomplete, out-of-date, or inconsistent data
  - Attributes that are well populated and relatively accurate will have >90% agreement
  - Error-filled or sparsely populated attributes have lowered weight scores

Gathering and assessing unmatched pairs

- **Gathering:** Because pairs are random, selected very quickly
  - Possible to randomly select pairs that match, but not likely
  - Larger the data set, the better the statistics
  - Usually 5,000,000 random pairs gathered, regardless of data size
- **Agreement:** Once pairs are pulled, the same question is asked: **Does the data agree?**
  - Tally: how many matched pairs 'agreed' on attribute
  - Statistically speaking, usually the unmatched agreement rate = (population frequency)^2
- **Usage:** Unmatched pair statistics represent odds of running across random matches
  - Unmatched pairs rarely agree, unless attribute has few choices (like Gender)

Figure 5-31. Gathering and assessing unmatched pairs ZZ7801.0

Notes:

Unmatched pairs also generate in multiple sets, focusing on a single attributes:

- **Gathering:** Because the pairs are random, they are selected very quickly
  - It is possible to randomly select pairs that match, but not likely
  - The larger the data set, the better the statistics – records may be used in multiple pairs
  - Usually 5,000,000 random pairs are gathered, regardless of sample data size
- **Agreement:** Once pairs are pulled, the same question is asked: **Does the data agree?**
  - A tally of how many unmatched pairs ended up 'agreeing' on name is kept
  - Statistically speaking, usually the unmatched agreement rate = (population frequency)^2
- **Usage:** Unmatched pair statistics represent the odds of running across random matches
  - Unmatched pairs rarely agree, unless the attribute has few choices (like Gender)

Calculating population frequency

- Simplest of weight calculation components:
- **Gathering:** individual value for attribute counted, then divided by total records
  - Simple process, but done on very large scale
  - When calculating name frequencies, every First, Middle, and Last name counted
    - How many John values?
    - How many Gupta values?
    - How many Lee values? (includes first, middle and last counts)
    - How many X values? (could be a middle initial, suffix, etc…)
- **Usage:** Population frequency stats used to get information about specific values
  - Other factors in score (matched and unmatched pairs) were generic stats
  - To know how much 'John' should weigh, we use population frequency
  - High population frequency leads to lower weight scores
  - Low population frequency leads to higher weight scores

Figure 5-32. Calculating population frequency ZZ7801.0

Notes:

Population frequency is the simplest of the three weight calculation components:

Gathering: Each individual value for the attribute is counted, then divided by total records

This is a simple process, but done on a very large scale

When calculating name frequencies, every First, Middle, and Last name is counted

How many John values?

How many Gupta values?

How many Lee values? (includes first, middle and last counts)

How many X values? (could be a middle initial, suffix, etc…)

Usage: The population frequency stats are used to get information about specific values

The other factors in the score (matched and unmatched pairs) were generic stats

To know how much 'John' should weigh, we use the population frequency

High population frequency leads to lower weight scores

Seeing the weight formula in action

- **log10(Population Frequency*Matched Agree Rate/Unmatched Agree Rate)**

	Smith	Gonzales	Chitsumungo
Value Frequency	92,982	6,372	39
Total Population	1,000,000	1,000,000	1,000,000
Population Frequency	**0.09298**	**0.00637**	**0.00004**
Unmatched Pairs Agreeing	43,228	203	8
Unmatched Pairs	5,000,000	5,000,000	5,000,000
Unmatched Agree Rate	**0.008645652**	**0.000040602**	**0.000000002**
Matched Pairs Agreeing	4,628,101	4,628,101	4,628,101
Matched Pairs	5,000,000	5,000,000	5,000,000
Matched Agree Rate	**0.9256**	**0.9256**	**0.9256**
Result	**0.998**	**2.162**	**4.375**

Figure 5-33. Seeing the weight formula in action ZZ7801.0

Notes:

The simplest form of weight calculations are frequency-based weights. Below is an example that uses three names and shows that, in a given population, these names will have weights that are inversely proportionate to their frequency. Remember, not all weights are calculated this way. Let us take the weight formula and boil it down to its most elemental parts:

log10(Population Frequecy*Matched Agree Rate/Unmatched Agree Rate)

which can be calculated as **Log 10(C * A/B)**.

A = Population Frequency: A is the number of times that a value occurs in population divided by the total population. So, if 9 of 100 people had the name Smith then 9%.

B = Unmatched Frequency: B is the likelihood that two random members would have the name Smith. This would be 9% of 9% of the random pairs or 0.81%.

C = Matched Frequency: C is the actual number of records that agreed (not just the Smiths, but simply had the same name) in a set of matched pairs. These matched pairs are identified by the algorithm in a similar process to the way you generated threshold analysis sample pairs. Typically, this is a high number, between 85-99%. In our example below, 92.6% of matched pairs agreed.

Probability(R in matched pairs) or C * A: The probability that two members have the same name is dependent upon two things: 1) how often does the name Smith appear in the general population and 2) what percentage of matched pairs have the same value for the name (some records might have empty or wrong information). When we multiply these two elements, we get the probability that two matched records would be named Smith (92.56% * 0.81% = 0.75%).

Probability(R in unmatched pairs) or B: The statistical likelihood that two random records pulled from your database would have the value you are measuring is based on how common that value is. If 9% of the population has the name Smith, then each of the two members in the random pairing have a 9% chance of being named Smith. So, to use it in the formula, the unmatched probability is the “square” of the value's population frequency or 0.81%.

Seven steps of weight generation

Weight generation steps can be run individually or in sequence:

Step 1	Delete artifacts from previous run
Step 2	Generate counts for all attribute values. This step calculates the population frequency statistics
Step 3	Generate random pairs of members (usually 5,000,000 unmatched pairs)
Step 4	Derive random data by comparing random
Step 5	Perform matched candidate pairs.
Step 6	Generate matched set, matched agreement statistics, and initial weights
Step 7	Iterate over previous step and check for convergence of weights

Figure 5-34. Seven steps of weight generation ZZ7801.0

Notes:

There are seven steps to weight generation.

Step 1

Delete artifacts from previous run (removes the contents of the weights folder in the ‘work’ directory on the server)

Step 2

Generate counts for all attribute values (uses the Generate Frequency Stats (mpxfreq) utility). This step calculates the population frequency statistics

Step 3

Generate random pairs of members (usually 5,000,000 unmatched pairs)

Step 4

Derive random data by comparing random members (builds multiple sets of pairs, by attribute analyzed)

Step 5

Perform matched candidate pairs reduction (similar to Threshold pairs process). This step calculates the unmatched agreement rate statistics.

Step 6

Generate matched set, matched agreement statistics, and initial weights

Step 7

Iterate over previous step and check for convergence of weights

Troubleshooting weights

- Bad weights can be caused by wide variety of reasons, here are a few:
  - **Fake data:** test data rarely generates good weights, it does not represent ‘real life’ errors
  - **Input file too small:** Too many matched pairs in random pair sample, or size of the matched set too small for reliable matched set statistics
  - **Duplication rate too small:** Size of matched set too small for reliable matched set statistics.
  - **Duplication rate too large:** Too many matched pairs in random pair sample or from the same source
  - **Match attributes sparsely populated:** not enough pairs for reliable statistics.
  - **Too few match attributes:** if only two attributes, e.g. Name and Address, to derive matched statistics, you would only be able to use Address to lookup matched pairs. You can't derive a good matched set from address alone. 5-7 match attributes are norm.

Figure 5-35. Troubleshooting weights ZZ7801.0

Notes:

Bad weights are most often the result of having bad or insufficient data. In many cases, attempts to create *fake* or *test* data fail to generate weights because they do not represent the real distribution of errors that are found in real life. Most commonly you will generate bad weights if the following conditions exist:

- **Fake data:** test data rarely generates good weights, it does not represent ‘real life’ errors
- **Input file too small:** there may be too many matched pairs in the random pair sample, or the size of the matched set is too small for reliable matched set statistics
- **Duplication rate too small:** the size of the matched set is too small for reliable matched set statistics. Duplication here involves similar records from multiple sources, same-source duplicates, and specific values appearing multiple times)
- **Duplication rate too large:** there may be too many matched pairs in the random pair sample or they are from the same source
- **Match attributes sparsely populated:** not enough pairs for reliable statistics. If an attribute is only populated in 10% of records, only about 1% of matched pairs will have members who both have the attribute.

- **Too few match attributes:** if there are only two attributes, say Name and Address, to derive the matched statistics, you would only be able to use Address to lookup matched pairs. You can't derive a good matched set from address alone. 5-7 match attributes are the norm.

How do you know your weights are bad?

- Some weights have normal patterns (if pattern is off = bad weights):
  - Using Excel, can graph weights to see patterns more easily
- What patterns are we looking for in wgt1dim?
  - **SSN:** Index 1 should be between 4 and 6 – all other numbers should continue to decrease
  - **DOB:** Index 1 should be a 0, wgtnval holds exact match scores, but all other numbers should continue to decrease
  - **AXP 1DIM:** Instead of decreasing from Index 1, the AXP scores should increase as the odds of having the same 6-digit number in two addresses randomly are very rare.
- What patterns are we looking for in wgt2dim?
  - **AXP:** All numbers should decrease from the Exact Phone/Exact Address score (across, down, and diagonally)

Figure 5-36. How do you know your weights are bad? ZZ7801.0

Notes:

Some weights have normal patterns – if the pattern is off, you have bad weights:

- Using Excel, you can graph the weights to see the patterns more easily

What patterns are we looking for in wgt1dim?

- **SSN:** Index 1 should be between 4 and 6 – all other numbers should continue to decrease
- **DOB:** Index 1 should be a 0, wgtnval holds exact match scores, but all other numbers should continue to decrease
- **AXP 1DIM:** Instead of decreasing from Index 1, the AXP scores should increase as the odds of having the same 6-digit number in two addresses randomly are very rare.

What patterns are we looking for in wgt2dim?

- **AXP:** All numbers should decrease from the Exact Phone/Exact Address score (across, down, and diagonally)

Exercise introduction

- In this exercise, you will:
  - Deploy Weights
  - View generated weights

Exercise: continue Generating Weights

Figure 5-37. Exercise introduction ZZ7801.0

Notes:

In this exercise, you will:

- Deploy Weights
- View generated weights

Exercise: continue Generating Weights

Unit summary

Having completed this unit, you should be able to:

- Understand how weights are generated
- Understand how to evaluate weights
- Understand various types of weights
  - Frequency based weights
  - Edit Distance weights
  - Parameterized weights

Figure 5-38. Unit summary ZZ7801.0

Notes:

Having completed this unit, you should be able to:

- Understand how weights are generated
- Understand how to evaluate weights
- Understand various types of weights
  - Frequency based weights
  - Edit Distance weights
  - Parameterized weights

Unit 6. Thresholds

What this unit is about

This unit describes how to create threshold for our PME using the Pair Manager and evaluating the threshold calculator.

What you should be able to do

After completing this unit, you should be able to:

- Understand Bulk Cross Match to analyze weights
- Understand the Enterprise ID assignment
- Understand how to create Thresholds

Unit objectives

After completing this unit, you should be able to:

- Understand Bulk Cross Match to analyze weights
- Understand the Enterprise ID assignment
- Understand how to create Thresholds

Figure 6-1. Unit objectives ZZ7801.0

Notes:

After completing this unit, you should be able to:

- Understand Bulk Cross Match to analyze weights
- Understand the Enterprise ID assignment
- Understand how to create Thresholds

What is a bulk cross match?

- Compares members on file system
  - Dramatically faster than comparing records individually in database
  - Three components to BXM:
    - **Compare Members in Bulk (mpxcomp):** pulls matched records and scores them
    - **Link Entities (mpxlink):** checks scores against thresholds to create entities or tasks
    - **Load UNLs to DB (madunlload):** Loads the entities and tasks into the database
  - Normally run during pre-production implementation exercises

- Takes about same time as Weigh Generation job
  - We will kick off our BXM first, then discuss the BXM process in detail while it runs

Figure 6-2. What is a bulk cross match? ZZ7801.0

Notes:

The Bulk Cross Match (BXM) compares your member records in binary form. The BXM process is made up of two primary jobs; Compare Members in Bulk (mpxcomp) and Link Entities (mpxlink). After running the compare and link, the data will need to be loaded into the database. BXMs are normally only run during pre-production implementation exercises. The BXM takes about the same amount of time as the Weigh Generation job

Exercise introduction

- In this exercise, you will:
  - Run the Bulk Cross Load process

Exercise: Bulk Cross Load

Figure 6-3. Exercise introduction ZZ7801.0

Notes:

In this exercise, you will:

- Run the Bulk Cross Load process

Exercise: Bulk Cross Load

Enterprise ID (EID) assignment

- Member records are assigned an EID, (identifies which Entity it belongs to):
  - If record finds match, EID is shared between records
  - If record is unique, becomes singleton Entity with own EID
  - The BXM assigns EIDs to all member records in dataset

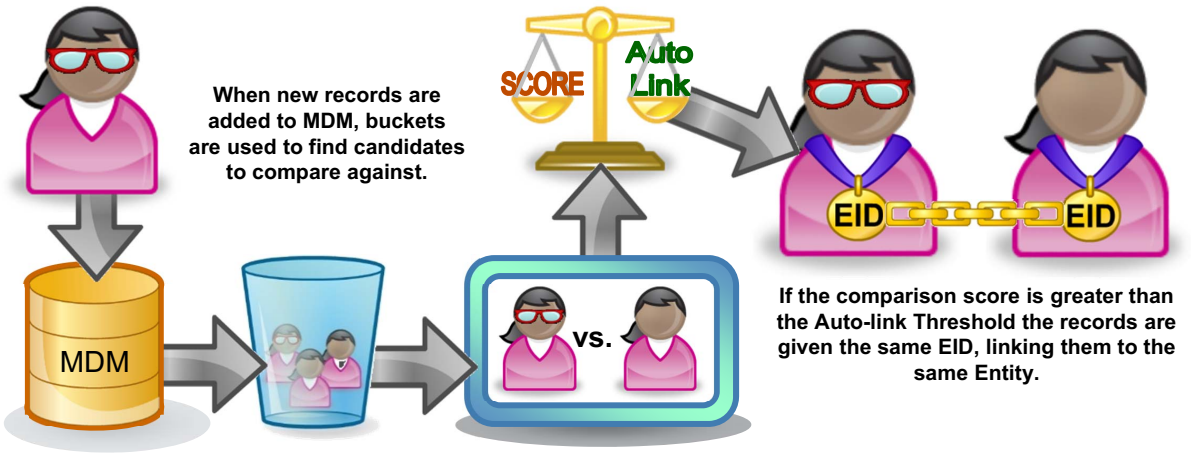


Figure 6-4. Enterprise ID (EID) assignment ZZ7801.0

Notes:

When a member is created in InfoSphere MDM, it is assigned an internal member record number (MemRecno) and an Enterprise ID (EID and/or EntRecno).

When new records are added to InfoSphere MDM, buckets are used to find candidates to compare against, if the comparison score is greater than the Auto-link Threshold the records are given the same EID, linking them to the same Entity. If the record is unique, it becomes a singleton Entity with its own EID. The BXM process will assign EIDs to all of the member records in the dataset.

BXM steps

- The BXM really starts at data derivation and ends with data load

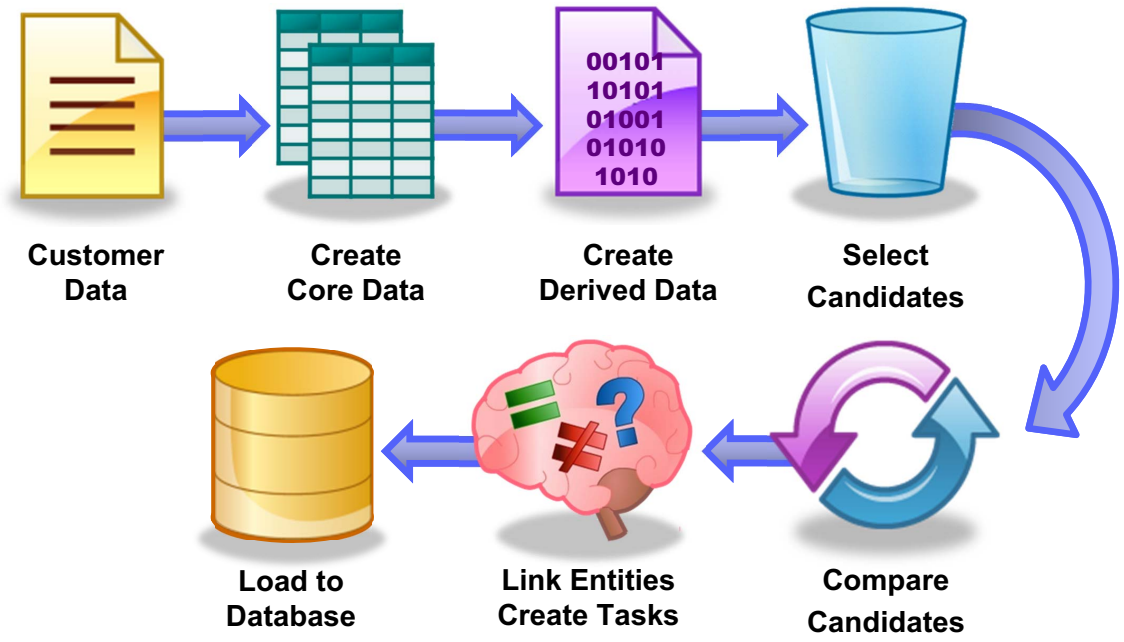


Figure 6-5. BXM steps ZZ7801.0

Notes:

Next we will talk about the BXM steps.

