Ingeniería en Computación

Herramienta flexible para integración de datos de fuentes heterogéneas

---

PROYECTO DE GRADO

Hernán Esteves — Matías Rodal — Nicolás Tomeo

Tutores: Dra. Adriana Marotta – Ing. Sebastián García

Montevideo, Uruguay
Marzo 2017

Resumen

Hoy en día, la integración de datos es una tarea con la que muchas empresas deben enfrentarse. En particular, por diversas razones, la problemática de identificar dos registros de distintas fuentes que corresponden a la misma entidad en el mundo real debe ser resuelta una y otra vez, y no es una tarea trivial.

Este proyecto de grado abarca el desarrollo de un framework web para la integración de datos de fuentes heterogéneas (en cuanto a modelo, representación, formatos, etc.) desarrollado con tecnolgías como Python, Django, AngularJS, MongoDB, entre otras. El mismo tiene una arquitectura modular y flexible que facilita la tarea de los desarrolladores que quieran o necesiten extenderlo integrando sus propios módulos, adaptarlo y así poder satisfacer sus necesidades para cada proyecto.

Previo al desarollo se realizó un estudio profundo del estado del arte en el campo de la integración de datos donde se capturan los aspectos esenciales del proceso que debe seguirse para llevar a cabo un proyecto de integración de datos exitoso. De la misma forma, se probaron y analizaron distintas herramientas disponibles hoy en día que asisten en proyectos de integración de datos, tanto open source como comerciales, dando una idea de las capacidades de la tecnología actual y de cómo se compara la herramienta propuesta con éstas. En base a los objetivos propuestos en el marco de este proyecto, las herramientas que se analizaron no proveen la modularidad, flexibilidad y alcance que requiere un proyecto de integración de datos, lo cual supuso un desafío nuevo a superar en el desarrollo de la herramienta propuesta.

Tabla de contenidos

Aclaraciones previas

A lo largo de este informe se utilizarán los siguientes términos con las acepciones detalladas continuación:

- Matching: refiere a la acción de hacer determinar que dos o mas elementos corresponden a una misma entidad.
- Matchear: la acción de determinar que dos o más elementos corresponden a una misma entidad.
- Matches: dos o más elementos que se determinó que refieren a la misma entidad.

Capítulo 1

Introducción

Dado el avance de las tecnologías de la información en las últimas décadas y el acceso cada vez más fácil a nuevos sistemas de información que manejan datos de todo tipo, muchas empresas generan y tienen sus datos almacenados en sistemas distintos donde los datos se representan de diferentes formas. Esto es un gran problema para la organización a la hora de sacar provecho de todos los datos que posee, ya que no se tiene a una entidad representada en forma consistente en un solo registro, y probablemente la información contenga errores o no sea completa en una o más de las fuentes. Este tipo de problemas donde una misma entidad puede estar representada de varias formas en distintas fuentes son llamados problemas a nivel de instancia, pero además, pueden existir heterogeneidades a nivel de esquema, donde un mismo dominio es representado con distintos esquemas.

Este tipo de problemas ha motivado un sinfín de investigaciones y propuestas desde hace más de 20 años, y motiva también este proyecto de grado, que busca aportar a la comunidad, no desde el punto de vista científico y matemático de los algoritmos de integración, sino desde la definición y construcción de una herramienta que sirva de plataforma extensible para un amplio espectro de casos de uso de integración.

El framework propuesto está orientado a usuarios que requieran de la integración de diversas fuentes de datos, tanto estructurados como no estructurados, en donde la correspondencia de entidades es el problema fundamental. En la herramienta, el usuario puede seleccionar entre diversos tipos de fuentes de datos (bases de datos relacionales, bases NoSQL, CSVs, etc), seleccionar entre diversas técnicas de matching y transformación y optar por el formato de salida requerido.

El framework dispone también de una interfaz gráfica que le permite al usuario construir y experimentar con flujos de extracción, transformación, matching y generación de la salida. El usuario construye estos flujos utilizando módulos provistos por la plataforma. Estos módulos tienen una funcionalidad bien definida, con una interfaz de entrada de datos y una interfaz de salida, de manera de poder combinar las entradas con las salidas de otros módulos. El usuario puede resolver sus problemáticas de integración con la combinación de los distintos módulos provistos por la plataforma.

Un aspecto fundamental del sistema es que se trata de una plataforma abierta. Desarrolladores terceros pueden incorporar sus propios módulos, permitiendo así extender las capacidades del framework. Este aspecto es el que permitirá el crecimiento de la

plataforma y su adaptación a nuevas problemáticas.

El framework no sólo puede ser utilizado desde la interfaz gráfica, sino que también dispone de una API. Esta API permite que programadores terceros puedan utilizar el software desde sus propios sistemas, por medio de la invocación de servicios expuestos por la plataforma. Esto logra dos cometidos importantes: compartir las funcionalidades de la plataforma con sistemas de terceros y al mismo tiempo flexibilizar su uso, rompiendo con cualquier restricción que imponga la interfaz de usuario.

1.1. Objetivos

El objetivo de este proyecto es la construcción de un framework configurable y flexible para la integración de datos provenientes de fuentes de distintos tipos, que permita la combinación de diferentes técnicas y soluciones de integración. Para esto, primero se profundizará en el estado del arte en cuanto a teoría y herramientas existentes, y luego se pasará a la definición de requisitos, diseño e implementación del framework.

1.2. Resultados esperados

- Conocimiento del estado del arte en el campo de integración de datos para datos heterogéneos.
- Framework de integración de datos para datos heterogéneos, configurable, modular y extensible de forma tal que permita al usuario tanto utilizar los módulos como desarrollarlos.
- Documentación completa del proyecto.

1.3. Organización del documento

El documento se divide en 6 capítulos. El capítulo 1 busca introducir la problemática, la necesidad de este proyecto y finalmente los objetivos planteados y resultados esperados para resolver dicha problemática en el marco de un proyecto de grado.

En el capítulo 2 se desarrolla el estado del arte del campo de Integración de Datos, cubriendo puntos importantes como son Schema Matching, Data Matching, Data Fusion y la integración de datos en Big Data. Finalmente, hay una investigación de algunas de las herramientas de integración de datos que existen actualmente, tanto open source como comerciales.

En el capítulo 3 se describe el desarrollo del framework desglosado en 3 fases: análisis y requisitos , diseño y arquitectura e implementación. La sección de análisis y requisitos además incluye un apartado donde se describen los perfiles identificados de los usuarios finales de la herramienta. En la sección de diseño y arquitectura se incluye también un apartado de puesta en producción donde se plantea un caso hipotético y se realiza una

discusión planteando alternativas al instalar el framework en un ambiente de producción. La sección de implementación describe tanto la implementación del motor interno del framework como la API REST utilizada, la interfaz gráfica, y finalmente se incluye con una subsección de validación con usuarios del framework.

En el capítulo 4 se presenta un caso de uso completo del framework con capturas y cuyo objetivo es mostrar las cualidades que brinda framework en pleno uso.

Finalmente, en el capítulo 5 se dan las conclusiones del proyecto y se presentan puntos para seguir trabajando en un futuro, y el capítulo 6 contiene anexos mostrando el uso de otras herramientas con mayor detalle.

Capítulo 2

Estado del arte

En este capitulo se describe el estado del arte en el área de integración de datos según varias fuentes consultadas.

Integrar datos de distintas fuentes consiste en 3 tareas principales:

- **Schema matching**: esta etapa se encarga de identificar qué estructuras de los esquemas a integrar representan el mismo tipo de entidad de la realidad. Por ejemplo, en caso de integrar dos bases de datos, identificar que tablas de las dos bases representan las mismas entidades (como podrían ser clientes, ventas, productos), y además, que columnas de esas tablas representan la misma información (nombre, dirección, precio, etc.)
- **Data matching**: es la tarea de identificar y hacer corresponder registros individuales de distintas bases de datos (u otras fuentes) que refieren a las mismas entidades del mundo real. Un caso especial de data matching es la *detección de duplicados*, que es la tarea de identificar y corresponder registros que refieren a la misma entidad dentro de una misma base de datos.
- **Data fusion**: es la tarea final y consiste en combinar pares o grupos de registros que fueron clasificados como matches, dando como resultado un registro limpio y consistente que represente de forma correcta a la entidad. Cuando se aplica en una sola base de datos a este proceso se le llama *deduplicación*.

2.1. Schema Matching

Al intentar integrar datos de distintas fuentes, el primer obstáculo que aparece es el de identificar qué partes de la estructura de cada fuente se corresponde con las de otra fuente. Por ejemplo, considerando los esquemas de la figura 2.1, se tiene que lo que en la base B se almacena en la tabla Clientes, en la base A se almacena en una tabla llamada Usuarios, y algunas columnas también tienen nombres distintos aunque representan la misma información. El objetivo del schema matching es hallar estas correspondencias.

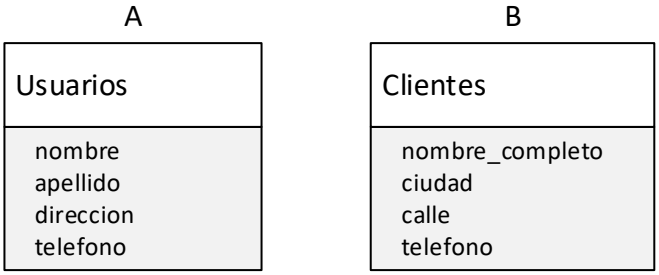

Figura 2.1: Esquema A y B de ejemplo

Históricamente, esta tarea ha sido realizada por expertos en el dominio de cada trabajo de integración, pero a pesar de que últimamente se han buscado alternativas que no dependan tanto de expertos, siguen siendo necesarios para tomar ciertas decisiones. Esta dificultad hace que la tarea de schema matching sea una de las más complicadas y el mayor obstáculo a enfrentar al momento de realizar una tarea de integración.

En [3], publicado en 1992, ya aparecen algunas nociones fuertes de integración de esquemas. Era común en ese entonces que para un proyecto grande de diseño de una base de datos se generaran decenas de esquemas distintos, que luego debían ser integrados generando un único esquema final que contemplara los aspectos relevantes de todos los esquemas. Las diferencias entre los esquemas mencionados pueden ser por varias causas, como por ejemplo, las distintas visiones ante el problema de los diferentes diseñadores.

El proceso que describe [3] es de ejecución manual y a realizar por expertos del dominio de los esquemas a integrar. Se basa en analizar dos tipos de conflictos: conflictos de nombre y estructurales.

Para resolver los conflictos de nombre, la técnica busca sinónimos y homónimos. Llama sinónimos a los elementos de los esquemas que tienen distinto nombre, pero representan el mismo elemento del dominio; y homónimos a los elementos de los esquemas que tienen el mismo nombre pero representan distintas entidades del dominio. Una vez identificados, se elige y asigna un nombre al par de sinónimos, mientras que a los homónimos se les asignan nombres distintos.

Para los conflictos estructurales se busca encontrar una representación común, y se clasifican en 3 categorías: idénticos, cuando la estructura es la misma y la unificación no realiza modificaciones; compatibles, cuando tienen distintas estructuras, pero no son contradictorias; e incompatibles, cuando se tienen diseños contradictorios (p. ej. distintas cardinalidades o distintas claves primarias). Para resolver los conflictos incompatibles se suele elegir uno sobre el otro, o buscar un intermedio que cumpla con restricciones de ambos esquemas.

2.1.1. Match workflows

En [2] se propone un proceso más automatizable. Para encontrar las correspondencias se definen secuencias de *matchers* llamados *match workflows*. Un matcher es un algoritmo que determina la probabilidad de que dos partes de un esquema se refieran al mismo elemento de la realidad. Los match workflows usualmente constan de los siguientes pasos:

1. **Preprocesamiento**: en este paso se transforman los esquemas a un formato interno estándar para poder ser procesados. También pueden aplicarse otras técnicas como las detalladas más adelante en 2.2.1 y 2.2.2 para acelerar el procesamiento.

2. **Ejecución de matchers**: es la parte central del procesamiento. La ejecución de los matchers puede ser:

   a) **Secuencial:** Donde el resultado de cada matcher es utilizado por el siguiente. Es útil cuando el siguiente matcher puede aprovechar el resultado del anterior.

   b) **En paralelo:** Se ejecutan todos independientemente. Permite que se ejecuten al mismo tiempo, mejorando el tiempo de procesamiento.

   c) **Mixta:** Combina los dos modos anteriores. Provee la mayor flexibilidad y permite definir flujos más complejos.

3. **Combinación de resultados**: los matchers arrojarán distintos resultados que deben ser unificados para tomar una decisión final. Esta podría ser tomar la unión, la intersección, ponderar los resultados de cada matcher, etc.

4. **Selección de correspondencias**: a partir del paso anterior se seleccionan las correspondencias finales. Una estrategia bastante común es tomar los pares cuya probabilidad de corresponder al mismo elemento de la realidad supera determinado umbral. Como el schema matching es una parte muy delicada que va a afectar el resto del proceso de integración, los resultados suelen presentarse como sugerencia que el usuario final tendrá que confirmar o corregir antes de finalizar.

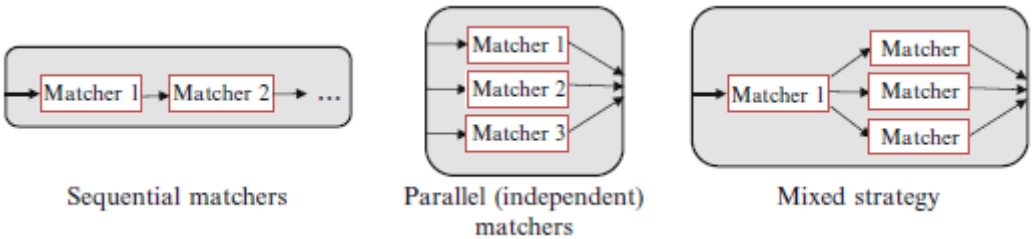

Figura 2.2: Formas de ejecución de matchers (extraído de [2])

2.1.2. Matchers

Existen variados algoritmos de matching, que se dividen en dos grandes grupos: los basados en metadatos y los basados en instancias.

2.1.2.1. Basados en metadatos

Los algoritmos basados en metadatos explotan características como restricciones de integridad, tipos de datos, relaciones entre elementos del esquema o nombres y descripción de elementos. Para esto último se puede utilizar información auxiliar como diccionarios, thesauri o listas de palabras para identificar elementos con nombres relacionados. En [16], se presenta un enfoque ordenado que describe como utilizar técnicas lingüísticas para el schema matching. Sus pasos son:

1. Preprocesamiento:

    a) Tokenización: los strings con varias palabras son partidos en listas de palabras o tokens.

    b) Eliminación de palabras vacías: se eliminan las palabras como "la", "el" y "de" ya que son irrelevantes.

    c) Eliminación de carácteres especiales: se eliminan carácteres como "-" y "/" que también son irrelevantes.

    d) Expansión de abreviaciones: se utiliza un diccionario de abreviaciones para expandirlas y de esta manera estandarizar todas las ocurrencias.

    e) Normalización de palabras a una forma estándar: las distintas representaciones de una misma palabra son llevadas a una única común, y los verbos son llevados a una misma forma verbal.

2. Ejecución de matchers de similaridad sintáctica: se ejecutan distintos algoritmos que evalúan la similaridad sintáctica de los elementos. En 2.2.3.1 se detallan algunos de estos algoritmos.

3. Ejecución de matchers de similaridad semántica: Se ejecutan distintos algoritmos que evalúan la similaridad semántica de los elementos. Se dividen en dos tipos:

    a) Basados en caminos: por ejemplo, un algoritmo calcula la distancia más corta entre los conceptos siguiendo una jerarquía ES-UN (p. ej. vaca es un animal) utilizando algún tipo de diccionario léxico como puede ser WordNet [20]. Otro es la medida de Wu and Palmer [21], que toma la distancia al ancestro en común mas cercano en la jerarquía.

    b) Basados en descripción: Se calcula el solapamiento de las descripciones de los distintos términos por ejemplo contando las palabras en común entre las descripciones. Lesk [10] utiliza este solapamiento para automatizar la desambiguación de palabras.

Si bien la mayoría de la literatura se centra en algoritmos que usan los nombres y descripciones de los elementos, también se pueden tener en cuenta las restricciones de integridad y tipos de datos, por ejemplo, asignando determinado peso extra cuando dos columnas tienen restricción de unicidad, cuando no pueden tener valores nulos o cuando contienen el mismo tipo de datos.

En [14] también se desarrolla toda una metodología de integración que utiliza, entre otras cosas, las relaciones entre los elementos de los esquemas (más correspondencias

indicadas por los usuarios), pero se centra en la integración de varias vistas en una vista global manteniendo las fuentes sin modificar, y requieren una gran cantidad de entrada de los usuarios para definir las correspondencias. Básicamete, los usuarios definen las correspondencias de las vistas, y el método encuentra una vista global con buenos atributos de calidad que las integra.

2.1.2.2. Basados en instancias

Los algoritmos que se basan en instancias determinan la similaridad de dos conceptos en base a la similaridad de sus datos. Prometen gran calidad en las correspondencias, pero como se explica en [2], es difícil conseguir suficientes datos de calidad como para lograrlo, por lo que allí se cataloga como método complementario.

El caso más simple es cuando se sabe que las dos fuentes a comparar comparten las mismas instancias (o la gran mayoría) en los datos. En este caso simplemente se comparan las entidades y se determina su correspondencia cuando tienen cierto porcentaje de instancias en común. Sin embargo esto no siempre sucede, por lo que también se define un caso general.

En un caso general, determinar la similaridad requiere determinar la similaridad entre las instancias, que es una variación del problema de data matching (2.2) detallado más adelante. Un enfoque para este caso es colocar todas las instancias de una entidad en un documento virtual y medir estos documentos virtuales de alguna forma, como por ejemplo con TF/IDF (Term Frequency/Inverse Document Frequency), generando un vector con la relevancia de las palabras de los documentos, y comparando el vector con el de otro documento.

2.1.3. Técnicas para escalar

Para cuando el tamaño de los esquemas a integrar hacen que la performance sea un problema, [2] propone algunas técnicas que reducen los tiempos necesarios para realizar el matching. Estas técnicas se pueden agrupar en tres grandes grupos: reducción del espacio de búsqueda, paralelización del procesamiento, y automatización de la configuración de los workflows.

2.1.3.1. Reducción de espacio de búsqueda

Este enfoque busca reducir la cantidad de comparaciones a realizar al momento del matching. Normalmente se compara todo con todo, pero esto tiene orden cuadrático en la cantidad de elementos, haciendo que para esquemas grandes pueda ser una complicación. Para reducir el espacio de búsqueda se aplican básicamente dos tipos de técnicas.

Eliminar pares al comienzo Este tipo de técnicas busca eliminar pares que tengan muy poca probabilidad de ser matches. Una opción es colocar un matcher al inicio del procesamiento con algún tipo de regla que sea poco costosa de evaluar para cada par y que descarte pares que tengan muy baja probabilidad de ser match. De esta forma, el resto de los matchers tendrán muchos menos pares que comparar.

Particionar el espacio de búsqueda Estas técnicas reducen la cantidad de operaciones particionando los esquemas de tal forma que cada partición del esquema A se compare con un subconjunto de particiones del esquema B. Si esto se hace correctamente, se logra reducir notablemente la cantidad de comparaciones a realizar. Si bien existen algunas implementaciones (por ejemplo la de COMA++ [13]), encontrar la mejor forma de particionar sigue siendo un problema abierto.

2.1.3.2. Matching en paralelo

Hay dos formas básicas de paralelizar: paralelizar la ejecución de matchers distintos (inter-matcher) y paralelizar la ejecución dentro de un matcher (intra-matcher).

En la paralelización inter-matcher la mejora en performance está limitada a la cantidad de matchers independientes en el flujo. No se puede paralelizar matchers que dependan uno de otro, y si no hay muchos matchers independientes, la mejora es marginal. Tampoco mejora la cantidad de memoria necesaria, ya que cada matcher tiene que evaluar los esquemas enteros.

En cambio, al paralelizar el trabajo de cada matcher se logra escalar mucho mejor ya que no existe la limitación de cantidad de matchers independientes, se puede dividir los pares a comparar tanto como se quiera. También mejora en el uso de memoria, ya que cada hilo requiere una porción de memoria menor que la necesaria para comparar los esquemas enteros.

2.1.3.3. Auto-tuning de workflows

Armar un workflow requiere que el usuario lo configure completamente, desde elegir qué matchers utilizar y en qué orden, elegir qué campos comparar y con qué algoritmos, elegir los parámetros de los algoritmos, umbrales, pesos, etc. Toda esta configuración requiere de un gran esfuerzo y conocimiento por parte del usuario para obtener buenos resultados, que quizás no lleguen a ser óptimos. Por esta razón se han desarrollado algunas técnicas (explicadas en [2]) que permiten automatizar la configuración de workflows.

Una forma de lograr esto es aplicar machine learning sobre trabajos de integración anteriores. Si se pudiera contar con una cantidad suficiente de configuraciones de workflows anteriores aplicadas sobre dominios parecidos (dado que los resultados para una configuración dada dependen de gran manera del dominio sobre el que se aplica) y con los resultados obtenidos de cada uno, sería posible inferir qué configuraciones logran los mejores resultados. Claramente, la mayor complicación para este enfoque es justamente obtener los datos de entrenamiento para aplicar machine learning.

Otro enfoque que requiere algo de trabajo manual pero es más fácil de aplicar que el anterior es extraer una parte representativa de los esquemas y resolver las correspondencias a mano. De esta forma, se cuenta con una muestra 100 % correctamente matcheada del problema, y se puede aplicar machine learning supervisado. El algoritmo intentará resolver nuevamente esa parte de los esquemas probando distintas configuraciones y se quedará con la que devuelva los resultados más parecidos a los de la muestra resuelta a mano.

2.2. Data Matching

Como se mencionó anteriormente, Data Matching es la segunda etapa en la integración de datos y es aquí donde se deben identificar aquellos registros que se corresponden con la misma entidad del mundo real.

En data matching se asume que la *extracción de información* ya fue previamente realizada y que los registros están bien persistidos y definidos en un archivo o base de datos.

El data matching se divide en cinco subtareas principales [4]:

1. **Preprocesamiento de los datos**: en este paso nos aseguramos que los datos de las fuentes queden en un formato estándar y consistente.
2. **Indexación**: comparar cada registro con todos los de otra base de datos tiene complejidad cuadrática lo que haría al proceso inviable. Este paso tiene como objetivo reducir dicha complejidad mediante el uso de estructuras que hacen más eficiente y efectiva la generación de pares de registros candidatos que posiblemente correspondan a la misma entidad y así reducir el espacio de comparaciones.
3. **Comparación de registros y campos**: en esta tercer etapa es donde se comparan los registros y se define la similitud entre los pares de registros comparados.
4. **Clasificación**: en este paso los pares de registros candidatos son clasificados en match, non-match o en match potencial (dependiendo del modelo de decisión) para los cuales habría que hacer una revisión manual.
5. **Evaluación**: en este paso se evalúa la calidad y la completitud de los datos matcheados.

2.2.1. Preprocesamiento

Se asume que los atributos en las bases de datos de entrada contienen valores que están separados mediante espacios. A estos valores se los denomina *tokens*.

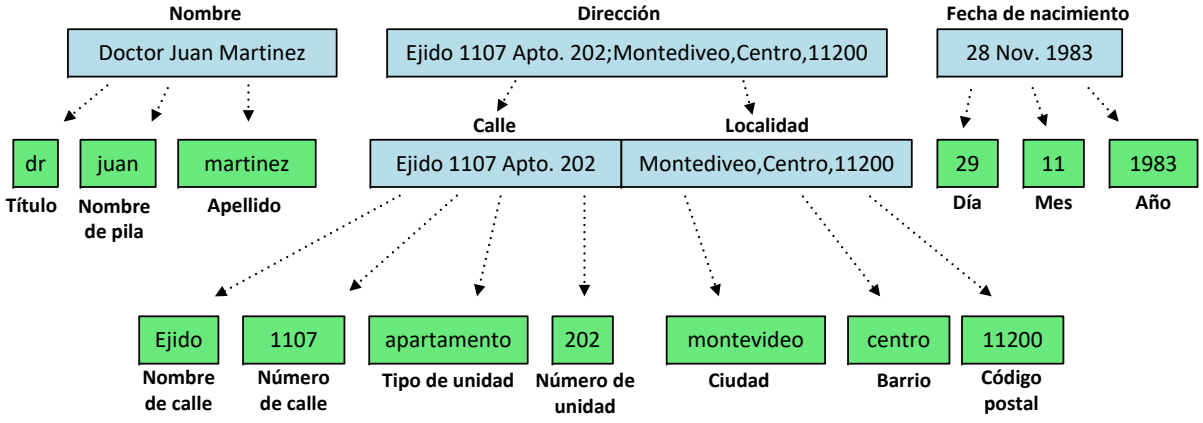

Figura 2.3: Ejemplo de preprocesamiento

El preprocesamiento a su vez consta de 4 subtareas:

1. **Remover carácteres no deseados y tokens**: este es el paso de la limpieza. Por ejemplo transformar todo a minúsculas, o corregir números ordinales (1ero a primero, etc.), remover carácteres como ? ' ", etc. Otra de las cosas que se debe hacer en este paso es reemplazar todas las ocurrencias seguidas de espacios blancos en una sola.

2. **Estandarización y tokenización**: en este segundo paso se trata de estandarizar los tokens detectando y corrigiendo errores o variaciones tipográficas conocidas, o reemplazando variaciones por su versión completa o sobrenombres en nombres completos (ej. Nate → Nathan).

   Aquí los tokens o grupos de tokens son analizados mediante tablas de look-up que contienen errores y variaciones conocidas y son reemplazados por su correspondiente representación (por ej. universidad de la reública := uni república, universidad república).

Etiqueta	**Descripción**	**Componente**
CP	Código postal	Dirección
TU	Tipo de unidad	Dirección
TI	Palabra de título (ej. 'sra', 'sr', 'ing', 'dr', etc.)	Nombre
NM	Nombre de persona masculino	Nombre
NF	Nombre de persona femenino	Nombre
AP	Apellido	Nombre

Tabla 2.1: Ejemplo de algunas etiquetas a utilizar en la tokenización

   Por otro lado en el proceso de tokenización a cada token le es comúnmente asignado una o más etiquetas de acuerdo al tipo de token donde fue encontrado en la tabla de lookup, o basado en alguna regla predefinida. Las etiquetas serán usadas en el próximo paso.

3. **Segmentación en campos**: el tercer paso es segmentar los valores de los atributos tokenizados y etiquetados en campos de salida que sean adecuados para el data matching. Es el paso más desafiante del preprocesamiento porque en general pueden haber varias asignaciones.

   El objetivo es que cada campo de salida contenga una sola pieza de información compuesta por uno o pocos tokens en vez de tener muchos valores en un solo atributo. Por ejemplo, a una dirección que contenga mucha información la podemos dividir en ciudad, departamento, código postal y país. A una fecha en: día, mes y año, etc.

   Pueden surgir ambigüedades al realizar la segmentación (p. ej. en el nombre Hernán Guzmán Esteves, Guzmán puede ser el segundo nombre o el primer apellido) y por eso existen varias técnicas para distintos tipos de datos de entrada, como nombres, direcciones, etc. Esta descomposición se ilustra con un ejemplo en la figura 2.3.

Normalmente se utilizan dos técnicas principales:

- *Segmentación basada en reglas*: se definen reglas en forma de "si token tiene etiqueta entonces le asigno un campo de salida determinado". Estas técnicas son usadas en atributos de los cuales se conoce bien su estructura, como teléfonos o nombres. Ejemplo:
  - si el token i contiene la etiqueta 'TI' entonces lo asigno como campo de salida "título".
  - si el token i contiene la etiqueta 'NM' y el token $i+1$ contiene la etiqueta 'AP' entonces asigno el token i al campo de salida "nombre" y el token $i+1$ a "apellido".

  También se pueden generar reglas automáticamente utilizando métodos de aprendizaje de reglas si existen datos de entrenamiento.
- *Segmentación basada en estadísticas*: se basan en la distribución de probabilidades que dan información acerca de la probabilidad de cual token debería ser asignado a cuál campo. Estas probabilidades son aprendidas desde datos de entrenamiento.

4. **Verificación**: este paso es opcional y consiste en verificar la correctitud de los valores asignados a los campos de salida y validarlos, p. ej. consultando bases de datos externas de direcciones.

2.2.2. Indexación

Este paso es un paso muy importante debido a que la comparación detallada de cada registro con todos los otros puede ser computacionalmente muy costosa. El objetivo de este paso es reducir lo más posible la cantidad de pares de registros que van a ser comparados en detalle, removiendo pares que es probable que no sean matches reales.

Lo que comúnmente se utiliza es una clave de bloque (*blocking key, BK*). Esta clave se define en función de los valores de un atributo o de un conjunto de atributos, por ejemplo en el blocking estándar si se elige la clave de bloque como el código postal entonces todos los pares que tengan el mismo código postal van a ser comparados.

Por lo tanto es clave definir la técnica de indexación teniendo en cuenta varias cosas, como la completitud de los atributos y la distribución de los valores en los atributos.
Para evaluar las distintas técnicas existen 3 métricas:

- Ratio de reducción: cuántos pares son comparados usando una técnica de indexación en relación al total de comparaciones posibles
- Completitud de pares (*recall*): cuántos pares de registros que se sabe que son matches verdaderos son incluídos en los pares de registros candidatos.
- Calidad de pares (*precision*): cuántos de los pares de registros candidatos corresponden a matches verdaderos.

Definiendo la clave de bloque La definición depende mucho del tipo de datos a ser matcheados, por ejemplo la similitud entre atributos puede ser definida en función de la forma de los carácteres, sonido, o similitud numérica (p. ej. edades cercanas).
Se deben tener en cuenta varias cosas al definir la clave de bloque:

- Calidad de los datos: un atributo que es utilizado para definir la clave de bloque debe ser de calidad, ya que cualquier error o la falta del mismo resultará en una clave errónea lo que significa que será insertado en el bloque erróneo.
- Frecuencia de los valores: se debe tomar en cuenta también la frecuencia de los valores en los atributos. Por ejemplo, si definimos una clave de bloque como el apellido, sabemos que "Rodríguez" es un apellido común lo que resultará en un conjunto de pares muy grande. Una buena clave debería generar bloques más o menos uniformes en cantidad de pares.
- Balance entre el número y tamaño de bloques: cuanto más específica la clave de bloque más chicos serán los bloques y por lo tanto menos comparaciones serán hechas.

2.2.2.1. Blocking estándar

En esta técnica cada identificador de cada registro es insertado en un solo bloque, mientras que las demás técnicas pueden insertar un registro en varios bloques. Una forma eficiente de implementarlo es definir un valor de clave de bloque (*blocking key value*, BKV) para cada registro e insertarlo en ese bloque. Por ejemplo, se puede establecer como clave de bloque la columna de código postal, entonces de esta forma todos aquellos registros que tengan el mismo código postal serán insertados en un único bloque.

2.2.2.2. Sorted Neighborhood Approach (SNA)

En esta estrategia, en vez de definir los bloques según la clave, se define una clave de ordenamiento (*sorting key*) de la misma forma que se define una clave de bloque y se ordenan los registros según la misma. Luego se define una ventana deslizante $w > 1$ y cada bloque queda definido según los registros dentro de esa ventana en un paso dado.

Mientras que una clave de bloque debe balancear la calidad de los pares candidatos generados con el tamaño de los bloques generados, la definición de la clave de ordenamiento debe ser más granular y específica para juntar los registros similares.

Para grandes bases de datos, esta técnica tiene un costo computacional de $O(n)$, es decir que es lineal en el tamaño de las bases a matchear. Sin embargo, hay que tomar en cuenta el ordenamiento también, lo cual se puede realizar en $O(nlog(n))$.

Alternativa Podemos definir una sorting key de forma "apellido-nombre", pero si el apellido y nombre son comunes pueden quedar afuera algunos registros con el mismo valor para la clave de ordenamiento. Una variación alternativa de esta técnica es definir un índice invertido con los valores de la clave de ordenamiento, ordenarlas, y definir una ventana deslizante sobre estos valores.

Resultados experimentales han mostrado que esta variación puede llevar a incluir más true matches en el conjunto de registros candidatos con el costo de una mayor cantidad de comparaciones [4].

El blocking estándar puede ser visto como SNA con una ventana que se mueve cada n pasos en vez de a uno definiendo bloques que no se solapan.

