

How does matching work in matchIT?

Overview

matchIT gives users the ability to search for duplicates using any data e.g. name, zip, full address, phone number, date of birth, account number – or anything else known about the customer. This search uses several proprietary “fuzzy matching” methodologies, including:

- Phonetic matching to match “sounds like” names such as Deighton and Dayton. Three levels of phonetic routine are available, the choice depending primarily on the nationality of the data.
- Lookup tables to match names such as Bill and William, The Guthrey Group and Guthrey Ltd.
- Acronym and initial matching to match inconsistencies like Bill and W. (e.g. Bill Deighton and Mr. W. Dayton), The Guthrey Group and TGG Inc.
- Non-phonetic fuzzy matching to match keying errors (such as transpositions like Wilson and Wislon) and reading errors (such as Morton and Horton).
- Element matching to match names with elements missing or reversed, such as Mr J R Gonzalez, Jose Gonzalez Jr and Gonzalez Jose.

matchIT grades duplicates by “score” – a higher matching score indicates a higher certainty of match. You can choose to individually inspect the results, and/or globally flag records that score above a certain threshold. Of course, you can print detailed reports at every stage in the process, or export the results for external database processing if desired.

matchIT will identify which of a pair of records is “best” and should be preserved, or combine data from duplicate records so nothing is lost. All the matching rules and tables are user-customizable to allow for the differing characteristics of data files and customer-specific requirements.

matchIT is highly flexible, fully parameterized and deals effectively with foreign names and addresses – so it is ideally suited to companies dealing with a wide variety of business and consumer data.

Key Generation

In *matchIT*, there are two critical steps: the first is the **Import** or **Key Generation** process, that performs **data standardization** and **enhancement** and generates **match keys**. This step also reports patterns in the data and allows the user to drill into potentially bad data. The second step is the actual **Find Matches** step itself.

Find Matches

Match Keys

After Import, you specify the **Match Keys** – a match key is something that groups of records within the database have in common, which indicates that detailed comparison of the records is worthwhile to see *how well* the records match each other in other respects. The records within these groups are only matching **candidates**, they may not be true matches – further processing is necessary to check if they are true matches.

In *matchIT*, by default, three separate keys are used to find all the possible matches. If, for example, one record in a pair of duplicates did not have a zip, we would miss the matching records if we relied just on phonetic key of last name plus some or all of the zip.

This is because the search requires at least part of the zip as well as the phonetic last name to be the same, in order for the two records to be in the same “candidate group”. However, if the street in both the records is the same (which will be the case if they are true matches),

then if we do a second search based on phonetic key of last name plus the phonetic key of the most significant word(s) in the street, we would find the match on the second scan of the database.

The third key that we use by default (for US data files) is the full address key plus the building/apartment number, which will allow duplicates to be identified where there is a non-phonetic error in the last names e.g. Morton and Horton. However, the user can choose different or additional keys, including customer-specific data – there is no limit to the number of match keys that you can use.

Match Scores

Match Keys help us cut down the number of potential matches that are fed into the second step, which is the **Match Score** step – the step that enables us to determine how well the other data matches in each pair of records. We now go through the record, field or block by field or block, and work out how similar they are. Each field or block can contribute to a **match score**, depending on how that data is in that pair of records. At the end, we have an overall score that tells us how alike two records are – the higher the score, the more similar the records.

When flagging duplicates with **matchIT**, you can enter a threshold score, for and above which **matchIT** will automatically delete one record from all matching pairs. With most data files, all pairs scoring above a particular score (say 90) will be true matches and anything below (say) 80 will be false matches – this leaves a grey area between (say) 80 and 90 where most of the pairs are true matches but some are false. For “underkill”, you can therefore enter (say) 91 as a threshold score for deletion. For “overkill”, you can enter 80. To get it spot on, you can flag all matches that score over (say) 90 and **Verify** interactively the matches in the grey area between 80 and 90, choosing whether each pair is a true or false match.

Because we know that everyone’s data is different, we have allowed the way that two records are compared to be customized, using a parameter table that tells us how much each field or block contributes to the overall matching score. Using this table, we can tell **matchIT** how important each field or block is in the matching process. We call this the **Weights** table, as it reflects the relative weighting that each field or block has towards the total match score.

Summary

The **Match Keys** act as a filter, filtering in candidate pairs that may be matches. Match Keys work best using phonetic keys, and combinations of fields. After that, each candidate pair of records has a **Match Score** calculated for it, based on the **Weights** table. This allows **matchIT** to work out how well each pair of records matches, as represented by the total match score. **matchIT** is unique in the degree to which it maximizes the number of true matches that it finds whilst minimizing both false matches and the manual effort needed to review the matches.