BXM steps – Derive Data

- The BXM process really starts at data derivation then ends with a data load

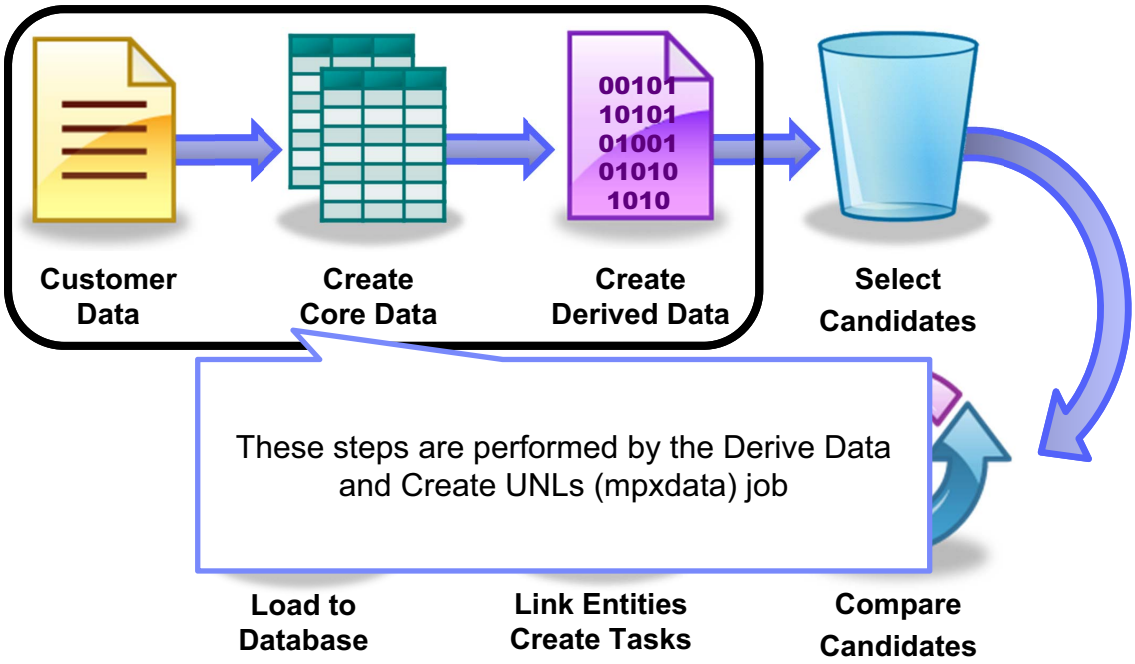


Figure 6-6. BXM steps – Derive Data ZZ7801.0

Notes:

Notice the 3 steps highlighted in the box, these steps are performed by the derive data and create UNLs utility. This is the utility you would typically use for an initial load (starting from scratch). This utility takes the file of customer data, and converts the core data into the InfoSphere MDM data model. For each table in the MDM data model, a new 'unl' text file (mpi_memname.unl, for example) is created which is eventually loaded into the database.

This utility also creates the derived data (buckets, comparison string) as well, and puts the derived data into two formats:

- 'unl' text file, to be loaded into the database later
- Binary files, which will be used in subsequent processing steps

BXM steps – Bulk Compare

- The BXM process really starts at data derivation then ends with a data load

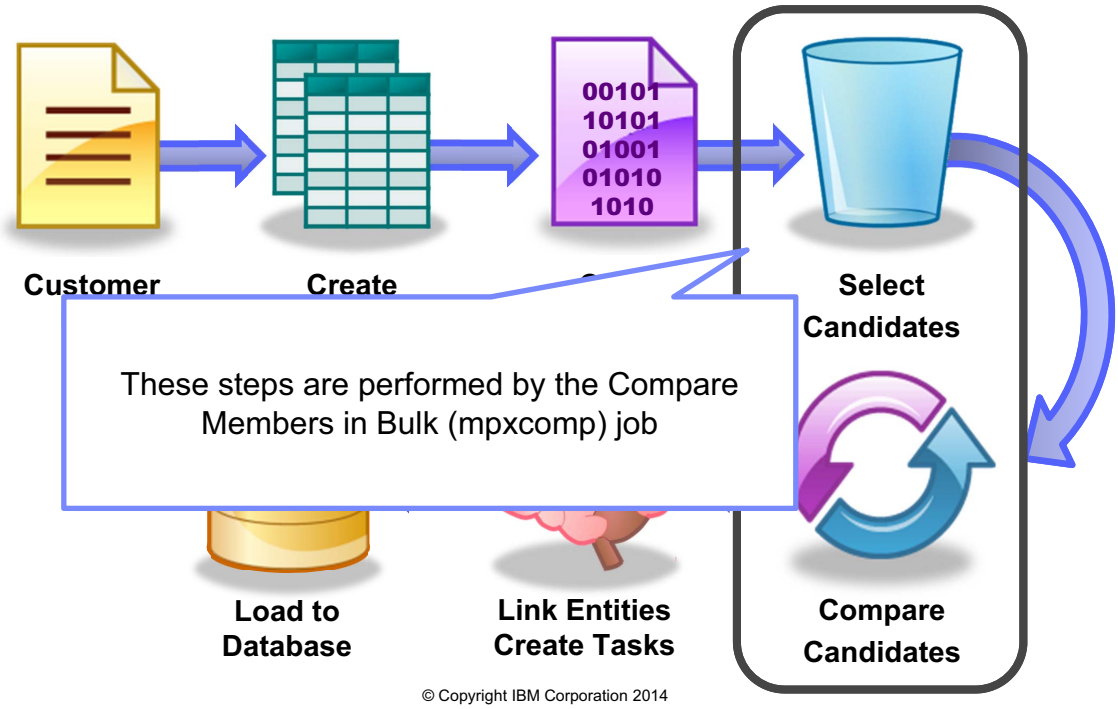


Figure 6-7. BXM steps – Bulk Compare ZZ7801.0

Notes:

Compare Members in Bulk (mpxcomp) utility: This utility iterates through all the buckets (selects candidates) and performs the comparison calculations for all the members in each bucket.

The input is the binary files of derived data (bucket and comparison string binary files) from the previous step. The binary files are read into memory to speed up all the comparison calculations.

The output is additional binary files that represent the entity link and task groupings.

BXM steps – Link Entities

- The BXM process really starts at data derivation then ends with a data load

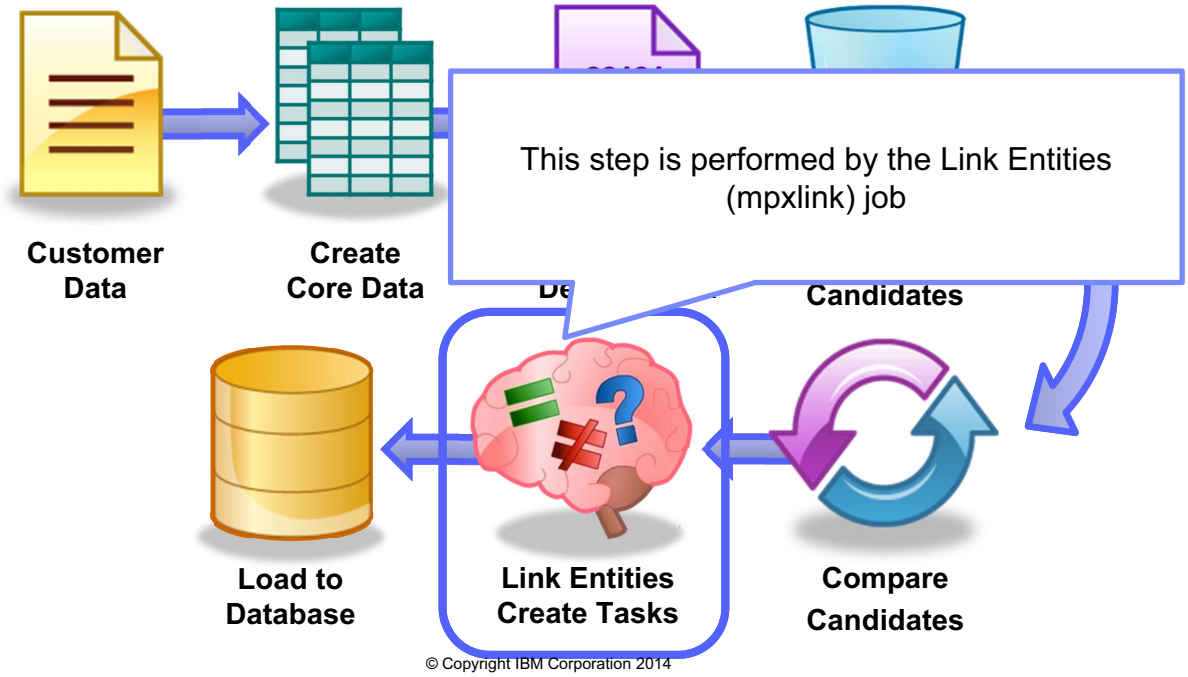


Figure 6-8. BXM steps – Link Entities ZZ7801.0

Notes:

Link Entities (mpxlink) utility: This utility takes the comparison results and creates entity link and task files that can be loaded into the database.

The input is the binary files of comparison results (entity link and task groupings) from the previous step. These files are read into memory for faster processing.

The output is additional ‘unl’ text files that contain the EID assignments (entlink), tasks (enttsk), and EID history (entxeia).

The Compare Members in Bulk (mpxcomp) and Link Entities (mpxlink) utilities must be run once for each type of entity (for example: 'identity' and 'household'), as the outcome will be different per entity type.

BXM steps – Load Database

- The BXM process really starts at data derivation then ends with a data load

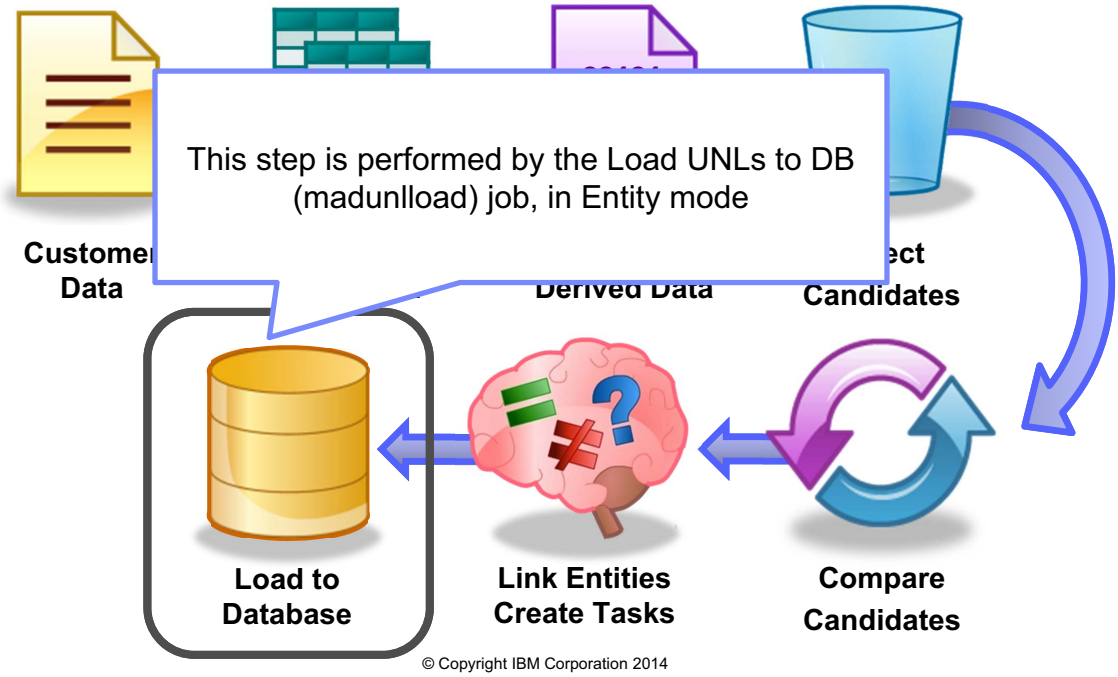


Figure 6-9. BXM steps – Load Database ZZ7801.0

Notes:

Load UNLs to DB (madunlload) utility: This utility takes the 'unl' text files created during previous steps and loads them into the database.

The Load UNLs to DB (madunlload) utility loads the core member data 'unl' files and the derived data 'unl' files (the outputs from the *Create Core Data* and *Create Derived Data* steps).

After the database is loaded, the MDM engine is started and real-time operation begins.

How do you analyze thresholds?

To run threshold analysis, you need to:

1. Run “Threshold Analysis Pair Generation” job
   - Generates pairs that have a weight above a certain limit
2. Answer “Yes,” “No” or “Maybe” to the pairs in Pair Manager
   - Provides manual answer to whether the pairs are a match
3. Run the Threshold Calculator
   - Calculates the weight distribution of matches and non-matches
4. Reach the desired set of Tasks or False Positive Rate for your metrics
   - Using analysis, determine which score you want as a manual task and autolink.
5. Deploy the new Thresholds to the Hub
6. Re-compare the records in your database

Figure 6-10. How do you analyze thresholds? ZZ7801.0

Notes:

To run threshold analysis, you need to:

a. Run “Threshold Analysis Pair Generation” job
   - Generates pairs that have a weight above a certain limit

b. Answer “Yes,” “No” or “Maybe” to the pairs in Pair Manager
   - Provides manual answer to whether the pairs are a match

c. Run the Threshold Calculator
   - Calculates the weight distribution of matches and non-matches

d. Reach the desired set of Tasks or False Positive Rate for your metrics
   - Using analysis, determine which score you want as a manual task and autolink.

e. Deploy the new Thresholds to the Hub

f. Re-compare the records in your database

g. Re-compare the records in your database

Why do you analyze thresholds?

- Determine score where you are confident two records represent the same person
- Scan data sample for False Positives and False Negatives
- Determine threshold where you want to manually review potential duplicates
- Validate algorithm we have configured
- Confirm weights generated make sense
- Identify general data quality issues

Figure 6-11. Why do you analyze thresholds? ZZ7801.0

Notes:

We perform threshold analysis for various reasons:

- Determine the threshold at which you are confident that two records represent the same person
- Determine the threshold at which you want to manually review potential duplicates
- Validate that the algorithm we have configured is producing meaningful matches
- Confirm that the weights we generated make sense
- Identify general data quality issues

False positives and false negatives

- False Positives and False Negatives occur when the “probability” of match does not correlate to actuality of match.
  - False Positives: usually Twins, Siblings, or Parent/Child records
  - False Negative: usually caused by sparse or inaccurate data in records

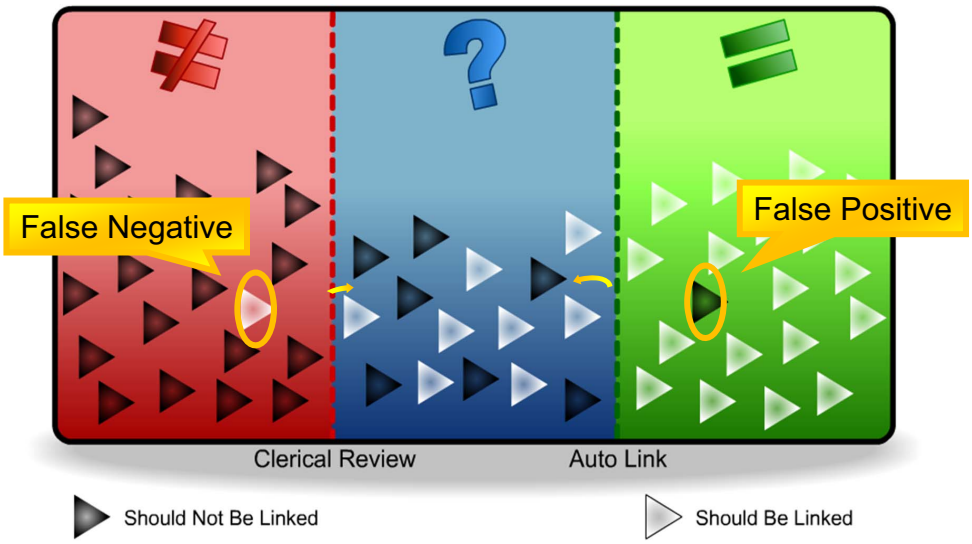


Figure 6-12. False positives and false negatives ZZ7801.0

Notes:

InfoSphere MDM uses set scores to determine what action to take as a result of a comparison. These scores are referred to as thresholds.

The **Autolink (AL) Threshold** is the cutoff score where two members will be linked together. This upper threshold reflects the score at which the organization is confident that a match represents the same person. We set this based on the organization's tolerance for false positives.

The **Clerical Review (CR) Threshold** is the cutoff score where two members will be assigned a task for manual review by a user. This lower threshold reflects the score at which the organization wants to manually review matches. We set this based both on the organization's tolerance for false negatives, and on the number of matches they are willing to review based on the cost of manually reviewing these tasks.

Members whose scores fall below the CR threshold will not be linked or assigned to a task - it is assumed that these members are not related given the information currently available. If the member records are updated causing the scores to improve, then InfoSphere MDM might assign them to a task or link them.

Let us learn more about the possible matching errors that can take place based on the available information.

The accuracy of our solution is measured in terms of two potential types of errors and we establish the two thresholds to minimize both types of errors: **False Negatives and False Positives.**

- Auto Link minimizes False Positive matches. These occur when you match two records that do not represent the same person.
- Clerical Review minimizes False Negative matches. These occur when you fail to match two records that represent the same person.

False negatives

The empty arrows on the left are members that *should be linked* but are not because of insufficient data. These are False Negatives. If all that is known is a name, then there is not enough data to make a sound linking judgment. False negatives can possibly be reduced by tweaking the algorithm, but are most commonly the result of poor data collection.