2.2.2.3. Indexación basada en Q-grams

Para datos que están sucios y contienen muchos errores y variaciones, tanto el blocking estándar como el SNA podrían no insertar los registros en el mismo bloque, por ejemplo si el comienzo del valor de la clave de ordenamiento es diferente para dos variaciones del nombre.
Un Q-gram es un substring de q carácteres. P. ej. christen → ch, hr, ri, is, st, te, en.
En este tipo de indexación se crea una clave de bloque, y para cada valor de la clave de bloque se definen sublistas de los q-gram de ese valor. Primero se remueve un Q-gram y luego se va removiendo recursivamente. Por ejemplo: si tenemos una clave de bloque definida por el apellido y para un registro el valor de la clave es "miller", y utilizamos bigrams con un mínimo de sublistas de largo 3 obtenemos las siguientes sublistas:

- mi,il,ll,le,er
- il,ll,le,er - mi,ll,le,er - mi,il,le,er - mi,il,ll,er - mi,il,ll,le
- ll,le,er - il,le,er - il,ll,er - il,ll,le - mi,le,er - mi,ll,er - mi,ll,le - mi,il,er - mi,il,le - mi,il,ll

Las claves del índice se definen como la concatenación de estas sublistas.
Una gran desventaja de este método es que una gran cantidad de sublistas (y por lo tanto claves de índice) son generadas, y cada registro es probable que sea insertado en muchos bloques. Además, la generación recursiva de sublistas es computacionalmente costosa.

2.2.2.4. Indexación basada en Array de Sufijos

Este método es parecido al anterior. Tiene los mismos objetivos de lidiar con errores y variaciones en los valores de la clave de bloque. Lo que se hace es generar varios sufijos y cada sufijo forma parte de una clave de índice. Por ejemplo, si en un registro particular el valor de la clave de bloque es katherina y el mínimo largo de sufijos es 4 obtendremos los sufijos: katherina, atherina, therina, herina, erina, rina.

Se debe definir, además, el mínimo largo del sufijo y limitar el tamaño de los bloques mediante otro parámetro que establezca el tamaño máximo de los bloques. Todos los bloques que contengan más de esa cantidad definida de registros serán borrados. Esta estrategia de poda se basa en que cada identificador del registro es probable que sea insertado en varios bloques.

2.2.3. Comparación de Registros y Campos

En este paso es donde los registros candidatos son comparados y se generan los vectores de similitud.

$s = sim(a_i, a_j)$ es una función de similitud que calcula la similitud entre los dos atributos a_i y a_j. Se asume que $0 \leq s \leq 1$, y es 1 cuando son exactamente iguales, 0 cuando son totalmente distintos y un valor entre 0 y 1 si existe determinada similitud intermedia. Por lo tanto el vector de similitud queda conformado por una tupla de valores entre 0 y 1 donde el largo de la tupla es la cantidad de atributos en los registros.

2.2.3.1. Comparación de strings

Distancia de edición Levenshtein En esta técnica de comparación se cuenta la cantidad menor de operaciones de edición requeridas para convertir un string s_1 en otro s_2 ($dist_{lev}(s_1, s_2)$). Cuenta la menor cantidad de inserciones, borrados y sustituciones de carácteres que son requeridas para convertir un string en el otro. En su forma básica, cada operación tiene el mismo costo unitario asociado.

Finalmente la similitud es definida como:

$$sim_{lev}(s_1, s_2) = 1.0 - \frac{dist_{lev}(s_1, s_2)}{max(|s_1|, |s_2|)}$$

También es posible definir costos particulares, por ejemplo si se sabe que los datos provienen de OCR entonces el costo de sustituir "n" en "m" debería ser bajo en comparación a otros.

Distancia de edición Smith-Waterman Es otra técnica de comparación por distancia de edición pero es más compleja que la de Levenshtein dado que incluye cinco operaciones con distinto puntaje y además existen puntajes negativos.

Q-grams Ambos strings a comparar se parten en substrings de largo q (q-grams) y se cuentan cuantos de estos q-grams ocurren en ambos strings de entrada.

Padded Q-grams La primera variación es utilizar un carácter especial para el comienzo y otro para el final. Por ejemplo, si utilizamos bigrams en el string "Hernan" obtendriamos: *h, he, er, rn, na, an, n+ (* y + son los carácteres especiales).

Positional Q-grams La segunda variación, es añadir información posicional en los q-grams. Cada q-gram se da junto a su posición de ocurrencia en el string. Cuando se cuentan los q-gram en común sólo se cuentan los que ocurren en la misma posición o a determinada distancia.

Luego de que los q-grams son generados en ambos strings s_1 y s_2 la similitud se calcula basada en la cantidad de q-grams que ambos strings tienen en común. Sea c_{com} esta cantidad, c_1 la cantidad de q-grams en s_1 y c_2 la cantidad de q-grams en s_2 entonces la similitud entre ambos strings puede ser calculada utilizando uno de tres coeficientes:

- **Coeficiente de solapamiento**

$$sim_{overlap}(s_1, s_2) = \frac{c_{com}}{min(c_1, c_2)}$$

- **Coeficiente de Jaccard**

$$sim_{jaccard}(s_1, s_2) = \frac{c_{com}}{c_1 + c_2 - c_{com}}$$

- **Coeficiente de Dice**

$$sim_{dice}(s_1, s_2) = \frac{2 \times c_{com}}{c_1 + c_2}$$

Longest Common Substring Consiste en encontrar y remover el string más largo en común s_c. El proceso se repite hasta que el substring en común mas largo encontrado sea menor a l_{min} (comunmente 2 o 3). Luego la suma de los largos de todos los strings comunes encontrados l_c es usado para calcular el valor de similitud utilizando los coeficientes previamente mencionados (Sobrelapamiento, Jaccard, Dice).

Jaro Combinación de distancia de edición y q-gram. Cuenta la cantidad de carácteres que son comunes en una cierta ventana de carácteres.

Winkler Variación de la comparación Jaro que incrementa la similitud entre dos strings si el comienzo es el mismo y las diferencias ocurren en el medio y final de ambos strings.

Jaccard El coeficiente de Jaccard se define como:

$$sim_{jaccard} = \frac{|A \cap B|}{|A \cup B|}$$

Calcula la similitud entre dos conjuntos de ítems (o tokens), A y B, como el número de ítems contenidos en la intersección de los dos dividido por el número de ítems contenidos en la unión de ambos conjuntos.

Monge-Elkan Diseñado específicamente para la comparación de strings con muchas palabras. Primero se extraen las palabras en ambos strings y luego se encuentran los pares que mejor matchean utilizando otra técnica de comparación (sim'). La similitud queda definida como:

$$sim_{monge\_elkan}(s_1, s_2) = \frac{1}{|A|} \sum_{i=1}^{|A|} \max_{j=1}^{|B|} sim(A_i, B_i)$$

2.2.3.2. Comparación numérica

Para comparar números n_1 y n_2 existen dos estrategias. En la primera se establece la máxima diferencia absoluta d_{max} que se tolerara independientemente de sus valores y la similitud se define como:

$$sim_{num\_abs} = \begin{cases} 1.0 - \left(\frac{|n_1 - n_2|}{d_{\text{máx}}}\right) & \text{si } |n_1 - n_2| < d_{\text{máx}} \\ 0.0 \text{ si no} \end{cases}$$

Por ejemplo, si tenemos valores de precios y lo máximo que toleraremos como diferencia son \$1000 y tenemos que comparar \$1000 y \$1500 entonces la similitud es $1 - 500/1000 = 0.5$. Por otro lado, si vamos a comparar \$850.000 y \$850.250 entonces la

similitud es $1 - 250/1000 = 0.75$.

Como se puede ver, existe un problema y es que la similitudes anteriores no dependen de la diferencia relativa a su valor y para esto se utiliza la otra estrategia que toma en cuenta esta diferencia relativa. Quisierámos que en este último caso la similitud sea mayor. Primero se define lo que se llama la diferencia de porcentaje como:

$$dp = \frac{|n_1 - n_2|}{max(|n_1|, |n_2|)} \cdot 100$$

Y luego, como en método por diferencia absoluta, se define un máximo porcentaje de diferencia dp_{max} y la similitud es calculada como:

$$sim_{num\_porc} = \begin{cases} 1.0 - \left(\frac{dp}{dp_{\text{máx}}}\right) & \text{si } dp < dp_{\text{máx}} \\ 0.0 \text{ si no} \end{cases}$$

Utilizando este segundo método con $dp = 30\,\%$ para $n_1 = \$850.000$ y $n_2 = \$850.250$ obtenemos una similitud de 0.9901.

2.2.3.3. Comparación de fechas, edades y tiempo

Las fechas pueden ser vistas como casos especiales de datos numéricos. Para fechas, la estrategia más usada es calcular la diferencia en días y luego emplear una diferencia absoluta entre estos valores.

Sin embargo, en las fechas existe un problema y es si el día y mes están intercambiados. Existen casos donde esto se puede detectar pero en otros no, p. ej. 12/10/2010 y 10/12/2010.

2.2.3.4. Comparación de distancia geográfica

En vez de comparar directamente los strings de las direcciones, una alternativa para calcular la similitud de dos direcciones es calcular la distancia geográfica y usar este valor numérico en una comparacion numérica. Para esto se debe realizar una geocodificación inversa a partir de los strings.

2.2.3.5. Comparación de datos complejos

Se manejan distintas técnicas para datos complejos. Por ejemplo, para XML una posible estrategia es mapear los documentos XML en una tabla de base de datos relacional, donde cada documento es representado mediante varias tuplas de tipo cada una consistiendo de un valor y un nombre de tipo. Las que tengan el mismo tipo luego pueden ser comparadas con alguna de las técnicas vistas anteriormente.

Para datos multimedia, existen varias técnicas. Consisten en extraer features de un archivo de multimedia, por ejemplo un histograma de colores de una imagen y guardar dichas features en feature vectors. Luego las similitudes son calculadas comparando estos feature vectors.

2.2.4. Clasificación

Una estrategia de clasificación puede ser supervisada o no-supervisada. Las no supervisadas clasifican a los pares de registros basado en similitudes sin tener ninguna información respecto a los true matches o true non-matches. Las estrategias supervisadas, requieren datos de entrenamiento que son etiquetados como true matches o true non-matches.

2.2.4.1. Clasificación basada en umbrales

La forma más simple de clasificar matches y non-matches es sumar los valores de similitud en el vector de similitud (llamado "SimSum") y luego aplicar un umbral t de similitud de forma tal que:

$$SimSum[r_i, r_j] \geq t \Rightarrow [r_i, r_j] \rightarrow Match$$

$$SimSum[r_i, r_j] < t \Rightarrow [r_i, r_j] \rightarrow Non - match$$

También se puede usar otro umbral para definir un tercer grupo, el de matches potenciales. Ésta última forma de clasificar (con dos umbrales) es llamada la clasificación de Fellegi y Sunter.

Este método tiene dos desventajas mayores. La primera es que, todas las similitudes de los atributos contribuyen al $SimSum$ con el mismo peso. Esto se puede solucionar asignándole un peso a cada valor de similitud en el vector.
La segunda es que sumar las similitudes hace que la información detallada de las similitudes en cada atributo sea perdida.

2.2.4.2. Clasificación basada en reglas

En este tipo de clasificación se definen reglas de la forma $P \Rightarrow C$ donde P es un predicado que se aplica en los valores de similitud para un par de registros y C es el resultado de la clasificación. Por ejemplo una regla podría ser:

$$(s(GivenName)[r_i, r_j] \geq 0.9) \wedge (s(Surname)[r_i, r_j] = 1.0) \Rightarrow [r_i, r_j] \rightarrow Match$$

Para generar estas reglas un experto en el dominio podría manualmente generarlas, pero es un trabajo tedioso de realizar. Otra alternativa es aprender las reglas a partir de datos que consisten de pares de registros candidatos y su estado de match real.

2.2.4.3. Métodos de clasificación supervisados

Si existen datos de entrenamiento en forma de pares de registro con su estado de match real entonces un método de clasificación supervisada puede ser empleado para entrenar un modelo de clasificación usando estos datos. El modelo entrenado luego es usado para clasificar los pares de registros con un estado desconocido en cuanto al match.
Consiste en 3 pasos:

1. Una técnica de clasificación es seleccionada y un modelo de clasificación es construido a partir del entrenamiento del clasificador utilizando los datos de entrenamiento.

2. La precisión del modelo de clasificacion construido es evaluada mediante un conjunto de pares de prueba con el mismo formato que los datos de entrenamiento. Si la precisión reportada no es lo suficientemente buena entonces se vuelve al paso 1 para

cambiar algunos parámetros o emplear alguna otra técnica. Para distintos tipos de datos convienen distintas técnicas.

3. Cuando una precisión satisfactoria es alcanzada en este paso se comienza a clasificar nuevos datos no vistos anteriormente.

A continuación se presentan dos técnicas de clasificación supervisada.

Inducción de Árboles de Decisión La Inducción de Árboles de Decisión es una de las técnicas más populares usada en data mining y machine learning. Consiste en generar un árbol de decisión a partir de los datos de entrenamiento. Si tomamos como ejemplo los datos de entrenamiento de la tabla 2.2 se pueden deducir los dos árboles de ejemplo en la figura 2.4.

ID del par	Nombre	Apellido	NumDir	NomDir	Ciudad	DiaNac	MesNac	AñoNac	Match
(a1,b1)	0.6	0.8	0.0	1.0	0.6	0.5	0.5	1.0	✓
(a1,b2)	0.0	0.15	0.0	0.5	0.0	0.5	0.0	0.75	✗
(a2,b1)	0.2	0.0	0.0	0.1	0.15	0.0	0.0	0.75	✗
(a2,b2)	0.0	0.25	1.0	0.4	0.6	1.0	1.0	0.75	✓

Tabla 2.2: Datos de entrenamiento (tomado de [4])

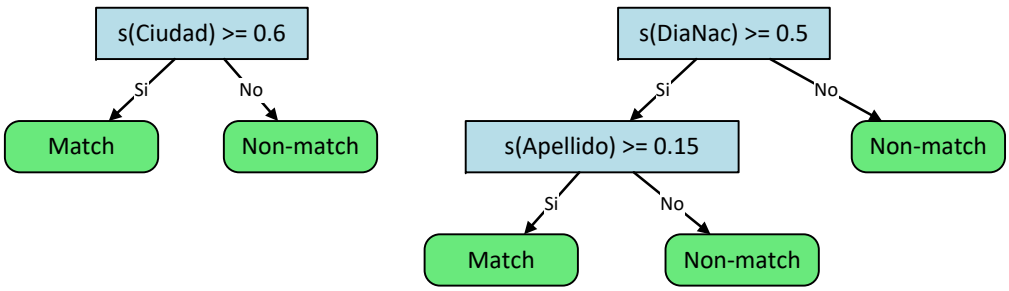

Figura 2.4: Dos árboles de decisión de ejemplo resultantes de cuatro vectores de comparación de entrenamiento de la tabla 2.2 (tomado de [4]).

Además, los árboles de decisión pueden ser convertidos en un conjunto de reglas.

Máquinas de vectores de soporte (SVM) Es una técnica desarrollada en los años noventa. Consiste en mapear los datos de entrenamiento con su estado real en un espacio multidimensional. Una frontera de decisión corrresponde a un hiperplano (una línea en dos dimensiones p. ej.) y una frontera de decisión óptima es aquella que tiene los márgenes más amplios a los registros de entrenamiento en ambas clases (match y non-match). El mapeo al hiperplano se realiza mediante las llamadas funciones kernel. En la figura siguiente (2.5) se ilustra un ejemplo en dos dimensiones.

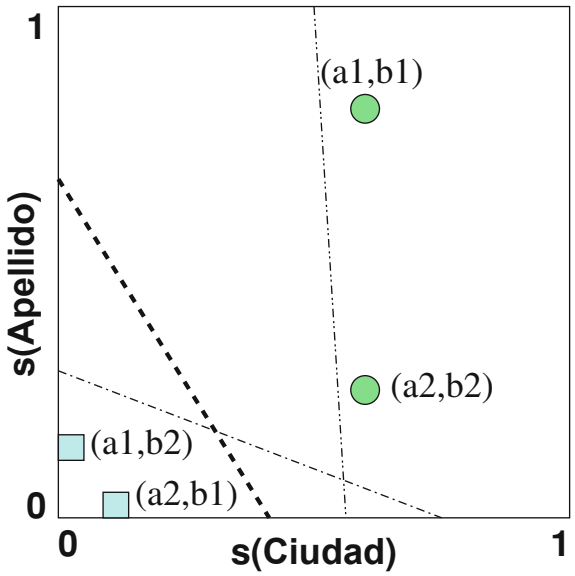

Figura 2.5: La línea punteada gruesa es la que divide las clases con mayor margen (SVM) (tomado de [4]).

2.2.4.4. El problema de Clausura transitiva

La clausura transitiva refiere a la situación cuando dos pares de registros (r_i, r_j) y (r_i, r_k) fueron clasificados como matches pero el par (r_j, r_k) fue clasificado como non-match. En bases de datos reales, este tipo de problemas suele ocurrir raramente porque el espacio de todos los valores posibles en los atributos es muy grande. La probabilidad de que los registros que no son matcheados y tengan una gran similitud entre ellos es muy chica.
Los siguientes métodos solucionan este problema ya que clasifican grupos de registros como matches en vez de pares individuales.

2.2.4.5. Estrategias basadas en Clustering

Este método visualiza la clasificación como una estrategia de clustering (agrupamiento) donde cada clúster contiene los registros que refieren a una entidad. El objetivo es crear grupos donde todos los registros que estén dentro de esos grupos sean similares entre sí, mientras que registros en distintos grupos no lo sean. Generalmente este método se realiza de forma no supervisada. Es conveniente para la deduplicación ya que para dos o más bases de datos todos los registros deben ser insertados en un conjunto común antes.

2.2.4.6. Clasificación Colectiva

Todos los métodos de clasificación vistos realizan decisiones locales, sin tener en cuenta las relaciones que tienen los registros con otros en la base de datos entera.
El primer paso es generar el grafo de relaciones (ejemplo en figura 2.6), que puede consistir de relaciones entre distinto tipos de entidades. Las relaciones puede ser 'hard' (donde se sabe con seguridad a partir de los datos) o conexiones que tienen probabilidad o peso asociado. Estos pesos pueden ser calculados mediante las similitudes calculadas cuando los pares son comparados.

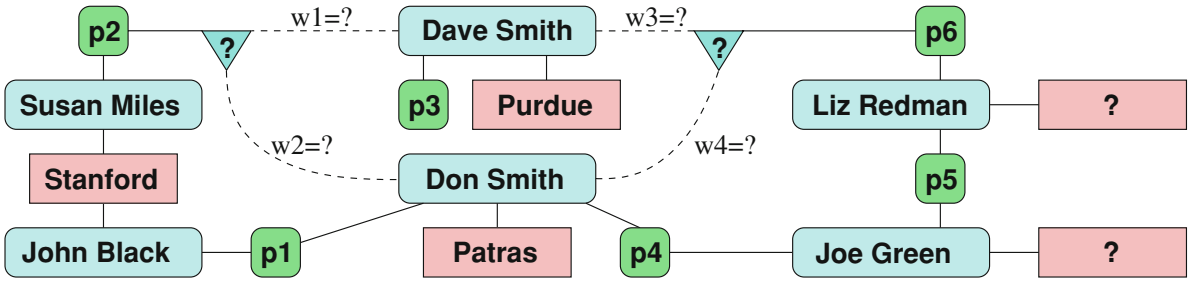

Figura 2.6: Ejemplo del matching colectivo basado en grafos de registros bibliográficos (extraído de [4]). La tarea es identificar si el autor 'D Smith' en los papers p2 y p6 refieren a Don Smith o a Dave Smith. Dado que Don Smith ha sido coautor de p1 con Jhon Black, que está afiliado a Stanford, y Susan Miles esta tambien afiliada a Stanford hay mayor probabilidad de que Don Smith, en vez de Dave Smith sea el coautor del paper p2 porque Dave Smith no tiene ninguna conexión con Stanford. De la misma forma, dado que Don Smith ha escrito el paper p4 con Joe Green, que ha sido coautor de p5 con Liz Redman hay mayor probabilidad de que Don Smith sea también el segundo coautor del paper p6 en vez de Dave Smith que no tiene conexión con Joe Green.

2.3. Data Fusion

En las primeras dos etapas de integración (schema matching y data matching) se solucionan los problemas de esquema e identificación de entidades. Sin embargo pueden quedar remanentes conflictos entre los valores de los registros clasificados como match, que ocurren cuando uno o más atributos equivalentes difieren en algún valor.

Dichos conflictos se solucionan en la etapa de data fusion, obteniendo como resultado un único registro a partir de cada grupo que contenga dos o más registros correspondientes a la misma entidad. Las dificultades que se deben superar residen en la existencia de valores nulos, contradicciones en los valores, problemas en metadatos, mantenimiento del linaje[1] e implementaciones de data fusion en manejadores de base de datos.

2.3.1. Conflictos entre los datos

Los conflictos entre los datos pueden clasificarse en incertidumbres y contradicciones [5].

Una incertidumbre es un conflicto entre un valor no nulo y uno o más valores nulos usados para describir la misma propiedad del objeto. Son causados por falta de información. En general se considera un caso más simple que el de las contradicciones debido a que a pesar de las distintas semánticas que puede tomar el valor null (desconocido, no aplicable o no autorizado) la mayoría de las técnicas y estrategias utilizadas para resolverlos se quedan con el valor no nulo a la hora de integrar.

Una contradicción es un conflicto entre dos o más valores no nulos diferentes que describen la misma propiedad de un objeto.

[1] Origen y transformaciones del dato a través del tiempo.

2.3.1.1. Clasificación de estrategias

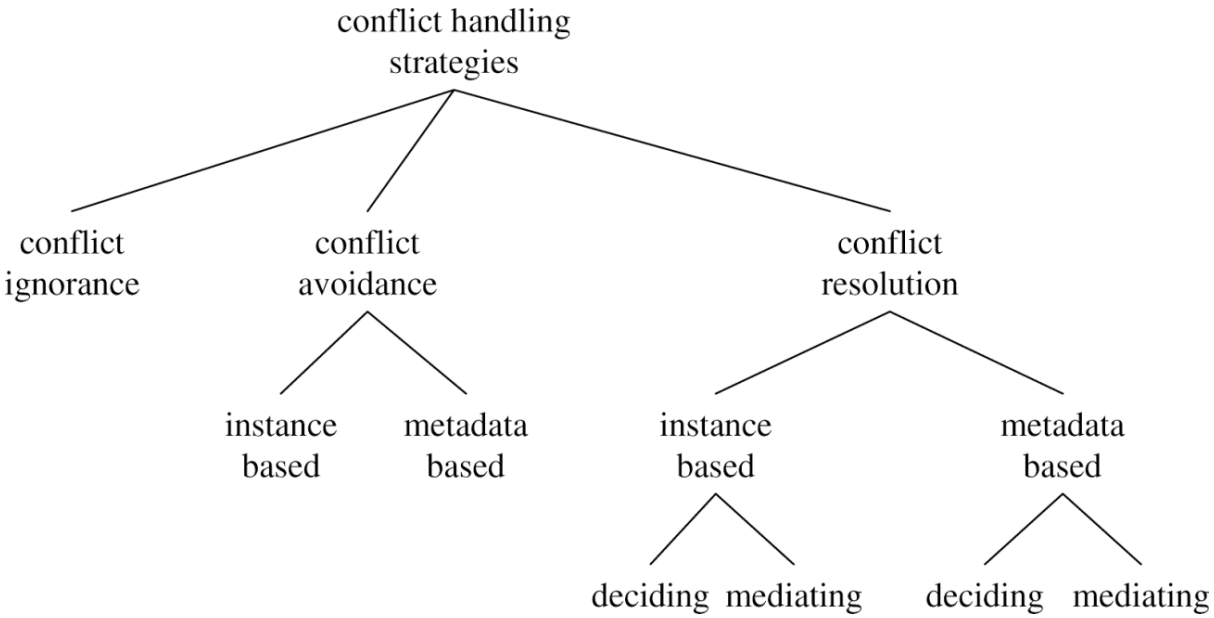

Figura 2.7: Clasificación de estrategias (de [5])

Estrategia	Clasificación	Descripción
PASS IT ON	Ignoring	Escala el conflicto al usuario o aplicación
CONSIDER ALL POSSIBILITIES	Ignoring	Crea todas las posibles combinaciones de valores
TAKE THE INFORMATION	Avoiding, Instance based	Prefiere valores sobre nulos
NO GOSSIPING	Avoiding, Instance based	Retorna solo tuplas consistentes
TRUST YOUR FRIENDS	Avoiding, Metadata based	Toma el valor de una fuente preferida
CRY WITH THE WOLVES	Resolution, Instance based, Deciding	Toma el valor mas recurrente
ROLL THE DICE	Resolution, Instance based, Deciding	Toma un valor al azar
MEET IN THE MIDDLE	Resolution, Instance based, Mediating	Toma un valor promedio
KEEP UP TO DATE	Resolution, Metadata based, Mediating	Toma el valor mas reciente

Tabla 2.3: Ejemplos de estrategias (de [5])

En las figuras 2.7 y 2.3 se presenta la clasificación de estrategias.

Las estrategias Conflict Ignoring ignoran el conflicto o no deciden qué hacer con el mismo. Un ejemplo es la estrategia Pass it On que consiste en pasarle todos los valores al usuario y que este decida.

Las estrategias Conflict Avoiding reconocen la existencia de conflictos pero no plantean soluciones para cada conflicto en particular. Sino que manejan los mismos aplicando una única solución para todos los datos, como por ejemplo preferir siempre la fuente más confiable (Trust your friends).

En contraste las estrategias Conflict Resolution tienen en cuenta todos los datos y metadatos antes de decidir cómo resolver el conflicto. Se subdividen en estrategias de decisión (toman un valor entre los posibles existentes) y de mediación (crean un nuevo valor) un ejemplo es “Keep up to date” que toma en cuenta metadatos de timestamp de la instancia y se queda con el más reciente.

Ambas estrategias, Conflict Avoiding y Conflict Resolution pueden a su vez clasificarse en Instance based si tienen en cuenta los valores en conflicto y Metadata based si tienen en cuenta metadatos como la confianza en la fuente o la frescura del dato.

2.3.2. Data Fusion Answers

Dado un sistema de información integrado la idea es realizar consultas que retornen respuestas denominadas Data Fusion Answers que pueden clasificarse en una de las siguientes categorías [8]:

- **Completa**: debe contener todos los objetos (completitud extensional) y todos los atributos (completitud intencional) de las diferentes fuentes.
- **Concisa**: debe contener todos los objetos del mundo real (concisión intencional) y los atributos semánticamente equivalentes (concisión extensional) una sola vez.
- **Consistente**: contiene todas las tuplas de todas las fuentes que son consistentes respecto a determinado conjunto de restricciones de integridad. En este sentido una respuesta consistente no implica completa, dado que pueden quedar afuera representaciones de objetos del mundo real que no cumplan las restricciones. Por otro lado si es concisa en forma extensional en caso de que una de las restricciones sea de clave en un identificador del mundo real.
- **Completa y consistente**: comprende todas las representaciones de objetos de la realidad de todas las fuentes cumpliendo con restricciones de clave en algún atributo del mundo real. Dicho resultado contiene todos los atributos de todas las fuentes combinando los semánticamente equivalentes en uno.

El objetivo es el de desarrollar sistemas de información integrados cuyas respuestas sean completas y consistentes.

2.3.2.1. Operadores relacionales aplicados a Data Fusion

En Data Fusion se utilizan operadores relacionales para fusionar datos de diferentes fuentes. Se asume que los operadores son binarios entre dos tablas, si bien la extensión a más de dos tablas es directa no siempre se cumple la asociatividad [8].

Una de las opciones al momento de fusionar los duplicados es utilizar joins. Este enfoque aumenta la completitud intencional debido a que todos los atributos son incluidos pero esto no garantiza la completitud por extensión excepto en el caso del `FULL OUTER JOIN`. Con respecto a la concisión extensional, el enfoque "join" funciona bien siempre que confiemos en identificadores globales únicos y no existan atributos duplicados en las fuentes.

Otro enfoque es el de utilizar los operadores de unión. Son el enfoque natural para conseguir completitud extensional, pero en contraste con el `JOIN`, la operación de unión no da buenos resultados respecto a concisión.

Finalmente el autor Naumann propone el diseño e implementación de un nuevo operador denominado `FUSE BY` [7]. Dicho operador brinda una simple forma de expresar consultas para fusionar múltiples tuplas que describen el mismo objeto de la realidad en una única

tupla, mientras que resuelve las incertidumbres y contradicciones. Se basa en la sintaxis estándar de SQL y refleja similitudes en sintaxis y semántica con la instrucción `GROUP BY`. El `FUSE BY` fue implementado en un prototipo que se presenta a continuación.

2.3.3. Prototipo: The Humboldt-Merger (HumMer)

Es un sistema de información integrada que permite leer y combinar datos relacionales, archivos XML y datos no estructurados en una estructura común [15].

Funciona en tres pasos, primero schema matching eliminando la heterogeneidad de las distintas fuentes de datos alineando los atributos correspondientes. Luego utiliza técnicas de detección de duplicados y finalmente realiza la fusión de los datos resolviendo los conflictos y obteniendo una única, consistente y limpia representación de los mismos.

Se plantea tanto como un framework permitiendo a los investigadores implementar nuevas partes del proceso de integración e instalarlas, así como un sistema de integración de datos.

Posee dos interfaces gráficas, un Wizard GUI (que se observa en la figura 2.8) y un SQL GUI.

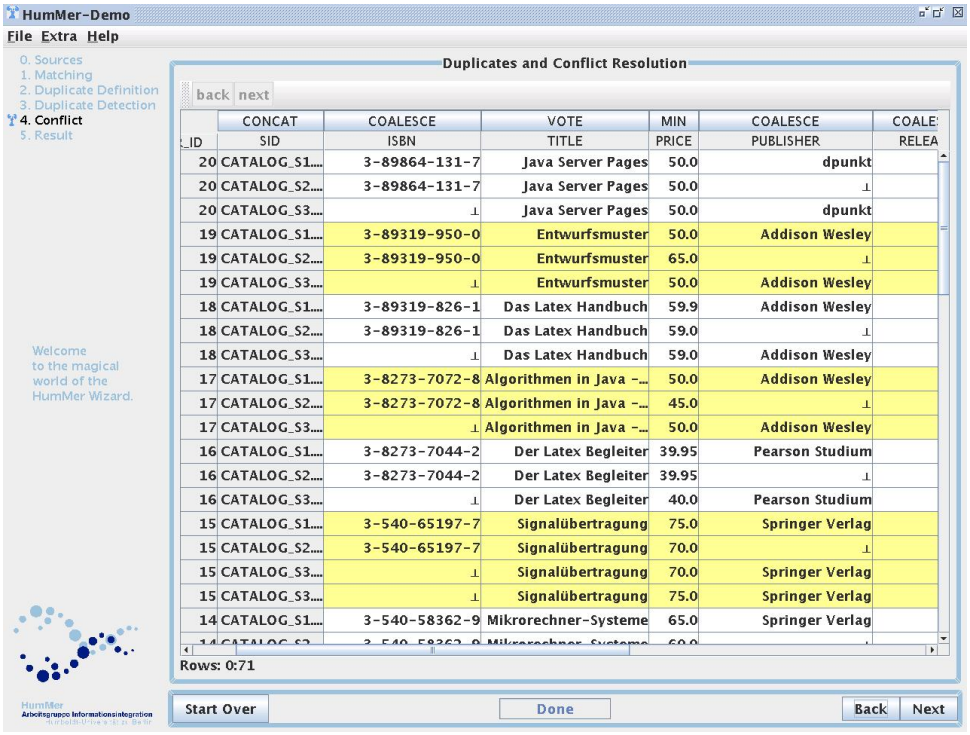

Figura 2.8: Captura de pantalla de la herramienta HumMer en la etapa de resolución de conflictos en los matches (extraída de [15])

El Wizard es manejado en base a la interacción con el usuario y permite visualizar cada paso del proceso de integración:

1. Elección de fuentes de datos.
2. Matcheo de atributos e integración de las fuentes de datos.

3. Selección de los atributos que identifican al objeto.
4. Detección de duplicados.
5. Elección de resolución de conflictos y fusión.
6. Chequeo del resultado.

Por otro lado la interfaz gráfica SQL GUI provee de una implementación de la instrucción `FUSE BY` como se mencionó anteriormente.