False positives

The dark arrow on the right that *should not be linked* is the result of data that is very similar; usually a set of twins, or parent and child records. This is a False Positive. There are several ways to deal with false positives. Most commonly, we use the False Positive Filter to issue weight penalties for subtle differences between records.

Matched pair review

- To help set Thresholds, review Matched Pairs
  - Matched pairs consist of roughly 4,000 pairs (10 pairs per .1 points)
  - Matched Pairs can be reviewed through Excel or Pair Manager
  - Customers indicate Yes, No, or Maybe to confirm if pairs match

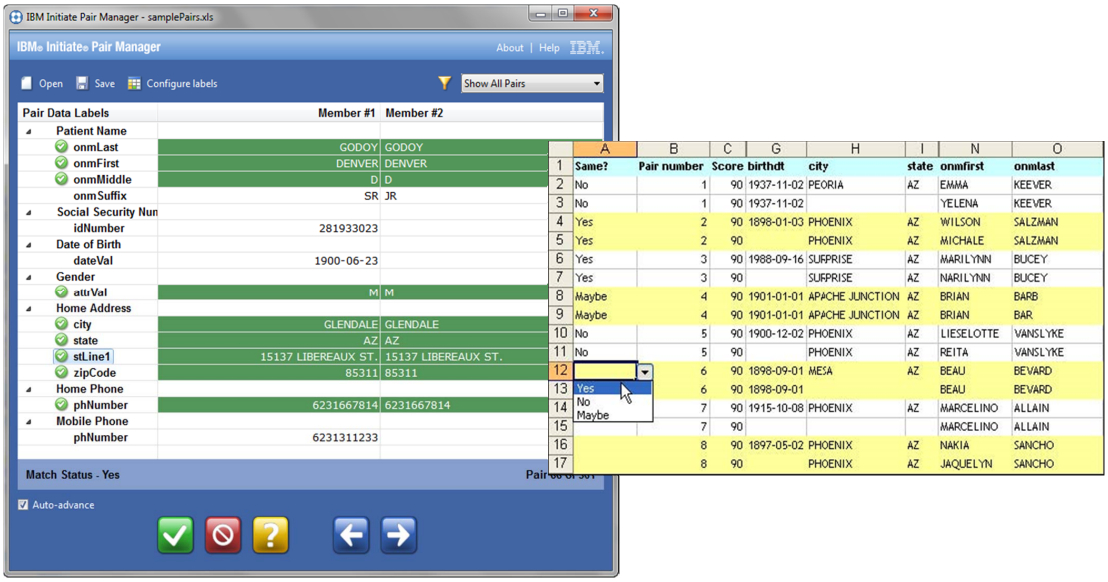


Figure 6-13. Matched pair review ZZ7801.0

Notes:

To help set Thresholds, we have Customers review Matched Pairs

- Matched pairs consist of roughly 4,000 pairs (10 pairs per .1 points)
- Matched Pairs can be reviewed through Excel or Pair Manager
- Customers indicate Yes, No, or Maybe to confirm if the pairs match

Threshold calculation

- Matched Pairs Review results used to calculate Thresholds
  - **False Positive Rate**
    Goal: <1 in 1 million
  - **False Negative Rate**
    Goal: <1 in 10 thousand
  - **Estimated # of Tasks**
    Dependent on Resources
  - **# of Common Attributes**
    Sources with fewer attributes cannot score as high as sources with more attributes

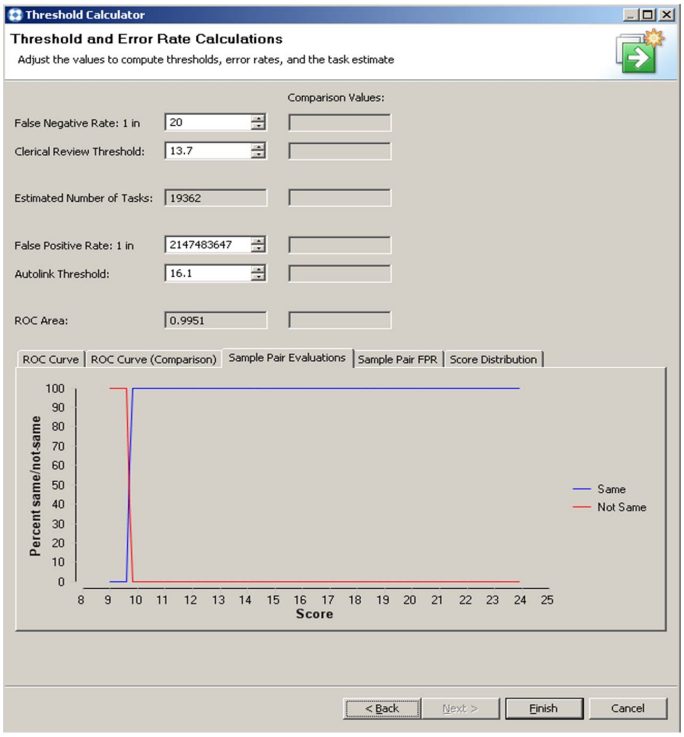


Figure 6-14. Threshold calculation ZZ7801.0

Notes:

Once they have evaluated and scored the sample matches, we can make algorithm improvements and perform another round of threshold analysis. We then use the organization's evaluations to determine the appropriate thresholds.

- The goal here is to have
  - <1 in 1 million False Positive Rate
  - <1 in 10 thousand False Negative Rate
  - Estimated # of Tasks Dependent on Resources
  - # of Common Attributes Sources with fewer attributes cannot score as high as sources with more attributes

Exercise introduction

- In this exercise, you will:
  - Generate some sample Pairs
  - Run Pair Manager to evaluate pairs
  - Run Threshold Calculator
  - Update Thresholds
  - Redeploy configuration

Exercise: Pair Manager

Figure 6-15. Exercise introduction ZZ7801.0

Notes:

In this exercise, you will:

- Generate some sample Pairs
- Run Pair Manager to evaluate pairs
- Run Threshold Calculator
- Update Thresholds
- Redeploy configuration

Exercise: Pair Manager

Reiterating the process

- Reiteration involves applying same steps we've already gone through
  - **Deploy hub configuration:** Deploy new anonymous strings, algorithm settings, and thresholds to server
  - **Re-derive data:** Regenerate comparison strings, buckets, and binary files from existing sample data
  - **Load new comparison strings & buckets:** Load UNLs to add newly re-derived data to database
  - **Bulk cross match:** Re-run BXM process, making sure to load the new tasks that exist now that our thresholds are 10 and 15
  - **Analyze and review:** Check the analytics again to verify entity composition and bucket sizing

Figure 6-16. Reiterating the process ZZ7801.0

Notes:

Reiteration involves applying several of the same steps we've already gone through

- **Deploy hub configuration:** Deploy the new anonymous strings, algorithm settings, and thresholds to the server
- **Re-derive data:** Regenerate comparison strings, buckets, and binary files from the existing sample data
- **Load new comparison strings & buckets:** Run another load UNLs job to add the newly re-derived data to the database
- **Bulk cross match:** Re-run the BXM process, making sure to load the new tasks that exist now that our thresholds are 10 and 15
- **Analyze and review:** Check the analytics again to verify entity composition and bucket sizing

Entity Analytics Options

- After matches have occurred we can analyze our Entities
  - **Entities By Size:** list of entities by their size
  - **Entity Composition:** details from the memcmpd table for each member in entity
  - **Entity Size Distribution:** chart showing number of entities by member count
  - **Member Comparisons:** interface for comparing members and seeing score
  - **Member Entity Frequency:** chart showing number of entities that members belong to
  - **Member Entity Values:** list showing entities that specific members belong to
  - **Members By Entity Count:** list showing members and how many entities they belong to
  - **Score Distribution:** chart showing count of entities linked at certain comparison scores

Figure 6-17. Entity Analytics Options ZZ7801.0

Notes:

There are a lot entity analysis queries available in the workbench.

- **Entities By Size:** list of entities by their size
- **Entity Composition:** details from the memcmpd table for each member in an entity
- **Entity Size Distribution:** chart showing number of entities by member count
- **Member Comparisons:** interface for comparing members and seeing score details
- **Member Entity Frequency:** chart showing the number of entities that members belong to
- **Member Entity Values:** list showing the entities that specific members belong to
- **Members By Entity Count:** list showing members and how many entities they belong to
- **Score Distribution:** chart showing count of entities linked at certain comparison scores

Reading member comparisons

- The aggregate of individual attribute weight scores that is calculated when two records are compared

Comparison Score

Attribute Weights

18.3					
	DATE	20050101	20050101	E	+3.90
	SSN	464036743	464306734	D	+3.15
	SEX	F	F	E	+0.30
⊟	Name	WALDNER|LARUE|L|	WALDNER|LARUE|B|	D	+5.36
		WALDNER	WALDNER	X	+4.16
		LARUE	LARUE	X	+4.16
		L	B	D	-0.98
⊞	AXP	12493|MERE|LN|PH...	12493|MERE|LN|PH...	P	+5.62

Match Codes: (E=Equal, D=Different, P=Partial, M=Missing, X=Equal Token, etc…)

Figure 6-18. Reading member comparisons ZZ7801.0

Notes:

This slide shows we use aggregate of individual attribute weight scores when comparing two records.

Entity Analytics Concerns

- When you run Entity Analysis look for:
  - Entities with a large number of members
  - Entities with only one member (singletons)
  - Members belonging to more than one entity
  - Score ranges of your linked entities
  - Composition and comparison scores of your largest entities

Figure 6-19. Entity Analytics Concerns ZZ7801.0

Notes:

Here are some concerns when you run Entity Analysis:

- Entities with a large number of members
- Entities with only one member (singletons)
- Members belonging to more than one entity
- Score ranges of your linked entities
- Composition and comparison scores of your largest entities

Exercise introduction

- In this exercise, you will:
  - Run Entity Analytics
  - Review Weight values

Exercise: Testing our Algorithm

Figure 6-20. Exercise introduction ZZ7801.0

Notes:

In this exercise, you will:

- Run Entity Analytics
- Review Weight values

Exercise: Testing our Algorithm

Unit summary

Having completed this unit, you should be able to:

- Understand Bulk Cross Match to analyze weights
- Understand the Enterprise ID assignment
- Understand how to create Thresholds

Figure 6-21. Unit summary ZZ7801.0

Notes:

Having completed this unit, you should be able to:

- Understand Bulk Cross Match to analyze weights
- Understand the Enterprise ID assignment
- Understand how to create Thresholds

Unit 7. PME and Physical MDM

What this unit is about

This unit describes the key concepts behind the InfoSphere MDM Physical module, including terminology and the basics of matching.

What you should be able to do

After completing this unit, you should be able to:

- Describe key concepts of Physical MDM
- Understand Physical MDM terminology
- Understand how the PME is used by the Physical MDM

Unit objectives

After completing this unit, you should be able to:

- Understand InfoSphere MDM Physical module
- Understand how InfoSphere MDM handles duplicates
- Understand how InfoSphere MDM uses the PME for Duplicate Suspect Processing
- Understand the high level integration with the PME.

Figure 7-1. Unit objectives ZZ7801.0

Notes:

After completing this unit, you should be able to:

- Understand InfoSphere MDM Physical module
- Understand how InfoSphere MDM handles duplicates
- Understand how InfoSphere MDM uses the PME for Duplicate Suspect Processing
- Understand the high level integration with the PME.

Exercise introduction

- In this exercise, you will:
  - Reset the MDM database

Exercise: Reset the MDM database

Figure 7-2. Exercise introduction ZZ7801.0

Notes:

In this exercise, you will:

- Reset the MDM database

Exercise: Reset the MDM database

Physical module overview

- Building the Single View of our Entities
  - Consolidates information from multiple sources
  - Becomes System of Record for our entities
  - Create complete, accurate, up to data information for consumers of Master Data
  - Matching and consolidating Records critical process

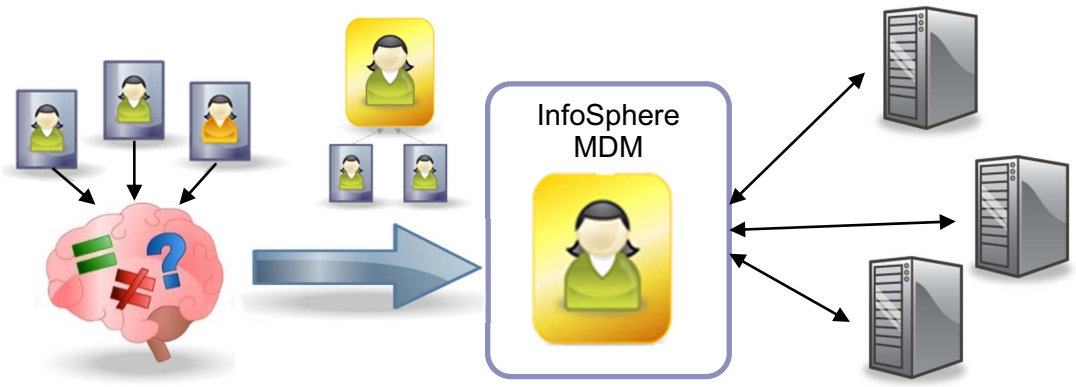


Figure 7-3. Physical module overview ZZ7801.0

Notes:

The Physical module of InfoSphere MDM behaves as a centralized-style MDM implementation. The Physical module consolidation records from any number of sources and create a single persisted view of the record. Once consolidation is complete, the physical module become the centralized system of record of our entities.

All update and access of our master data, in the enterprise, will go through the InfoSphere MDM.

The biggest strength of a centralized implementation is that it creates a single record across the enterprise and ensures that the master data is complete, accurate and up to date for all consumers of the master data in the enterprise.

To create the single record, an important process is the matching of records from the various sources of information.

Workbench

- Workbench is the MDM's primary configuration tool:
  - PME Configuration (Algorithms, Member Model, etc)
    - This our focus in this course
  - Data extensions (Additions, Extensions, Specs)
  - Service extensions (Behavior extension, Composite Services, etc)
  - Interface customization (i.e. Service Mapping and Tailoring)
  - Analytics (Buckets, Entities, etc)

Figure 7-4. Workbench ZZ7801.0

Notes:

The Information MDM Workbench is a plugin to the Rational Application Developer (RAD) environment. RAD is an eclipse based development environment. The Workbench is used to customize and configure various aspects of the InfoSphere MDM. For the Physical module this includes:

- **PME Configuration**: The workbench is used to create and customize the algorithms and member model used for matching and searching. We will use the workbench throughout this course to customize the default PME configuration. There are various tool provided with the workbench that include generating weight, loading data into the Virtual module and deriving data that we will also use when optimizing our algorithms.
- **Data extension**: MDM Physical data model can be extended by creating new entities or extending existing entities with new attributes
- **Service extension**: Using the workbench, you can generate or customize existing services for the Physical MDM
- **Interface customization**: Map external services to the InfoSphere MDM service or customized the exposed web service to only include entities that you will use in your organization

- **Analytics**: the workbench provides reports on buckets and entities to evaluate our algorithms.

NOTE: Service Extensions, Data Extensions and Interface customizations are covered in the ZZ840 course.

Pair Manager

- Used to:
  - Assist in Threshold setting exercises
  - Review matched pairs for accuracy
  - Locate possible False Positive record pairs
  - Assess how well the Algorithm is matching

NOTE: Used for same purposes as Virtual MDM

Figure 7-5. Pair Manager ZZ7801.0

Notes:

Pair Manager is a stand-alone client that is used during the Threshold Analysis stage to allow end-users to look at a sample matched pair of records and determine if they are the same; not the same; or not enough information to determine same or not (basically determining that a suspect should be created for review).

It is used to help in determining what the Algorithm thresholds should be set to.

It allows the end user to look for and irregularities that may need to have the configuration adjusted to handle. Items such as False Positives.

Virtual PME vs Physical MDM

- Configuration of Physical PME is done using Virtual MDM
- Using Virtual PME
  - Bucket Analysis
  - Entity Analysis
  - Thresholds

NOTE: Sample data is needed for the Virtual MDM during configuration

- Once configuration complete, PME algorithm is exported to Physical MDM
- During operation, BObjs are mapped to Virtual type data model

Figure 7-6. Virtual PME vs Physical MDM ZZ7801.0

Notes:

As we will see, configuring the Physical PME algorithms is done with the help of Virtual MDM, therefore all the steps we performed in the Virtual PME configuration apply when we are working with the Physical MDM PME.

During the configuration we will deploy sample data to the Virtual MDM in order to configure the PME for the Physical, this will require us to configure the Virtual Data Model to match our Physical MDM data model (or have a mapping between the two).

Similar to the Virtual configuration, we will use the Virtual MDM to help us with Bucket Analysis, Entity Analysis and Thresholds.

Once the configuration of the PME algorithm is complete we will not deploy the algorithm to the Virtual MDM but will export the configuration and deploy to the Physical MDM.

During the operation of the Physical MDM, the BObjs will go through a process of mapping to the Virtual type data model. We will examine this process in this course.

Business Object (BObj)

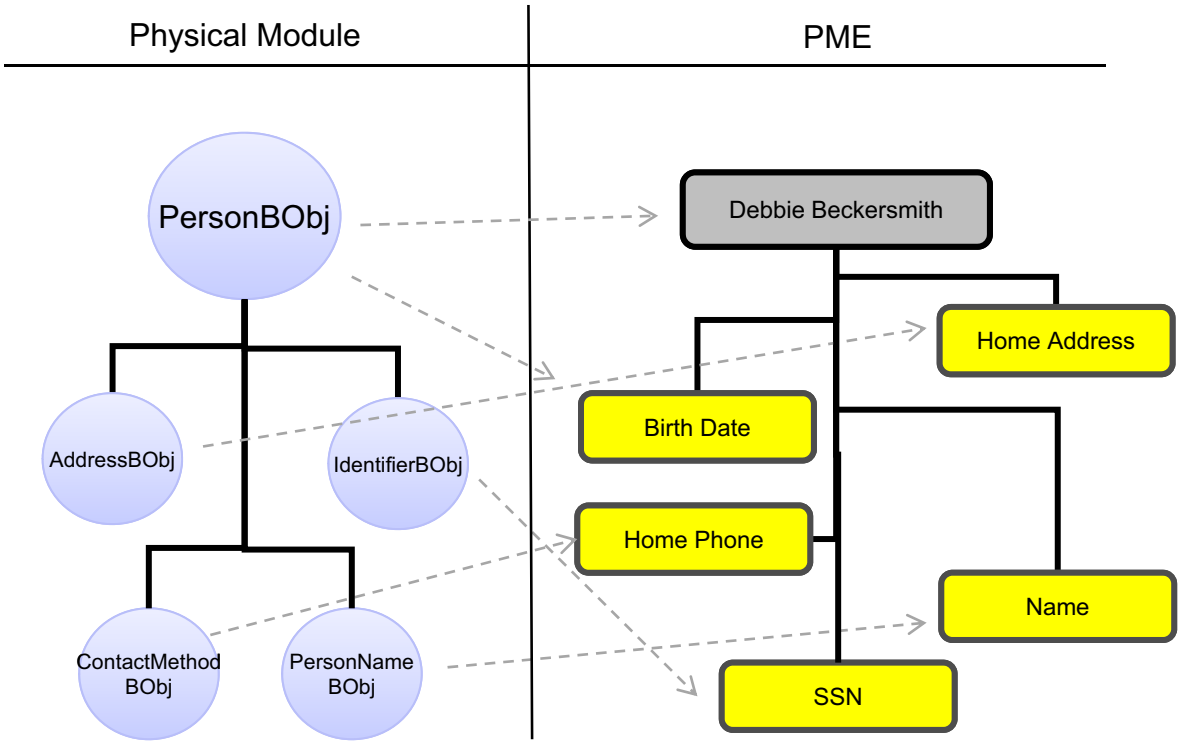


Figure 7-7. Business Object (BObj) ZZ7801.0

Notes:

Business Object in Physical Module are similar to the Attributes on the PME. When we run Matching or Searching we will map the Bobjs (e.g. AddressBObj) to the attributes on the PME (e.g. HomeAddr)

Suspect Duplicate Processing

- Triggers invoke SDP
  - Adds or updates of critical data
- Probabilistic Matching Engine is used to detect matches
  - Uses an **Algorithm** to generate match score
  - Match score produces A1, B, or C match
- Physical BObjs are mapped to PME Attributes
- Physical Module handles actions for A1, B and C matches
  - C: Do nothing
  - B: Mark records as possible duplicates
  - A1: Automatically collapse records into single record

Figure 7-8. Suspect Duplicate Processing ZZ7801.0

Notes:

Since matching and consolidating records in InfoSphere MDM Physical module is so critical it is important to understand how this process works.

The process of matching and consolidating 2 records into a single persisted view is called Suspect Duplicate Processing or SDP for short.

SDP is triggered by critical data being updated or added to the InfoSphere MDM (e.g. a Person's address change may provide more insight into a duplicate in the system)

Once SDP is triggered, the Probabilistic Matching Engine (PME) detects whether the records match any of the existing records in the database. When it is comparing records, the PME creates a match score based on an algorithm that determines how closely the records match. The higher the score the closer the records match

We also have thresholds defined that translate the match score into an A1, B or C match.

- A C match means the score was low and that the parties are not a match
- A B match means the score was high enough that we need to investigate more to determine if the records are a match. In this case we will mark the records for review

- An A1 match means the score was high enough that we know the records are the same and we will automatically collapse the records into a single record.

The focus of this course will be to examine how the algorithms determine the match score.

Probabilistic Search

- searchPersonProbabilistic & searchOrganizationProbabilistic
  - Also uses the PME
  - Input: Person or Org record
  - Output: Matches with Match Score
  - Can provide min match score

- We will use searchPersonProbabilistic to test our Algorithms

Figure 7-9. Probabilistic Search ZZ7801.0

Notes:

InfoSphere MDM also provides the capabilities to search Person and Organization records using the PME. The services include searchPersonProbabilistic and searchOrgProbabilistic. The probabilistic searches use the same PME algorithm as the SDP.

The input to the probabilistic search is a Person or Organzation records that is passed to the PME for matching. The output will include all records that match and have a minimum match scores (passed as a optional parameter in the search.

We will use the searchPersonProbabilistic search service to test out algorithms after we've done some customizations in our exercises.

The InfoSphere MDM Duplicate Process

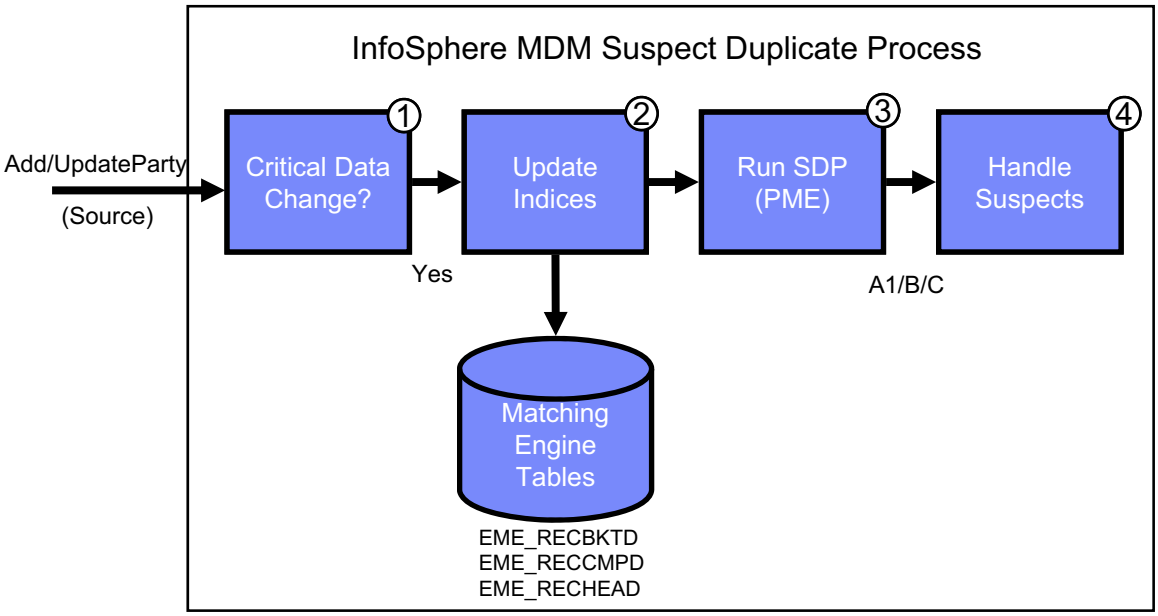


Figure 7-10. The InfoSphere MDM Duplicate Process ZZ7801.0

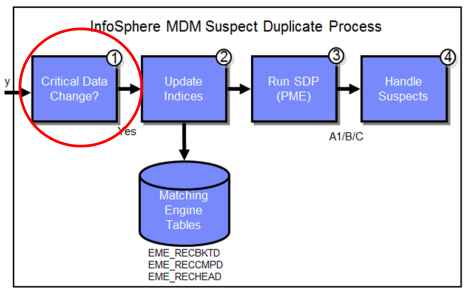
Notes:

When a record is added to the InfoSphere MDM, the following process occurs:

1. **Critical Data Change** - InfoSphere MDM determines which business attributes were updated. If any critical data was updated, then Suspect Duplicate Processing will be started. The critical data is database driven and can be modified to meet the requirements of the organization
2. **Update Indices** - InfoSphere MDM copies the critical attributes of each business object to a tables that is used by the PME. Some attributes are used only for matching and will not trigger Suspect Duplicate Processing, other will trigger the process since it could provide additional insight into duplicates.
3. **Run Matching (PME)** - The PME is run to detect whether there are any matches to the record that was sent in. This will involve bucketing and matching algorithms to produce a match score that will use the Threshold to find out what Action Type the records produce (A1, B or C)
4. Handling Suspect involve 3 external rules, a rule for A1 matched, B matches, and C matches, these can be customized to meet the business requirements of the organization.

1. Critical Data Change

PURPOSE: Detect whether a business object was changed and if the information should to passed to the Matching Engine



Before Image <> After Image ➔ Data has changed

Occurs in the Post Transaction

Figure 7-11. 1. Critical Data Change ZZ7801.0

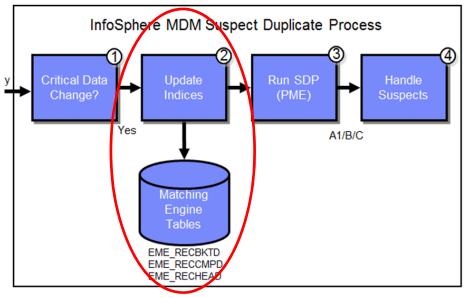
Notes:

The purpose of the critical data change module is to detect whether any attributes were change during this service request. This occurs on the post transaction level and can be customized for each business object it is handling. The default behavior will be to check the before image and compare it to the after image (these are both contained inside the BObj). If a change occurred, those business objects are sent to the Update Indices module.

2. Update Indices

If the element was changed and is found in the CRITICALDATAELEMENT table the fields are sent to the Matching Engine Tables

Once fields are in the Matching Engine Tables, they can be used for Matching



Which element to synchronize

Purpose of Synchronization 1 = Synchronize 2=SDP (run SDP)

Specific type of instance (e.g. only Legal Name)

GROUP_NAME	ELEMENT_NAME	SYNCPURPOSE_TP_CD	ENTITY_TYPE	INSTANCE_PK	…
Person	GenderType	1			
PersonName	LastName	2	NameUsageType	1	
…					

CRITICALDATALEMENT

Figure 7-12. 2. Update Indices ZZ7801.0

Notes:

The PME Engine contains a number of tables that store the derived data and buckets to the record information. The tables are used for the bucketing and matching during the Suspect Duplicate Processing, therefore when there is an update to the critical data, these tables need updating.

InfoSphere MDM stores which attributes are critical data in the CRITICALDATAELEMENT table where it indicate the purpose as 1, synchronize the data, or 2, synchronize the data and run SDP.

3. Run Suspect Duplicate Processing

PURPOSE: Search for Party Matches and determine the Match Action Category (A1/B/C)

Uses Standardized data and Buckets from previous step

We will focus on this step in this course

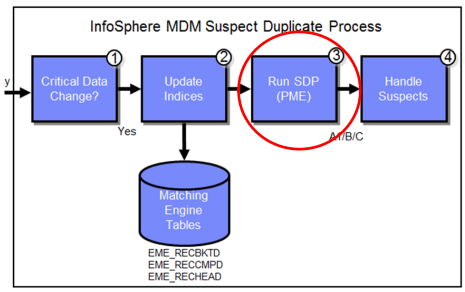


Figure 7-13. 3. Run Suspect Duplicate Processing ZZ7801.0

Notes:

The next step in the process is to run matching on the records stored in InfoSphere MDM. This is done through the PME and it's tables (that were updated in the last step). The PME uses an Algorithm that created standardized data and buckets for quicker comparison during the Update Indices step. In this step it will use the standardized data and buckets to generate a match score (as defined in the algorithm). We will be focused on this step of the SDP process to examine the algorithm and customize it to meet our requirements in the exercises.

Default Matching on Person

- **Name:** The Match Score is based on the best match across any of the name attributes, e.g., if Legal Name from record 1 exactly matches Alias Name from record 2, the score from that comparison may be used for the Name score. Exotic name tokens will score higher than common tokens, e.g. a match of "Keyser Söze" will score higher than a match of "John Doe".
- **Address and Phone Number combined**: The Match Score is based on the best match across any of the person address attributes, combined with the best match across any of the person phone attributes. For example if comparing a person primary residence and business phone number with another person's temporary residence and their mobile phone yields the highest match score, that those entities will be used.

- **Social Security Number (SSN):** This is straightforward comparison of one person's social security number with another person's social security number. Unlike a deterministic match engine, numbers that differ by only one or two digits, will still add to the match score, although less than a perfect match would.
- **Social Insurance Number (SIN):** same as SSN
- **Gender:** This is a simple yes/no comparison since there cannot be partial matches.

Default Matching on Organization

- **Name:** The match score is based on the best match across any of the attributes. For example when comparing two organizations, if the Legal Name of the first organization yields the highest match score when it's compared to the Doing Business As name, then those will be used.
- **Business Address and Business Phone**: The Match score is based on the best match across any of the address attributes, combined with the best match across any of the business phone attributes.
- **Org DUNS number:** This is straightforward comparison of one number with another. Unlike the MDMS deterministic match engine, numbers that differ by only one or two digits, will still add to the match score, although less than a perfect match would.
- **Org Corporate Tax ID:** Same as above

No Match (C)

- For Persons: 0 *to* less than 7
- For Organizations: 0 to less than 9

Possible Match (B)

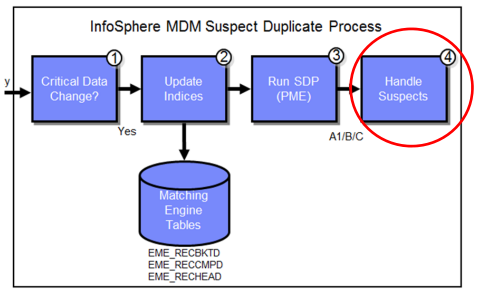
- For Persons: 7 *to* less than 15
- For Organizations: 9 *to* less than 12

Match (A1)

- For Persons: 15 or higher
- For Organizations: 12 or higher

4. Handle Suspects

PURPOSE: Collapse Parties if a match is found, add a suspect record to the database if needs more investigation.



External Rule: A1SuspectsActionRule

External Rule: A2SuspectsActionRule

External Rule: BSuspectsActionRule

External Rule: CSuspectsActionRule

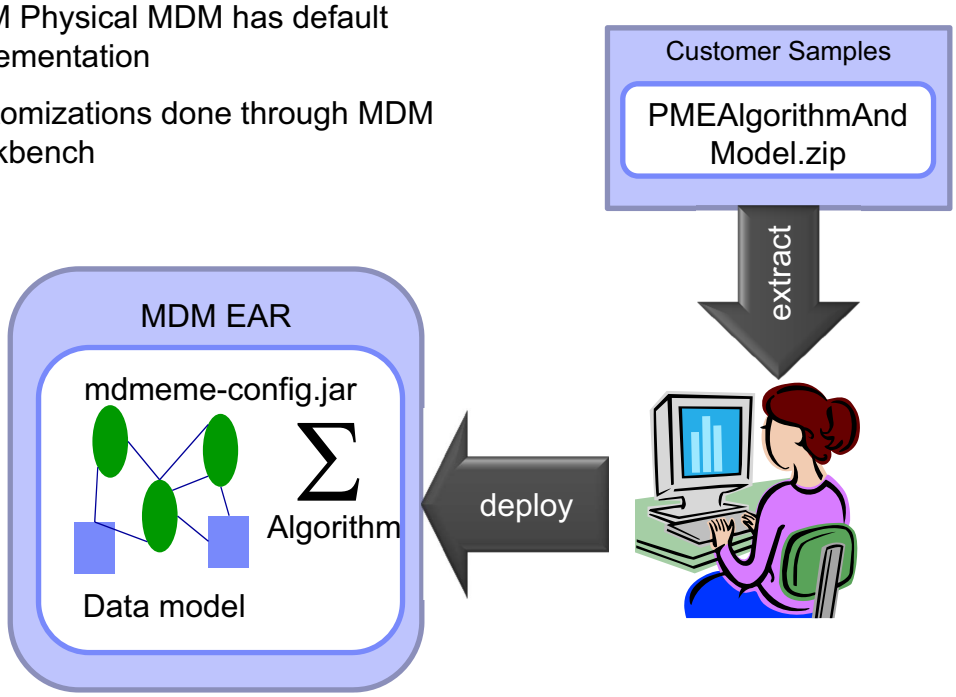
Figure 7-14. 4. Handle Suspects ZZ7801.0

Notes:

Handling Suspect involve 4 external rules, a rule for A1 matched, A2 matches, B matches, and C matches. The A2 match rule is not used with PME and only used with deterministic and QualityStage matching. Each of these rules is an external rule which can be customized to meet the business requirements of the organization.

Configuring the Model And Algorithm

- MDM Physical MDM has default implementation
- Customizations done through MDM Workbench



16

Figure 7-15. Configuring the Model And Algorithm ZZ7801.0

Notes:

MDM is preconfigured with a default model and matching algorithm contained in mdmeme-config.jar.

Clients can tailor the algorithm by extracting the eclipse project from customer samples and tailoring it in the InfoSphere MDM Worbench

Clients can generate a new mdmeme-config.jar and replace the one in MDM

Exercise introduction

- In this exercise, you will:
  - Add a Party to the Physical MDM
  - Run a number of Probabilistic Search
  - Evaluate the weight score from searches

Exercise: Testing the Default PME Configuration

Figure 7-16. Exercise introduction ZZ7801.0

Notes:

In this exercise, you will:

- Add a Party to the Physical MDM
- Run a number of Probabilistic Search
- Evaluate the weight score from searches

Exercise: Testing the Default PME Configuration

Unit summary

Having completed this unit, you should be able to:

- Understand InfoSphere MDM Physical module
- Understand how InfoSphere MDM handles duplicates
- Understand how InfoSphere MDM uses the PME for Duplicate Suspect Processing
- Understand the high level integration with the PME.

Figure 7-17. Unit summary ZZ7801.0

Notes:

Having completed this unit, you should be able to:

- Understand InfoSphere MDM Physical module
- Understand how InfoSphere MDM handles duplicates
- Understand how InfoSphere MDM uses the PME for Duplicate Suspect Processing
- Understand the high level integration with the PME.