Fue posible comunicarse con el profesor Naumann, director del proyecto Hummer, acerca del estado actual del mismo. Naumann explicó que el proyecto estaba finalizado y no se continuaba trabajando en el. Se le solicitó acceso a la herramienta, pero no fue otorgado.

2.4. Integración en Big Data

Cuando se entra en el campo de la big data, muchos algoritmos que trabajan sobre los datos comienzan a ser muy costosos de ejecutar. En esta sección se presentan técnicas que pueden mejorar las tareas de integración sobre big data.

En la literatura en general, y en particular en [6] se habla de las siguientes "dimensiones V" de desafíos que presenta la integración de big data.

- **Volumen:** quizás es el primero que se viene a la mente. Se refiere al volumen de datos.
- **Velocidad:** se refiere a tanto la velocidad de generación de datos nuevos, como por ejemplo los datos de sensores; así como a la velocidad de actualización en los datos en ciertos escenarios y a la velocidad con la que aparecen nuevas fuentes de datos.
- **Variedad:** los datos pueden provenir de una gran cantidad (decenas, cientos o miles) de fuentes distintas, que pueden diferir tanto en la estructura de los datos como en la semántica.
- **Veracidad:** describe los distintos niveles de calidad de los datos provistos por distintas fuentes. Distintas fuentes proveen datos con distintas calidades que por ejemplo pueden variar en frescura, precisión o completitud.

En [6] se mencionan varias características de big data que permitirían facilitar la integración de esta información. Éstas son:

- **Redundancia:** en muchos casos existe una gran redundancia, ya que muchas fuentes proveen los mismos datos. Esto permite por ejemplo atacar la dimensión de veracidad, ya que al existir muchas fuentes proveyendo los mismos datos, se podría hacer algún tipo de votación para determinar cuáles datos son correctos.
- **Datos históricos:** también llamado long data. Si se almacenan las evoluciones de los valores a través del tiempo de varias fuentes, técnicas de Data Matching podrían aprovechar el hecho que las evoluciones suelen ser graduales, y por ejemplo descartar posibles matches determinando que es muy poco probable que varíen tanto en un corto período de tiempo.

- **Plataformas para trabajar sobre big data:** Hadoop es un framework que permite trabajar en paralelo sobre datos distribuidos en múltiples nodos. Uno de sus modelos de programación es MapReduce, básicamente opera en dos etapas: la etapa de map, donde los datos asignados a ese nodo son procesados de alguna forma para luego ser enviados a la etapa reduce con ciertos criterios; y la etapa reduce, donde cada reducer aplica cierta operación a los datos y tiene como salida una versión más reducida.

  Hadoop junto a MapReduce y el resto de algoritmos y técnicas que nacieron para trabajar con datos de estas características se pueden aprovechar en las tareas de integración. Se verán algunas técnicas que los usan.

2.4.1. Schema Matching probabilístico

A priori no se conoce la semántica de cada atributo de cada fuente. Las fuentes pueden tener atributos con mismo nombre pero significados distintos. En esta técnica se particionan los atributos de forma que todos los atributos que parezcan tener la misma semántica estén en la misma partición. Ya que puede ser casi imposible encontrar una partición perfecta en caso de que exista, se definen varias formas de particionar los mismos atributos, y a cada una se le define una probabilidad de ser la que mas se ajusta en base a la similaridad de los atributos y/o de una muestra de sus datos.

Esta técnica se usa particularmente en sistemas de consulta en tiempo real como por ejemplo sitios de búsqueda de hoteles que obtienen de resultados de varias fuentes. En [6] se puede profundizar más sobre las definiciones estadísticas de la propuesta.

2.4.2. MapReduce en Data Matching

Un enfoque básico de aplicación de MapReduce en data matching consiste en utilizar los mappers para particionar los datos según la clave de bloque, y que luego los reducers en paralelo se encarguen de las tareas de matching. Sin embargo se debe tener cuidado en la selección de la clave, ya que los tamaños de bloque pueden diferir mucho (por ejemplo si se utiliza el apellido de personas, el bloque de Rodríguez va a ser mucho mayor al de Zapdos), y esto limita la mejora que se pueda alcanzar.

En [9] se describen dos estrategias para balancear la carga en los reducers. La primera, llamada *BlockSplit*, subdivide el trabajo de cada bloque en varias tareas de match y paralelizando estas subtareas. Ésta no garantiza totalmente un buen balanceo, pero en muchos casos mejora considerablemente la performance.

La segunda, llamada *PairRange*, garantiza un buen balanceo de carga asignando una cantidad similar de comparaciones de pares a cada reducer. Para lograr esto, PairRange crea una enumeración virtual de los pares a comparar, y a cada reducer le asigna una cantidad similar de pares.

2.4.3. Meta-indexación

Esta técnica propone construir un grafo donde los nodos son los registros a comparar, y las aristas indican que los nodos comparten al menos un bloque luego de haberse generado las particiones. Las aristas son ponderadas, y su peso intenta indicar cuan probable es que los registros pertenezcan a la misma entidad. Lo que se intenta hacer luego es recortar ramas que tengan baja probabilidad de ser match para alivianar el procesamiento.

Una simple forma de ponderar las aristas es indicar en cuántos bloques coexisten los nodos que une. Una más complicada podría tener en cuenta otros aspectos como por ejemplo la cantidad de elementos de los bloques, y aprovechar el hecho de que cuantos mas registros tenga un bloque, menos probable es que dos registros refieran a la misma entidad.

En cuanto a las técnicas de recorte de ramas, [12] propone dos técnicas. La primera, llamada "centrada en aristas", selecciona las mejores aristas con una vista global, mientras que la segunda, llamada "centrada en nodos", selecciona los mejores locales a cada nodo. También propone dos formas de descartar pares, una que descarta cuando el peso de la aristas es menor a determinado valor, y otro que se queda con las mejores k aristas del nodo.

Los resultados presentados en [12] indican que la meta-indexación mejora la eficiencia del bloqueo por 1 o 2 órdenes de magnitud frente a otras técnicas, y manteniendo alta la completitud de pares. La técnica de recorte de ramas centrada en aristas da mejor performance que la basada en nodos, y mantiene la calidad de los resultados. También se encontró que descartar pares usando el peso de la arista obtiene resultados de mejor calidad que quedarse con las mejores k aristas.

2.5. Herramientas

A continuación se presentan análisis de diferentes herramientas de Integración de datos tanto comerciales como prototipos académicos. Varias de las herramientas open source analizadas fueron obtenidas de un listado muy completo presente en [4] y las demás, especialmente las comerciales, fueron obtenidas de búsquedas en la Web. No todas cubren todas las etapas de la integración de datos que se vieron previamente.

El objetivo de este análisis es conocer el estado del arte actual con respecto a las herramientas existentes. Al mismo tiempo obtener ideas y familiarizarse con las necesidades presentes tanto en la industria como en la academia para poder diseñar una herramienta propia.

Algunos de los puntos tenidos en cuenta fueron el tipo de fuentes de datos soportadas, técnicas de indexación utilizadas y funciones de comparación.

Cabe destacar también que actualmente existen varios proveedores de motores de bases de datos (Oracle, IBM, Microsoft) y otras empresas a cargo de herramientas de ETL como Pentaho Data Integration y Talend Data Integration cuyos productos ofrecen características de integración de datos, pero no resuelven el problema completamente. La mayoría de ellas se han enfocado en crear herramientas amigables (aunque necesitan de mucha configuración manual) para los procesos ETL, pero no brindan un ambiente para

la integración de datos como tal.

La investigación realizada queda resumida en una tabla comparativa presentada al final de esta sección (figura 2.4).

2.5.1. Merge Toolbox

Es un software de comparación de registros y deduplicación programado en Java que surge como un proyecto de la Fundación de Investigación Alemana (German Research Foundation) bajo la dirección del Prof. Dr. Rainer Schenell siendo a partir del 2011 mantenido por el centro alemán de Record Linkage (German Record Linkage Center) [1].

Es una herramienta de uso gratuito únicamente con fines académicos.

Posee una interfaz gráfica simple pero es necesario leer el manual para poder utilizarla dado que carece de ayuda integrada y maneja conceptos de Data Matching para los que se requieren conocimientos previos.

Trabaja con archivos de texto plano (donde se debe definir un separador como por ejemplo la coma en CSV) o Stata tanto para la entrada como la salida.

No se dispone de funcionalidades de preprocesamiento pero para la indexación (blocking) la herramienta cuenta con dos modos: tradicional y Canopy Clustering. Con respecto a las funciones de comparación se cuenta con 24 algoritmos distintos en base a dos enfoques: identificación basada en distancia o en probabilidad.

Merge-Toolbox es un desarrollo puramente académico que requiere un conocimiento profundo de los algoritmos a utilizar y poco utilizable en un marco empresarial dado que por ejemplo carece de módulos de conexión a bases de datos y preprocesamiento.

2.5.2. WizSame

WizSame es una herramienta comercial para realizar deduplicación desarrollada por la empresa WizSoft con sede en EEUU e Israel. Fue posible contactarse con dicha empresa, la cual proporcionó un demo de la herramienta.

El software está orientado a usuarios administrativos que se encuentran con problemas clásicos como auditorías (encontrar facturas o pagos duplicados) y CRM (encontrar clientes duplicados) [17].

Para considerar que dos registros son duplicados la herramienta verifica un conjunto de reglas respecto a los campos de los registros a procesar. Dichas reglas son ingresadas por el usuario y chequeadas utilizando un algoritmo de similaridad propietario y un diccionario que puede ser actualizado por el usuario.

Presenta una interfaz relativamente sencilla con ayuda integrada que redirige a un manual de usuario bien organizado.

WizSame admite como entrada tablas de bases de datos mediante OLE DB y ODBC, MS Access, archivos Excel, dBase y texto ASCII [19] y exporta datos a Excel, texto ASCII o MS Access.

Como preprocesamiento la herramienta permite establecer el formato de los tipos de campos clasificándolos automáticamente como como Quality (carácteres alfa-numéricos), Number (números a los que se le pueden aplicar cálculos matemáticos) o Date (datos de fecha en alguna forma estándar). Para cada campo se puede seleccionar si ignorarlo, si deben coincidir exactamente para considerarse match, o si se considera match si uno de los registros tiene el campo vacío. También se pueden aplicar condiciones por determinados valores que tomen los campos para filtrar registros a través de reglas que se relacionan lógicamente.

El software no cuenta con estrategias de indexación y con respecto a la similaridad entre registros, la misma es realizada mediante un algoritmo propietario explicado en [18] y también utiliza un diccionario de sinónimos que puede ser actualizado por el usuario.

Esta herramienta plantea un enfoque muy básico al problema de identificación de entidades. No brinda al usuario demasiadas alternativas pues solo posee un algoritmo de similaridad y no permite configurar umbrales.

2.5.3. FEBRL

FEBRL[2] (Freely Extensible Biomedical Record Linkage) es una herramienta desarrollada por Peter Christen, autor de [4], cuyo desarrollo comenzó en 2002 en el marco de un proyecto de investigación llevado a cabo por la Universidad Nacional Australiana en colaboración con el Departamento de salud de New South Wales. El objetivo principal fue desarrollar técnicas de limpieza de datos, estandarización, deduplicación y record linkage sobre bases de datos de sistemas de salud. Está escrita en Python, es de código abierto y su última versión es del 2013.

Brinda tres modos principales de operación: estandarización, deduplicación y record linkage. Los formatos de entrada aceptados son sólo CSV, COL y TAB; no ofrece tipos mas complejos como bases de datos o documentos JSON. En cuanto a las técnicas implementadas, ofrece 7 algoritmos de indexación con 9 funciones de codificación de índices, 26 funciones de comparación de campos y 4 algoritmos de clasificación incluyendo supervisados y no supervisados.

El formato de salida de los resultados es bastante incómodo de procesar; genera IDs para cada match, y retorna dos CSVs (uno por cada set de entrada) con los datos y una columna extra que indica a que matches pertenece. También genera una gráfica que muestra la distribución de la similaridad de los pares comparados.R

[2] https://sourceforge.net/projects/febrl/

	Data Matching Enterprise	WinPure Clean & Match	Match (DQ Global)	FEBRL	Merge Toolbox	WizSame
Comercial	✓	✓	✓	✗	✗	✓
Fuentes	SQL Server, Oracle, Fixed Width Text File, Text delimited, DBF, MySQL, Ole DB, ODBC, Excel	Access, Excel, .txt, CSV, dBase, SQL Server, MySQL, Outlook	CSV, ODBC, Excel, Access	CSV, TAB, COL	Stata, cualquier archvivo de texto plano (se establece separador)	Texto plano, ODBC, OLE DB, Access, Excel, dBase
NoSQL	✗	✗	✗	✗	✗	✗
Data fusion	✓	✓	✗	✗	✗	✗
Preprocesamiento	Limpieza de datos y estandarización manual	Módulo para limpieza de datos manual	Múltiples métodos para datos comunes. Scripts personalizados	Personalización para nombres, direcciones, teléfonos y fechas	✗	Personalizar formato de números y fechas
Indexación	No se sabe si utiliza técnicas indexación de fondo (algoritmos propietarios)	✗	DQ Fonetix, Soundex y Metaphone	7 técnicas de indexación y 8 fonéticas de selección de clave de índice	Exacta o Canopy Clustering (Q-grams y distancia de Jaccard)	✗
Funciones de comparación	Fuzzy (algoritmo propietario), Fonético, Exacto y Numérico	Fuzzy Matching (algoritmo propietario) y matching exacto	No permite elegir algoritmos de comparación	26 funciones de comparación para strings, números, fechas y lugares	24 algoritmos de comparación. Comparaciones probabilísticas o por distancia	Funciones de comparación de string simples y un algoritmo propietario de similitud
Umbrales	Permite establecer pesos en la comparación de atributos y nivel	Si se elige Fuzzy matching es posible definir el umbral	Umbral para marcar para revisión y otro para marcar como match	Dependiendo del método se definen manualmente o automáticamente	Dependiendo del método se pueden definir manualmente o automáticamente	✗
Exportación	Exportación de matches en SQL Server, Oracle, Fixed Width Text File, CSV, XML, DBF, MySQL, Ole DB, ODBC, E xcel	Access, Excel, .txt, CSV, dBase, SQL Server, MySQL, Outlook	Dedup directo en la fuente, exportación de matches en CSV	Exporta histograma, vectores de similitud y estado de matches en CSV	Stata o texto plano	Dedup directo en la fuente, exportar a texto plano

Tabla 2.4: Tabla comparativa de herramientas

Capítulo 3

Desarrollo

3.1. Análisis y requisitos

En esta sección se detalla el análisis realizado y definición de los requisitos que debe cumplir la herramienta a desarrollar.

Se plantea como objetivo el desarrollo de un framework de integración de datos que permita realizar proyectos de integración entre dos fuentes siguiendo los lineamientos y utilizando las técnicas de integración mencionadas en el capítulo 2.

Dicha herramienta debe poder ser ejecutada en forma autónoma pero a su vez, por su condición de framework, debe brindar servicios que puedan ser consumidos por otros sistemas a través de una API. De este modo, con el objetivo de brindar una experiencia de usuario atractiva se hace necesario contar con una interfaz gráfica que permita a los usuarios configurar y ejecutar proyectos de integración de datos de manera amigable, mientras que través de la API, los programadores puedan no solo enriquecer sus propias aplicaciones sino también automatizar proyectos de integración rutinarios.

Muchas veces puede ser necesario integrar datos generados periódicamente por dos fuentes, como por ejemplo reportes de ventas de dos empresas distintas. Aquí alcanza con configurar una vez el flujo, y luego ejecutarlo las veces que sea necesario sobre los datos nuevos. Para esto se definen entonces 2 modos de uso: modo de diseño, donde el usuario utiliza la interfaz gráfica para configurar el trabajo de integración, y modo producción, donde se utiliza la configuración hecha durante el modo diseño de forma automatizada a través de scripts que pueden ser parametrizados y que se comunican con la API de la herramienta para realizar la integración.

Por otro lado es importante resaltar que uno de los objetivos principales del prototipo es que sea modular y permita que terceros puedan extenderlo con sus propios módulos de integración. Esto genera la necesidad de una arquitectura que modularice las tareas involucradas en el proceso de integración y por otro lado, una interfaz gráfica flexible y también fácilmente extensible.

Analizando las herramientas existentes y teniendo presente lo estudiado en el estado del arte se encontraron cuáles pasos son los importantes para incluir en la herramienta. Por ejemplo, se determinó que el paso de segmentación aporta un gran valor al flujo de

integración, sin embargo, no está presente en ninguna de las herramientas analizadas.

Finalmente, otra funcionalidad que el framework debe brindar al usuario, es la posibilidad de generar un script ejecutable a partir de un proyecto ya creado. Este script ejecutará un flujo de integración con la misma configuración, y puede ser temporizado para facilitar las tareas de automatización. De ser necesario también podrá ser parametrizado.

Para lograr la modularidad de la herramienta, para cada paso se definen distintos tipos de módulos que ejecutan distintos algoritmos relacionados con el paso (por ejemplo los módulos de comparación consisten en algoritmos de comparación, los de extracción extraen datos de distintas fuentes, etc.).

Los pasos definidos para la herramienta son los siguientes:

1. **Extracción**: es el primer paso y en el que se obtienen los datos fuente y se transforman al formato interno. El usuario debe seleccionar un módulo de extracción por cada fuente a integrar.

2. **Limpieza**: por cada columna de los datos originales, el usuario puede elegir un conjunto de módulos de limpieza para por ejemplo eliminar caracteres extraños o pasar a minúscula. Esto mejora la calidad de los resultados de integración.

3. **Estandarización y etiquetado**: para cada columna de los datos originales, el usuario puede elegir un módulo de estandarización y de etiquetado para estandarizar las palabras y asignarle posibles etiquetas que puede o no usar el próximo paso.

4. **Segmentación**: para cada columna de los datos originales, el usuario puede elegir un módulo de segmentación que generará posibles campos de salida (vistos en la sección 2.2.1, también llamados segmentos) para cada registro. Estos campos de salida son los que se compararán en el paso de comparación.

   De las herramientas analizadas, sólo FEBRL cuenta con segmentación, y se hace sobre nombres y direcciones. Sin embargo no permite ningún tipo de configuración ni aparece en la interfaz, por lo que este paso es un agregado muy útil a la herramienta, ya que permite que las comparaciones sean más precisas porque se comparan independientemente las distintas partes por las que está compuesta una columna (por ejemplo una dirección).

   En este paso también se genera un nuevo esquema con la unión de los campos de salida encontrados en cada columna de cada registro. El usuario puede optar por saltarse este paso (no segmentar) y utilizar el valor de las columnas en el paso de comparación.

5. **Schema Matching**: se matchean las columnas de los dos esquemas que luego serán comparadas para encontrar pares de registros correspondientes. También debe generar un esquema global, y transformar todos los registros para que sigan este esquema único desde este paso en adelante.

Durante el desarrollo del estado del arte este paso siempre se mencionó al comienzo de los trabajos de integración, pero para la herramienta se decidió colocarlo luego de los pasos de estandarización y segmentación ya que la información añadida por éstos es útil y puede ser aprovechada por algún módulo de schema matching que utilice técnicas basadas en las instancias para identificar correspondencias en los esquemas.

6. **Indexación**: se agrupan los registros a comparar en grupos más pequeños para reducir la cantidad de comparaciones a realizar en el siguiente paso.

7. **Comparación**: por cada campo de salida (o columna si el usuario decidió saltarse la segmentación), el usuario debe seleccionar distintos módulos de comparación con sus ponderaciones para ser aplicados. Por cada par de registros comparados se genera un vector de similitud que indica cuán parecidos son los registros.

8. **Clasificación**: en este paso se determina si los pares comparados corresponden a un match, no match o match potencial en base a los vectores de similitud generados anteriormente y al algoritmo de clasificación elegido.

9. **Data Fusion**: se aplica un algoritmo de data fusion sobre los registros clasificados como matches para generar un registro único por cada par.

10. **Exportación**: finalmente se permite al usuario exportar los datos fusionados en distintos formatos. Opcionalmente, se puede elegir exportar también los registros que no fueron clasificados como match.

La diferencia entre un paso y módulo está dada por la tarea que realiza cada uno. Los pasos definen la forma en la cual se ejecutan los módulos y cómo se guarda la información resultante de la ejecución de los mismos mientras que los módulos son más granulares y ejecutan algoritmos o tareas más básicas.

A nivel de usuario final, en general, los pasos consisten simplemente en la selección de los módulos y a diferencia de estos últimos, no se puede elegir entre distintas implementaciones de los pasos. En un principio se evaluó la posibilidad de permitir distintas implementaciones de los pasos, pero se tomó esta decisión para acotar el alcance del prototipo. El enfoque está en la modularidad y extensibilidad que proveen los módulos.

Existen dos tipos de módulos clasificados según su complejidad: los ***módulos de pasos*** y los ***módulos de utilidad***. Los módulos de utilidad son módulos que no son invocados directamente en ningún paso y cuya función está limitada a alguna tarea tan simple que puede ser utilizado por otros módulos de pasos. Los módulos de pasos son aquellos cuya elección es requerida por los pasos.

En la versión inicial del framework, los módulos de codificación son los únicos de utilidad. Específicamente, los módulos de codificación son utilizados en el módulo indexación blocking estándard (sección 2.2.2.1) donde se debe elegir la clave en base a módulos de codificación para formar la clave. Por ejemplo, un módulo de codificación podría codificar un string tomando sus primeros n caracteres.

Esta diferenciación de módulos incrementa la reusabilidad, permitiendo así que los desarrolladores no implementen funciones básicas en cada uno de sus módulos que podrían ser

reusadas entre unos y otros.

A continuación se describen los distintos tipos de módulos clasificados según su tarea.

Extracción	
Resumen	Extraen datos de una fuente.
Entrada	Información sobre la fuente.
Salida	El conjunto de registros extraído de la fuente.
Limpieza	
Resumen	Aplican un algoritmo de limpieza sobre un campo.
Entrada	Un campo.
Salida	El mismo campo luego de aplicarle la limpieza.
Estandarización y etiquetado	
Resumen	Aplican un algoritmo de estandarización y etiquetado sobre un campo.
Entrada	Un campo.
Salida	El mismo campo (o un conjunto de campos, ya que estos módulos pueden tokenizar los campos) estandarizado(s) y con posibles etiquetas asignadas.
Segmentación	
Resumen	Para cada columna y su conjunto de campos, agrupa estos campos en distintos campos de salida.
Entrada	Un conjunto de campos.
Salida	El mismo conjunto de campos, pero cada uno puede tener asignado un campo de salida.
Indexación	
Resumen	Indexan un conjunto de registros agrupándolos de alguna forma.
Entrada	Un conjunto de registros.
Salida	Uno o más subconjuntos de registros agrupados según el criterio definido por el algoritmo.
Codificación	
Resumen	Codifica un valor.
Entrada	Un valor de un campo.
Salida	El valor codificado.
Comparación de strings	
Resumen	Ejecuta algún algoritmo de comparación entre dos strings y retorna la similitud.
Entrada	Dos strings s_1 y s_2.
Salida	Número entre 0 y 1 que representa la similitud entre s_1 y s_2.

Clasificación	
Resumen	Determinan si un vector de similitud corresponde a un match o no. También puede devolver otros estados como por ejemplo el caso en que se está en duda y se requiere intervención de un operador.
Entrada	Un vector de similitud.
Salida	Devuelve si es match o no.
Data Fusion	
Resumen	Para cada par que se encontró como match, devuelve un registro con los datos fusionados según el algoritmo que implemente para la fusión.
Entrada	Un par de registros clasificados como match.
Salida	Un registro con los datos fusionados.
Exportación	
Resumen	Extrae los resultados de la integración a algún formato externo.
Entrada	Un conjunto de registros.
Salida	El conjunto de registros representado en un formato externo.

Tabla 3.1: Descripción de tipos de módulos

Por otro lado, los módulos que se implementarán para este prototipo serán los siguientes:

- **Extracción**:
  - Extracción de archivos CSV.
  - Extracción de bases de datos PostgreSQL.
  - Extracción de bases de datos MongoDB.
- **Limpieza**:
  - Conversión a minúsculas.
  - Eliminación de caracteres no deseados.
  - Transformación a tipo fecha.
  - Eliminación de espacios en blanco al comienzo y fin de los valores.
- **Estandarización y etiquetado**:
  - Módulo de etiquetado (sin transformación/estandarización) basado en expresiones regulares para direcciones uruguayas (nombre de calle y número entre 3 y 4 dígitos).
  - Módulo de etiquetado de nombres y apellidos en base a tablas de lookup.
- **Segmentación**:
  - Parsing para direcciones uruguayas que asigna dos campos de salida (Nombre de calle y Número de calle) si se encontó la etiqueta de nombre de calle y la de número entre 3 y 4 dígitos.

    - Parsing de nombres que asigna los siguientes campos de salida: primer nombre, segundo nombre y apellido.

- **Schema Matching**:
    - Matcheo de esquemas manual[1].

- **Indexación**:
    - Indexación de índice completo, donde se comparan todos los registros con todos.
    - Indexación estándar por clave, donde el usuario puede elegir una clave por la que se agruparán los registros. Esta clave puede estar formada por la concatenación de varios campos del registro codificados.

- **Codificación**:
    - Quedarse con los primeros N caracteres.

- **Comparación de strings**:
    - Igualdad.
    - Distancia de edición Levenshtein.
    - Jaro.
    - Jaro-Winkler.
    - Token Sort.
    - Token Set.
    - Ratio.

- **Clasificación**:
    - Clasificación Fellegi y Sunter.
    - Clasificación basada en reglas.

- **Data Fusion**:
    - Módulo de selección de fuente de confianza, donde frente a un conflicto (dos columnas matcheadas que difieren en los datos), el registro final tiene los datos de la fuente de confianza elegida.

- **Exportación**:
    - Exportación a base de datos MongoDB.
    - Exportación a archivo CSV.

[1]Dada la complejidad de implementar algoritmos de matcheo de esquemas automáticos, este prototipo incluye un módulo que permite al usuario explicitar las correspondencias manualmente.

3.1.1. Perfiles de usuario

Para lograr un proceso de integración de calidad se requiere la intervención de distintos perfiles de usuarios. A continuación se detallan los perfiles identificados y el uso que podría darle cada uno a la herramienta.

Experto del dominio Un experto del dominio es aquel que conoce con detalle los datos que se desea integrar, su estructura y el significado de cada campo, y es el rol más importante para lograr buenos resultados. Es necesario para definir qué tipo de limpieza necesitan los datos, qué tipo de segmentación aplicar sobre los mismos, cómo matchear los esquemas, cómo fusionar los datos una vez comparados, e incluso para ayudar al experto en integración en la selección de algoritmos de comparación.

Experto en integración Los expertos en integración trabajan de forma muy cercana con los expertos del dominio, ya que por lo general los expertos del dominio no conocen de los algoritmos específicos que se usan a lo largo del proceso de integración. Los expertos en integración los ayudan determinando que algoritmos específicos de comparación utilizar según la naturaleza y el tipo de problemas de calidad de los datos (por ejemplo no es lo mismo que los datos tengan faltas de ortografía a que estén abreviados), así como a decidir qué tipo de indexación utilizar y con qué clave, a clasificar los resultados de comparación, y finalmente a fusionar los correspondientes.

Desarrollador El desarrollador tiene dos funciones principales. La primera consiste en desarrollar e integrar al framework funcionalidades adicionales (módulos) que se puedan necesitar o sean de utilidad para el proceso de integración, y la segunda función involucra el otro modo de uso del framework: el uso mediante la API. Los desarrolladores podrán integrar el framework a otros sistemas que necesiten de capacidades de integración, o también utilizar los scripts generados automáticamente para tareas programadas o batch.

3.2. Diseño y arquitectura

El prototipo consta de dos grandes entidades: la interfaz gráfica, y el servidor como se observa en la figura 3.1. El componente principal del servidor es el motor, que contiene toda la lógica de integración. La interfaz gráfica es una SPA (Single Page Application) que se comunica con el motor a través de una API REST. Coordinando el flujo de integración está el "WorkflowController", que es quien se encarga de llamar al paso que corresponda dependiendo del estado actual del trabajo de integración (de ahora en más llamado *proyecto*).

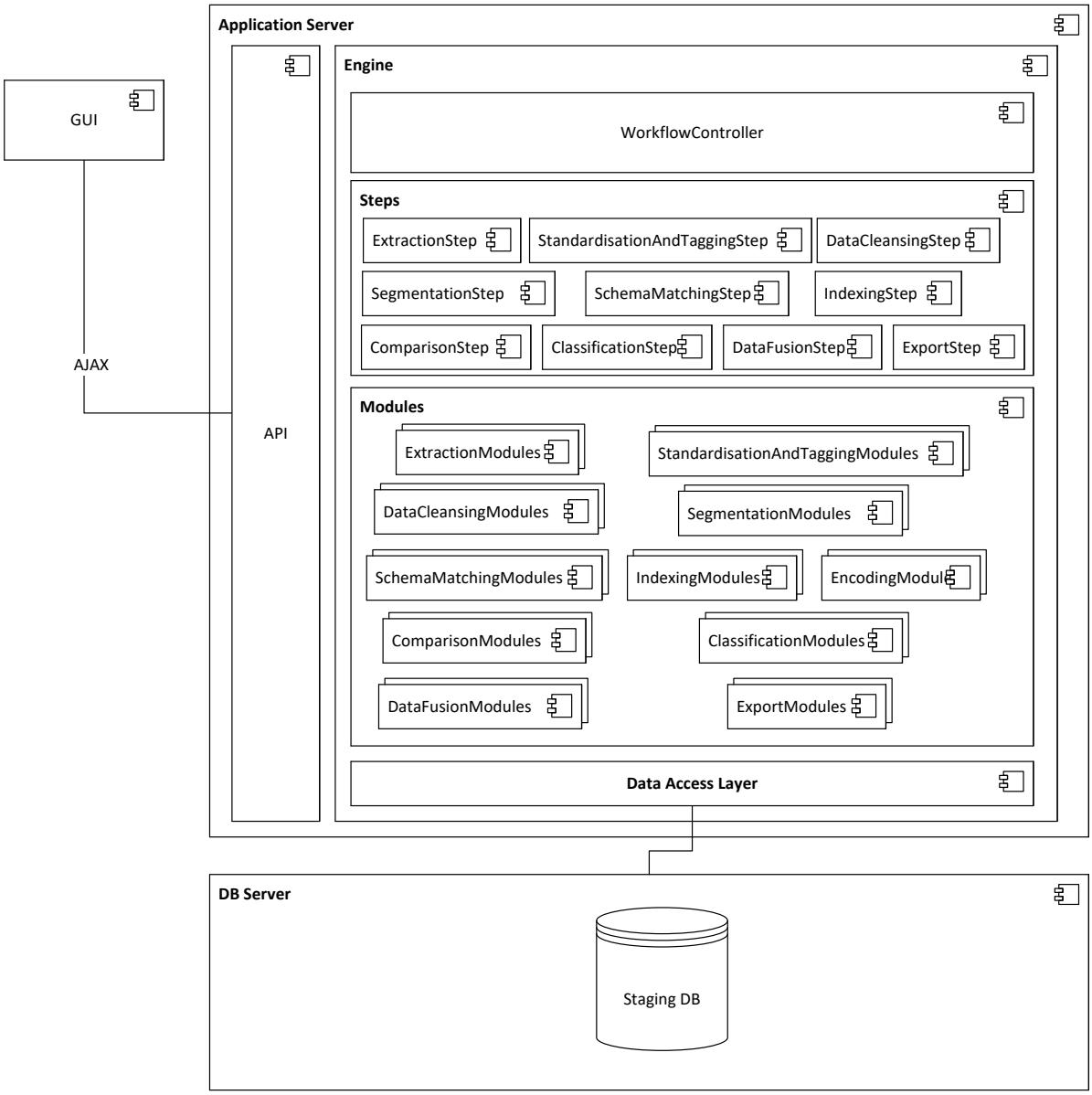

Figura 3.1: Arquitectura (diagrama de despliegue)

Cuando la API recibe un pedido de ejecutar el siguiente paso, ésta pasa el pedido y toda la configuración requerida al "WorkflowController", y éste procede a invocar al paso correspondiente enviándole también toda la configuración que recibió. En la configuración recibida se encuentran los módulos a ejecutar así como configuración específica para el paso. Cada paso invoca dinámicamente los módulos requeridos en la configuración, y una vez terminada su tarea, almacenan sus resultados en una base de staging.