Unit 8. Physical PME Data Model

What this unit is about

This unit describes the data model using the Physical MDM for the PME operations, including the where the derived data and buckets are stored.

What you should be able to do

After completing this unit, you should be able to:

- Understand the PME database tables
- Understand the configuration tables for the PME
- Understand the mapping between the BObjs and the PME data model

Unit objectives

After completing this unit, you should be able to:

- Understand the PME database tables
- Understand the configuration tables for the PME
- Understand the mapping between the BObjs and the PME data model

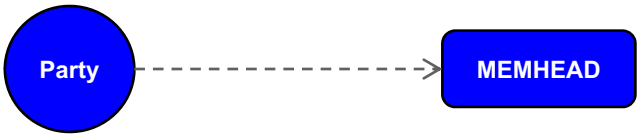
Figure 8-1. Unit objectives ZZ7801.0

Notes:

After completing this unit, you should be able to:

- Understand the PME database tables
- Understand the configuration tables for the PME
- Understand the mapping between the BObjs and the PME data model

Physical MDM/PME Data Models: Party



- Party in Physical MDM maps to MEMHEAD in PME
- Party can represent a Customer, Investor, Employee, Supplier, Patient, Hospital, etc.

Figure 8-2. Physical MDM/PME Data Models: Party ZZ7801.0

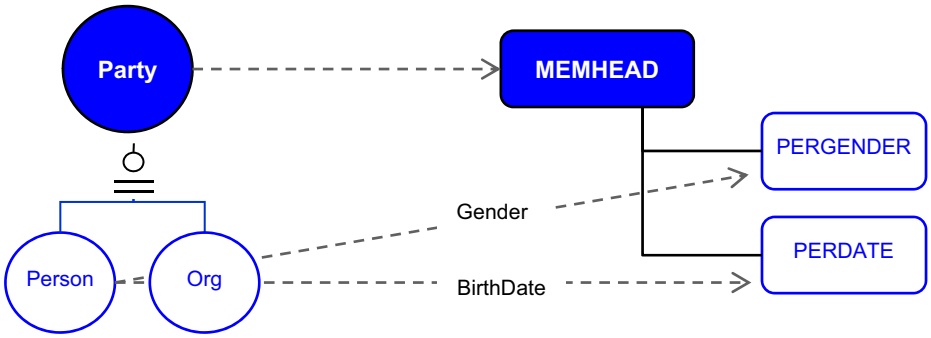
Notes:

We will start with the core entity of the Party domain of the Physical Module: the Party. The Party is stored in the table CONTACT and has a primary key cont_id on the database and PartyId in the Business Objects. A party could be a Customer, Investor, Employee, Supplier, etc.

On the PME, we have the MEMHEAD that stores similar information including it's own variation of a PartyId (memidnum).

When we synchronize with the PME, each party would produce a MEMHEAD record.

Physical MDM/PME Data Models: Person/Org



- Party in Physical MDM can be a Person or an Organization
- Person contains fields for BirthDate and Gender, these map to PERDATE and PERGENDER on PME respectively

Figure 8-3. Physical MDM/PME Data Models: Person/Org ZZ7801.0

Notes:

On the Physical MDM, a Party Entity has 2 subtypes, the Person or Organization. A Party therefore can be a Person or Organization but not both.

Person contains fields for BirthDaet and Gender, these map to PERDATE and PERGENDER on the PME respectively.

The Org business object on the Physical MDM does not map to any attribute type in PME by default as non of the attributes are used in matching and searching.

Physical MDM/PME Data Models: Names

- Physical MDM contains Business Objects for Person Name and Organization Name
- PME has similar attribute types: PERNAME and ORGNAME

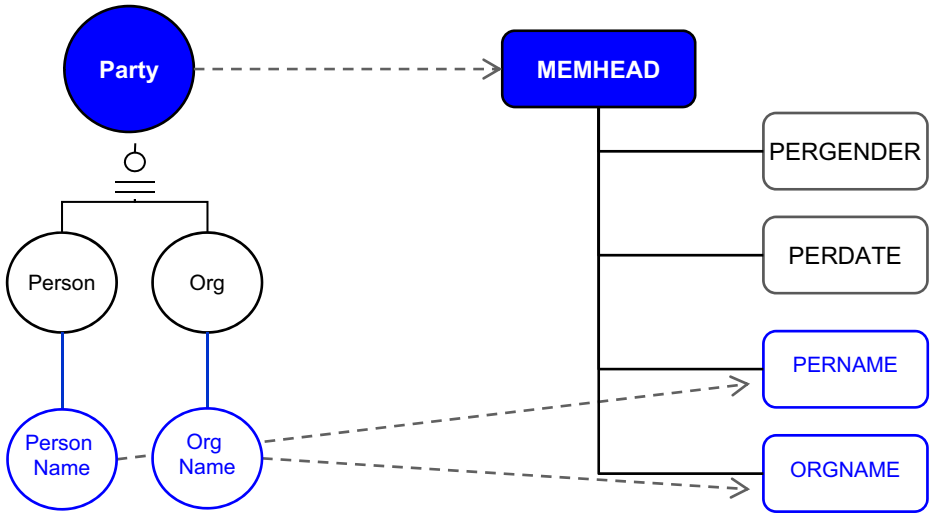


Figure 8-4. Physical MDM/PME Data Models: Names ZZ7801.0

Notes:

Physical MDM contains Business Objects for Person Name and Organization Name and are mapped to the PERNAME and ORGNAME attribute types on the PME.

For the Person, the following name types are passed to the PME for matching and searching:

- Legal Name
- Business Name
- Nickname
- AKA Name
- Maiden Name
- Alias Name
- Preferred Name
- Previous Name

For the Organization, the following name types are passed to the PME for matching and searching:

- Legal Name
- Business Name
- Abbreviated Name

Physical MDM/PME Data Models: Address/Phone

- Physical MDM Phone Numbers are stored in Contact Method Business Object
- PME has attribute types for Person and Organization Phones: PERPHONE and ORGPHONE

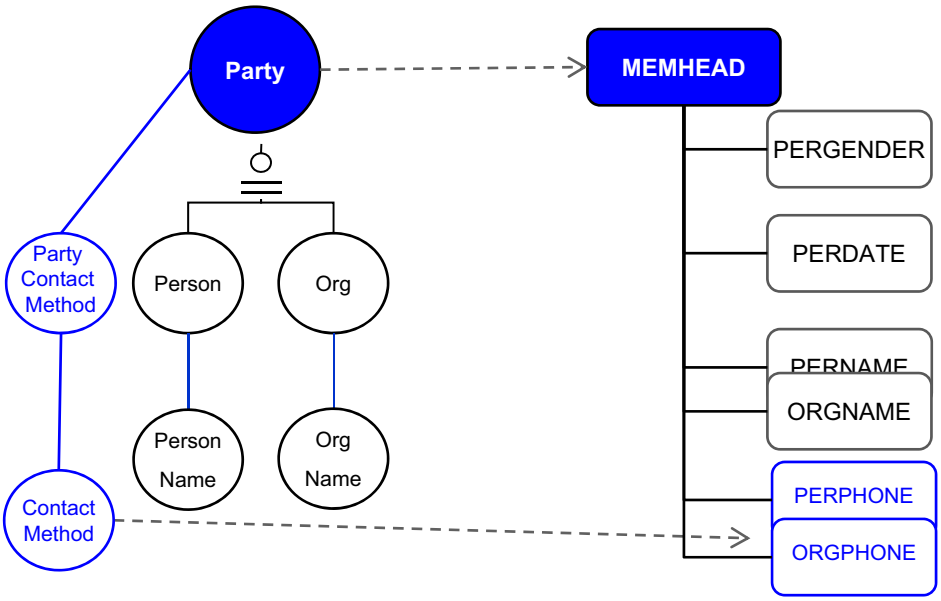


Figure 8-5. Physical MDM/PME Data Models: Address/Phone ZZ7801.0

Notes:

Physical MDM contains the following business objects for a Phone Number:

- ContactMethod: contains the actual phone number
- PartyContactMethod: Creates the link between the phone number and Party.

When the Physical MDM is synchronized with the PME, the Contact Method is mapped to the PERPHONE and ORGPHONE attribute types.

For the Person, the following phone types are passed to the PME for matching and searching:

- Home Phone
- Cell Phone
- Mobile Phone
- Business Phone

For the Organization, the following phone types are passed to the PME for matching and searching:

- Business Phone

Physical MDM/PME Data Models: Address

- Physical MDM Addresses are stored in Address Business Object
- PME has attribute types for Person and Organization Addresses: PERADDR and ORGADDR

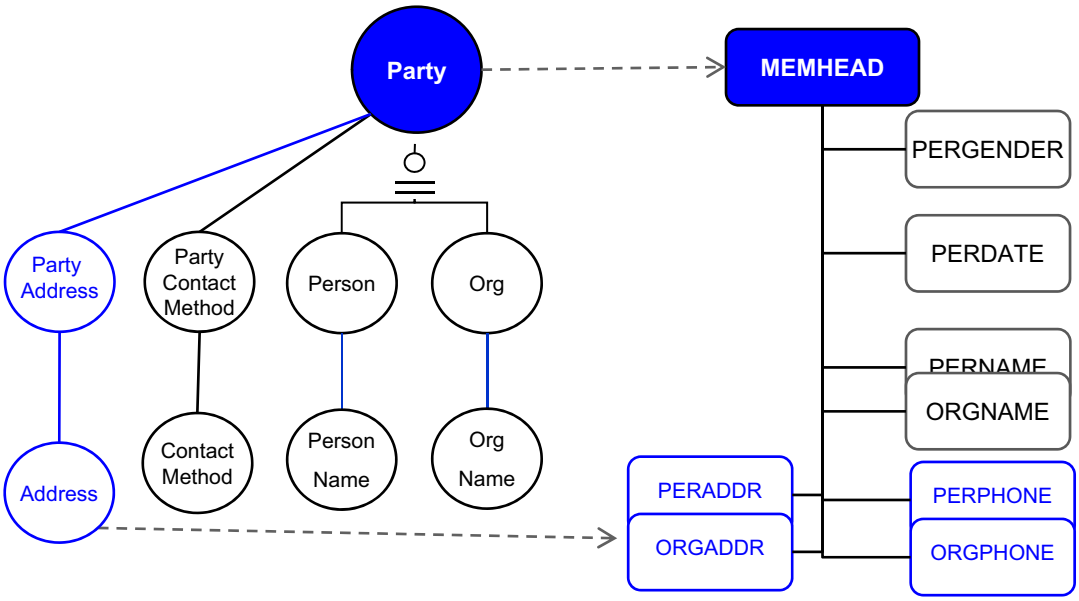


Figure 8-6. Physical MDM/PME Data Models: Address ZZ7801.0

Notes:

Physical MDM contains the following business objects for a Phone Number:

- Address: contains the actual Address information (addrline1, city, country, etc)
- PartyAddress: Creates the link between the address and Party. This allows a single address to be shared across multiple parties.

When the Physical MDM is synchronized with the PME, the Address is mapped to the PERADDR and ORGADDR for the Person an Organization respectively.

For the Person, the following address types are passed to the PME for matching and searching:

- Primary Residence
- Other Residence
- Business Address
- Mailing Address
- Summer Residence
- Temporary Address

- Second Residence

For the Organization, the following phone types are passed to the PME for matching and searching:

- Mailing Address
- Business Address

Physical MDM/PME Data Models: Identifier

- Physical MDM Identifications are stored in Party Identifier Business Object
- PME has attribute types for Person and Organization Identifications: PERIDENT and ORGIDENT

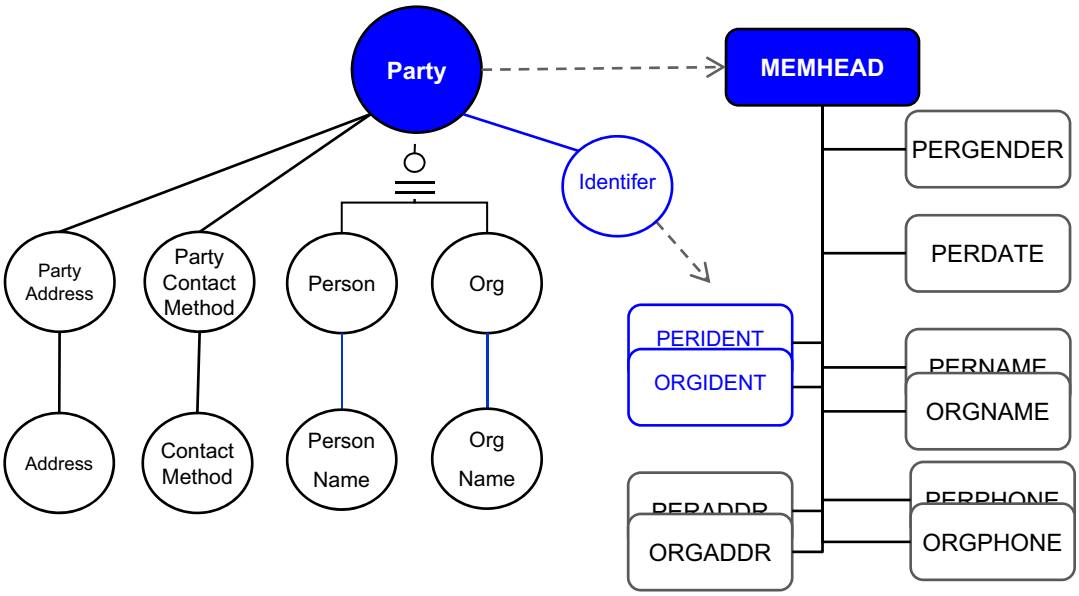


Figure 8-7. Physical MDM/PME Data Models: Address ZZ7801.0

Notes:

Physical MDM stores Identification (DUNS Numbers, SSN, SIN, etc) in the Party Identifications Business Object.

When the Physical MDM is synchronized with the PME, the Identifications are mapped to the PERIDENT and ORGIDENT attribute types.

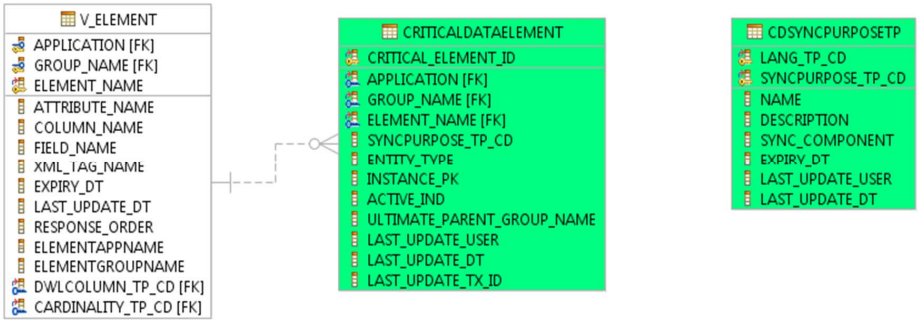
For the Person, the following address types are passed to the PME for matching and searching:

- SSN
- SIN
- Driving License (not by default but we will add this mapping)

For the Organization, the following identifications are passed to the PME for matching and searching:

- Corporate Tax Id
- DUNs Number

Defining Data worth Detecting



- CRITICALDATAELEMENT
  - Defines BObjs and fields worth monitoring (for changes)
- CDSYNCPURPOSETP
  - Why field in BObj is worth monitoring
  - 2 reasons
    - 1 Synchronize PME indices
    - 2 Re-identify suspects (PME Match)

9

Figure 8-8. Defining Data worth Detecting ZZ7801.0

Notes:

The critical attributes used in PME Matching and Searching are defined in the CRITICALDATALEMENT table. This table identifies the attributes that are synchronized with the PME and which attributes will trigger Suspect Duplicate Processing using the PME.

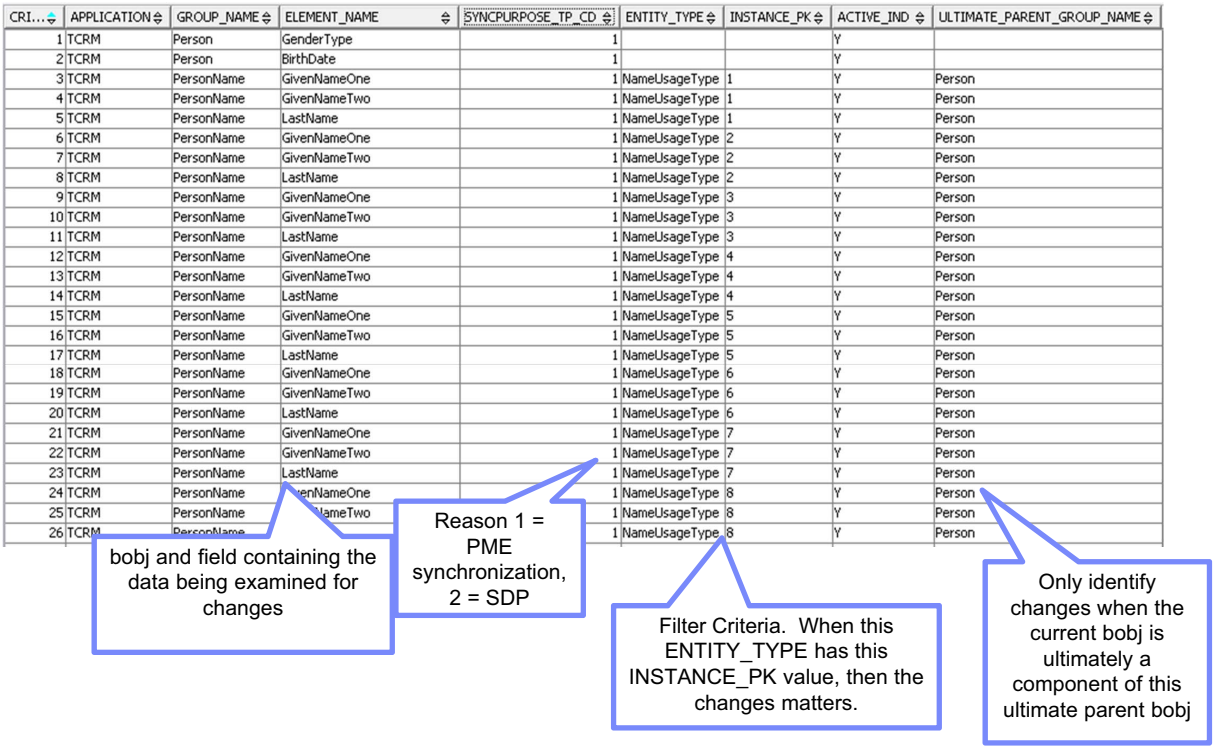
CRITICALDATAELEMENT

- Defines the Business Objects and their fields that constitute data worth monitoring for changes.

CDSYNCPURPOSETP

- Describes the reason why any field in a bobj is worth monitoring for changes
- Currently there are two ‘reasons’
  - 1 – detect data changes in order to synchronize PME indices
  - 2 – detect data changes in order to re-identify suspects.

CRITICALDATAELEMENT



CRI...	APPLICATION	GROUP_NAME	ELEMENT_NAME	SYNCPURPOSE_TP_CD	ENTITY_TYPE	INSTANCE_PK	ACTIVE_IND	ULTIMATE_PARENT_GROUP_NAME
1	TCRM	Person	GenderType	1			Y	
2	TCRM	Person	BirthDate	1			Y	
3	TCRM	PersonName	GivenNameOne	1	NameUsageType	1	Y	Person
4	TCRM	PersonName	GivenNameTwo	1	NameUsageType	1	Y	Person
5	TCRM	PersonName	LastName	1	NameUsageType	1	Y	Person
6	TCRM	PersonName	GivenNameOne	1	NameUsageType	2	Y	Person
7	TCRM	PersonName	GivenNameTwo	1	NameUsageType	2	Y	Person
8	TCRM	PersonName	LastName	1	NameUsageType	2	Y	Person
9	TCRM	PersonName	GivenNameOne	1	NameUsageType	3	Y	Person
10	TCRM	PersonName	GivenNameTwo	1	NameUsageType	3	Y	Person
11	TCRM	PersonName	LastName	1	NameUsageType	3	Y	Person
12	TCRM	PersonName	GivenNameOne	1	NameUsageType	4	Y	Person
13	TCRM	PersonName	GivenNameTwo	1	NameUsageType	4	Y	Person
14	TCRM	PersonName	LastName	1	NameUsageType	4	Y	Person
15	TCRM	PersonName	GivenNameOne	1	NameUsageType	5	Y	Person
16	TCRM	PersonName	GivenNameTwo	1	NameUsageType	5	Y	Person
17	TCRM	PersonName	LastName	1	NameUsageType	5	Y	Person
18	TCRM	PersonName	GivenNameOne	1	NameUsageType	6	Y	Person
19	TCRM	PersonName	GivenNameTwo	1	NameUsageType	6	Y	Person
20	TCRM	PersonName	LastName	1	NameUsageType	6	Y	Person
21	TCRM	PersonName	GivenNameOne	1	NameUsageType	7	Y	Person
22	TCRM	PersonName	GivenNameTwo	1	NameUsageType	7	Y	Person
23	TCRM	PersonName	LastName	1	NameUsageType	7	Y	Person
24	TCRM	PersonName	enNameOne	1	NameUsageType	8	Y	Person
25	TCRM	PersonName	ameTwo	1	NameUsageType	8	Y	Person
26	TCRM	PersonName		1	NameUsageType	8	Y	Person

Figure 8-9. CRITICALDATAELEMENT ZZ7801.0

Notes:

In the CRITICALDATALEMENT table you'll find the following columns:

- **GROUP_NAME** (BObj) and ELEMENT NAME (Bobj attribute): Identifies the attribute of the business object that will be monitored for changes
- **SYNCPURPOSE_TP_CD**: can have 2 values (1 or 2). 1 indicates that the attribute should be synchronize with the PME tables when changed. 2 indicates that a change in the attribute should trigger SDP to be run. In most cases you will find that an attribute has both 1 and 2 defined, since if it is changed we want it to be synchronize and SDP to be run.
- **Filter Criteria (ENTITY_TYPE, INSTANCE_PK):** narrows down the type of attribute that is being monitored (for example we want to monitor SSN but not Drivers License)
- **ULTIMATE_PARENT_GROUP_NAME**: Identities the Object that the attributes needs to be part of to be recognized for changes.

Synchronize PME Indices

- Process in post service (add or update)
- Driven by Data Change Detection
  - Data Change Purpose Type = 1
- Single synchronization adapter (DerivedDataSynchronizationAdapter).
  - Derived Data: term used in PME Data that has gone through Standardization and Bucketing
- On/Fff using configuration repository setting:
  - `/IBM/DWLCommonServices/DerivedDataSynchronization/enabled = true/false`

11

Figure 8-10. Synchronize PME Indices ZZ7801.0

Notes:

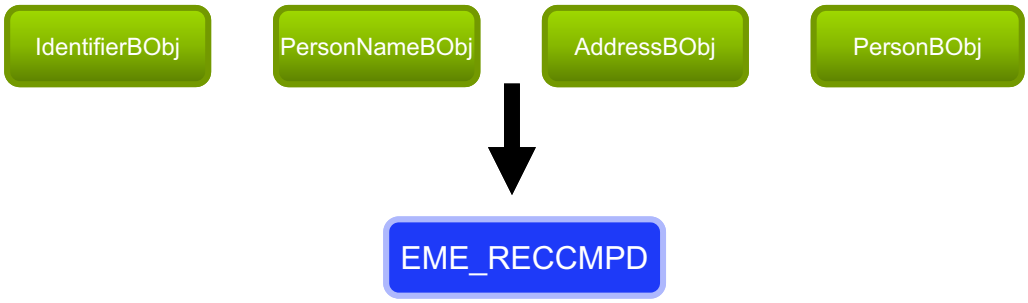
Synchronizing PME indices is a generic process in post execute of a Physical MDM service. It is driven by data change detection where the attribute id defined in the CRITICALDATAELEMENT table and the data change purpose type is 1.

On the technical side, there is a single synchronization java adapter (DerivedDataSynchronizationAdapter) that handles the synchronization of the attribute with the PME tables.

During the synchronization, the data goes through the Standardization and Bucketing processes and is then referred to as Derived Data.

Synchronization is turned on/off using the configuration repository setting:
`/IBM/DWLCommonServices/DerivedDataSynchronization/enabled = true/false`

Master Data Tables – Derived Tables



eme_reccmpd

MEMRECNO	SRCRECNO	CMPVAL	…
1	1	BROSE:PERRY:O:.^85217^N-5520:S-BUCHANAN:S-ST:.S-APACHE:S-JUNCTION:S-AZ:.N-85217:.	
2	1	TEEPLE:DANIAL:E:.^85283^N-30829:S-SIXTEENTH:S-ST:.S-TEMPE:S-AZ:.N-85283:.	
3	1	HOWSE:ADELAIDE:W:.^^N-14553:S-TRIANON:S-PLZ:.S-PHOENIX:S-AZ:..^312212702^3986314	
…			

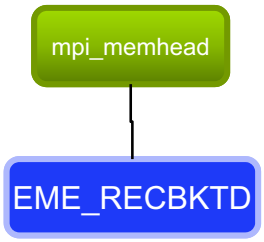
Figure 8-11. Master Data Tables – Derived Tables ZZ7801.0

Notes:

When the attributes are synchronized with the PME, the data will go through the Algorithm and create the derived data. After the standardization function the data is stored in the eme_reccmpd table. Some of the key fields of the eme_reccmpd table include:

- **MemRecNo**: The unique identifier of the record
- **SrcRecNo**: The source of the member record
- **CmpVal**: The derived data produced after the standardization function

Master Data Tables – Bucketing Table



eme_recbktd

MEMRECNO	SRCRECNO	BKTHASH	…
1	1	9059273197451191198	
1	1	2351273197451191198	
2	1	7252340299695367517	
…			

Figure 8-12. Master Data Tables – Bucketing Table ZZ7801.0

Notes:

Also still in the synchronization process, the standardized data is run through the bucketing process and the records are linked to the buckets that they are associated with. The eme_recbktd table stores the connection from the record to the bucket. The Bucket is identified by a hash code that represents the values, this improves the performance when retrieving members from a bucket. Each Hash Code represents a different bucket value (e.g. the bucket Hash 9059273197451191198 might be James Smith). We will see how these buckets are generated in the next unit.

Synchronize PME Indices

- synchronizeeME()
  - Facilitate on-demand synchronization of MDM data
  - Inputs
    - Entity Type (Person or Organization)
    - Instance PK
  - Ignores setting in configuration repository:
    - `/IBM/DWLCommonServices/DerivedDataSynchronization/enabled`

14

Figure 8-13. Synchronize PME Indices ZZ7801.0

Notes:

Besides the automatic trigger of synchronizing the PME there is also a service available for the Physical MDM to trigger the synchronization of the attributes: synchronizeeME()

The Inputs to the services include:

- Entity Type : Person or Organization
- Instance PK: the id of the entity for synchronization (i.e. ParytId)

This service ignores the setting in the configuration repository:

```
/IBM/DWLCommonServices/DerivedDataSynchronization/enabled = true/false
```

Exercise introduction

- In this exercise, you will:
  - Load default Physical PME configuration into RAD
  - Viewing derived data from previous exercise

Exercise: Loading the default PME Algorithm

Exercise Viewing the PME Derived Data

Figure 8-14. Exercise introduction ZZ7801.0

Notes:

In this exercise, you will:

- Load default Physical PME configuration into RAD
- Viewing derived data from previous exercise

Exercise: Loading the default PME Algorithm

Exercise Viewing the PME Derived Data

Unit summary

Having completed this unit, you should be able to:

- Understand the PME database tables
- Understand the configuration tables for the PME
- Understand the mapping between the BObjs and the PME data model

Figure 8-15. Unit summary ZZ7801.0

Notes:

Having completed this unit, you should be able to:

- Understand the PME database tables
- Understand the configuration tables for the PME
- Understand the mapping between the BObjs and the PME data model

Unit 9. Algorithms

What this unit is about

This unit describes the default algorithm used by the PME for the Physical MDM.

What you should be able to do

After completing this unit, you should be able to:

- Understand the default algorithm components for the Physical MDM PME implementation
- Customize the default algorithm to include 2 new fields

Unit objectives

After completing this unit, you should be able to:

- Understand default PME algorithm for Physical PME
- Understand what customization will be done in the exercise

Figure 9-1. Unit objectives ZZ7801.0

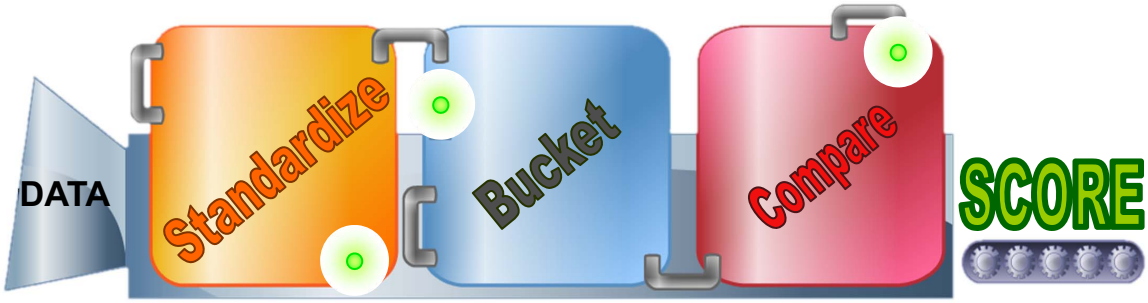
Notes:

After completing this unit, you should be able to:

- Understand default PME algorithm for Physical PME
- Understand what customization will be done in the exercise

Algorithms: the secret sauce / black box

The algorithm is the brain behind the searching and matching processes:



Converts data to simplest form for easier use during matching process.

Organizes records that share common values for faster search retrieval.

Compares pairs of records using the probabilistic method to calculate a score.

Jim Smith ➔ **SMITH:JIM** ➔ **JN + SNT** ➔ **Jim Smith vs. Jim Smith = 6.1**
➔ **Jim Smith vs. James Smith = 5.9**
➔ **Jim Smith vs. John Smith = 4.2**

Figure 9-2. Algorithms: the secret sauce / black box ZZ7801.0

Notes:

Algorithm: A series of processes (Standardization, Bucketing, and Comparison) used to probabilistically match records that results in a comparison score.

Data: is ingested by InfoSphere MDM and goes through the following steps….

Standardization: converts data into its simplest form for easier use during the searching and comparison processes. In other words, standardization is used to clean up the data by reformatting it into consistent chunks to be used by the algorithm.

Bucketing: organizes records that share common values for faster search retrieval. Think of bucketing this way: is it easier to find a needle in a hay stack or in a jar labeled “needles”?

Comparison: uses the probabilistic method to compare pairs of records and then assign a score based on the similarities and differences between the two records. In other words, the comparison function is a means for finding similarity or difference between two records and their attributes.

The output from the algorithm is the **Score** which is the accumulation of the weights that are determined during comparison.
For Example:

Using the name Jim Smith, the **Standardization** process (depending on the function chosen) simply removes punctuation and capitalizes the name (plus whatever additional the function you choose does) and produces cleansed outputs that would be a colon between the fields.

The **Bucketing** function in this example is using Equivalence & Phonetic, so it converts the nickname/alternate name/equivalent name of Jim to the formal James and then converts James to the key sound markers JN. Smith has no nickname, so it is just converted to the key sound markers SNT. These two are combined and placed in the “JN + SNT” bucket, which requires that both tokens are required for a search.

Comparison will occur if there is more than one record in the “JN + SNT” bucket. Here we see a Jim Smith compared to a Jim Smith. The exact match scores 6.1 points.

Next we see a Jim Smith compared to a James Smith. Jim is a nickname for James, so they score almost as high as if they were an exact match, in this case 5.9 points.

Also in this bucket is John Smith (John is also converted to JN). When John and Jim are compared, they still match phonetically… but the phonetic match is not scored as high as the Nickname or Exact Match. So the score is 4.2, which shows us that John Smith is not very far off.

While we have only reviewed this process for a Name, it is important to note that during the Algorithm’s analysis of the data, it runs parallel analyses of each of the attributes in the records for those attributes that have data available to use.

Default Physical Algorithm (Person Name)

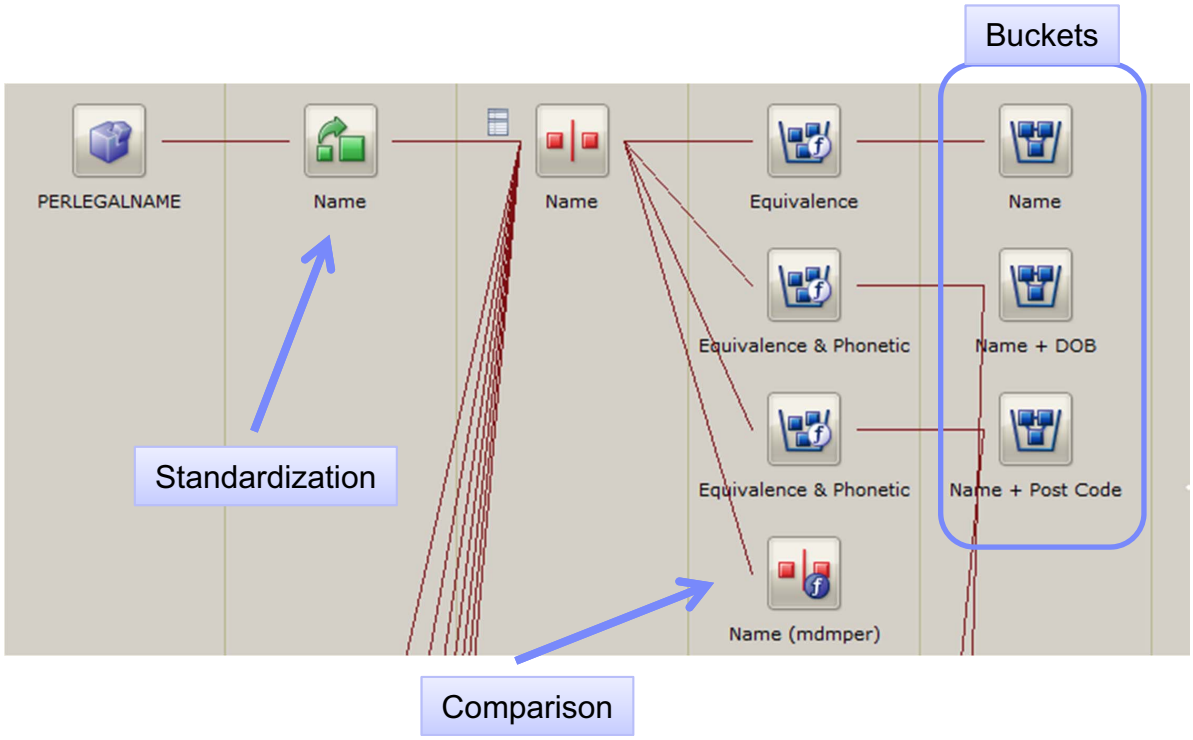


Figure 9-3. Default Physical Algorithm (Person Name) ZZ7801.0

Notes:

Attributes:

There are 8 name types used default Physical PME Algorithm (only PERLEGALNAME is shown in the slide). They are all placed in the same comparison role value (e.g. Legal names will be compared to Business Names, Maiden Names, etc)

- PERLEGALNAME
- PERALIASNAME
- PERMAIDENNAM
- PERPREFNAME
- PERPREVNAME
- PERAKANAME
- PERNICKNAME
- PERBUSNAME

Standardization

The standardization function for each of the names is the same (PXNM). The standardization accepts the lastname, givenname1, givenname2, prefix, generation, and givenname3 fields and uses an Anonymous string file to filter out values that would be considered anonymous (e.g. Boy).