Para la base de staging se optó por utilizar un DBMS documental, ya que se consideró que brinda gran comodidad al momento de almacenar y trabajar con datos en sus distintos esquemas a lo largo de todo el proceso de integración. En una base de datos documental se pueden almacenar los datos tal cual son, evitando así (comparando con un motor relacional) tener que definir un meta-esquema lo suficientemente genérico como para poder operar sobre los distintos esquemas que se deseen soportar. Otra razón es que este meta-esquema estaría compuesto de varias tablas, por lo que serían necesarios joins pa-

ra leer los datos, degradando considerablemente también el rendimiento del framework [11].

La cualidad *schema-less* de este tipo de bases de datos también permite un desarrollo más ágil, ya que se pueden hacer cambios en el modelo de datos de la aplicaión sin tener que reflejar esos cambios en la base de datos (como sí se haría en una base relacional), permitiendo así obtener el prototipo en menos tiempo.

Otra característica que si bien no es aprovechada por el prototipo en su versión inicial, pero que queda disponible para explotarse en un futuro, es la de *auto-sharding*. El sharding consiste en distribuir la base en varios servidores (escalamiento horizontal), almacenando distintas porciones de los datos en cada uno, y mejorando la velocidad de acceso los datos al distribuir la carga en varios nodos. Si bien esto se puede lograr con bases relacionales, suele ser muy complejo ya que no es una funcionalidad soportada de forma nativa y es necesario agregar elementos nuevos al sistema como software que maneje las consultas distribuidas, que decida donde almacenar los registros, etc. Los motores de bases documentales, por otro lado, suelen soportar esta funcionalidad de forma nativa, o sea que no requieren agregar ningún software ni hardware adicional.

Para comenzar con el diseño del framework se debe tener en cuenta que es necesario contar con un "formato" común entre todos los pasos, es decir objetos que representen a los datos de forma única para que los pasos y módulos puedan comunicarse. Por esta razón es que se define el formato interno (figura 3.2) de los datos en el framework. El formato interno es aquel sobre el cual el sistema operará durante todos los pasos y representa a los datos en determinado estado según el paso en el que se encuentre el proyecto.

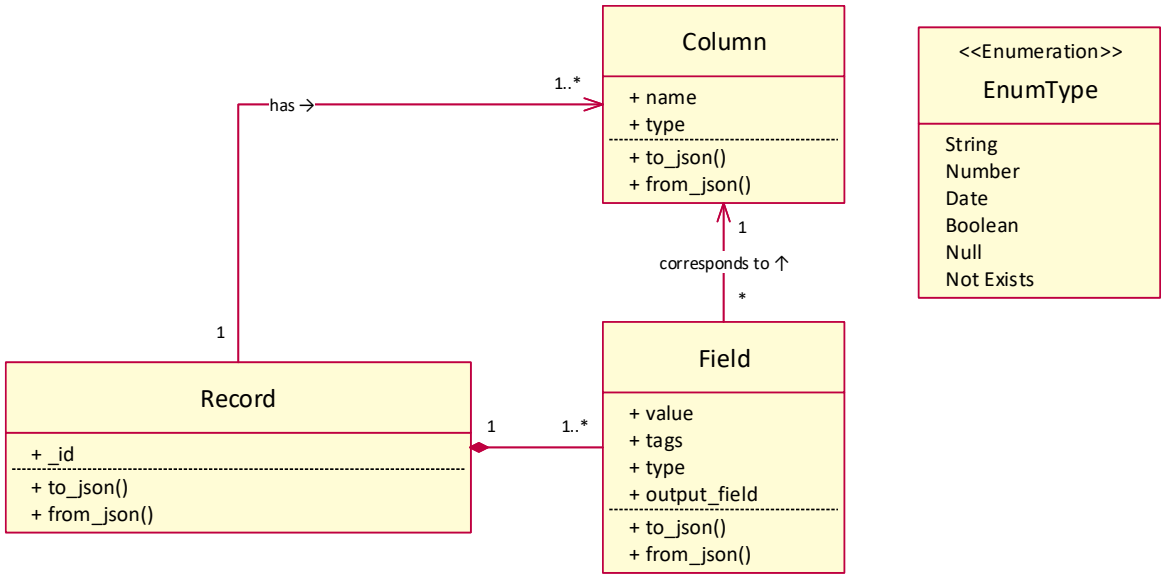

Figura 3.2: Diagrama de clases (formato interno)

La clase "Field" **tiene una semántica variable según el paso en el que nos encontremos**. A modo de ejemplo: luego del paso inicial de extracción, la clase "Field" representará los valores completos en cada columna sin ninguna etiqueta ni campo de salida, pero luego del paso "Estandarización y Etiquetado", una instancia de la clase "Field" representará un único token (o pocos) con posibles etiquetas asignadas dependiendo

del paso y/o módulo/s utilizados (ya que podrían usarse módulos que no generen etiquetas).

El enumerado "EnumType" (utilizado en la clase "Field") representa tipos de datos primitivos, tales como Integer, Null, String, Date, entre otros así como también se utiliza el valor "Not Exists" para aquellos campos cuyo valor está ausente, y se utilizaría por ejemplo en bases de datos documentales, donde no están las columnas predefinidas. Es de gran importancia incluir este metadato ya que si algún módulo de estandarización lo desea, podría inferir el tipo de dato y esto sería una gran ayuda para la automatización de la selección de algoritmos de comparación. El atributo de frescura es otro atributo que puede ser aprovechado por varios módulos si está disponible a partir de la fuente.

También se utilizan otras estructuras auxiliares (figura 3.3) a lo largo del proceso. La estructura "SchemaMatch" representa al matcheo entre esquemas, y consiste en una lista de pares de conjuntos de columnas, donde cada elemento de un par contiene columnas de un esquema. Así, por ejemplo un matcheo de los esquemas S1(nombre, apellido, calle, departamento, país) y S2(nombre_completo, dirección, nacionalidad) podría ser:

```
[
    ({nombre, apellido}, {nombre_completo}),
    ({calle, departamento}, {dirección}),
    ({país}, {nacionalidad})
]
```

La estructura "IndexingGroup" se utiliza a partir del paso de indexación, y representa un par de grupos de registros y una clave que los vincula. Esto es, que registros de ambas fuentes tienen la misma clave de indexación. Finalmente, también existen las estructuras "SimilarityVector", que vincula un par de registros con su vector de similitud; y "MatchResult", que vincula un par de registros con la desición final sobre su similitud (si son match o no).

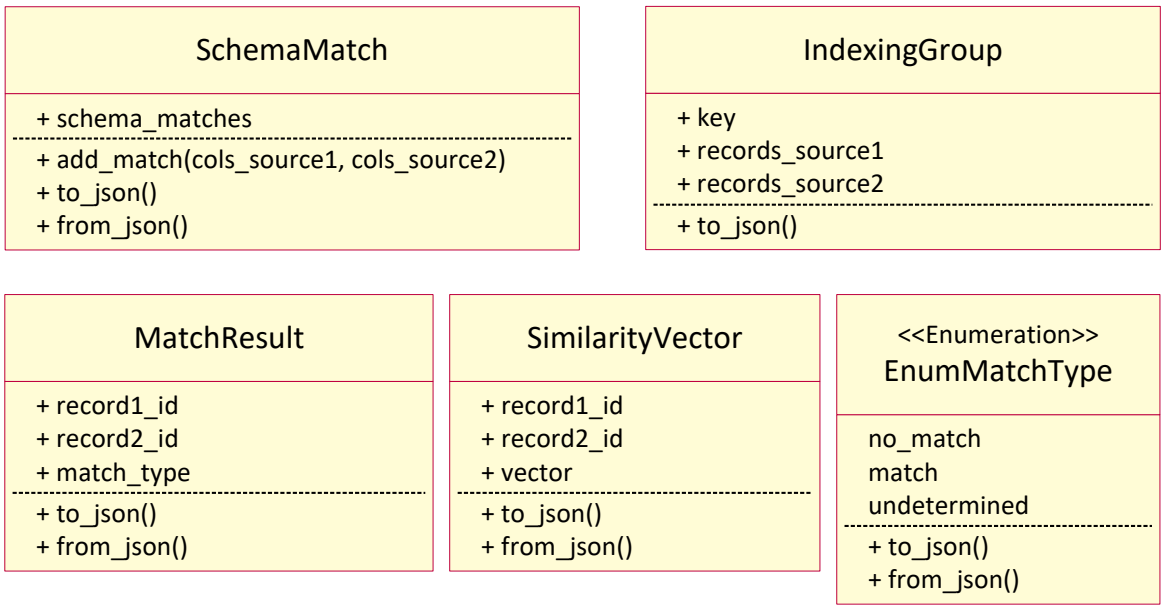

Figura 3.3: Otras estructuras auxiliares

Para la secuencia de pasos (figura 3.4) se emplea el patrón de diseño State, donde la clase "Workflow" hace el papel del objeto de "contexto", como es nombrado comúnmente al hablar de este patrón. Cada paso representa un estado distinto del patrón.

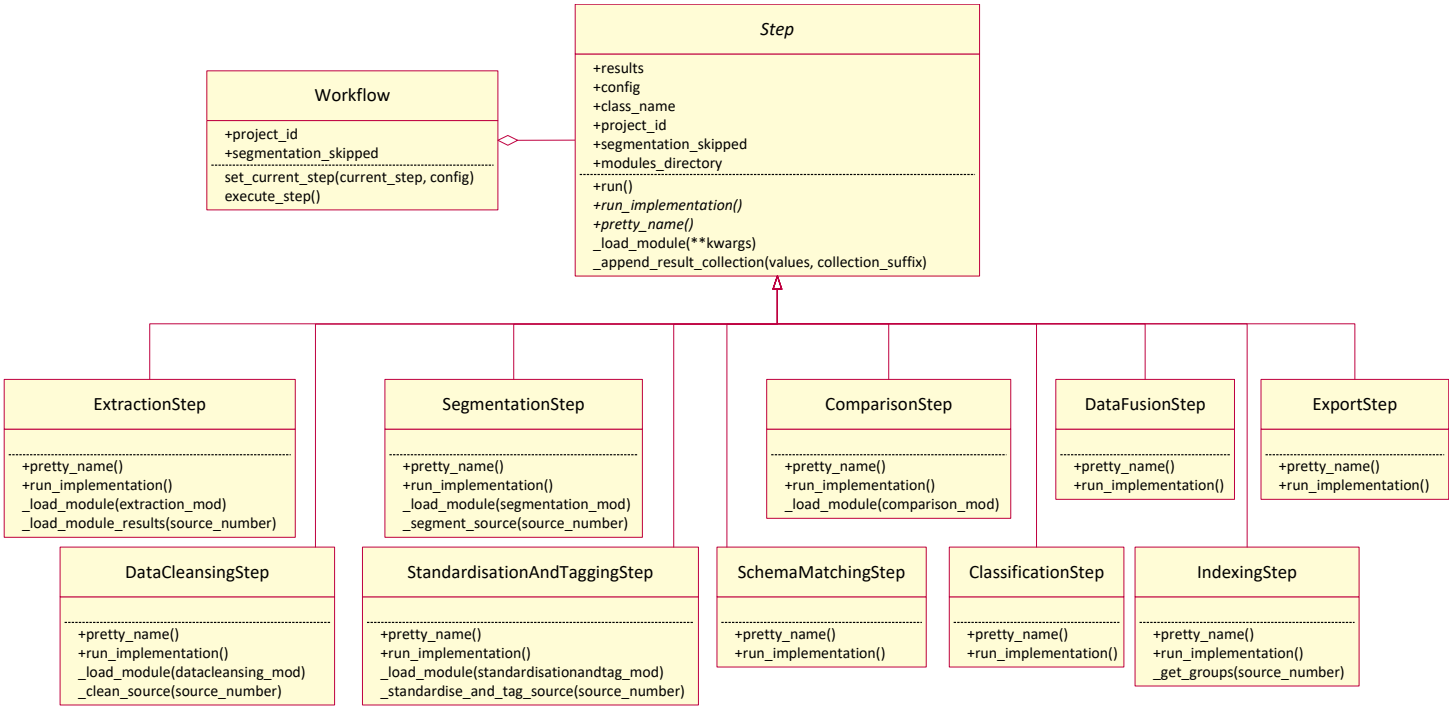

Figura 3.4: Diagrama de clases (flujo de trabajo y pasos)

En cuanto a los módulos, como se analizó, están clasificados de acuerdo a su funcionalidad. Cada tipo de módulo tendrá una clase abstracta que deberán extender las distintas implementaciones. En la figura 3.5 se muestra un diagrama de clases con las clases abstractas para cada tipo de módulo.

Cada módulo tiene información sobre cada campo de configuración que necesita para funcionar. El desarrollador de cada módulo debe implementar tanto el código con el algoritmo correspondiente al módulo (método *run*) como el que devuelve esta información de configuración (método *config_json*). La interfaz del framework genera dinámicamente los formularios de configuración de cada módulo en base a esta información en el momento que el módulo es seleccionado.

Como se mencionó en la sección 3.1, los módulos también se clasifican en base a su complejidad. Por un lado existen módulos de utilidad cuya función es ser utilizados por otros módulos. A nivel de diseño de clases los módulos de pasos y los módulos de utilidad no pueden diferenciarse. Los módulos de utilidad son utilizados como parte de la configuración del módulo de paso. Entonces si el desarrollador de un módulo quiere que su módulo sea utilizado por otros entonces debe incluir el identificador de su módulo de utilidad en su método *config_json* con el objetivo de que se pueda realizar la carga dinámica dentro de la ejecución del módulo que haga uso del mismo.

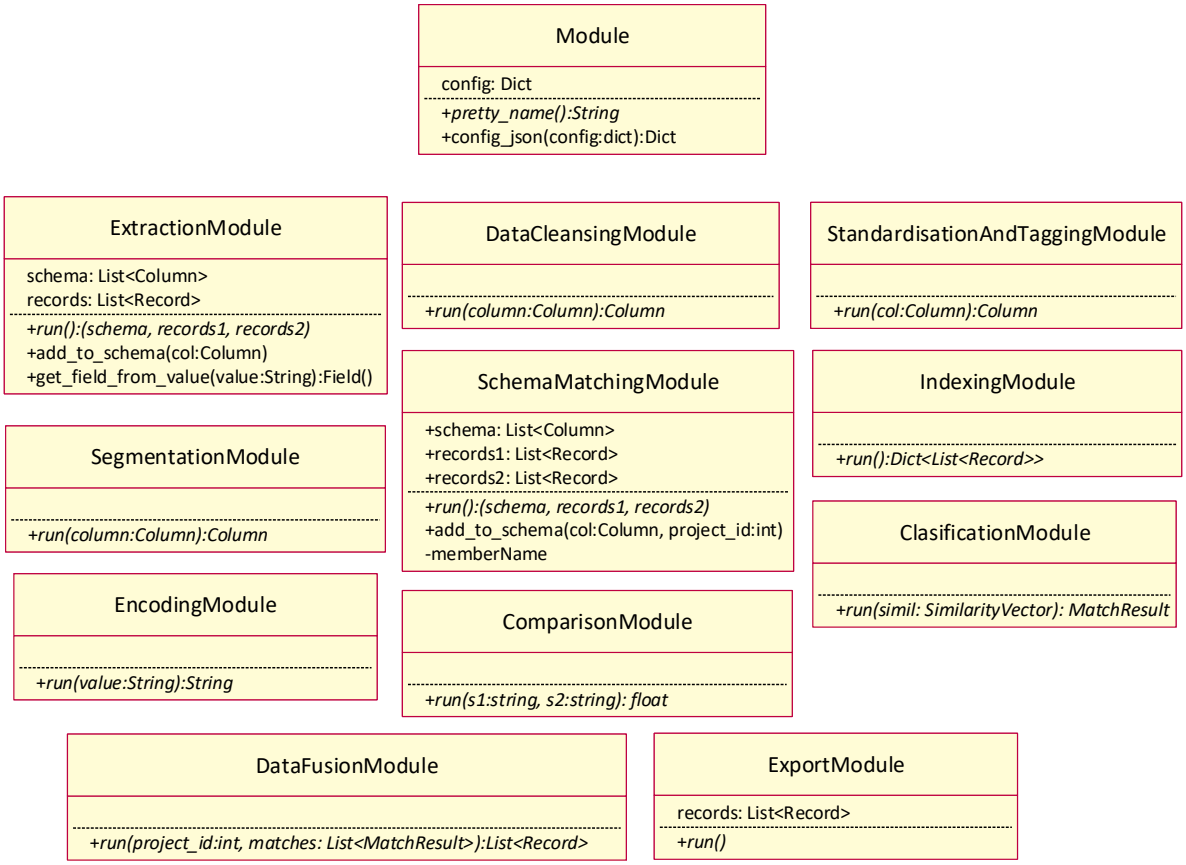

Figura 3.5: Clases abstractas por tipo de módulo

Luego de configurado un módulo, al continuar al siguiente paso, el framework envía cada configuración al módulo correspondiente de forma dinámica para evitar acoplamiento y simplificar lo más posible la integración de módulos adicionales.

3.2.1. Puesta en producción

Se plantea a continuación un caso hipotético de uso: una gran cadena de venta de productos informáticos quiere mantener actualizada su base de datos de productos (Oracle Database) en cuanto a cambios de precios en el resto del mercado, así como a productos nuevos que puedan aparecer. Para esto, en una reunión con el área de TI, se planteó el uso del framework de integración propuesto en este Proyecto de Grado para generar reportes mensuales que integren su base de datos de productos con los productos ofrecidos en MercadoLibre.

Para la puesta en producción se propone un despliegue como se muestra en la figura 3.1 al comienzo de la sección. Un servidor de aplicaciones que aloje tanto el motor de integración como la API, un servidor web para la interfaz gráfica, y un servidor de bases de datos. Estos 3 servidores pueden estar físicamente en un mismo equipo, o en equipos separados según la carga que tengan que soportar. Por ejemplo, si en la empresa hay un equipo con bases de datos que tenga recursos disponibles, se puede colocar la base allí y el motor en un equipo que se utilice como servidor web. Como en este caso se ejecutará un trabajo de

integración por mes, alcanza con que estén en un equipo compartido.

El equipo de TI ahora debe resolver la extracción de datos de productos de la API de MercadoLibre y de su base de datos Oracle. Dado que la herramienta no cuenta con un conector para bases de datos Oracle, se hace necesario desarrollar un módulo de extracción que haga esto. Para la extracción desde MercadoLibre se plantea primero la opción de generar un script que extraiga los datos a un CSV, y utilizar este CSV como entrada de la herramienta de integración utilizando el módulo de extracción ya implementado. El desarrollador plantea una mejor opción: desarrollar un módulo de extracción propio que consuma la API de MercadoLibre directamente. Esta segunda opción hace que la solución quede más compacta reduciendo la cantidad de componentes (elimina el script que extrae a CSVs) y permite agregar configuraciones a la extracción que puedan ser manejadas directamente desde la herramienta.

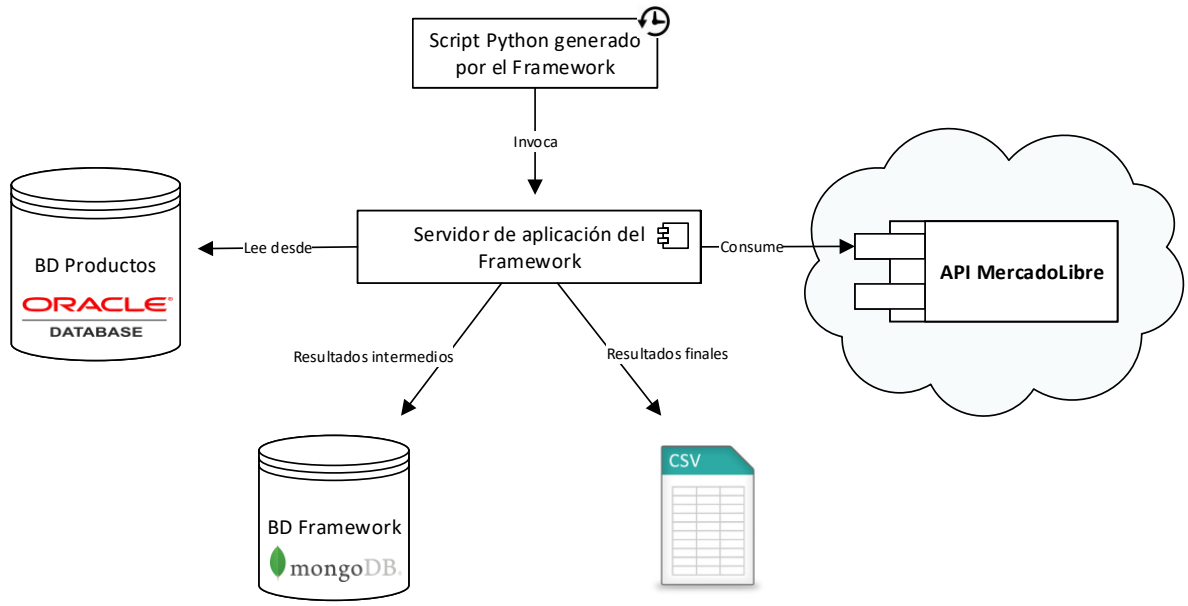

Figura 3.6: Diagrama del flujo de extracción en producción con la tarea programada

Una vez implementado el módulo de extracción, un experto del dominio de la empresa junto con un experto en integración utilizan la herramienta en modo diseño (modo diseño y modo producción se introducen al comienzo de la sección 3.1) configurando todo el flujo de integración sobre un conjunto de datos de muestra. Se recomienda hacer esto sobre una muestra de datos para agilizar el proceso, ya que hacerlo sobre el conjunto completo puede resultar en demoras en la ejecución de los pasos que dificulten el trabajo. Este conjunto de muestra debe ser una representativo de los datos reales para obtener mejores resultados cuando se pase al modo producción. En el capítulo 4 se presenta un caso de estudio donde se muestra una configuración de un flujo en detalle.

Concluido el diseño, se utiliza la funcionalidad de la herramienta que genera un script para reutilizar la configuración diseñada sobre otros conjuntos de datos. El equipo de TI parametriza este script de acuerdo a sus necesidades, y lo deja como tarea programada en un servidor para ser ejecutado una vez al mes. Este resultado mensual es luego utilizado por la empresa para generar un reporte con la información que necesiten. En la figura 3.6

se puede observar cómo queda el despliegue de los componentes una vez instalado y en producción el framework.

3.3. Implementación

3.3.1. Motor de integración

El motor de integración es quién contiene y se encarga de ejecutar los módulos de cada paso en un proyecto de integración de acuerdo a configuraciones específicas.

Está implementado en Python y realiza la carga dinámica de los módulos a medida que son requeridos. Es pensado y realizado en forma modular y muy flexible, brindándole al desarrollador facilidades para programar sus propios módulos.

A su vez, los resultados obtenidos son persistidos en bases de datos MongoDB a través de una capa de acceso que abstrae al programador de la tarea de comunicarse en forma directa con el motor de MongoDB.

3.3.1.1. Pasos

Lo primero a destacar y común a todos los pasos es la carga dinámica de los módulos. Cada paso recibe uno o varios identificadores (el nombre de su directorio) de módulos junto con su configuración, y estos son cargados e inicializados con dicha configuración dinámicamente, es decir, los pasos no "conocen" a los módulos que están llamando. Esta característica hace que el framework sea más fácil de extender con nuevos módulos.

nombre	**direccion**	**telefono**
Juan Galo	Rincon 1562	095315468
Leonardo Vaz	Canelones 1242	091457963

Tabla 3.2: Fuente A

first name	**last name**	**address**
Juan	Gallo	rincon 1526
Leonard	Vas	canelon 1240

Tabla 3.3: Fuente B

A continuación se explicarán con mayor detalle algunos pasos claves del proceso de integración. Para facilitar la comprensión, se utilizará como ejemplo un flujo para integrar las fuentes A y B (mostradas en las tablas 3.2 y 3.3), y se acompañará con los documentos JSON almacenados en cada paso, que son la traducción del formato interno que se maneja en memoria durante la ejecución.

```
{
    "columns" : [
        {
            "fields" : [
                {
                    "type" : 1,
                    "output_field" : null,
                    "value" : "Juan",
                    "tags" : []
                }
            ],
            "name" : "first name",
            "custom_name" : null,
            "is_new" : false
        },
        {
            "fields" : [
                {
                    "type" : 1,
                    "output_field" : null,
                    "value" : "Gallo",
                    "tags" : []
                }
            ],
            "name" : "last name",
            "custom_name" : null,
            "is_new" : false
        },
        {
            "fields" : [
                {
                    "type" : 1,
                    "output_field" : null,
                    "value" : "rincon 1526",
                    "tags" : []
                }
            ],
            "name" : "address",
            "custom_name" : null,
            "is_new" : false
        }
    ]
}
```


Figura 3.7: JSON correspondiente a un registro luego del paso de extracción

Extracción Los registros recién extraídos sólo tienen algunos metadatos que se pudieron extraer de la fuente; los campos aún no están estandarizados, etiquetados ni segmentados. Este paso almacena los registros por un lado y el esquema original por otro, ya que éste tiene información útil que es utilizada por módulos de pasos posteriores.

Los registros extraídos se almacenan como se muestra en la figura 3.7. El campo "fields" en este paso tiene un solo elemento que corresponde al valor entero de la columna. El campo "type" indica el tipo de dato (string, número, fecha, etc.) según un enumerado definido en el código y el campo "name" contiene el nombre de la columna. El resto de los campos se explicarán más adelante ya que no son relevantes en este paso.

```
{
    "columns" : [
        {
         "name" : "address",
            "fields" : [
                {
                    "value" : "rincon",
                    "tags" : [
                        "WN"
                    ],
                    ...
                },
                {
                    "value" : "1526",
                    "tags" : [
                        "N34"
                    ],
                    ...
                }
            ],
            ...
        },
        ...
    ]
}
```


Figura 3.8: JSON correspondiente a un registro luego de ser estandarizado y etiquetado

Estandarización y etiquetado Una vez llegado a este paso, los registros son procesados para identificar tokens relevantes dentro de los valores de las columnas. Mientras hasta aquí el campo "fields" contenía un solo elemento, ahora tiene un elemento por cada token identificado, y en el campo "tags" las etiquetas asignadas a ese token. En el ejemplo mostrado en la figura 3.8 se aplicó tokenización de direcciones, y se puede ver cómo el módulo separa el valor "rincon 1562" en calle (identificado por la etiqueta "WN", de "Wayfare name") y número (identificado por la etiqueta "N34", que se asigna a números de 3 o 4 dígitos). Los valores aparecen en minúscula porque en el paso de limpieza se aplicó el módulo de pasar a minúsculas.


```
{
    "columns" : [
        {
         "name" : "address",
            "fields" : [
                {
                    "output_field" : "Wayfare name",
                    "value" : "rincon",
                    ...
                },

                    "output_field" : "Wayfare number",
                    "value" : "1526",
                    ...
                }
            ],
            ...
        },
        ...
    ]
}
```


Figura 3.9: JSON correspondiente a un registro luego del paso de segmentación

```
{
    "name" : "address",
    "custom_name" : null,
    "fields" : [
        {
            "type" : 1,
            "output_field" : "Wayfare name",
            "value" : "n/A",
            "tags" : []
        },
        {
            "type" : 2,
            "output_field" : "Wayfare number",
            "value" : "n/A",
            "tags" : []
        }
    ],
    "is_new" : false
}
```


Figura 3.10: JSON correspondiente a parte del esquema almacenado en el paso de segmentación

Segmentación Este paso se encarga de darle semántica a los datos en base a su metadata. Mientras que en un principio los campos simplemente representaban un dato de la fuente original, ahora los campos representan información mucho más específica y con significado. En la estructura almacenada (figura 3.9), la semántica se encuentra representada en el campo "output_field" de cada campo. En el ejemplo mostrado se utilizó segmentación de direcciones para la columna dirección que dió como resultado 2 campos de salida, "Wayfare name" y "Wayfare number". Este paso almacena también los nuevos esquemas de las fuentes, que ahora incluyen los segmentos dentro de cada columna (figura 3.10), y que es utilizado en pasos posteriores.

Este paso puede ser salteado, y en ese caso los campos originales se mantienen como estaban, cambiando la forma de comparación, que pasa a utilizar las columnas completas.

Schema matching Este paso no tiene demasiada complejidad en su lógica, pero la clase de la cual hereda el resto de los módulos de schema matching tiene algunos detalles que vale la pena mencionar.

En este paso se define el esquema global (el esquema que a partir de aquí tendrán todos los registros de ambas fuentes), por lo que por cada match de columnas (cada match puede estar compuesto por N columnas de la fuente A con M de la fuente B), los módulos de schema matching deberán crear una columna nueva con la unión de los campos de cada conjunto de columnas. Continuando con el ejemplo, si se matchea la columna *nombre* de la fuente A con las columas $firstname$ y *lastname* de la fuente B, y *direccion* de la fuente A con *address* de la fuente B, los registros de la fuente B pasados al esquema global tendrán en la nueva columna *nombre_completo* la unión de las columnas mencionadas anteriormente (como se puede ver en la figura 3.12).

```
{
    "columns" : [
        {
            "fields" : [
                {
                    "type" : 1,
                    "output_field" : "First name",
                    "value" : "juan",
                    "tags" : [
                     "Given name"
                    ]
                },
                {
                 "type" : 1,
                    "output_field" : "Surname",
                    "value" : "gallo",
                    "tags" : [
                     "Given name"
                    ]
                }
            ],
            "name" : "__new__nombre__first name-last name",
            "custom_name" : "nombre completo",
            "is_new" : true
        },
        {
            "fields" : [
                {
                    "type" : 1,
                    "output_field" : "Wayfare name",
                    "value" : "rincon",
                    "tags" : [
                        "WN"
                    ]
                },
                {
                    "type" : 2,
                    "output_field" : "Wayfare number",
                    "value" : "1526",
                    "tags" : [
                        "N34"
                    ]
                }
            ],
            "name" : "__new__direccion__address",
            "custom_name" : "direccion",
            "is_new" : true
        },
        {
            "fields" : [],
            "name" : "s1_telefono",
            "custom_name" : null,
            "is_new" : false
        }
    ]
}
```


Figura 3.11: JSON correspondiente a un registro con el esquema global luego del paso de schema matching

Luego, a cada registro se le generan columnas vacías por cada columna no matcheada y perteneciente al otro esquema (a los registros de la fuente 1 se los completa con el resto de las columnas de la fuente 2 y viceversa). En la figura 3.12 sucedió esto mismo, los valores de nombre y apellido se fusionaron en una columna, y se agregó la columna *telefono* que pertenecía a la otra fuente.

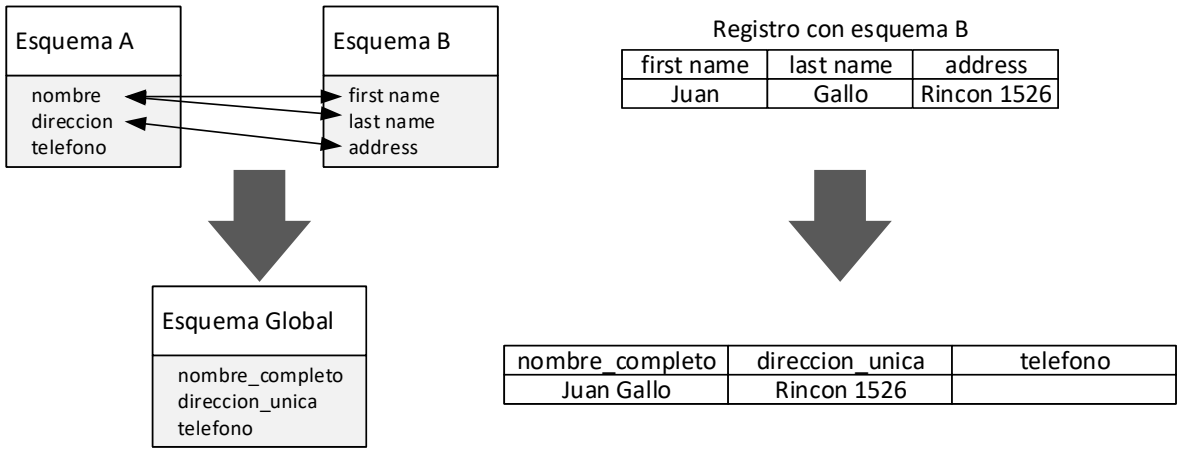

Figura 3.12: Generación del esquema global (izquierda) y pasaje de un registro al esquema global (derecha)

La clase de la cual heredan los módulos de schema matching se encarga de ir manteniendo el nuevo esquema global con todos los campos de salida de ambas fuentes para poder almacenarlo una vez finalizado la ejecución del módulo. Este nuevo esquema queda a disposición para ser utilizado por módulos de pasos posteriores.

En la figura 3.11 se puede ver como se almacenan los registros con el nuevo esquema. Las columnas matcheadas cuentan con un nombre personalizado en el campo "custom_name" que se mostrará en los pasos posteriores, y para identificarlas se les enciende la bandera "is_new".

```
{
    "source" : 1,
    "key" : "ri",
    "columns" : [
        {
         "name" : "__new__direccion__address",
            "fields" : [
                {
                    "output_field" : "Wayfare name",
                    "value" : "rincon",
                    ...
                },
                ...
            ],
            ...
        },
        ...
    ]
}
```

Figura 3.13: JSON correspondiente a un registro con el grupo de indexación asignado

Indexación Durante la indexación se generan los grupos de registros a comparar entre sí, y se almacenan como se puede ver en la figura 3.13, donde se aplicó indexación blocking standard con una codificación que toma los primeros 2 caracteres de la columna de dirección. Quedan igual que en el paso anterior, pero se les agrega un campo "key" que identifica al grupo al cual pertenecen, y un campo "source" que indica la fuente de origen.

Fuente	nombre	direccion	telefono
A	Juan Galo	Rincon 1562	095315468
B	Juan Gallo	rincon 1526	-
A+B	Juan Galo	Rincon 1562	095315468

Tabla 3.4: Fusión de registros

```
{
    "record1" : ObjectId("58c58bec5eedd322a0652c25"),
    "record2" : ObjectId("58c58bec5eedd322a0652c2a"),
    "comparisons" : [
        [
            "juan galo",
            "juan gallo"
        ],
        [
            "rincon",
            "rincon"
        ],
        [
            "1562",
            "1526"
        ]
    ],
    "vector" : [
        0.95,
        1.0,
        0.5
    ]
}
```

Figura 3.14: JSON correspondiente a una comparación realizada

Comparación En la comparación es donde se generan los vectores de similitud que indican cuán parecidos son los registros entre sí. En la figura 3.14 se puede ver cómo se almacenan estos vectores. El vector se compone por los identificadores de los registros comparados, las comparaciones realizadas (en el campo "comparisons") y las similitudes de cada comparación (en el campo "vector").

```
{
    "record1" : ObjectId("58c58bec5eedd322a0652c26"),
    "record2" : ObjectId("58c58bec5eedd322a0652c2b"),
    "match_type" : 1
}
```

Figura 3.15: JSON correspondiente a una comparación realizada

Clasificación Una vez calculados los vectores de similitud, se debe determinar si los pares de registros corresponden o no a las mismas entidades. Este paso se encarga de eso, y almacena el resultado con el formato mostrado en la figura 3.15, el campo "match_type" indica si el par es match (valor 1), no match (valor 0) o posible match (valor 2).

Data fusion Para finalizar, los pares de registros matcheados son unificados de acuerdo al módulo de data fusion elegido. En la tabla 3.4 se muestra cómo se genera un registro final suponiendo que el módulo elige la fuente 1 por ser de confianza en caso de conflictos.

3.3.1.2. Cómo agregar un módulo propio

Para integrar un nuevo módulo al framework basta con crear un directorio con un archivo Python dentro que extienda la clase abstracta correspondiente a los módulos del paso (mostradas en la figura 3.5). Todas estas clases a su vez también extienden una clase abstracta "Module" que define métodos y atributos comunes a todos los módulos, por lo que también tendrán que implementarse los métodos abstractos de esta clase.

En la figura 3.16 se muestra la estructura de directorios que los contiene. Consiste de un directorio principal que aloja un directorio por cada paso de integración, y estos tienen dentro un directorio por cada módulo que se debe llamar "module.py". Si por ejemplo se quisiera incluir un nuevo módulo de limpieza, se debe crear un nuevo directorio dentro del directorio del módulos de limpieza, y dentro del mismo crear el archivo module.py.

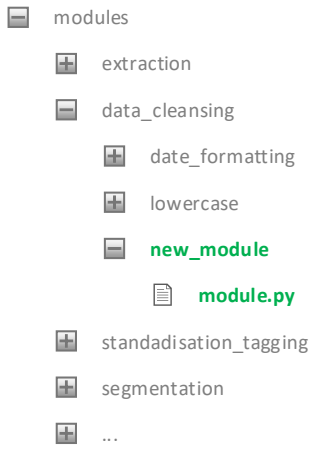

Figura 3.16: Estructura de directorios de módulos

El archivo de implementación debe contener una clase (no hay restricciones sobre su nombre) que extienda la que corresponda según el tipo de módulo. Para extenderla, el método inicializador debe recibir un parámetro especial de Python, "**kwargs", y pasarlo al inicializador de la clase padre. Este parámetro permite encapsular los parámetros que recibe la clase padre, y de esta forma se mantiene el código más ordenado y mantenible evitando tener que repetir los mismos parámetros en todas las clases. Si bien pueden tener todos los métodos que desee el desarrollador, es obligatorio implementar los siguientes tres o de lo contrario se dispara la excepción "NotImplementedError":

1. ***pretty_name***: debe retornar un nombre descriptivo del módulo, y es utilizado en la interfaz al momento de mostrar los módulos disponibles.

2. ***config_json***: debe retornar un diccionario que detalle los campos de configuración del módulo. Este diccionario será utilizado por la interfaz para generar dinámicamente los formularios de configuración, y debe seguir la estructura de los campos dinámicos explicados en 3.3.3.1. El formato del JSON de configuración para cada campo dinámico está documentado en la directiva que lo implementa.

3. **Método que ejecuta el algoritmo principal del módulo (*run*)**: debe ejecutar el algoritmo deseado. Los argumentos que recibe y el valor de retorno dependen del tipo de módulo, y se pueden ver en 3.5.

El desarrollador puede también guiarse con los módulos ya implementados.

3.3.1.3. Etiquetas y campos de salida

El framework provee etiquetas y campos de salida globales para que los desarrolladores puedan importarlos y hacer uso de los mismos en sus módulos. Actualmente el framework cuenta con las etiquetas y campos de salida presentadas en [4], entre otros.

Por otro lado, ya que cada campo de salida tiene su propia semántica, los que están integrados en el framework tienen como fin proveer compatibilidad entre todos los módulos de segmentación que pueden coexistir en el framework de forma tal que si dos módulos de segmentación son usados en distintas columnas y luego deben compararse entonces hay seguridad que se están comparando los mismos campos de salida. Sin embargo, está permitido que el desarrollador use sus campos de salida personalizados, pero quien lo haga deberá tener esto en cuenta.

Algo similar ocurre con las etiquetas: si bien existen etiquetas predefinidas globales al framework para que el desarrollador las importe en su módulo es posible que utilice sus propias etiquetas, pero si existen módulos de segmentación que no reconocen dichas etiquetas (etiquetas por fuera del framework) entonces no se podrá realizar tal segmentación.

El objetivo es que las etiquetas y campos de salida globales al framework crezcan a medida que los desarrolladores implementen sus módulos y necesiten agregar más para así evitar el uso de etiquetas y campos de salida personalizados que puede llevar a incompatibilidades entre módulos.

3.3.2. API REST

Para implementar la API REST se utilizaron los frameworks de Python, Django y Django REST. Django brinda la posibilidad de implementar una aplicación web en forma ágil, siguiendo un patrón Model-View-Template, lo que permite poner el foco en el motor del sistema.

A su vez Django REST Framework brinda serializadores que facilitan la tarea de convertir los objetos del sistema a formato JSON para interactuar con el usuario. Además también genera automáticamente vistas HTML de la API que permiten su navegación en forma amigable (figura 3.17).

3.3.2.1. Servicios

Las entidades mantenidas por la aplicación constan de Proyectos (contiene nombre, identificador y paso actual), los cuales tienen Configuraciones de pasos (con el nombre del paso ejecutado, configuración e identificador). Dichas entidades son persistidas en una base SQLite mediante un ORM[2], lo que agiliza el desarrollo puesto que no es fundamental ocuparse de las consultas SQL.

[2]Django brinda soporte para distintas bases de datos por lo que en el caso de un ambiente de producción no sería problema utilizar un motor de base de datos mas robusto.

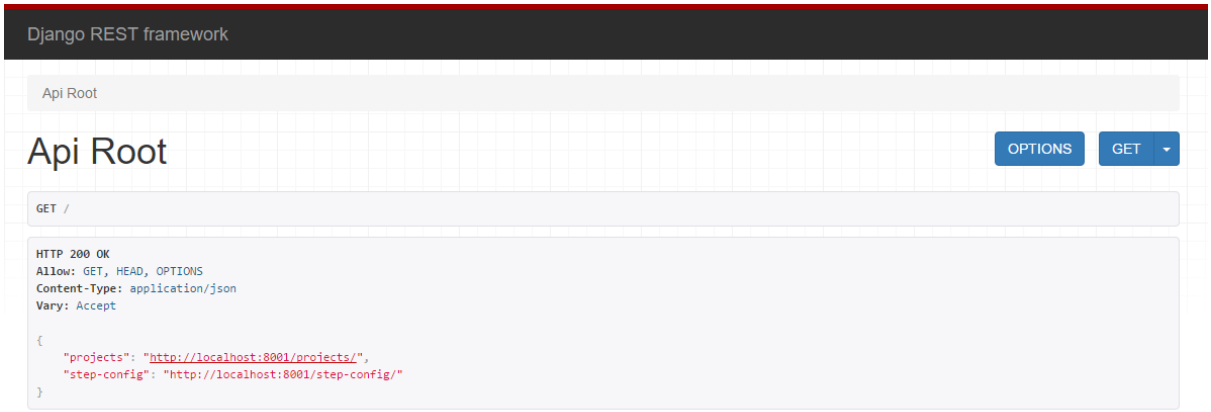

Figura 3.17: Vista HTML de la API

A través de la API es posible crear y borrar proyectos así como ejecutar pasos del proyecto enviando la configuración necesaria para cada uno. La ejecución de los pasos supone naturalmente la creación de configuraciones de pasos que registran lo realizado. Sin embargo la API también brinda la posibilidad de crear, editar y borrarlos en forma directa.

```
{
   "project_id":"29",
   "step":"DataCleansingStep",
   "config":{
      "source1":{
         "CLIENTE":[
            {
               "name":"delete_chars",
               "config":{
                  "chars":".,"
               }
            }
         ]
      },
      "source2":{
         "GRUPO":[
            {
               "name":"lowercase",
               "config":{

               }
            }
         ]
      }
   }
}
```

Figura 3.18: Ejemplo de JSON con la configuración que recibe la API para ejecutar el paso de limpieza de un proyecto

A modo de ejemplo, como se observa en la figura 3.18, basta enviar el identificador de un proyecto, el paso y su configuración para que dicho paso sea ejecutado.

Por otro lado se brindan otros servicios auxiliares como por ejemplo información de módulos disponibles para cada paso, datos necesarios para generar las vistas de previsualización de resultados de determinado paso en la interfaz gráfica y soporte de subida y descarga de archivos.

Finalmente la API permite generar un script Python que el usuario puede descargar y utilizar en la automatización de la creación y ejecución de proyectos de integración, en base a un proyecto previo ya ejecutado del cual obtiene la configuración de cada paso. Se utiliza la información almacenada en las configuraciones de paso del proyecto para generar un pequeño programa que hace uso de las librerías de Python de manejo de JSON y pedidos HTTP de manera de comunicarse con la API del mismo modo que lo hiciera un usuario a través de la interfaz gráfica.

```
Creating project Proyecto2
Project created
Executing Extraction Step...
{"status": "ok"}
Executing Data Cleansing Step...
{"status": "ok"}
Executing Standardisation And Tagging Step...
{"status": "ok"}
Executing Segmentation Step...
{"status": "ok"}
Executing Schema Matching Step...
{"status": "ok"}
Executing Indexing Step...
{"status": "ok"}
Executing Comparison Step...
{"status": "ok"}
Executing Classification Step...
{"status": "ok"}
Executing Data Fusion Step...
{"status": "ok"}
Executing Export Step...
Download file in http://localhost:8001/download-file/7eada2b7-96a1-41e1-98ad-b91f49180df7.csv/Datos
```

Figura 3.19: Salida de la ejecución en consola del script generado

En la figura 3.19 se observa la salida de la ejecución del script generado, el cual recibe como parámetro el nombre del nuevo proyecto y luego imprime cada paso que ejecuta y el resultado.

```
print "Executing Segmentation Step..."
config = json.loads(r'''{"source2": {}, "source1": {}, "skipstep": true}''')
data = {"project_id": project_id,"step": "SegmentationStep", "config": config }
req = urllib2.Request("http://%s/run/" % host)
req.add_header("Content-Type", "application/json")
response = urllib2.urlopen(req, json.dumps(data))
print response.read()
```

Figura 3.20: Fragmento de script mostrando la parte de segmentación

En la figura 3.20 se observan algunas líneas del script que puede generarse. En este caso, el fragmento carga la configuración del paso de segmentación en un JSON que luego envía a través de un POST HTTP a la API, para luego mostrar la respuesta del sistema al usuario.

3.3.3. Interfaz gráfica

La interfaz gráfica (GUI), es uno de los componentes fundamentales del framework ya que debe poder reflejar y soportar la modularidad y dinamicidad de los mismos, por lo tanto presenta un desafío con una complejidad comparable a la lógica aplicativa.

La interfaz gráfica está implementada con el framework AngularJS que nos provee una interfaz web moderna y fácil de utilizar. Cada paso está hecho mediante una directiva de AngularJS. Dichas directivas son compiladas en tiempo de ejecución lo que significó un esfuerzo extra a la hora de programar su funcionamiento.

3.3.3.1. Campos dinámicos

Los campos dinámicos son el componente más importante de la interfaz gráfica ya que la existencia de los mismos es fundamental para la modularidad y flexibilidad del framework. Cada desarrollador tiene a disposición una serie de campos para obtener entradas del usuario. Dichos campos son, al igual que los pasos, directivas de AngularJS pero son compilados y mostrados en tiempo real a medida que el usuario selecciona distintos módulos, en base al JSON de configuración que el desarrollador haya específicado. A continuación se describen los campos dinámicos implementados.

Subida de archivos Permite la subida de archivos.

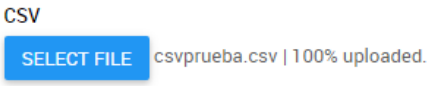

Figura 3.21: Subida de archivos

Texto Permite el ingreso de texto libre.

Figura 3.22: Texto

Número entero Permite el ingreso de un número entero solamente.

Figura 3.23: Número entero

Contraseña Permite el ingreso de texto libre. El texto ingresado es enmascarado.

Figura 3.24: Contraseña

Dropdown con selección simple Permite la selección de una opción entre una o más.

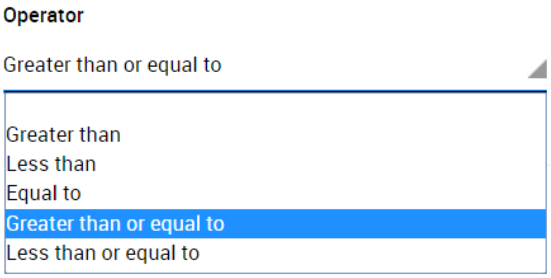

Figura 3.25: Dropdown con única selección

Dropdown con selección múltiple Permite la selección de una o más opciónes entre una o más.

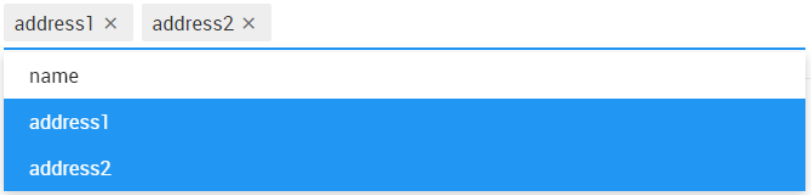

Figura 3.26: Dropdown con múltiple selección

Radio Permite la selección de una opción entre una o más.

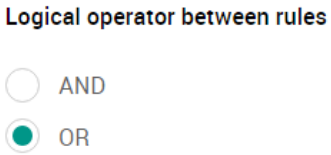

Figura 3.27: Radio

Radio (en una línea) Permite la selección de una opción entre una o más.

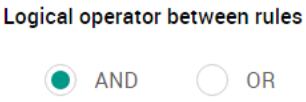

Figura 3.28: Radio (en una línea)

Deslizador Permite la selección de un número con la ayuda de un deslizador gráfico. Se puede definir la etiqueta mostrada, granularidad, color, comienzo y fin.

Figura 3.29: Deslizador

Deslizador (selección de rango) Permite la selección de un rango con la ayuda de un deslizador gráfico. Se puede definir la etiqueta mostrada, granularidad, color, comienzo y fin.

Figura 3.30: Deslizador (selección de rango)

Checkbox Permite la selección de una o más opciónes dentro de una lista.

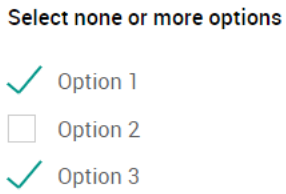

Figura 3.31: Checkbox

Toggle Switch Permite activar o desactivar una opción. Se puede especificar el color.

Figura 3.32: Toggle Switch

Entrada oculta Tipo especial de campo dinámico cuyo fin dependerá del módulo desarrollado. Es útil utilizar este campo dinámico si se quiere que el módulo sea utilizado por otros módulos (como es el caso de los de codificación). Para esto se debe incluir el identificador del módulo en el valor de este campo.

Fila Tipo especial de campo dinámico cuyo fin es contener hasta 4 campos dinámicos simples para poder mostrarlos en una sola fila.

Figura 3.33: Fila

Filas Tipo especial de campo dinámico cuyo fin es contener campos dinámicos de tipo fila de forma tal que el usuario sea capaz remover y añadir filas.

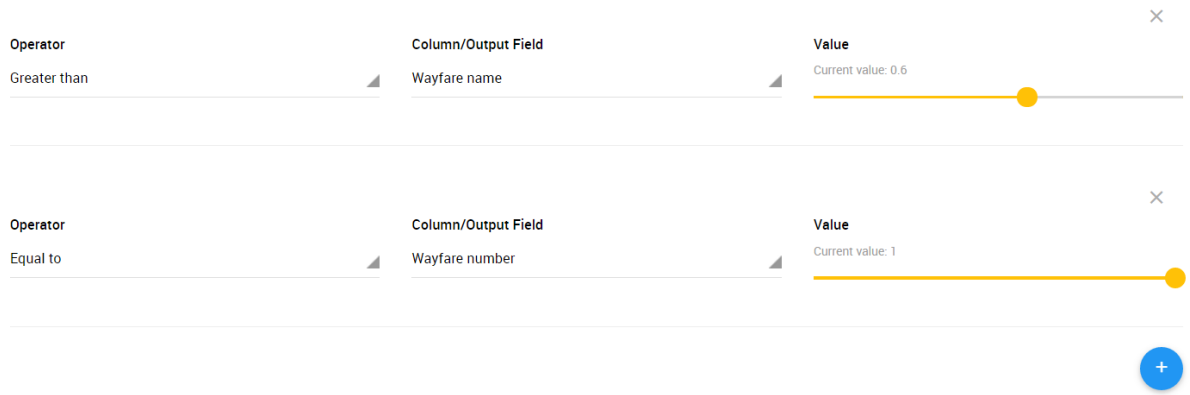

Figura 3.34: Filas

Cabe destacar al estar implementados mediante directivas de AngularJS es sencillo agregar un nuevo tipo de campo dinámico si se desea.

3.3.4. Validación con usuarios

La empresa IDATHA es una empresa uruguaya dedicada al procesamiento de datos provenientes de redes sociales y Big Data en general, procesamiento del lenguaje natural y machine learning, entre otras cosas. Fue identificada como un usuario potencial del framework ya que en muchos de sus proyectos surgen problemas de Entity Matching que deben resolver una y otra vez en cada uno de ellos. Para este proyecto sirvió como usuario para la validación de la herramienta desarrollada.

Logramos reunirnos con uno de los directores y desarrolladores de la empresa, el Ing. Benjamín Machín, quién ya tiene experiencia desarrollando en Python. Se obtuvieron comentarios positivos en general; primero se le mostró la herramienta en un flujo normal, luego fue interactuando con la herramienta de forma libre y finalmente se le presentó el código y la manera de integrar un nuevo módulo.

Un punto a destacar es que a la hora de utilizar la herramienta rápidamente sintió la necesidad de extender, modificar y crear varios módulos. Por ejemplo, al ver el módulo de extracción de PostgreSQL, se le ocurrió, además de las configuraciones ya requeridas, un campo opcional de “Consulta” donde el usuario pueda extraer datos a partir de una consulta SQL que el usuario escribe, haciéndonos notar el caso donde se quisiera hacer

una extracción de datos que provengan de la unión de dos o más tablas. Por otro lado, se le ocurrió que un módulo útil en el paso de limpieza sería uno que reemplace contenido en base a una expresión regular que el usuario ingrese.

A la hora de llegar a los pasos de estandarización/etiquetado y segmentación encontró dificultades para entender la diferencia entre ambos, lo cual es esperable ya que son pasos similares. Luego de entender la diferencia se preguntó si ambos pasos estarían acoplados en cuanto a las etiquetas (ya que un paso las añade y el siguiente las lee). No es el caso, ya que por esta razoń es que se incluyeron las etiquetas globales, así como también los segmentos globales tal como se explica en el apartado 3.3.1.3.

Por otra parte, Benjamín nos mostró una tecnología de inteligencia artificial de Microsoft que usan llamada LUIS (*Language Understanding Intelligent Service*) que realiza procesamiento del lenguaje natural en base a aprendizaje automático. Este servicio permite consultar una API con un texto arbitrario y el servicio responde con las entidades encontradas en el mismo según se haya definido en los datos de entrenamiento. Esto es un claro ejemplo de segmentación utilizando aprendizaje automático y sería posible implementar un módulo que haga uso del mismo.

También se le ocurrió una nueva funcionalidad que sería la posibildiad de agregar columnas calculadas, por ejemplo, promediar los precios en dos listados de productos. Si bien esto escapa al concepto de integración de datos, es una función útil que se podría agregar en un futuro.

La necesidad de modificar y/o integrar nuevos módulos es una de las más importantes y se tuvo en cuenta a lo largo de todo el proceso de desarrollo de la herramienta, por lo que está completamente cubierta, permitiendo dicha integración y/o modificación de módulos de manera fácil, rápida y aislada del resto del framework.

En cuanto a la usabilidad de la interfaz gŕafica, Benjamín señaló varios puntos a mejorar. Entre ellos:

- Mayor utilización de los espacios.
- Permitir avanzar al siguiente paso haciendo click en su pestaña además del botón de siguiente.
- Mostrar un resumen de la configuración de los pasos anteriores sin tener que volver atrás navegando por las pestañas.
- Resaltar el flujo de los pasos (por ejemplo, mostrando flechas entre las pestañas).

Aún así, elogió la usabilidad y dinamicidad de la misma, así como también la gran utilidad que encontró en la visualización de resultados parciales en cada paso.

Finalmente, concluyó que sin dudas podría ser utilizada en sus proyectos, y que desarrollarían módulos que se adapten a los mismos.

Capítulo 4

Caso de estudio

En este capítulo se presenta un ejemplo de uso completo y paso a paso del framework, destacando aquellos puntos que brindan al usuario grandes beneficios a la hora de integrar datos de dos fuentes.

Ya que cada proyecto de integración de datos es un mundo particular y requiere un análisis propio en cada caso, en el ejemplo que sigue a continuación **no se hace énfasis en la elección de módulos para obtener una integración óptima**, si no que el énfasis está en la modularidad, personalización, visualización de información y funcionalidades en general que brinda el framework.

Para dar comienzo al caso de estudio, se cuenta con dos fuentes: un archivo CSV con datos de empresas (Dirección, Contacto, Teléfono, Empresa) y por otro lado, datos de empresas alojados en una base de datos de MongoDB que contiene información como razón social, ciudad, dirección, persona a cargo y teléfono.

Comenzando con la extracción, para este ejemplo se utilizan los módulos de extracción para CSV y de MongoDB para las fuentes 1 y 2 respectivamente. Como se puede observar en la figura 4.1, al seleccionar el módulo de extracción para cada fuente se muestran los campos generados dinámicamente y al vuelo.

Al ejecutar el paso los datos comienzan a extraerse de las fuentes y los módulos transforman los datos al formato interno del framework para que sean guardados.

Al terminar la extracción se pasa al siguiente paso: limpieza. A partir de éste se pueden previsualizar el estado de los datos para cada paso en la parte inferior de la página. Por ejemplo, cuando se está configurando el paso de limpieza, se puede observar una muestra de registros luego de la extracción. Además para ese caso particular se informa la cantidad de registros extraídos para cada fuente.

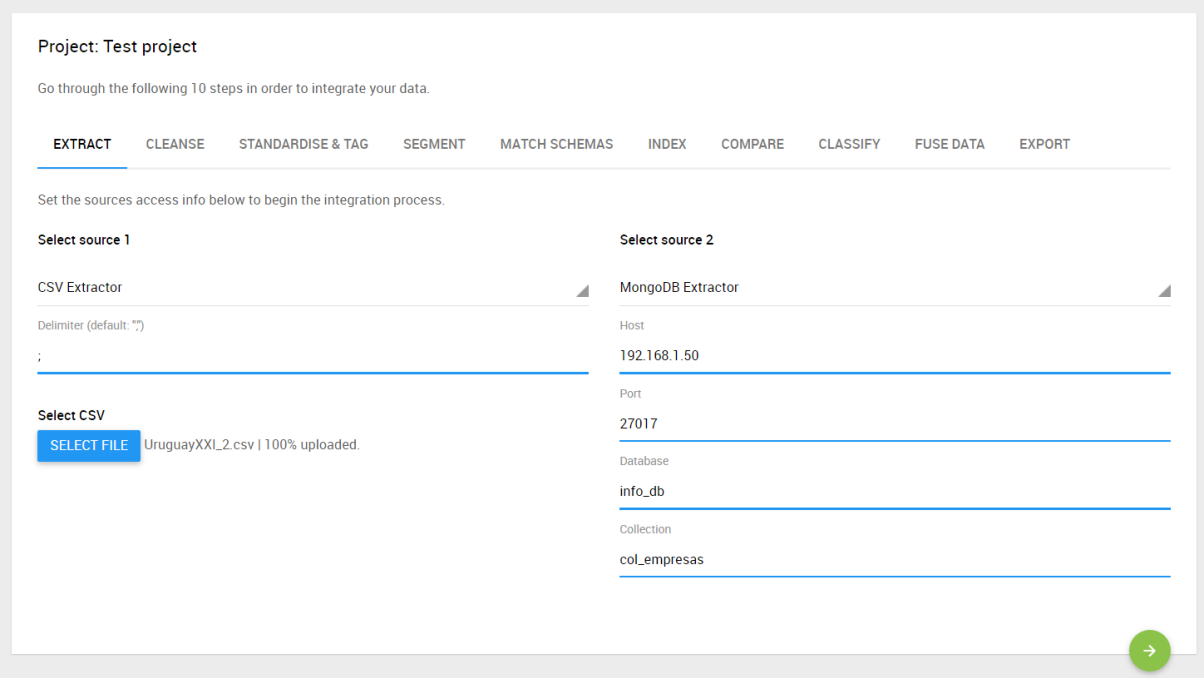

Figura 4.1: Captura del paso de extracción mostrando la configuración para la exracción de un CSV y de MongoDB

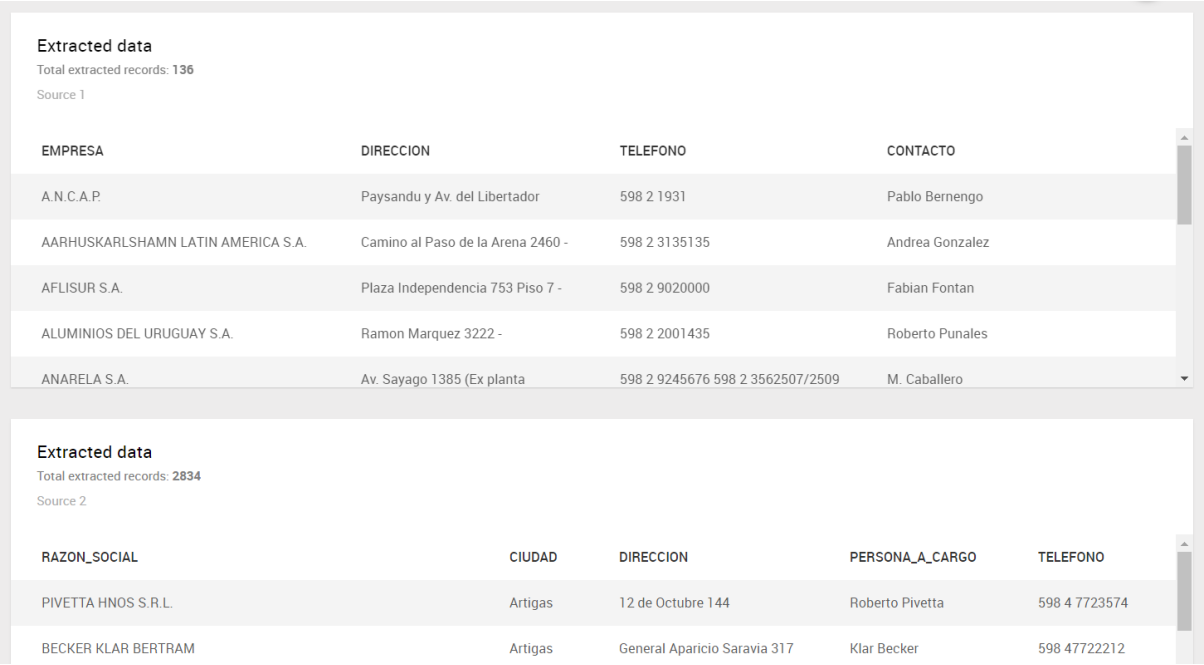

Figura 4.2: Información resultante de la extracción en el paso de limpieza

El próximo paso es el de limpieza de datos. En la siguiente imágen se puede ver en qué consiste la selección de módulos y columnas. En primer lugar, se pueden agregar una cantidad arbitraria de pares (columna, módulo) para aplicar limpieza. Si se selecciona más de un módulo para una misma columna entonces la limpieza será ejecutada en el orden especificado. Para este caso simplemente se aplica el pasaje a minúsculas ya que será suficiente, pero también hay otros como el de borrado de carácteres específicos, o de

espacios en blanco.

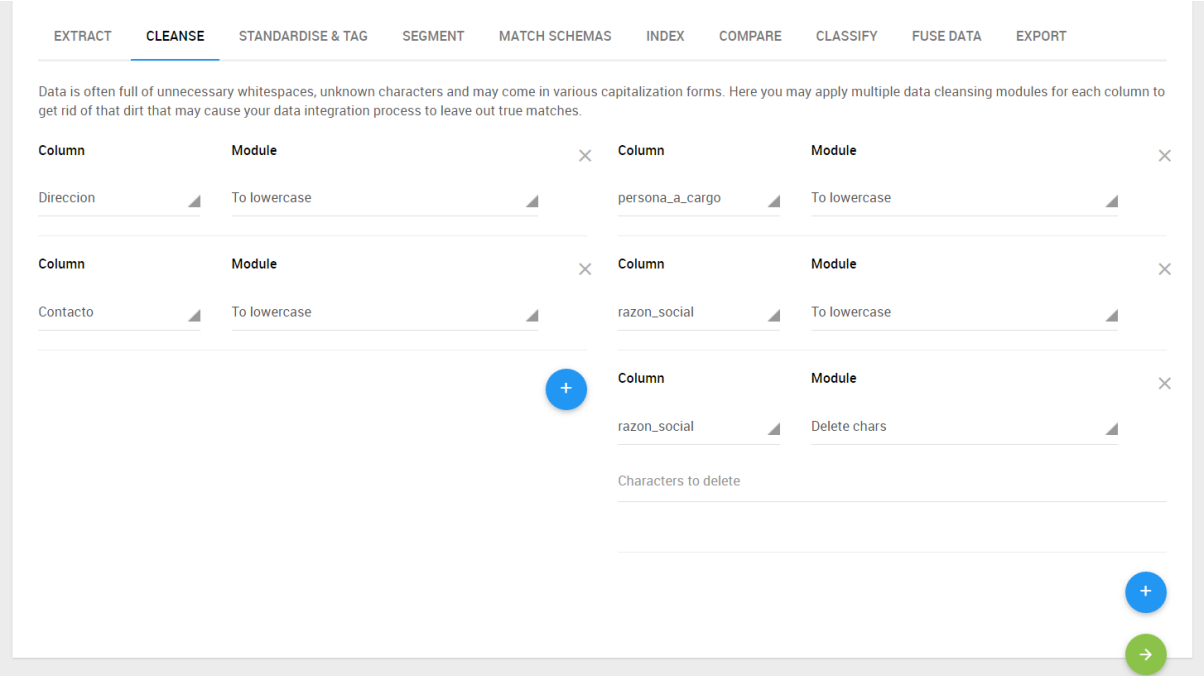

Figura 4.3: Captura del paso de extracción mostrando la configuración para la extracción de un CSV y de MongoDB

Al terminar de ejecutar la limpieza se continúa con el paso de estandarización y etiquetado pero antes de realizar la configuración se puede obervar nuevamente el estado de los datos luego de la limpieza (Figura 4.4).