Buckets

There are 3 buckets defined for the Name derived data. The first one is an exact match on 2 of the tokens from the name (e.g. givenname1, lastname). The second bucket is a combination between the Phonetic version of the Name and Match of the DOB. The third Bucket is a combination of the Phonetic version of the Name and the Postal Code (or ZIP code).

Comparison

The comparison of the name uses the QXNM Metaphone function. This will produce a value that is added to the overall match score between records that belong to the same bucket. The QXNM will compare the tokens in the name and produce a value for matches (higher score for rare values). It will also provide bonus points for tokens that are in the same order.

Default Physical Algorithm (Person Address)

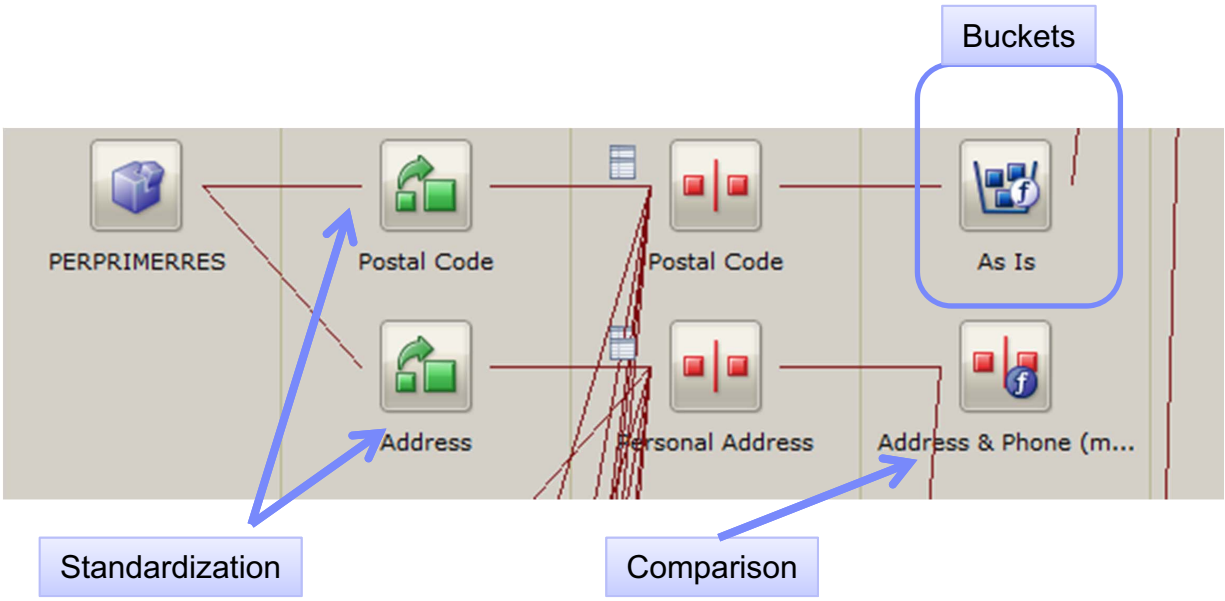


Figure 9-4. Default Physical Algorithm (Person Address) ZZ7801.0

Notes:

Attributes:

There are 6 Address Types used in this part of the Physical PME Algorithms (only PERPRIMERRES is shown in the slide)

- PERPRIMERRES: primary address
- PERMAILADDR: mail address
- PEROTHERRES: other residence
- PERSECONDRES: secondary residence
- PERSUMMERRES: summer residence
- PERTEMPADDR: temporary address

Standardization

There are 2 standardization functions that our Addresses go through.

The first separates the zip code form the address and provides the zip code to the Name bucket described in the previous slide.

The second is used for comparison, and takes the full address and standardizes it to be fed into the Address and Phone Comparison function.

Comparison

In the case of the Address and Phone there are no buckets, this means that if only the address matches, there will not be a score as it will not be compared to any other record (only records from the same bucket are compared to each other.

The Address and Phone uses a 2 dimensional weight that combines the comparison of the Address and the comparison of the phone to come up with a value. We will look at the weights in the next unit.

Default Physical Algorithm (Phone)

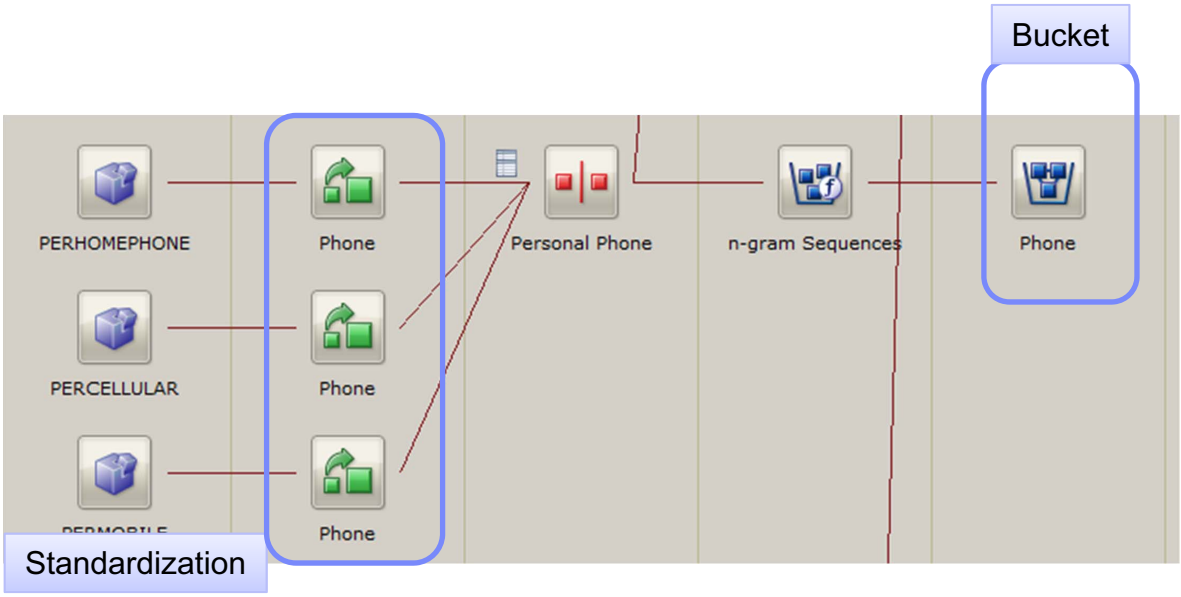


Figure 9-5. Default Physical Algorithm (Phone) ZZ7801.0

Notes:

The next part of the algorithms uses the phone information.

Attributes:

There are 3 phone types used in this part of the default Physical PME Algorithm:

- PERHOMEPHONE: primary address
- PERCELLULAR: mail address
- PERMOBILE: other residence

Standardization

The phone standardization uses the Phone2 Standardization function. This function takes the phone number and essentially removes not numeric values and removes the area code:

- If the length is greater than or equal to 10 digits, use digits 4 through 10.
- If the length is greater than or equal to 7 digits, use the first 7 digits

Bucketing

The bucketing uses a n-gram Sequences bucketing function (using 6 as the N value) which allows values that contain 6 digits of the phone number to belong to the same bucket.

Comparison

The Phone information is used in the 2 comparison along with the address information.

Default Physical Algorithm (Business)

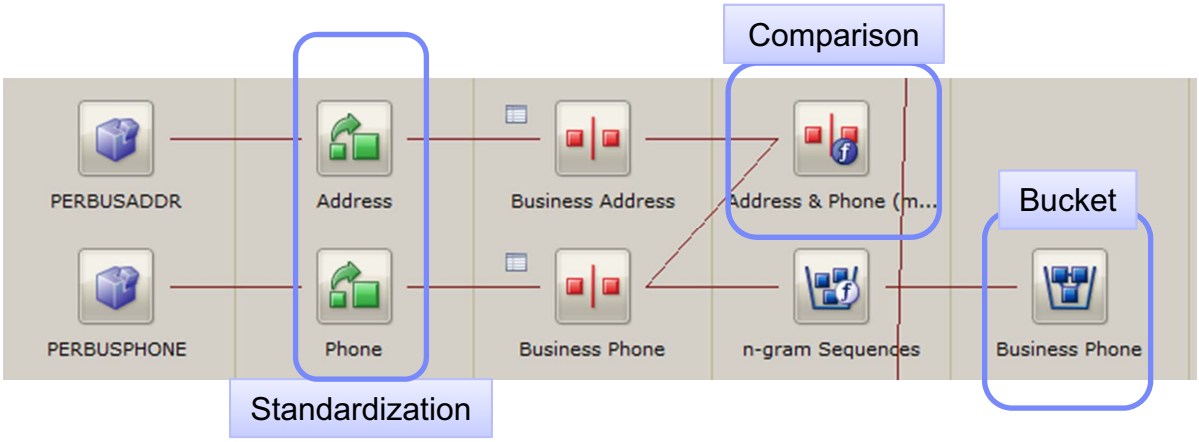


Figure 9-6. Default Physical Algorithm (Business) ZZ7801.0

Notes:

The Business Address and Business Phone also contain a Comparison and Bucket that is separate from the other Address and Phone information. Although the Standardization Comparison and Buckets are the same they are separated so that the Primary Address is not compared to the Business Address or Home Phone number is not compared to the Business Phone Number.

Default Physical Algorithm (SSN and SIN)

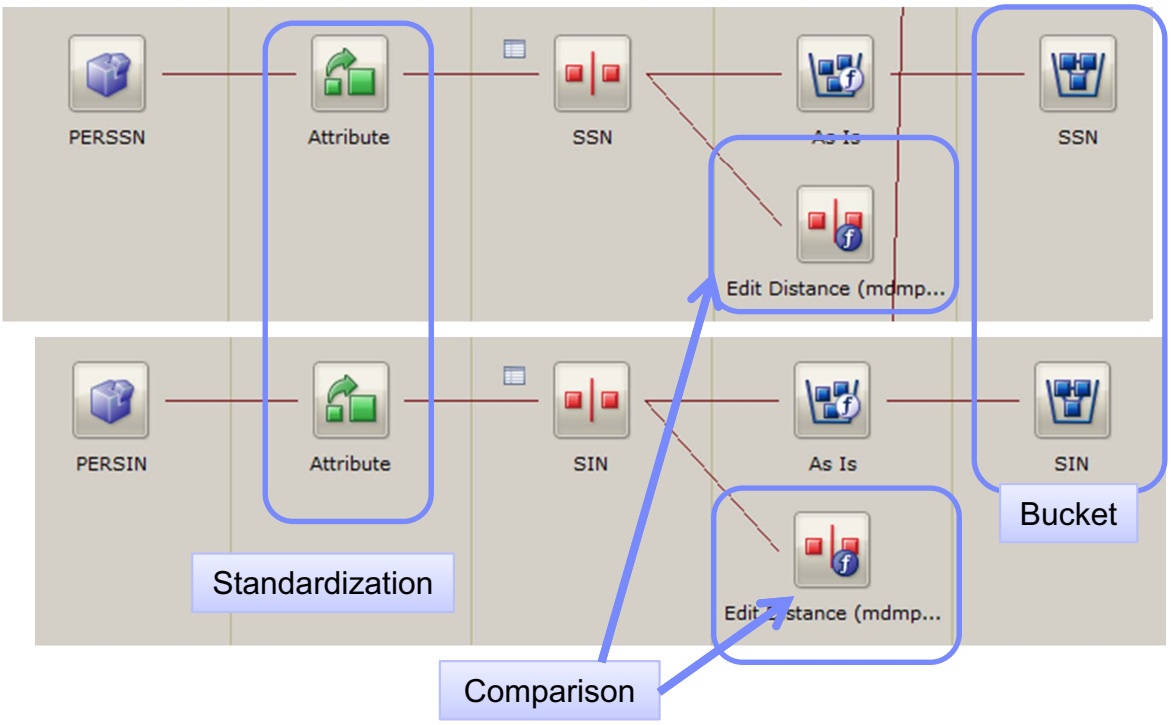


Figure 9-7. Default Physical Algorithm (SSN and SIN) ZZ7801.0

Notes:

Next in the default algorithm, you will find the Person SSN and Person SIN. Both of these attribute are run through the same Standardization, Bucketing and Comparison functions.

Standardization

The standardization is a simple Attribute standardization function with Anonymous values to remove values that should not be evaluated (e.g. 99999999)

Bucket

The buckets for the SSN and SIN take the standardized data and create buckets As-Is, meaning that the SIN or SSN would have to essentially match to be in the same bucket

Comparison

The comparison uses Edit Distance which allows the data to be off by a couple of edit and still produce a weight. This is useful for fields such as SSN and SIN as there may be small errors when this data in entered in to the system.

Default Physical Algorithm (Gender)

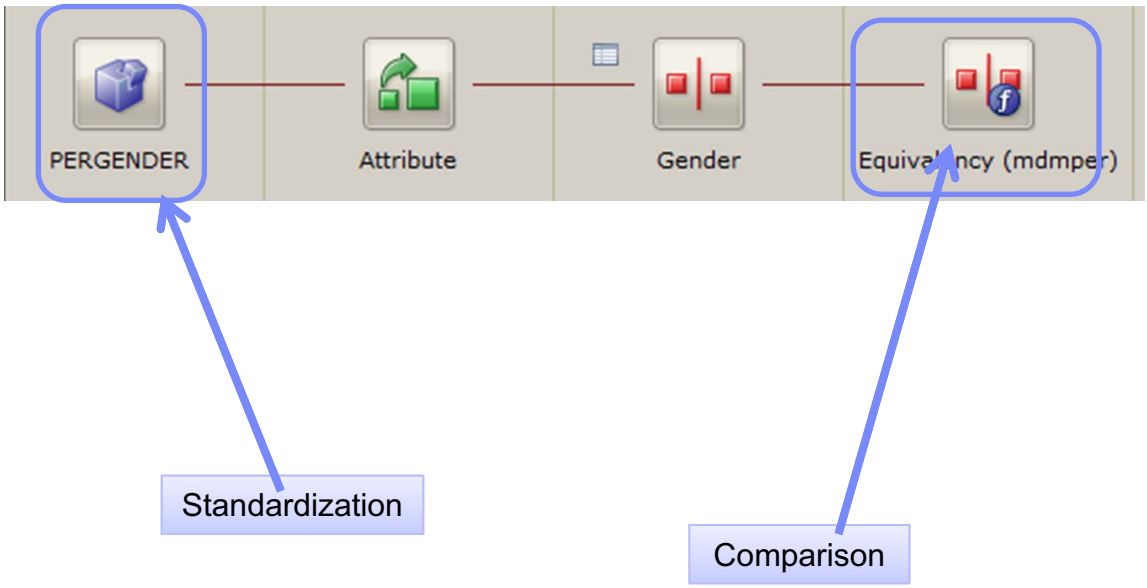


Figure 9-8. Default Physical Algorithm (Gender) ZZ7801.0

Notes:

The Gender is also compared in the default algorithm. There is no bucketing as the buckets would be too large however we do provide a score in the Comparison if the gender matches. The Comparison has to match exactly, therefore in the comparison function we provide a equivalent string code that will translate certain words into their equivalent values (e.g. Male – M)

Default Physical Algorithm (Birth Date)

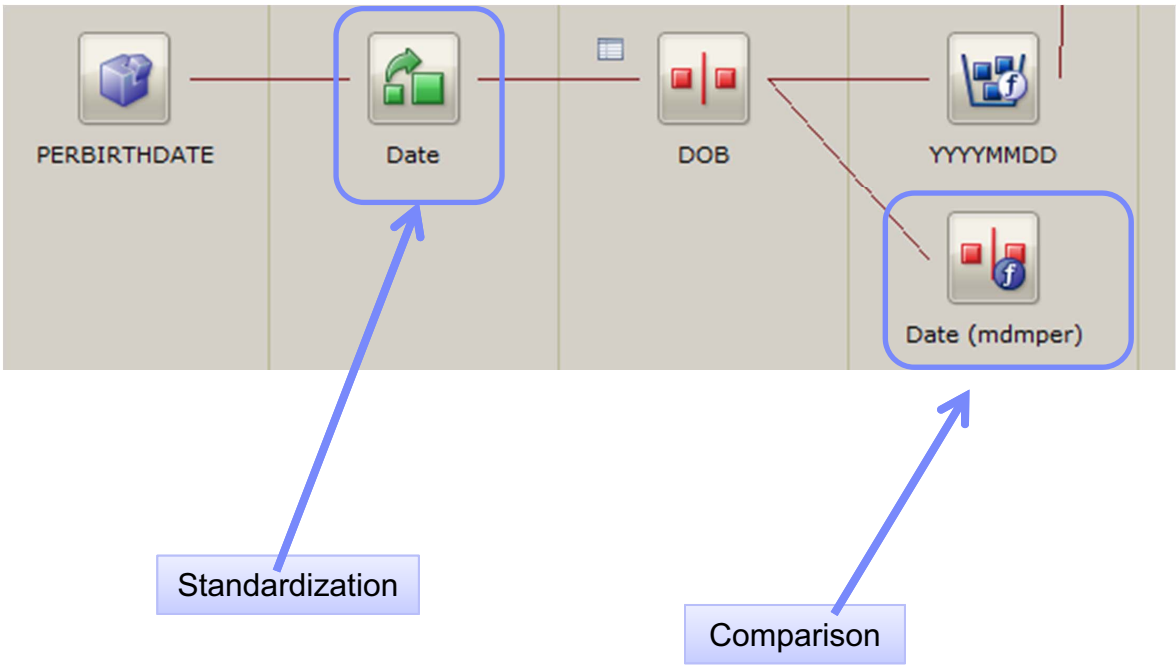


Figure 9-9. Default Physical Algorithm (Birth Date) ZZ7801.0

Notes:

The Person birth date is also used in the Algorithm for both a Bucket (along with a name) and a Comparison function.