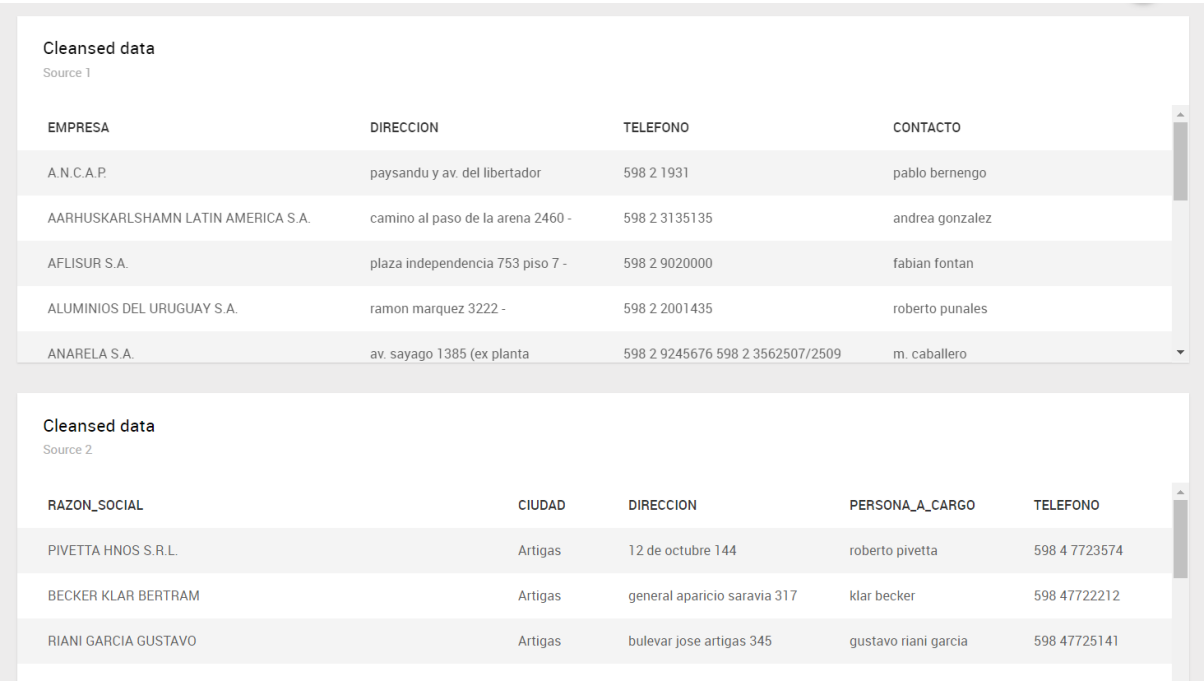

Figura 4.4: Previsualización de datos luego de la limpieza

En el paso de estandarización y etiquetado se usan los módulos de etiquetado para las direcciones uruguayas y para nombres de personas. En este paso solamente se permite la configuración de un módulo por columna, como se puede ver en la figura 4.5.

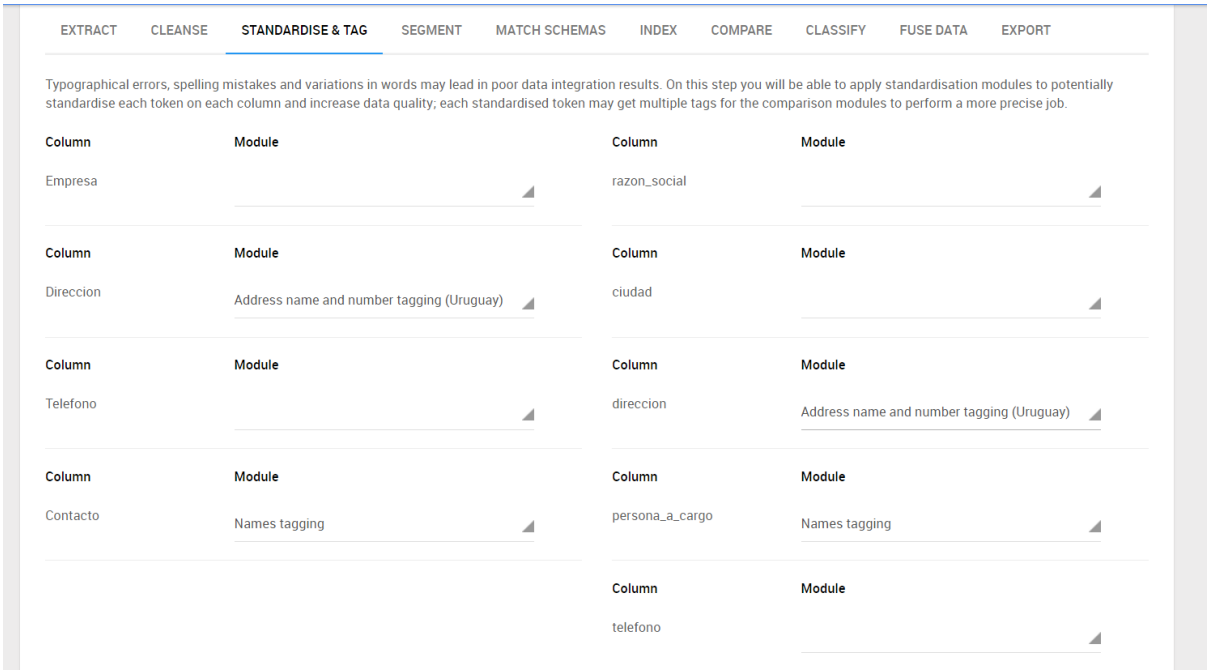

Figura 4.5: Captura del paso de estandarización y etiquetado

Al ejecutar el paso y pasar al siguiente se puede observar cómo fueron asignadas las etiquetas. Particularmente en el tercer registro de la muestra de datos de la fuente 1 (figura 4.6) se puede ver que se asignaron las etiquetas "GN" (correspondiente a Nombre de pila) a los tokens "ana" y "laura". Esto permite que posteriormente un módulo de segmentación pueda asignar el segmento "Primer nombre" a "ana" y el segmento "Segundo nombre" a "laura". En cuanto a la fuente 2 (figura 4.7) se puede ver que varios nombres no están etiquetados; esto es debido a que el módulo contiene una tabla de nombres predefinida y si un nombre no está en dicha tabla no se asigna ninguna etiqueta.

Figura 4.6: Previsualización de una muestra de datos luego de aplicar la estandarización y etiquetado (fuente 1)

Figura 4.7: Previsualización de una muestra de datos luego de aplicar la estandarización y etiquetado (fuente 2)

Por otro lado, también se aplicó el módulo de etiquetado de direcciones uruguayas, que simplemente utiliza una expresión regular y asigna la etiqueta “WN” (correspondiente a “Nombre de calle”) al primer set de tokens que estén formados por caracteres alfabéticos y la etiqueta “N34” (correspondiente a “Número entre 3 y 4 dígitos) al token que le siga y que contenga entre 3 y 4 dígitos. Este es un módulo de prueba, ya que en la práctica un módulo de este tipo debería fijarse en una tabla de lookup con nombres de calles y asignar las etiquetas en base a ello. Si bien en este ejemplo los módulos de etiquetado asignan una etiqueta por token, se podría haber asignado más de una por token, a diferencia de la segmentación, que solo puede asignar un campo de salida por cada set de tokens y éste debe ser único por columna.

El próximo paso es el de segmentación, y es aquí donde las etiquetas son usadas. Debe considerarse que este paso es el único opcional y como se puede ver en la figura 4.8 existe un toggle para saltearselo o no. En dicho paso se seleccionan los módulos de parsing para las direcciones y para los nombres para que segmente en (Nombre de calle, Número de calle) y (Primer nombre, Segundo nombre, Apellido) respectivamente.

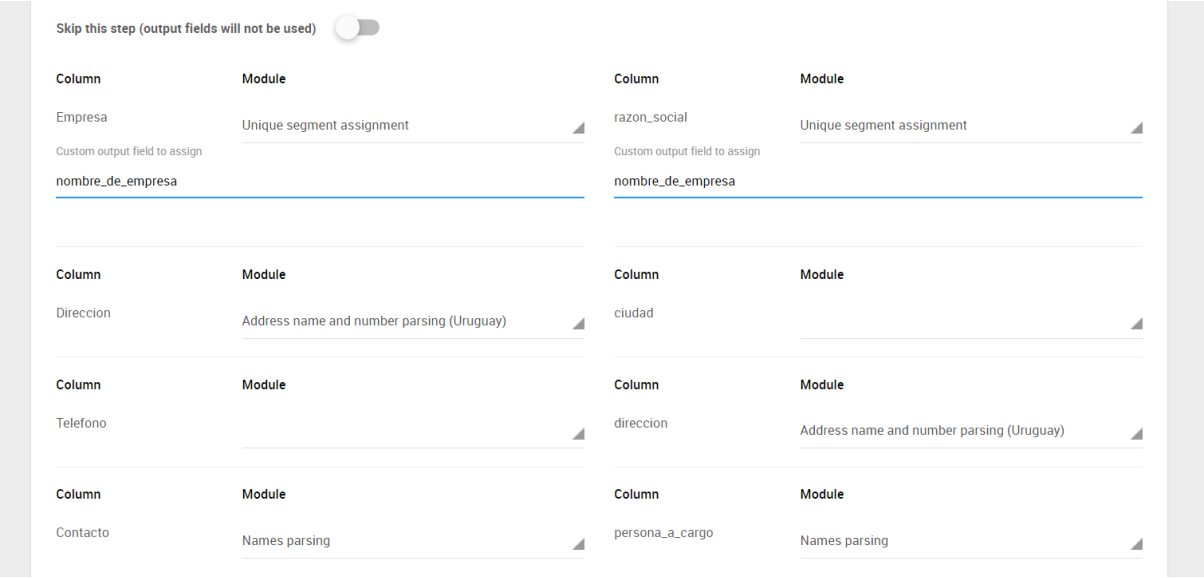

Figura 4.8: Captura del paso de segmentación

Pero ¿qué pasaria si se quisiera también comparar el nombre de la empresa sin querer segmentarlo? Esto no es posible si solamente se segmentara el nombre de empresa y dirección ya que el nombre de la empresa no tendría ningún segmento. Tampoco es posible segmentarlo ya que no existen módulos de etiquetado para nombres de empresas. Sin embargo, existe un módulo de segmentación que se puede ver seleccionado en la figura 4.8 (Unique segment assignment) que permite asignar un solo campo de salida personalizado a toda la columna. En este caso, se asigna "nombre_de_empresa"; de esta forma en todos los registros de ambas fuentes a la columna de nombre de la empresa se le asignará el mismo campo de salida, de modo de asegurar que estos valores serán tomados en cuenta a la hora de comparar.

Luego de ejecutada la segmentación se puede ver en las figuras 4.9 y 4.10 cómo quedan los datos. Los datos que están tachados son aquellos que no tienen ningún campo de salida asignado, y por ende, no van a ser usados a la hora de comparar. Se puede observar que efectivamente todos los strings de la columna que corresponden a "Empresa" y "razon_social" fueron asignados el mismo campo de salida ("nombre_de_empresa"). Por otro lado, se puede observar en particular que el string "ana laura deslissague" fue correctamente segmentado ya que el módulo de parsing de nombres asigna el campo de salida "Segundo nombre" en caso de que ya haya encontrado el primer nombre (dado por la etiqueta "GN").

Figura 4.9: Captura del los datos segmentados (fuente 1)

Figura 4.10: Captura del los datos segmentados (fuente 2)

El próximo paso es el de schema matching. En este paso el framework solo cuenta con un módulo: el de matching manual. En este módulo se pueden seleccionar N columnas de la

fuente 1 y matchearlas con M de la fuente 2. Para este ejemplo, se matchean las columnas que sabemos que tienen los mismos campos de salida ya que en la comparación solamente se compararán aquellos campos de salida que sean iguales dentro de la/s columna/s matcheada/s. En la figura 4.11 se puede ver este mapeo; y además se puede observar un toggle que permite agregar el resto de las columnas al esquema. Esto es, el esquema global tendrá las nuevas columnas que fueron matcheadas y asignadas un nombre particular como se puede observar y además el resto de las columnas de ambas fuentes que no fueron matcheadas. Para simplificar el ejemplo, se tomarán solamente las columnas matcheadas.

En la figura 4.12 se muestra el esquema resultante, la primer fila muestra las columnas y la segunda los campos de salida.

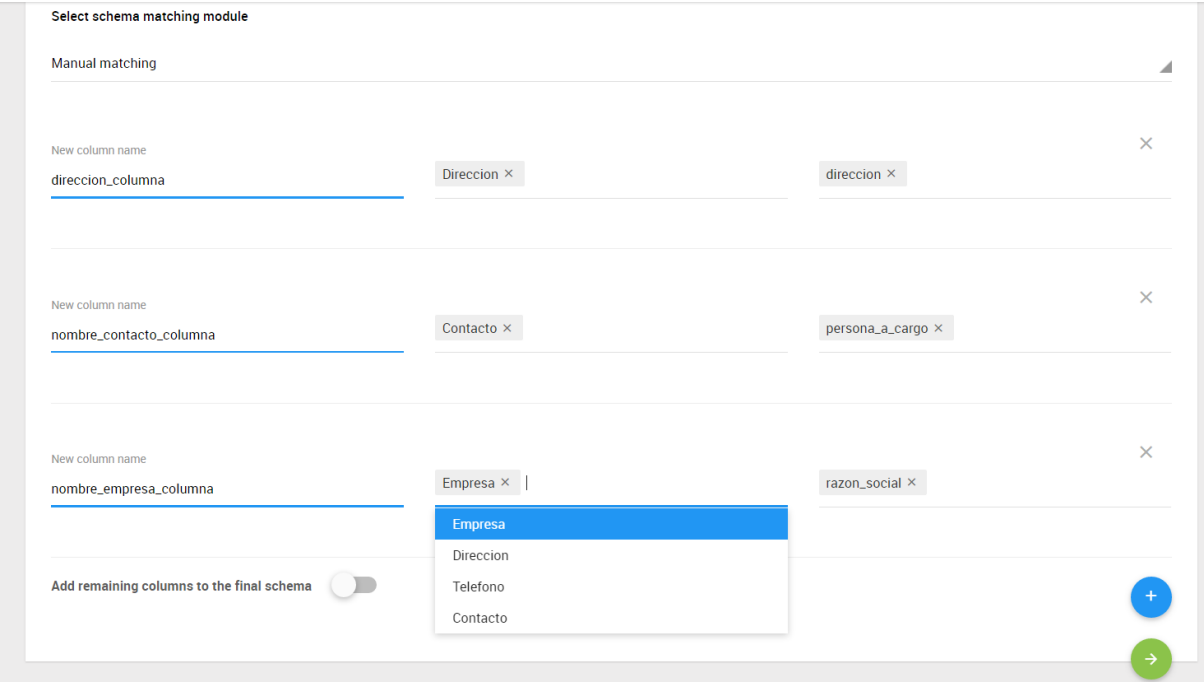

Figura 4.11: Captura de la configuración de schema matching manual

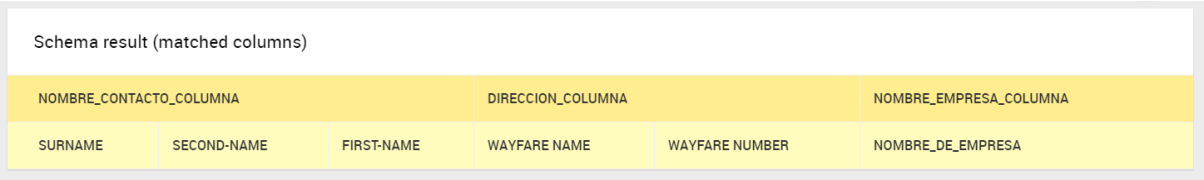

Figura 4.12: Esquema global generado luego de realizado el matching manual

El próximo paso es el de indexación. El prototipo cuenta con módulos que ofrecen índice completo (se comparan todos con todos) y blocking estándar. En este ejemplo, se usa el módulo de blocking estándar. En la figura 4.13 al seleccionar el módulo blocking estándar puede observarse como se generan los campos dinámicos de tipo "filas", permitiendo agregar una cantidad arbitraria de filas. Lo que se hace es crear la clave que definirán a los bloques. En este caso la clave de cada grupo está formada por los primeros dos caracteres del nombre de la persona seguidos por el primer carácter de la direccion, formando una

clave de 3 caracteres.

También se puede notar en la figura 4.13 que al seleccionar el módulo de blocking estándar se puede a su vez seleccionar módulos de codificación cuya configuración es también cargada dinámicamente en tiempo de ejecución.

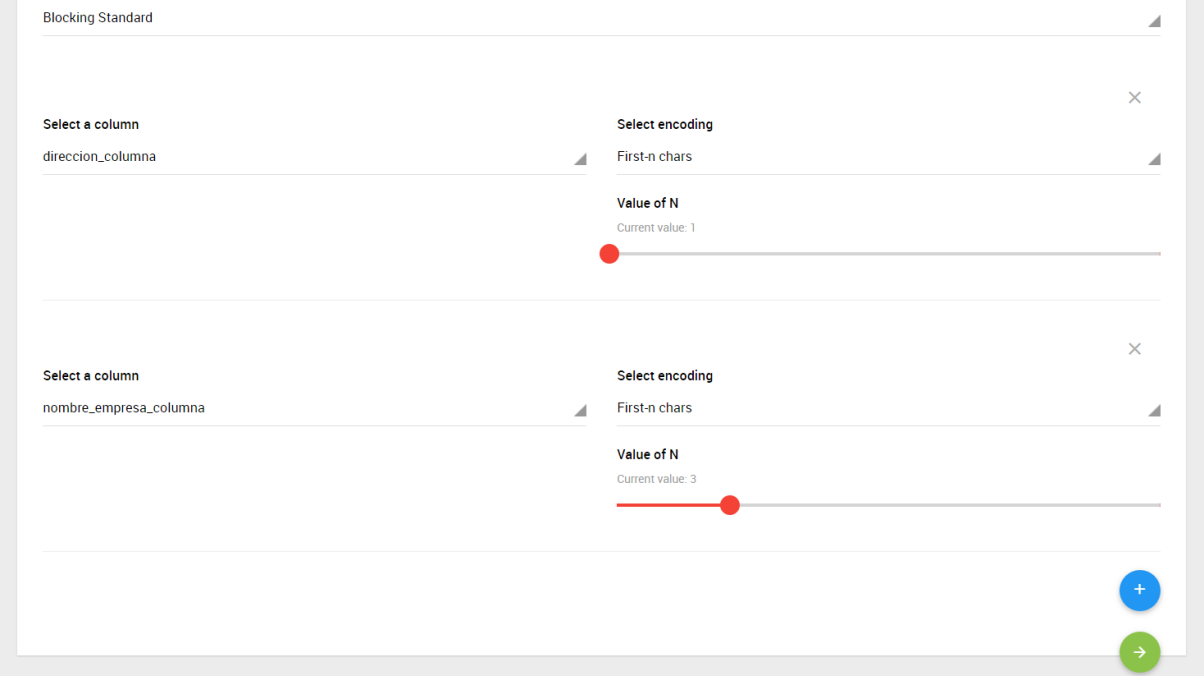

Figura 4.13: Selección de módulo de indexación blocking estándar y configuración del mismo usando codificación

Al ejecutar el paso, se puede ver información asociada al resultado de la indexación (figura 4.14): cantidad de comparaciones a realizar luego de indexar, cantidad de comparaciones que se hubieran hecho sin haber utilizado indexación y el porcentaje de reducción. Además se listan todos los grupos generados y la distribución de los registros de cada fuente por grupo generado.

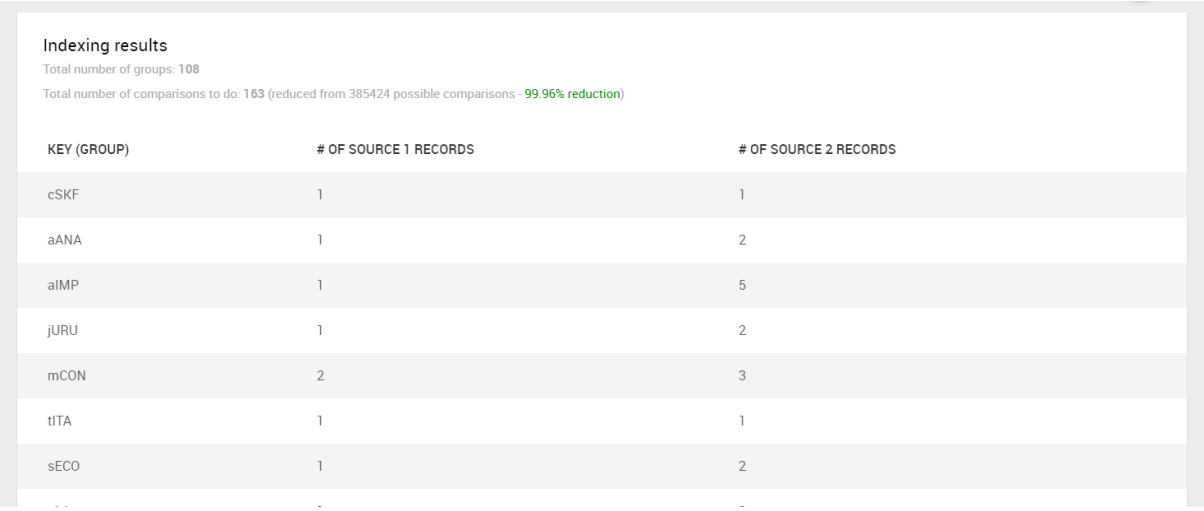

Figura 4.14: Información resultante de la indexación

El siguente paso es el de la comparación. Aquí se debe elegir un módulo de comparación por campo de salida (o columna si la segmentación fue salteada) y un peso (por defecto son todos 1). Esta configuración se puede observar en la figura 4.15

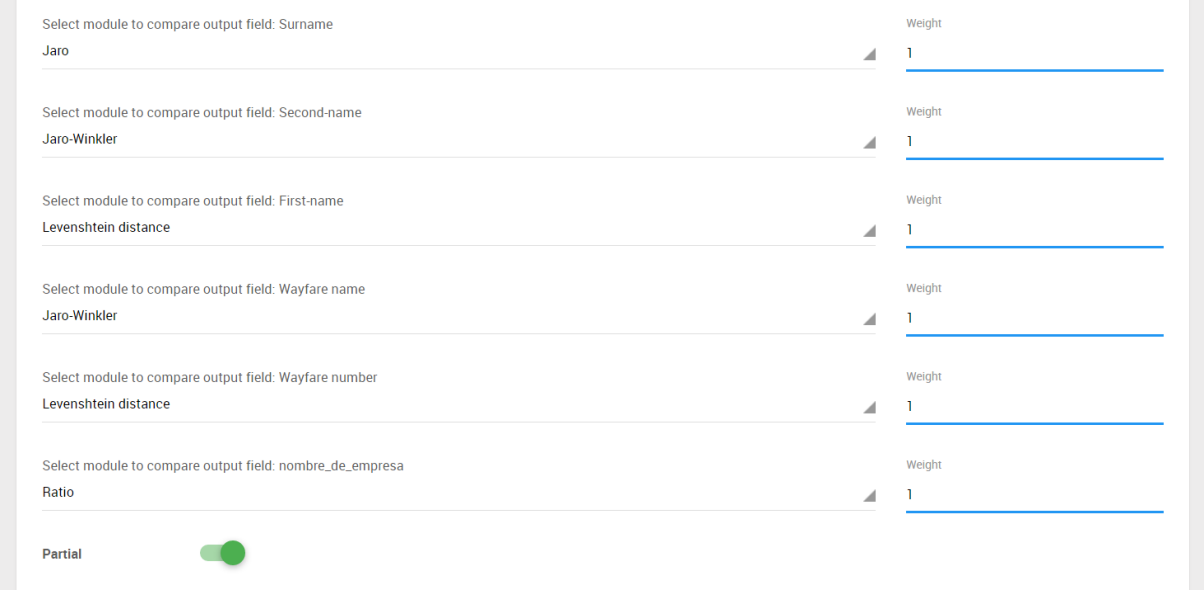

Figura 4.15: Pantalla de configuración del paso de comparación

Luego de ejecutada la comparación se podrán ver algunos de los vectores de similitud generados (ver figura 4.16). Los valores en los vectores de similitud generados siempre son entre 0 y 1. El paso se encarga de normalizarlos si algún peso fue asignado a las comparaciones.

Comparison results

Total comparisons made: **163**

	rodriguez	walter	1474	cebollati 1474, of	OXITENO URUGUAY S.A.
0	1	1	1	1	1
palma		daniel	200		BADER INTERNATIONAL SUC. URUGUAY
palma		daniel	200		BADER INTERNATIONAL SUC. URUGUAY
1	0	1	1	0	1
			716	av. general flores	QUIMICA ORIENTAL S.A.
		fabricio	3488	asilo	QUINQUIN S.R.L.
0	0	0	0	0.16666666666666663	0.33333333333333337
		guillermo	3857	pestalozzi	MANUEL BOULLOS.A. S.A.
	boullosa	guillermo	3857	pestalozzi	MANUEL BOULLOS.A. S.A.
0	0	1	1	1	1

Figura 4.16: Resultados parciales de la comparación. Se muestran los registros comparados y debajo el correspondiente vector de similitud

En el siguiente paso, que es el de clasificación, se debe seleccionar el módulo que tomará como entrada los vectores de similitud y los clasificará en tres categorías: match,

non-match y match potencial en base a determinado criterio que haya implementado el desarrollador del módulo. El framework, en esta versión, cuenta con dos módulos de clasificación; por un lado se tiene el módulo de Fellegi-Sunter que es muy simple: promedia los valores del vector y clasifica los según un rango especificado. Por otra parte se tiene un módulo mas complejo que es el de clasificación basada en reglas que implica establecer una cantidad arbitraria de reglas lógicas por entrada del vector. En el ejemplo se usa este último.

En la figura 4.17 se puede ver que este módulo genera dos tipos de campos dinámicos: un botón de radio para elegir qué operador lógico se aplicará entre las reglas y un campo dinámico de tipo filas para crear las reglas. A su vez, por cada regla (fila) debemos configurar el campo de salida (o columna si aplica), el operador y el valor del vector de similitud.

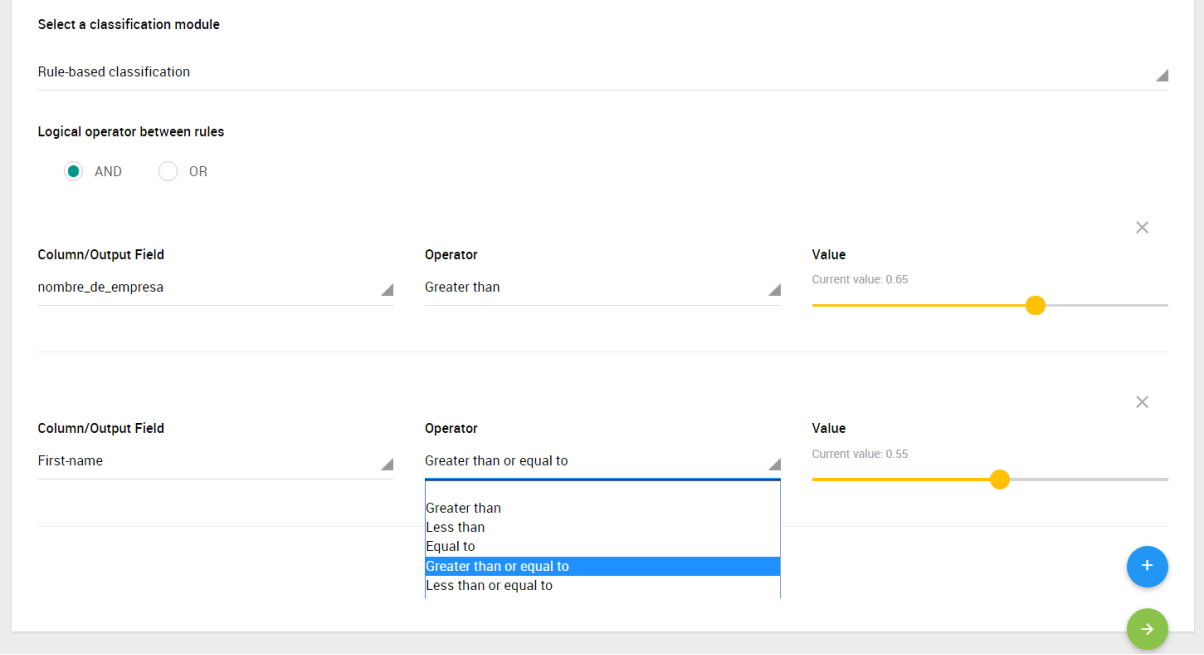

Figura 4.17: Resultados parciales de la comparación. Se muestran los registros comparados y debajo el correspondiente vector de similitud

Al terminar la clasificación se puede ver información asociada al resultado como la cantidad de matches, no matches y matches potenciales junto a su porcentaje sobre el total de par de registros comparados. A su vez se puede observar una muestra de a lo sumo 5 pares por clase. En la figura 4.18 se puede ver una muestra de pares de registros clasificados como match (color verde) y no matches (color rojo). El módulo de clasificación basada en reglas solamente clasifica en matches y no matches.

Si en vez de la clasificación basada en reglas se selecciona el módulo de clasificación Fellegi-Sunter con una configuración como muestra la figura 4.19 se obtendrán algunos matches potenciales. En la figura 4.20 se muestran algunos de los pares que serían clasificados como matches potenciales (color amarillo) si se hubiera usado la clasificación de Fellegi-Sunter.

Para este ejemplo, se seguirá con los resultados del módulo basado en reglas.

Figura 4.18: Muestra de pares clasificados como matches (verde) y no matches (rojo)

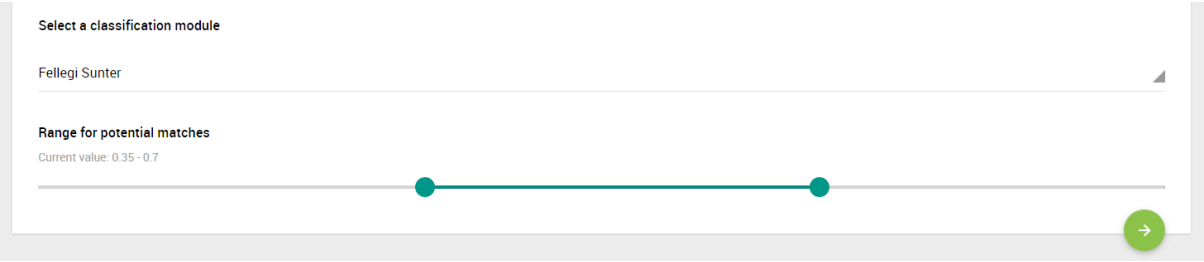

Figura 4.19: Configuración del módulo de clasificación Fellegi-Sunter con el campo dinámico de deslizador de rangos para seleccionar el rango de matches potenciales. Los que estén por debajo de ese rango serán clasificados como no matches y los que estén por encima como matches.

En el próximo paso, Data Fusion, se debe seleccionar un módulo para fusionar los pares de registros clasificados como match. El framework cuenta con un módulo de data fusion el cual permite elegir una fuente de confianza y en caso de match quedarse con el registro de la fuente confiable. En la figura 4.21 se puede observar esta selección.