Standardization (Date2)

Creates an 8 character string (MMDDYYYY)

- If year, month or day is invalid, those portions of the date are replaced by a string of 0's (zeros)

Comparison

Provides values for Year, MonthDay (if month and day match), Month and Day. This allow the date to be partially correct and still receive some value.

Exercise Algorithm synopsis

2 new attributes:
PERDRVRLICEN, PERUSERNAME (new attribute)

2 standardizations:
Attribute

2 comparison roles (RECCMPD):

2 bucketing functions:
n-gram Sequences

2 bucketing groups (RECBKTD):
DRIVLIC, USERNM

2 comparisons:
Edit Distance

Figure 9-10. Exercise Algorithm synopsis ZZ7801.0

Notes:

This slide shows you the Algorithm additions we will make in the exercise.

We will be configuring our algorithms using 2 new attributes.

From each of the attributes, we will be doing the following:

1. Standardization function (Attribute)
2. Comparison Role
3. Bucketing Function (n-gram Sequence)
4. Bucketing Group
5. Comparison Function (Edit Distance)

Exercise introduction

- In this exercise, you will:
  - Add a 2 attributes to the default algorithm
    - Person Username
    - Person Driving License

Exercise: Customizing the Physical algorithm

Figure 9-11. Exercise introduction ZZ7801.0

Notes:

In this exercise, you will:

- Add a 2 attributes to the default algorithm
  - Person Username
  - Person Driving License

Exercise: Customizing the Physical algorithm

Exercise introduction

- In this exercise, you will:
  - Deploy new Configuration to Virtual Module
  - Load Sample Data
  - Run Bucket Analysis
  - Update Algorithm to improve buckets

Exercise: Bucket Analysis

Figure 9-12. Exercise introduction ZZ7801.0

Notes:

In this exercise, you will:

- Deploy new Configuration to Virtual Module
- Load Sample Data
- Run Bucket Analysis
- Update Algorithm to improve buckets

Exercise: Bucket Analysis

Deploying the PME Algorithm for Physical

- Generate PME Bundle Project
- Deploy OSGi bundle to WAS
  - "Bolt" new bundle to MDM
- Update CONFIGELEMENT
  - Indicate new algorithm

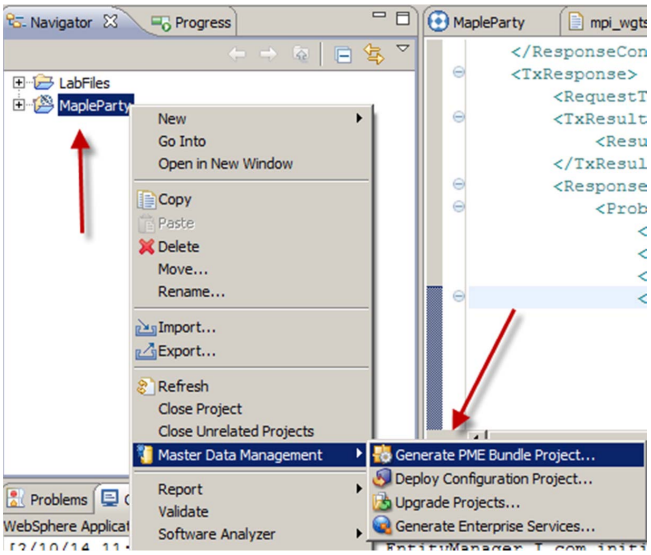


Figure 9-13. Deploying the PME Algorithm for Physical ZZ7801.0

Notes:

The whole process of customizing the PME Algorithm for the Physical MDM has been done using the Virtual MDM space. In order to deploy the PME Algorithm to the Physical MDM we will need to go through the following steps:

1. Generate PME Bundle Project – this creates an OSGi bundle that can be deployed to the WAS server. Physical MDM will look for the configuration service in the bundle.
2. Deploy OSGi bundle to WAS and attach the bundle to the MDM runtime
3. Update CONFIGELEMENT table. The configuration will indicate which configuration to use for the Person and Organization entities.

Exercise introduction

- In this exercise, you will:
  - Generate Weights
  - Deploy Weights
  - View generated weights
  - Deploy to Physical MDM
  - Rerun Search Services

Exercise: Generating Weights

Figure 9-14. Exercise introduction ZZ7801.0

Notes:

In this exercise, you will:

- Generate Weights
- Deploy Weights
- View generated weights
- Deploy to Physical MDM
- Rerun Search Services

Exercise: Generating Weights

Unit summary

Having completed this unit, you should be able to:

- Understand default PME algorithm for Physical PME
- Understand what customization will be done in the exercise

Figure 9-15. Unit summary ZZ7801.0

Notes:

Having completed this unit, you should be able to:

- Understand default PME algorithm for Physical PME
- Understand what customization will be done in the exercise

Unit 10. Adapters and Converters

What this unit is about

This unit describes the Adapters and Converters used in the Physical Module and PME integration.

What you should be able to do

After completing this unit, you should be able to:

- Understand how MDM synchronizes with PME
- Understand role of PME Adapters
- Understand role of PME Converters
- Understand how to customize the default PME Converters

Unit objectives

After completing this unit, you should be able to:

- Understand how MDM synchronizes with PME
- Understand role of PME Adapters
- Understand role of PME Converters
- Understand how to customize the default PME Converters

Figure 10-1. Unit objectives ZZ7801.0

Notes:

After completing this unit, you should be able to:

- Understand how MDM synchronizes with PME
- Understand role of PME Adapters
- Understand role of PME Converters
- Understand how to customize the default PME Converters

Adapters and Converters

- Adapters
  - Overall business logic used to carry out matching, searching, and synchronization using the PME
  - Interfaces between MDM and PME
- Converters
  - Converters ‘convert’ business objects into PME records and back again

3

Figure 10-2. Adapters and Converters ZZ7801.0

Notes:

The basic architecture for embedding PME consists of Adapters and Converters

Adapters

- Adapters contain the business logic used to carry out matching, searching, and synchronization using the PME
- Adapters are the interfaces between MDM and PME

Converters

- Converters ‘convert’ business objects into PME records and back again
- We can create our own converters

Embedded PME – Overview

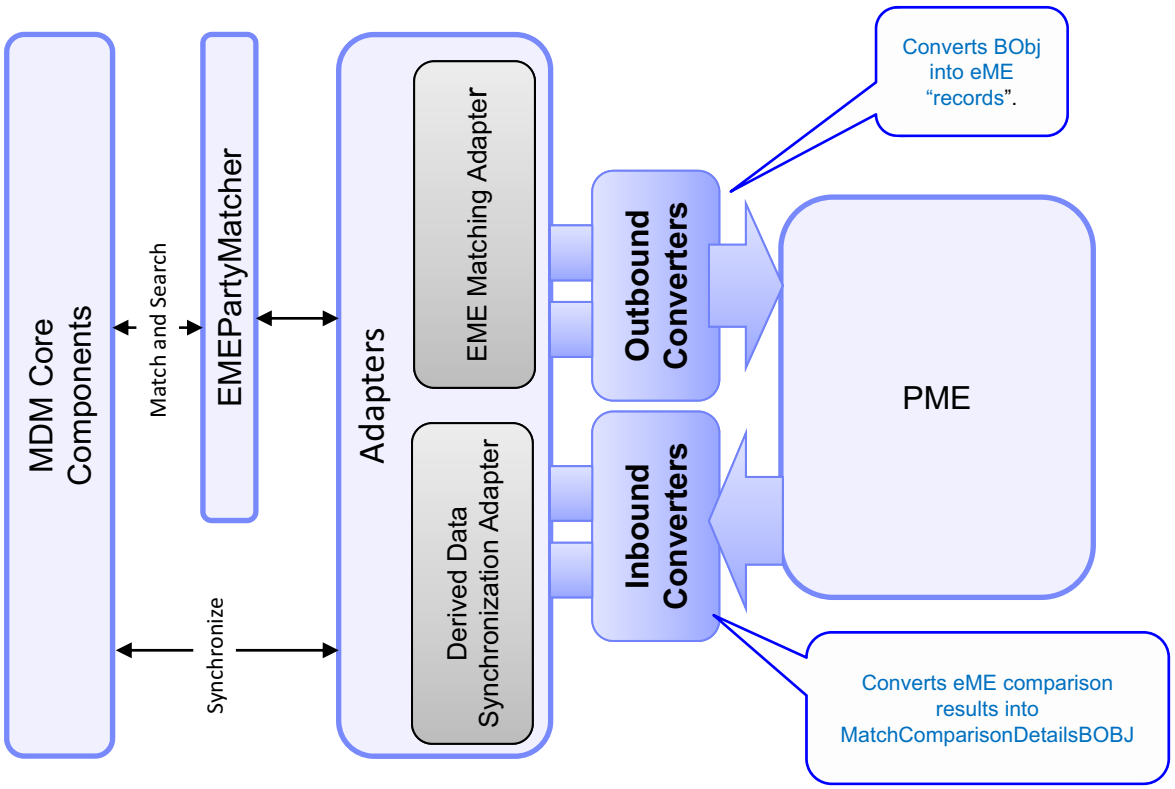


Figure 10-3. Embedded PME – Overview ZZ7801.0

Notes:

The architecture of the InfoSphere MDM Probabilistic Matching Engine within an InfoSphere MDM solution consists of a suite of adapters and converters that enable communication between InfoSphere MDM core components and the InfoSphere MDM Probabilistic Matching Engine.

In order for the PME to run the algorithm, the data from Bobjs must be converted to the PME record format. This is done through the converters.

- Outbound converters convert business objects into eME "records" and pass those records to eME for matching.
- Inbound Converters convert eME comparison results into MatchComparisonDetailsBOBJ business objects and return those to MDM's Core components for further processing or displaying to users.

Converters

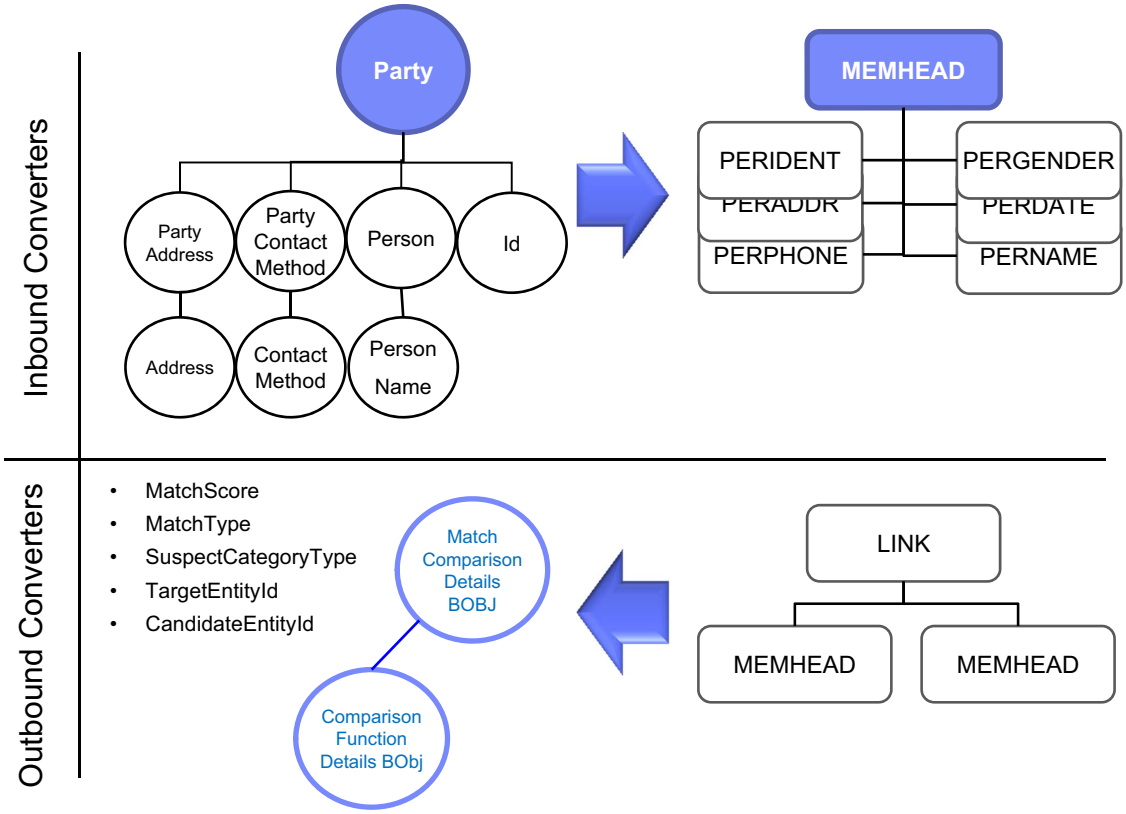


Figure 10-4. Converters ZZ7801.0

Notes:

Converters carry out the conversion of data from InfoSphere MDM formats, such as business objects, to InfoSphere MDM Probabilistic Matching Engine Record formats.

These converters are delivered in the default external rules directory and already convert a predefined subset of the InfoSphere MDM business objects and their attributes.

From the Physical Module, the Inbound Converter takes a PersonBObj (or OrganizationBobj) and converts the objects into a Member (MemHead with all its attributes). This is done during Synchronization and Probabilistic Search.

The Outbound Converter takes a link between 2 or more members and creates a MatchComparisonDetailsBObj. The business object contains the match score, match type (eME), suspect category type (A1, B, C), target entity id and candidate entity id.

The MatchComparison DetailsBObj also contains ComparisonFunctionDetailsBObj describing the details of the match between attributes.

Custom Converters

PersonDerivedDataConverter.java

```
personRecord.addAttribute(buildAttribute("PERUSERNAME", "idnum",
                personObj.getDisplayName());
```

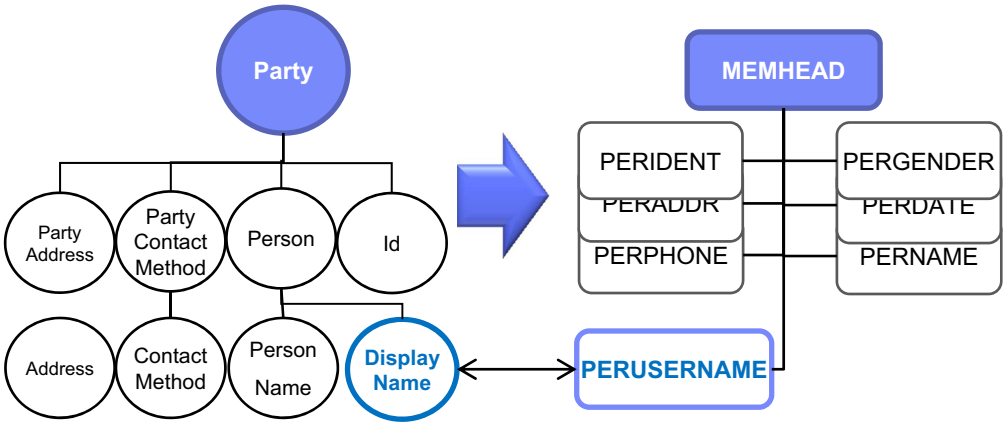


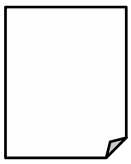
Figure 10-5. Custom Converters ZZ7801.0

Notes:

We can customize the current converter by adding additional conversions between the Business Objects and the Records. In the exercise we will add the code to create the PERUSERNAME record from the person display name.

Exposing new Converters (blueprint)

- *Interface*
  - *com.ibm.mdm.common.converter.ConverterFactoryService*
- *Convert Types:*
  - *synchronization.eme.converter*
  - *eme.matching.converter*
  - *eme.searchSuspects.converter*
- *Bean Class*
  - *com.ibm.mdm.common.converter.ConverterFactoryServiceImpl*
- *Object to Convert*
  - *TCRMPersonBObj*



blueprint

Figure 10-6. Exposing new Converters (blueprint) ZZ7801.0

Notes:

To expose our new Converter to the InfoSphere MDM, we must expose the ConverterFactoryService Interface. When an Object is sent to the adapter, the adapter will look for this service and the Object that it handles (e.g. TCRMPersonBObj). There is also 3 converter types that you can implement (Synchronization, Matching, SearchSuspects) used for the 3 process that invoke the PME.

We will see a concrete example of the blueprint file in the exercise

Exercise introduction

- In this exercise, you will:
  - Import default converter
  - Customize converter (add username)
  - Update blueprint file
  - Deploy new converter
  - Test new converter

Exercise: Customizing MDM Converters

Figure 10-7. Exercise introduction ZZ7801.0

Notes:

In this exercise, you will:

- Import default converter
- Customize converter (add username)
- Update blueprint file
- Deploy new converter
- Test new converter

Exercise: Customizing MDM Converters

Unit summary

Having completed this unit, you should be able to:

- Understand how MDM synchronizes with PME
- Understand role of PME Adapters
- Understand role of PME Converters
- Understand how to customize the default PME Converters

Figure 10-8. Unit summary ZZ7801.0

Notes:

Having completed this unit, you should be able to:

- Understand how MDM synchronizes with PME
- Understand role of PME Adapters
- Understand role of PME Converters
- Understand how to customize the default PME Converters