Figura 4.20: Muestra de pares clasificados como matches potenciales (amarillo) y matches (verde)

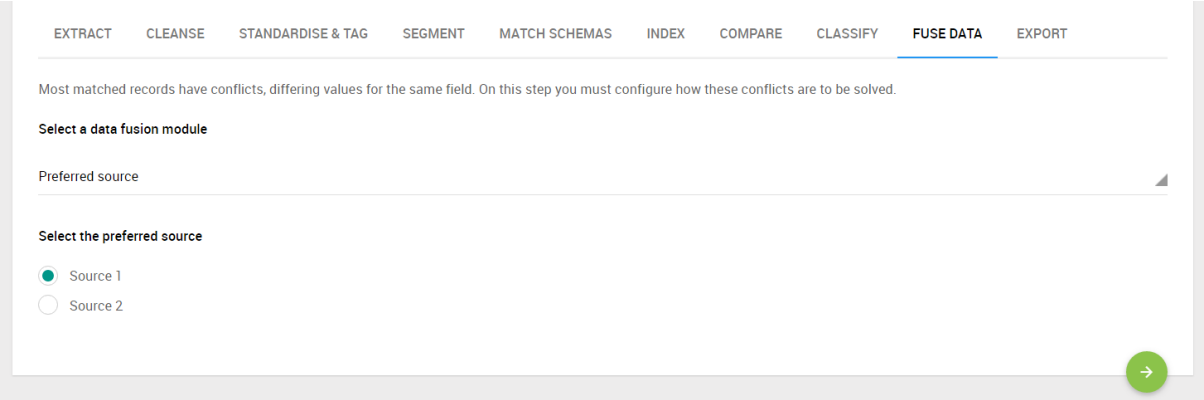

Figura 4.21: Pantalla de configuración del módulo de data fusion: selección de fuente de confianza

Figura 4.22: Registros fusionados

Al terminar el paso de data fusion, una muestra de los registros fusionados y finales a exportar pueden ser vistos en la previsualización (figura 4.22). Y es aquí donde finalmente se debe seleccionar el módulo de exportación. El framework cuenta con exportación a CSV o a MongoDB.

El paso de exportación incluye además una opción que permite incluir solamente los registros que fueron matcheados. Por defecto se exportan los registros que fueron fusionados y además los que no. En la figura 4.23 se ilustra esto.

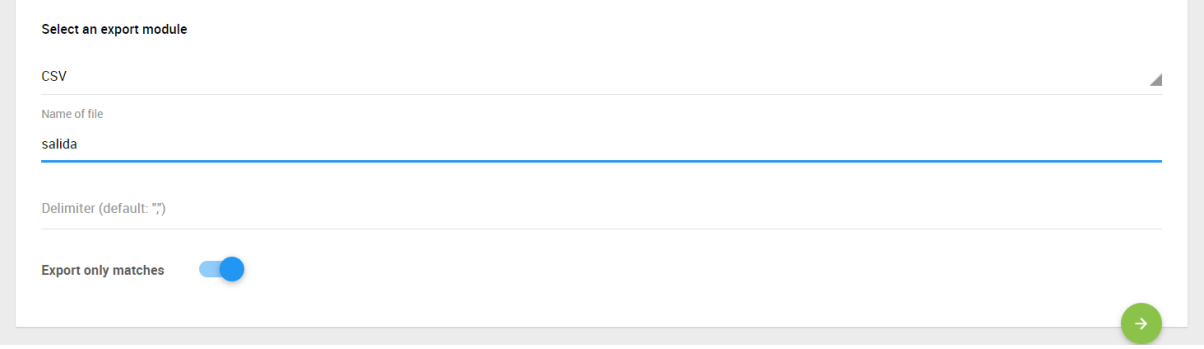

Figura 4.23: Paso de exportación

Hasta ahora se ejecutó un flujo completo de integración, pero ¿si no se esta satisfecho con los resultados de, por ejemplo, la comparación? En estos casos el framework permite al usuario volver a cualquier paso previamente ejecutado y volver a configurarlo para poder ejecutarlo nuevamente. En la figura 4.16 se puede ver que muchos registros no contienen apellido y en general hay campos vacíos, por lo que haber usado tres campos de salida en la columna de nombres pudo no haber sido lo más conveniente; entonces el usuario puede volver al paso de segmentación y asignar un único segmento a las columnas de nombres (4.24). Al momento de ejecutar este paso algunas de las configuraciones de los pasos que le siguen serán perdidas y será necesario volver a configurarlos pues algunos pasos dependen del anterior.

A la hora de comparar los nombres, al tener un solo segmento que incluye tanto el primer nombre, segundo nombre y apellido el usuario puede encontrar conveniente seleccionar el módulo de Token Set que es un módulo de comparación para strings con múltiples tokens.

Luego de ejecutar el paso de comparación con la nueva configuración el usuario puede volver a evaluar el resultado observando la muestra de datos en la previsualización y decidir si proseguir con el siguiente paso o volver a cualquier paso anterior y reconfigurar los módulos/pasos para obtener una integración de datos óptima.

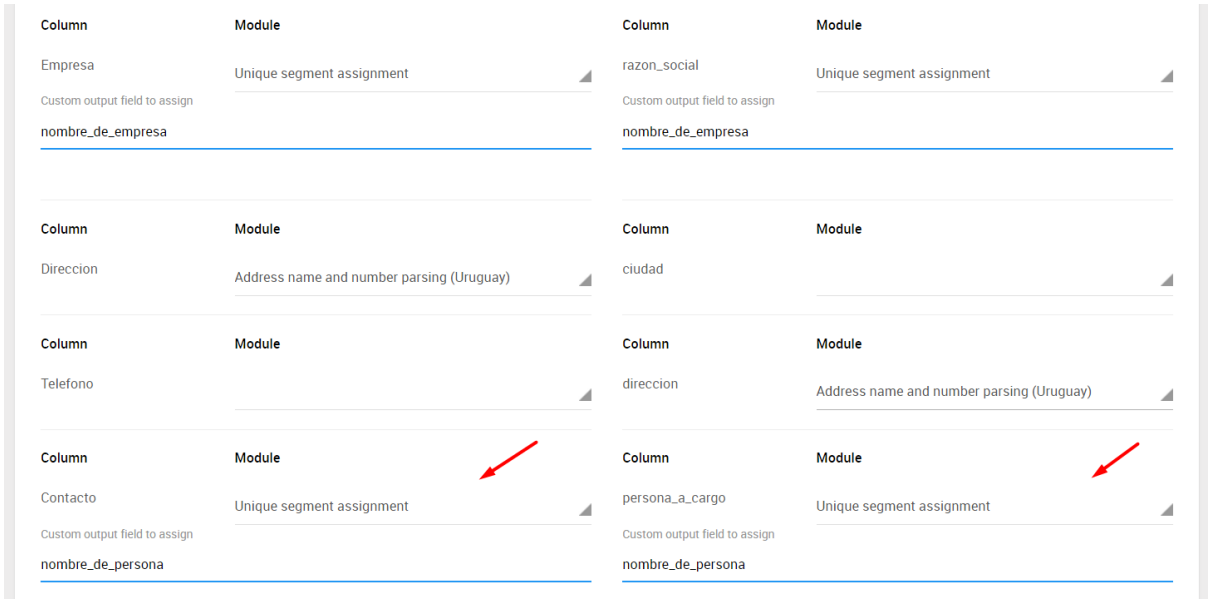

Figura 4.24: Nueva configuración del paso de segmentación. Se asigna un único campo de salida a las columnas que corresponden a nombres en ambas fuentes

Capítulo 5

Conclusiones y trabajo a futuro

El proceso de investigación realizado nos permitió obtener información principalmente acerca de diferentes algoritmos de distintas etapas de integración, fundamentalmente de data matching, pero en nuestra opinión, nos encontramos con falta de procesos y técnicas de integración que abarcaran todas las etapas necesarias para integrar datos. Por tanto, fue necesario combinar distintas técnicas de integración de varios autores para desarrollar los pasos de integración que utiliza el framework implementado.

Durante el análisis y diseño de la herramienta hubo decisiones difíciles, pero consideramos que se lograron superar de forma satisfactoria. Una de estas fue la definición del formato interno, que debía ser lo suficientemente flexible como para soportar distintos tipos de fuente, pero lo suficientemente estructurado para guiar al desarrollador de nuevos módulos en su tarea. Otra de las decisiones clave fue la de incluir o no el paso de segmentación, ya que es de los más complejos y determina el resto del flujo, porque la calidad de la segmentación es decisiva; datos mal segmentados van a dar resultados erróneos. Además de esto, estaba la problemática de qué sucedería si no hubieran módulos de segmentación que se ajustaran a los datos a integrar; no se puede obligar al usuario a desarrollar módulos de segmentación para todos los tipos de datos que utilice. Esto se solucionó con dos funcionalidades: por un lado, permitir saltear la segmentación, en cuyo caso se utilizan las columnas completas; y por otro, el módulo de segmentación que asigna un único campo de salida a la columna completa. Esto último permite utilizar segmentación en campos que tengan módulos de segmentación que se adecuen, pero al mismo tiempo comparar columnas con datos que no tengan módulos de segmentación útiles.

En base a la validación con IDATHA, se concluye que la herramienta es útil en casos reales; en particular fue muy valorada la fácil extensibilidad que permite ajustar la herramienta a las necesidades de la empresa. Se encontró particularmente interesante la inteligencia artificial de Microsoft, LUIS, con la que se pueden generar módulos de segmentación que aprovechen esta tecnología ya desarrollada y soportada por una empresa de gran porte.

El alto nivel de soporte de MongoDB (módulos de extracción/exportación) agrega un gran valor al framework frente al resto de las herramientas analizadas, ya que hoy en día dicho sistema de base de datos es muy utilizado en todo tipo de entornos y ninguna de las herramientas analizadas provee ningún tipo de soporte con fuentes NoSQL.

A pesar de que existen un sinfín de estudios al respecto, aún no se han logrado algoritmos

que logren solucionar problemas de integración completos (todos los pasos). Las principales trabas que han evitado la automatización completa de estos trabajos, y para las cuales alentamos realizar futuros proyectos de grado son: el problema de schema matching, para el cual sigue siendo necesaria la intervención de seres humanos que conozcan el dominio a integrar, y la elección de algoritmos de comparación que se adecuen mejor a los datos.

Si bien el prototipo implementado incluye una variada cantidad de módulos y funcionalidades que ya lo hacen de gran utilidad, y tuvo una buena aceptación en general, siempre hay lugar a mejoras y nuevas funcionalidades. A continuación se mencionan algunas de estas que no se incluyeron en el prototipo, pero quedan planteadas como trabajo a futuro.

Módulos de recomendaciones Una de las funcionalidades más interesantes y desafiantes para agregar consiste en módulos semi automáticos que se auto configuren y permitan al usuario ajustar esta configuración. Esta configuración recomendada puede ser de gran utilidad por ejemplo en el paso de schema matching, donde puede volverse tedioso ingresar manualmente cada correspondencia. Estas recomendaciones se realizarían en base a metadatos como el esquema de las fuentes o en los datos mismos.

Concepto de "entidad" en formato interno El formato interno actual está pensado para trabajar sobre pares de conjuntos de registros que corresponden a un sola sola entidad del mundo real (clientes, usuarios, ventas, etc.). Viéndolo desde el punto de vista relacional, está pensado para integrar una tabla con otra (cada fuente corresponde a una tabla). Pero no permite, por ejemplo, integrar dos bases de datos enteras, o un conjunto de tablas con otro conjunto de tablas. Para permitir esto, se pensó en incluir al formato interno el concepto de "entidad", que representaría a un conjunto de registros de cierta entidad, y una fuente consistiría en un conjunto de entidades a ser integrado con otro conjunto de entidades. Este sería un cambio medular en el prototipo que llevaría a una reestructuración completa de todo el framework. Se decidió dejarlo afuera debido al balance entre el valor agregado y el costo adicional de implementarlo.

Deduplicación Un caso de uso similar al de integración de datos (pero con algún paso menos) que se podría agregar, es el de deduplicación. En la deduplicación se tiene un solo conjunto de registros, y el objetivo es hallar entidades repetidas y quedarse con una sola versión de cada duplicado.

Optimización de carga y procesamiento mediante paralelización En cuanto al rendimiento del framework de integración, una optimización casi mandatoria a futuro, sobre todo si se pretende trabajar con grandes conjuntos de datos o poner la herramienta en producción, es la de paralelizar el procesamiento en la ejecución de cada paso. En un principio, utilizando varios procesos o hilos en un mismo nodo, y en caso de ser necesario, reestructurando el framework para permitir el procesamiento distribuido en varios nodos.

Permitir otras interacciónes con el usuario Si bien los campos dinámicos ofrecen un alto grado de flexibilidad, una vez que se cargan, el contenido del formulario es estático. En algunos casos puede ser útil que ciertos campos extra se carguen dependiendo de una configuración seleccionada previamente (dentro de el mismo paso), por ejemplo, un botón

que en base a una configuración cargue otra para que el usuario continúe configurando.

También se podría agregar interacción durante la corrida del paso para resolver alguna duda que surja durante la ejecución, como por ejemplo, algún módulo de clasificación que en base a un rango especificado entre 0 y 1 le pregunte al usuario si el par corresponde a un match o no.

Exportación de resultados de pasos individuales Muchas veces no solo es importante el resultado final con los registros integrados, sino que suelen también ser útiles resultados intermedios de la integración, como por ejemplo los vectores de similitud entre cada registro, o las agrupaciónes del paso de indexación. Si bien estos resultados intermedios están disponibles gracias a que se mantienen almacenados en la base de staging, en un futuro sería de utilidad permitir descargarlos desde la interfaz gráfica.

Bibliografía

[1] Tobias Bachteler. *Merge ToolBox, Version 0.74, Getting Started*. German Record Linkage Center, 25 de mayo de 2012. URL: https://www.uni-due.de/~hq0215/documents/mtb_gettingstarted.pdf.

[2] Zohra Bellahsene, Angela Bonifati y Erhard Rahm, eds. *Schema Matching and Mapping*. 2011 edition. Heidelberg: Springer, 14 de feb. de 2011. 320 págs. ISBN: 978-3-642-16517-7.

[3] Stefano Ceri, Carol Batini y Shamkant B. Navathe. *Conceptual Database Design: An Entity-Relation Approach*. 1992.

[4] Peter Christen. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. 2012 edition. Springer, 4 de jul. de 2012. 290 págs.

[5] Xin Luna Dong y Felix Naumann. *Data Fusion - Resolving Data Conflicts for Integration*. 2009.

[6] Xin Luna Dong y Divesh Srivastava. *Big Data Integration*. URL: http://www.morganclaypool.com/doi/abs/10.2200/S00578ED1V01Y201404DTM040 (visitado 17-05-2016).

[7] Jens Bleiholder y Felix Naumann. *Declarative data fusion – syntax, semantics, and implementation*. ADBIS'05 Proceedings of the 9th East European conference on Advances in Databases e Information Systems, 2005. URL: http://dx.doi.org/10.1007/11547686_5.

[8] Jens Bleiholder y Felix Naumann. *Journal ACM Computing Surveys Volume 41 Issue 1, Article No. 1, Data Fusion*. ACM. URL: http://doi.acm.org/10.1145/1456650.1456651.

[9] Lars Kolb, Andreas Thor y Erhard Rahm. *Load Balancing for MapReduce-based Entity Resolution*. Abr. de 2012. URL: https://arxiv.org/pdf/1108.1631.pdf (visitado 18-05-2016).

[10] Michael Lesk. *Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone*. Abr. de 2012. URL: http://dl.acm.org/citation.cfm?id=318728 (visitado 18-05-2016).

[11] *NoSQL Databases Explained*. URL: https://www.mongodb.com/nosql-explained (visitado 03-01-2017).

[12] George Papadakis, Georgia Koutrika y Themis Palpanas. *Meta-Blocking: Taking Entity Resolutionto the Next Level*. Abr. de 2012. URL: http://ieeexplore.ieee.org/document/6487505/ (visitado 18-05-2016).

[13] *Schema and Ontology Matching with COMA++*. URL: http://dbs.uni-leipzig.de/file/comaplusplus.pdf (visitado 17-05-2016).

[14] Stefano Spaccapietra y Christine Parent. *View integration: a step forward in solving structural conflicts*. 1994. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=277770 (visitado 27-10-2016).

[15] *The Humboldt-Merger*. URL: https://hpi.de/naumann/sites/hummer/.

[16] Ozgul Unal y Hamideh Afsarmanesh. *Using Linguistic Techniques for Schema Matching*. 2006. URL: http://home.iitk.ac.in/~veena/ICSOFT/Volume%202/Information%20Systems%20and%20Data%20Management/Short%20Papers/C4_265_Unal.pdf (visitado 17-05-2016).

[17] WizSoft. *WizSame® duplicate records discovery*. URL: http://www.wizsoft.com.

[18] WizSoft. *WizSame Matching Algorithm*. URL: http://www.wizsoft.com.

[19] WizSoft. *WizSame User's Guide*. URL: http://www.wizsoft.com.

[20] *Wordnet: Base de datos léxica en inglés*. URL: https://wordnet.princeton.edu/ (visitado 17-05-2016).

[21] Zhibiao Wu y Martha Palmer. *Verb Semantics And Lexical Selection*. Abr. de 2012. URL: http://delivery.acm.org/10.1145/990000/981751/p133-wu.pdf (visitado 18-05-2016).

Capítulo 6

Anexos

6.1. Uso de la herramienta Merge Toolbox

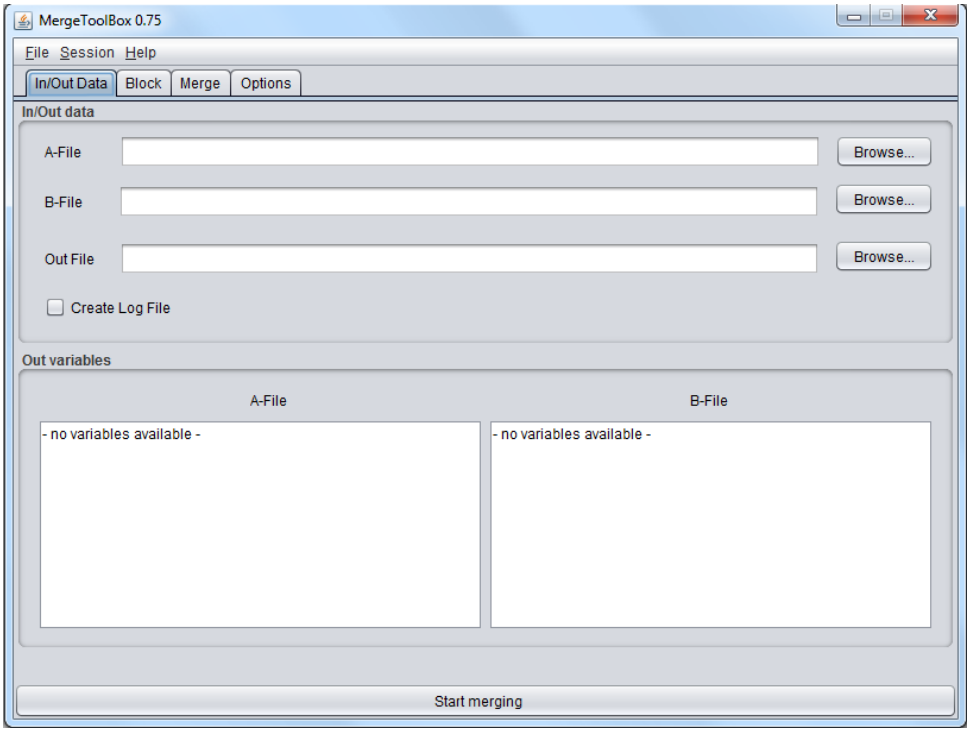

Figura 6.1: Captura de pantalla de la herramienta Merge Toolbox en la etapa de entrada

En primer lugar se deben ingresar dos archivos en formato Stata o texto plano como se observa en la figura 6.1 y 6.2 respectivamente. En caso de ser texto plano se debe definir el separador y la codificación. Para realizar deduplicación se debe establecer la opción en la pestaña Options y en ese caso solo se toma en cuenta el archivo A.

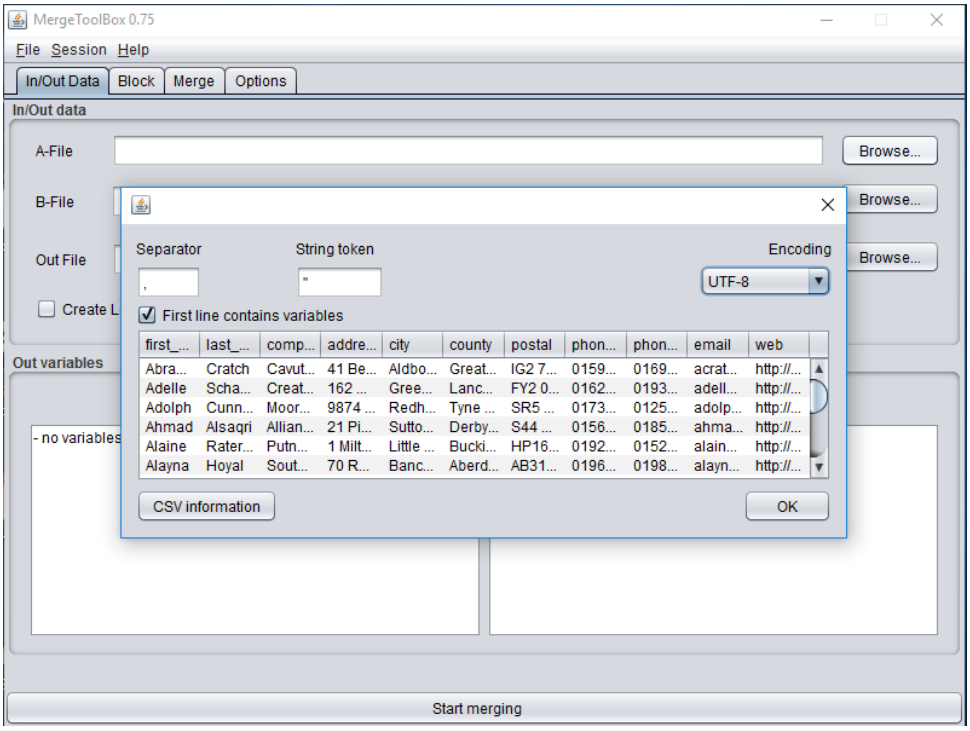

Figura 6.2: Captura de pantalla de la herramienta Merge Toolbox en la etapa de entrada de archivo de texto plano

También se debe especificar el archivo de salida que puede ser Stata o texto plano y elegir los atributos que lo compondrán como se visualiza en la figura 6.3.

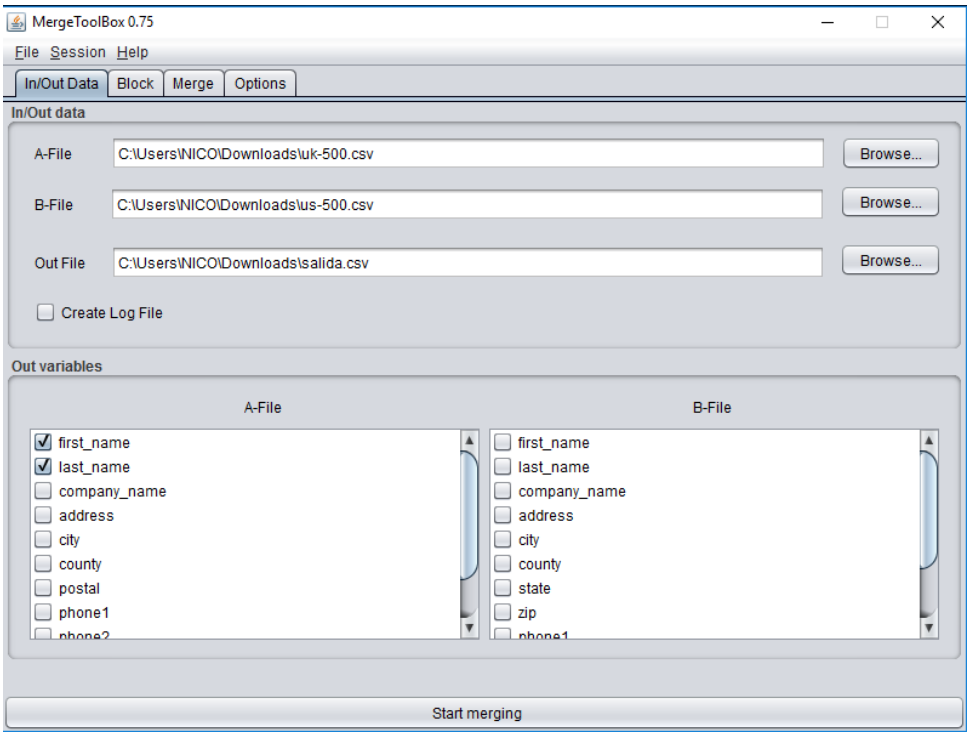

Figura 6.3: Captura de pantalla de la herramienta Merge Toolbox de especificación de archivo de salida

Se recomienda ingresar el archivo mas pequeño como A-File para optimizar la performance [1].

No se dispone de funcionalidades de preprocesamiento por lo que como siguiente paso conviene elegir el número de registros del archivo de entrada B que se consideran como match respecto a un registro del archivo de entrada A, los cuales serán ingresados al archivo de salida (Number of best matches, se observa en la figura 6.4). Asimismo puede determinarse un rango de registros a escribir de acuerdo a limites superior e inferior en el puntaje de match calculado mediante los campos Low cut out (límite inferior) y High cut out (superior).

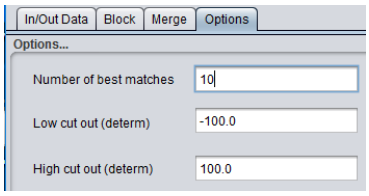

Figura 6.4: Captura de pantalla de la herramienta Merge Toolbox, configurando Number of best match en Options

Luego se debe escoger la estrategia de Indexación (blocking) para lo cual la herramienta posee dos modos, tradicional (Exact) y Canopy como se presentan en la figura 6.5 y 6.6 respectivamente.

El método tradicional consiste en seleccionar las variables que determinaran los bloques especificando el tipo de comparación exacta o numérica con +/- 1 de error.

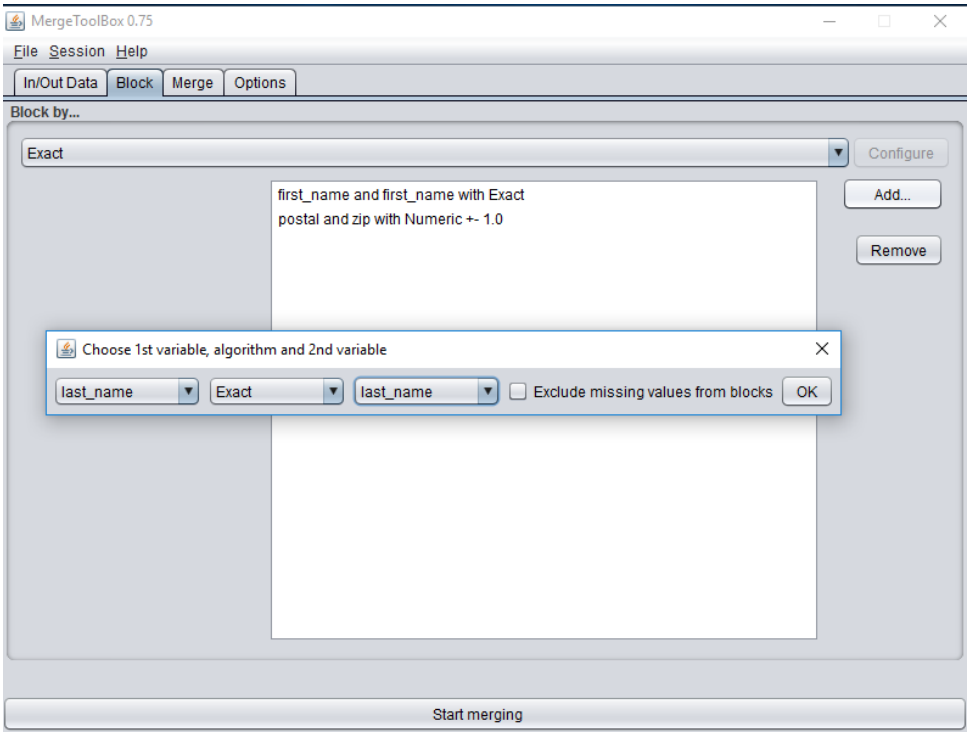

Figura 6.5: Captura de pantalla de la herramienta Merge Toolbox, en la etapa de Block, Exact

El método Canopy Clustering explicado en [4] está implementado en Merge Toolbox utilizando q-grams (n-grams para la aplicación) como función de similaridad y la función de Jaccard para formar las canoas con la posibilidad de configurar el n, utilización de padding y el umbral.

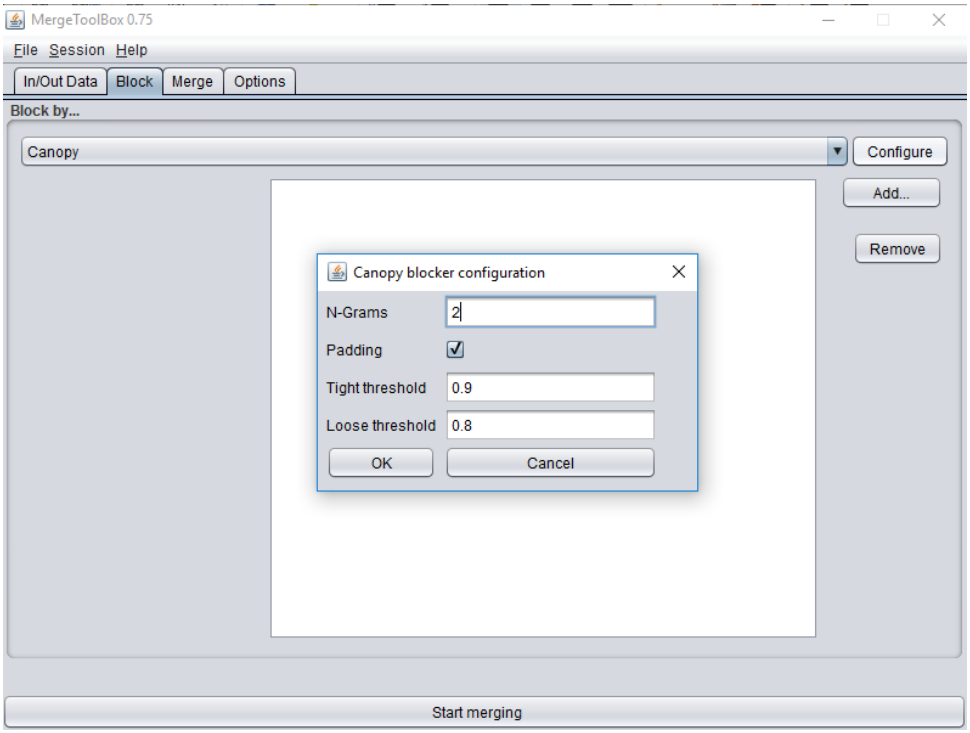

Figura 6.6: Captura de pantalla de la herramienta Merge Toolbox, en la etapa de Block, Canopy

Pasando al Matching propiamente dicho el Merge Toolbox ofrece dos posibilidades, matching basado en distancia y matching probabilístico.

Para utilizar matching basado en distancia (figura 6.7) se debe establecer dicha opción en la pestaña opciones y luego en la pestaña de Merge elegir los pares de atributos a comparar y el respectivo algoritmo. La herramienta provee 24 algoritmos distintos con sus respectivas configuraciones. Finalmente se hace click en Start Merge para realizar el procesamiento.

Para utilizar un abordaje de matching probabilístico se debe desmarcar la opción en la pestaña Options como se muestra en la figura 6.8. Luego en la pestaña Merge se seleccionan pares de atributos o de arreglos de atributos a comparar. Además del algoritmo de comparación también se pueden configurar para cada par los valores de m y u.

Otras de las funcionalidades que Merge Toolbox ofrece relacionadas al enfoque probabilístico se encuentran en la pestaña Options donde es posible configurar valores globales de m y u, cálculo de umbrales, parámetros del algoritmo EM para estimación, entre otros.

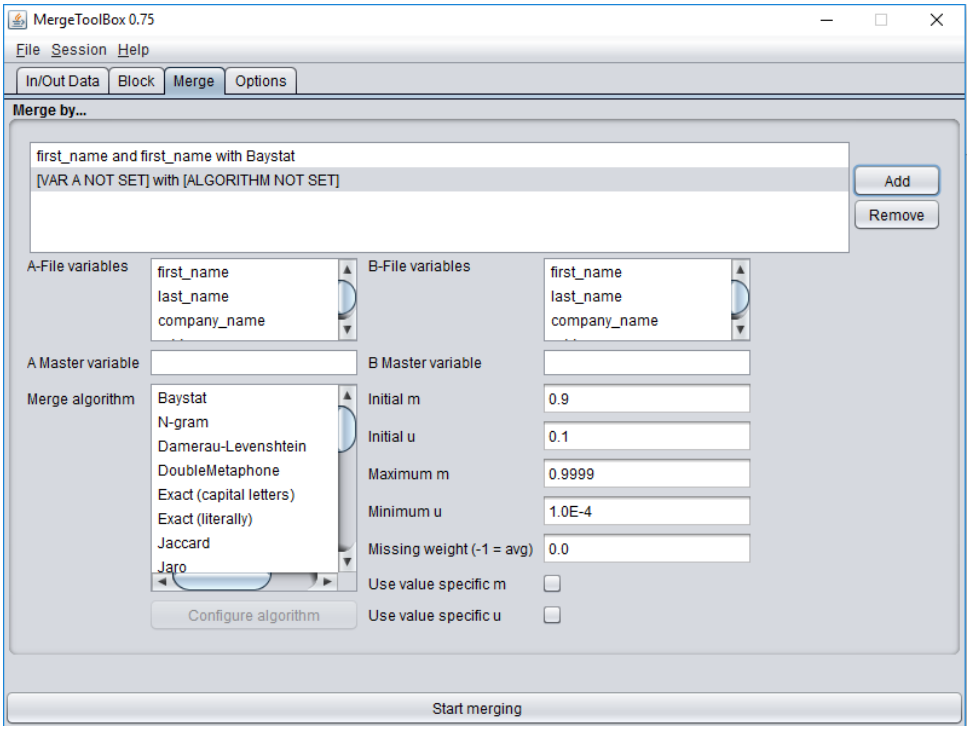

Figura 6.7: Captura de pantalla de la herramienta Merge Toolbox, en la etapa de Merge por distancia

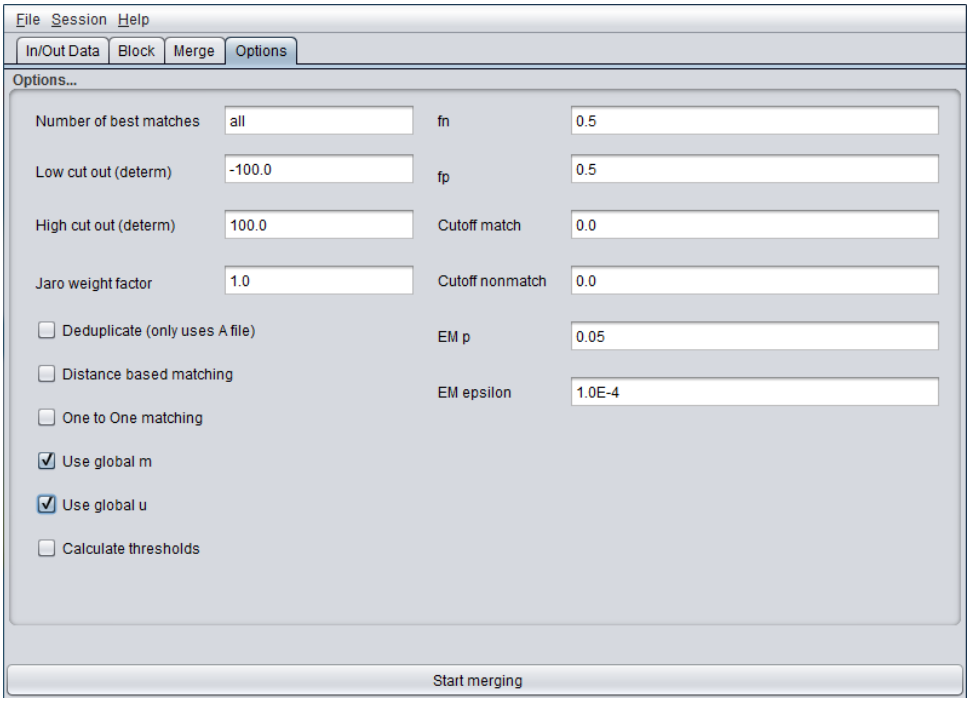

Figura 6.8: Captura de pantalla de la herramienta Merge Toolbox, en la pestaña de Options

La herramienta permite guardar las configuraciones de cada sesión y luego procesarlas en modo batch. Además posee algunas funcionalidades para reporte y análisis de resultados como por ejemplo, un histograma que refleja la cantidad de pares de registros por puntaje de match obtenido (figura 6.9).

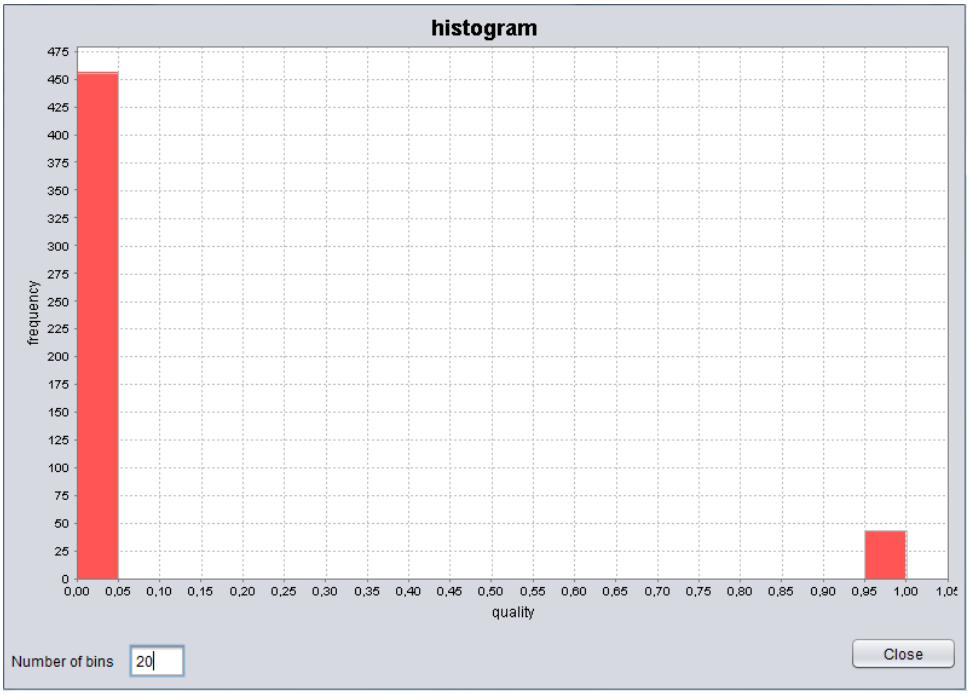

Figura 6.9: Captura de pantalla de la herramienta Merge Toolbox, visor de Histograma

6.2. Uso de la herramienta WizSame

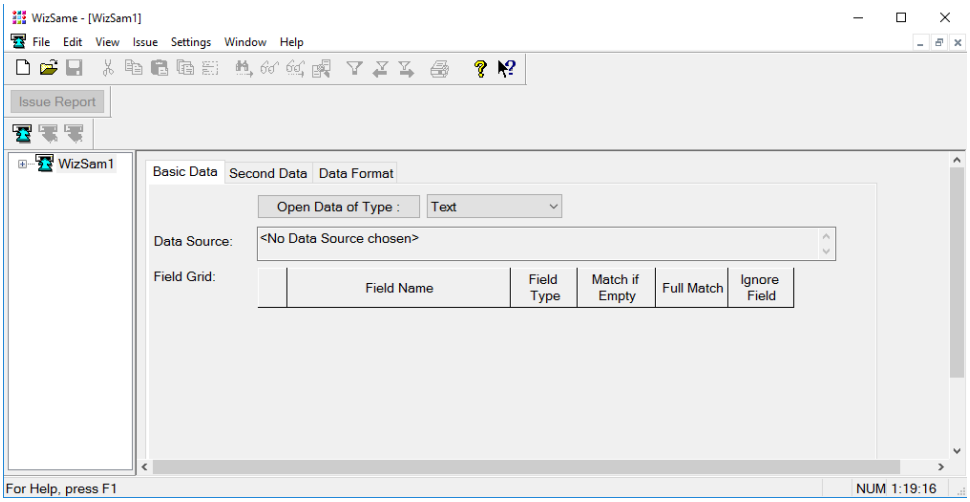

Figura 6.10: Captura de pantalla de la herramienta WizSame, pantalla inicial

En primer lugar se debe ingresar la fuente a procesar como se ve en la figura 6.10. WizSame admite como entrada tablas de bases de datos mediante OLE DB y ODBC, MS Access, archivos Excel, dBase y texto ASCII. Puede tomar los registros de una o dos tablas (Basic Data y Second Data) [19] (figura 6.11).

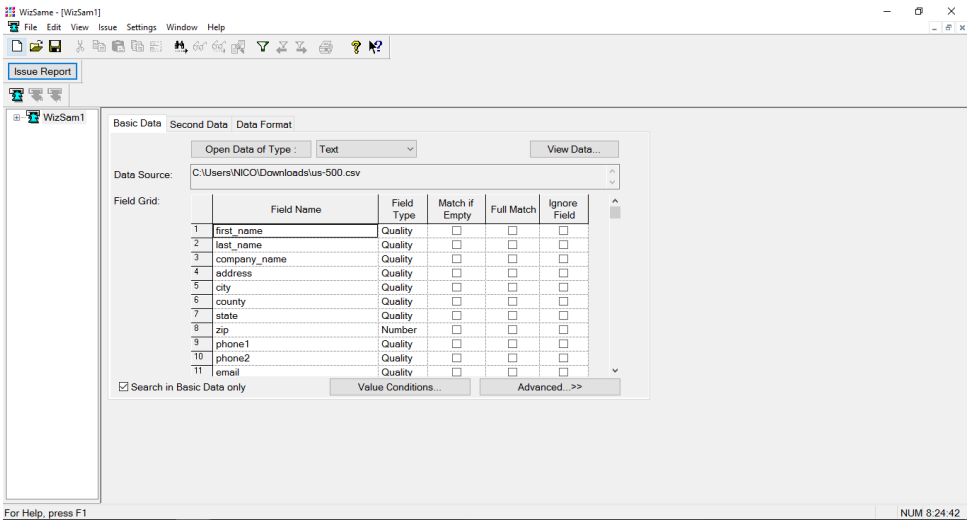

Figura 6.11: Captura de pantalla de la herramienta WizSame, archivo de entrada

También permite modificar el formato de los datos en la pestaña de Data Format como se presenta en la figura 6.12.

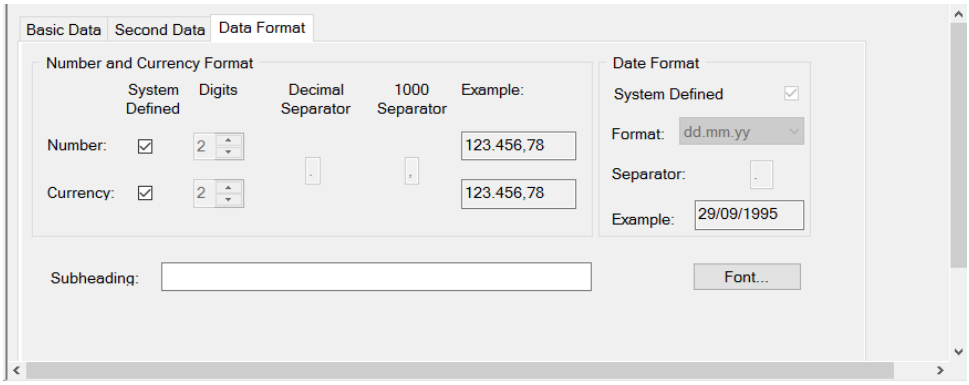

Figura 6.12: Captura de pantalla de la herramienta WizSame, formato de datos

Una vez cargada la tabla se obtienen las columnas y la herramienta clasifica automáticamente cada campo como Quality (carácteres alfa-numéricos), Number (números a los que se le pueden aplicar cálculos matemáticos) o Date (datos de fecha en alguna forma standard).

Para cada campo se puede seleccionar si ignorarlo (ignore field), si deben coincidir exactamente para considerarse match (full match) o si se considera match si uno de los registros tiene el campo vacío (match if empty).

Se pueden aplicar condiciones por determinados valores que tomen los campos para filtrar registros a través de reglas que se relacionan lógicamente y se configuran en base a un campo, un operador y un valor (figura 6.13).

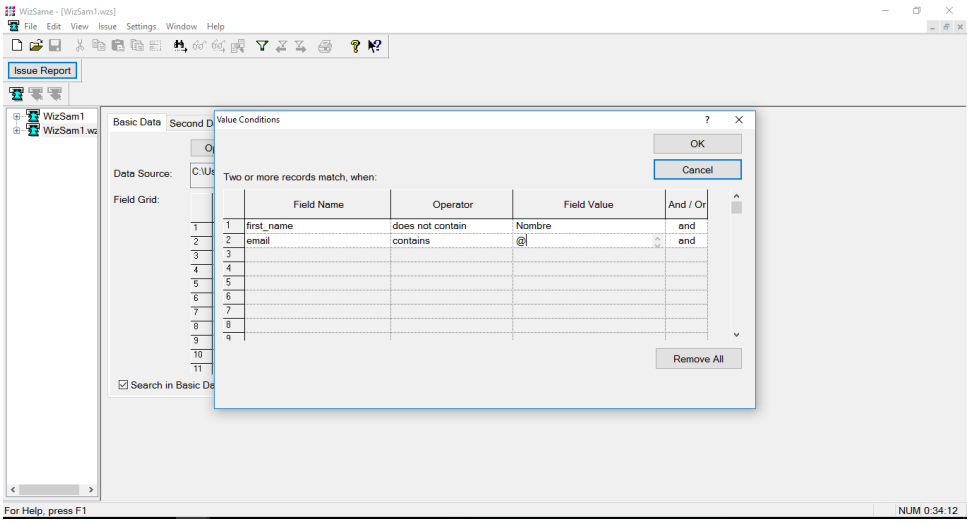

Figura 6.13: Captura de pantalla de la herramienta WizSame, filtrado por valores

En forma similar se define cuando un registro está duplicado a través de un conjunto de reglas relacionadas lógicamente para las cuáles debe determinarse el campo y el algoritmo a utilizar para comparar (figura 6.14). Dicho algoritmo puede ser igualdad o similaridad. WizSame determina dicha similaridad de acuerdo a un algoritmo propietario explicado en [18] y también utiliza un diccionario de sinónimos que puede ser actualizado por el usuario.

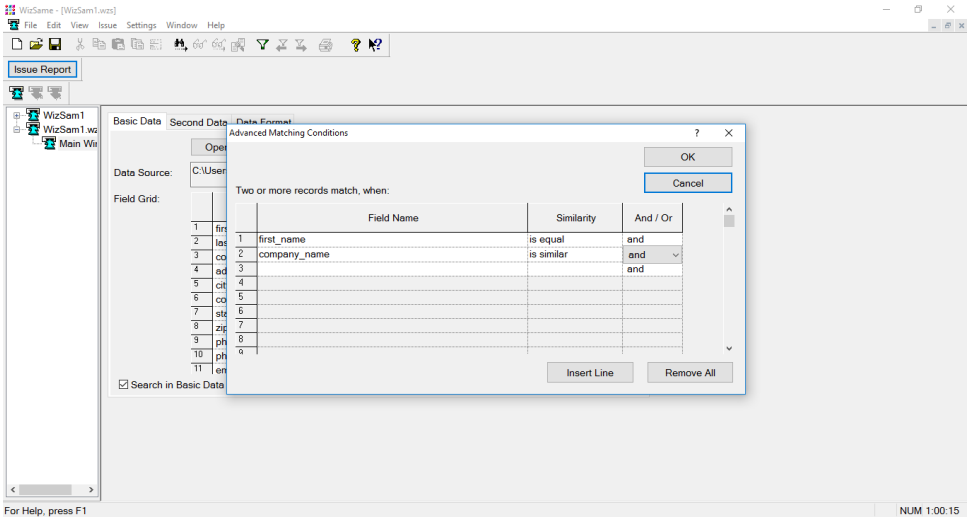

Figura 6.14: Captura de pantalla de la herramienta WizSame, reglas de matching

Luego en Issue, Issue Report el usuario ejecuta el matching y puede luego navegar entre los registros que fueron clasificados como duplicados donde se indica cuales campos son iguales (tick verde) y cuales similares (tick azul) como se observa en la figura 6.15.

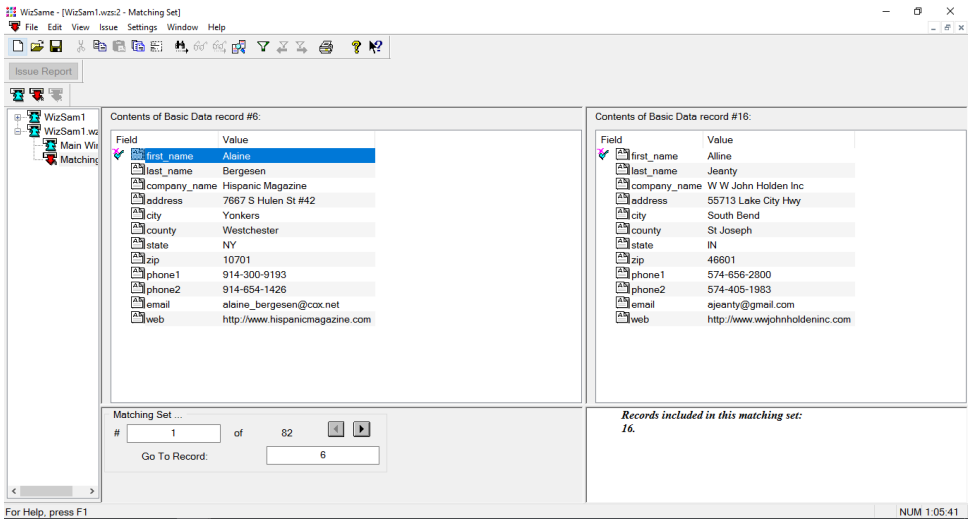

Figura 6.15: Captura de pantalla de la herramienta WizSame, vista de registros duplicados

Finalmente WizSame permite exportar una versión de la tabla con los duplicados eliminados en texto plano, Excel o Access (figura 6.16).

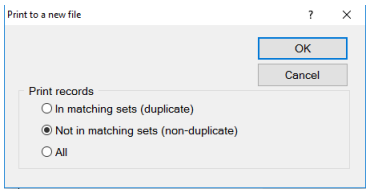

Figura 6.16: Captura de pantalla de la herramienta WizSame, exportar a archivo

6.3. Uso de la herramienta FEBRL

A continuación se detalla un ejemplo de uso de la herramienta.

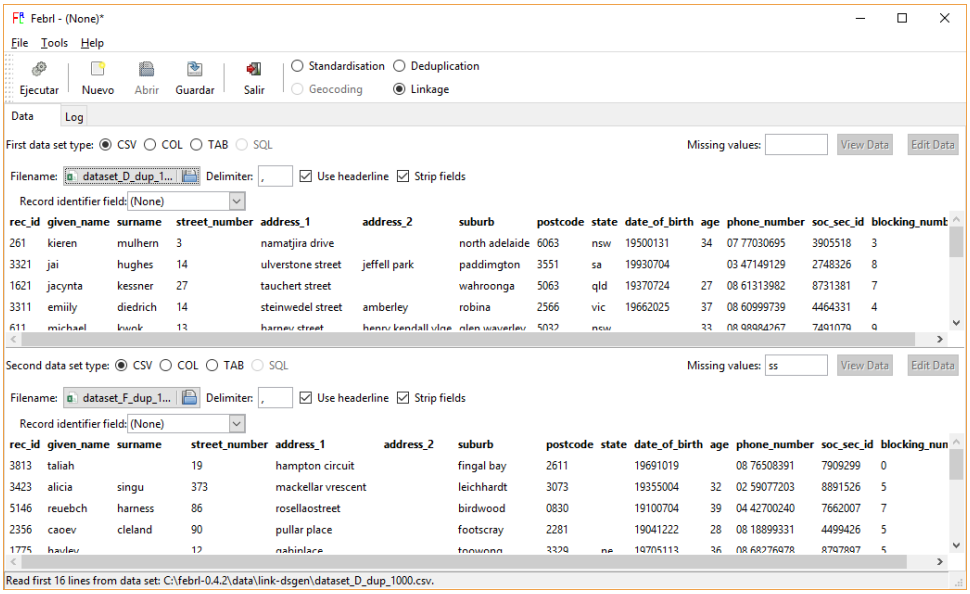

Figura 6.17: Carga de las fuentes

Lo primero es cargar los sets de datos a vincular como se observa en la captura 6.17. La entrada puede ser en formato CSV, COL o TAB. Aquí se puede elegir cosas como el separador de campos del CSV, si la primera línea indica las columnas, y también se puede elegir la columna que identifica a los registros. Una vez completado este paso, se debe presionar el botón "Ejecutar" para pasar a la siguiente fase. Siempre se hace esto para pasar de fase.

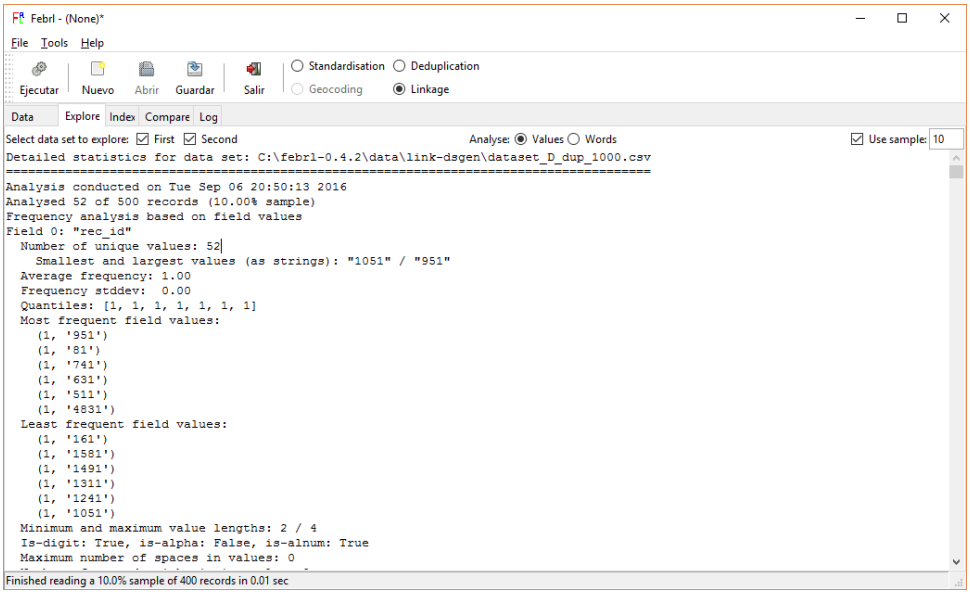

Figura 6.18: Datos estadísticos sobre los sets de datos

Al presionar "Ejecutar" se habilitan tres nuevas pestañas: "Explore", "Index" y "Compare". En la pestaña "Explore" se pueden obtener datos estadísticos sobre los sets de datos como los valores más y menos comunes de cada atributo, los valores más cortos y más largos de cada uno, si los campos contienen valores numéricos o alfanuméricos, etc (figura 6.18). Este paso es opcional.

Pasando a la pestaña "Index" es posible elegir el método de indexado de entre 7 distintos como se visualiza en la figura 6.19. Una vez elegido el método, se pueden agregar todos los indices que se desee, y estos tienen que ser configurados a mano. Febrl brinda 9 funciones de codificación de índices. Una vez configurado el indexado se presiona nuevamente el botón "Ejecutar".

En la pestaña "Compare" se eligen los campos a comparar para determinar que registros corresponden a una misma entidad. Por cada comparación se debe configurar los campos a comparar, la función de comparación (de entre 26 posibles), y las distintas configuraciones que requieren las distintas funciones de comparación (figura 6.20). Al presionar "Ejecutar", en este caso se habilita la siguiente pestaña: "Classify".

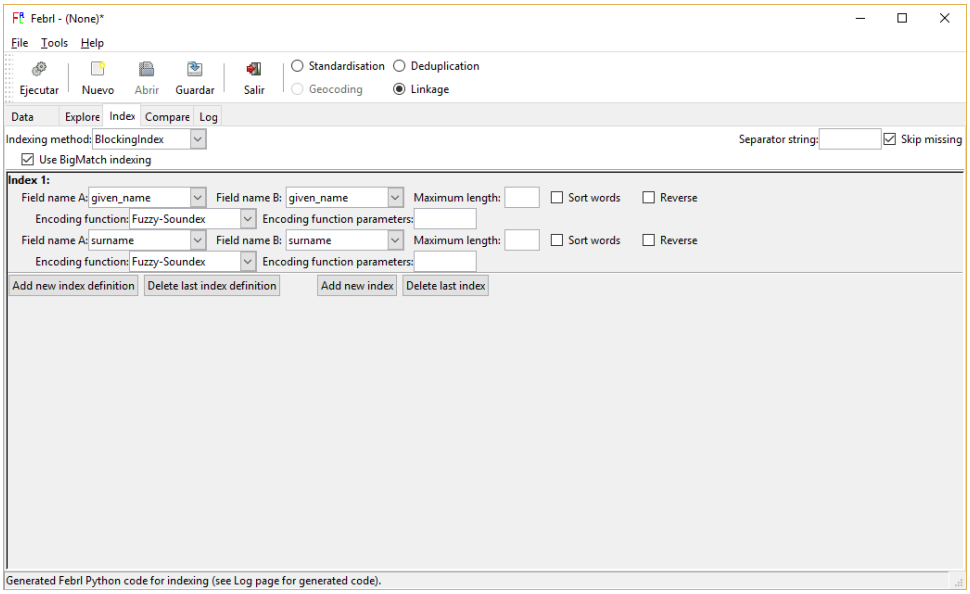

Figura 6.19: Configuración de indexado

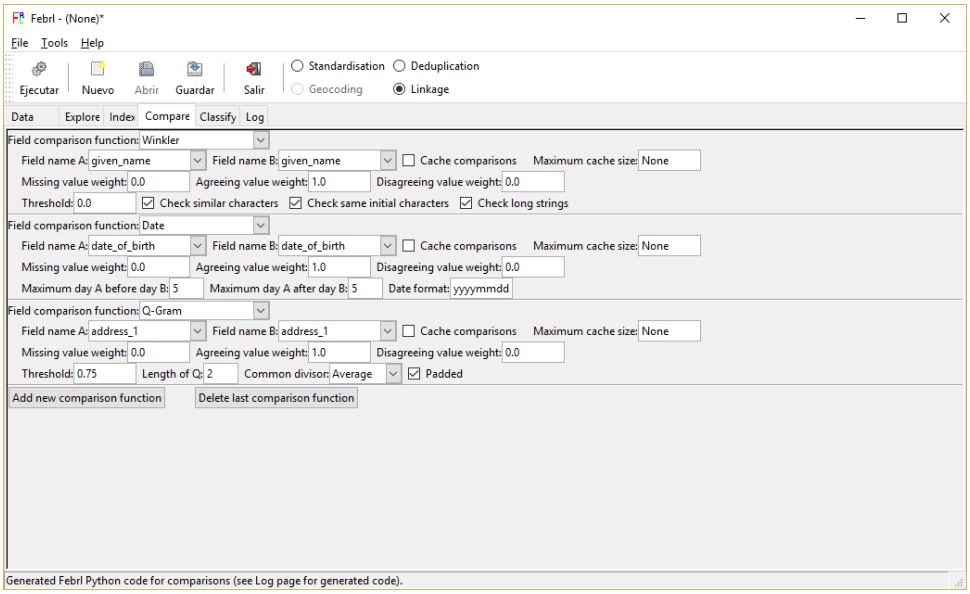

Figura 6.20: Configuración de funciones de comparación

En “Classify” se puede elegir el método para determinar cuando una comparación resulta en una correspondencia como se visualiza en la figura 6.21. El más simple es el de Fellegi y Sunter, que simplemente recibe 2 umbrales, uno inferior que separa no matches de posibles matches, y uno superior que separa posibles matches de matches. Febrl también brinda métodos supervisados como Optimal Threshold y Support Vector Machine y métodos no supervisados. La siguiente pestaña que se habilita es la que permite correr finalmente el matcher y permite guardar los resultados obtenidos.

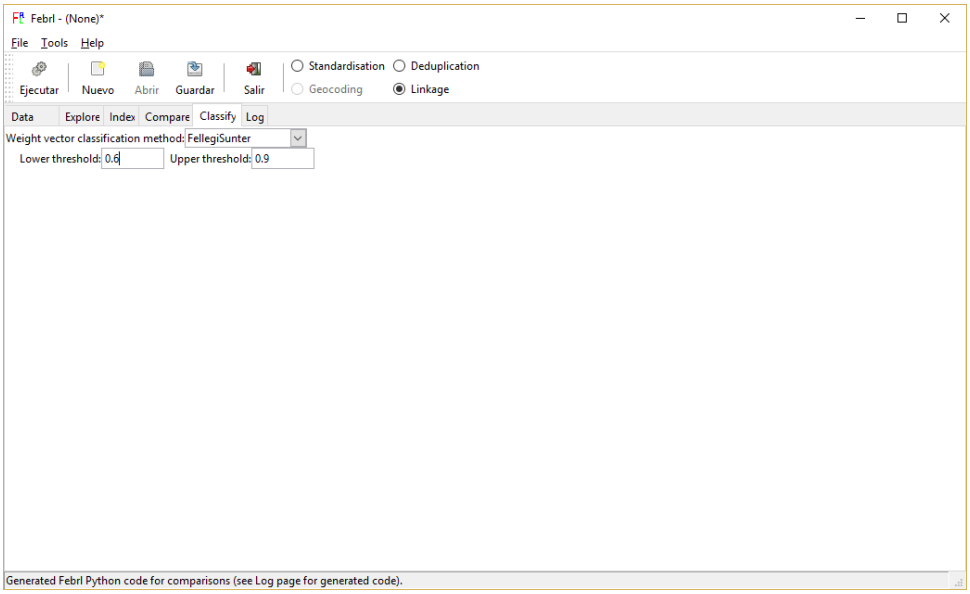

Figura 6.21: Configuración de métodos de clasificación

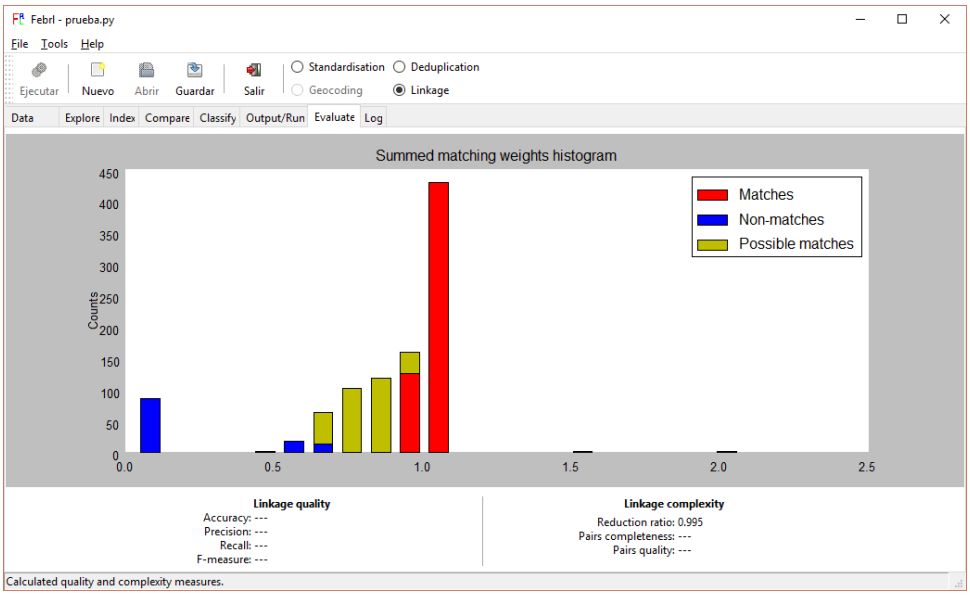

Figura 6.22: Histograma de comparaciones

El formato de resultados es bastante incómodo de procesar; genera IDs para cada match, y retorna dos CSVs (uno por cada set de entrada) con los datos una columna extra que indica a que matches pertenece. También genera una gráfica que muestra la distribución de la similaridad de los pares comparados (figura 6.22).