Universidad Central “Marta Abreu” de Las Villas
Facultad de Matemática, Física y Computación

Trabajo para optar por el Título Académico Máster en Ciencia de la Computación

Procedimiento para la mejora de la completitud en registros bibliográficos con formato MARC 21

Autor:

Lic. Juan Luis García Mendoza

Tutores:

MSc. Lisandra Díaz de la Paz

Dra. Luisa Manuela González González

Consultante:

Dr. Amed Abel Leiva Mederos

Marzo, 2017

Hago constar que el presente trabajo fue realizado en la Universidad Central “Marta Abreu” de Las Villas como parte de la culminación de los estudios de la Maestría en Ciencia de la Computación, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma total como parcial y que además no podrá ser presentado en eventos ni publicado sin previa autorización de la Universidad.

_____

Firma del autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del tutor

Firma del jefe del Seminario de Tecnologías de Programación y Sistemas de Información

Dedicatoria

A mi mamá:

Por ser la mejor madre del mundo, esa que lo dio todo por mí y que no pensó en bienestar propio más que el mío.

Por haber existido y darme tanto amor y cariño de manera incondicional.

Por tus consejos y valores que me ayudan a ser un mejor hombre.

Por haber sido mi confidente, amiga y hermana.

Por brindarme su sabiduría, su alegría y los momentos más felices de mi vida.

Por tu comprensión cuando más la necesitaba.

Por ser mi motivo de inspiración y guía en la vida.

A mi abuela:

Por ser más que mi abuela mi segunda madre, la que me ha criado y cuidado toda mi vida.

Por motivarme constantemente a ser alguien a través del estudio.

Por sus enseñanzas que sin duda han contribuido a mi formación como persona.

Por su atención constante y estar pendiente de mí en todo momento.

Por ser la mejor abuela del mundo.

Por haber tenido una hija tan maravillosa, mi mamá.

Por tu sacrificio al cuidarme, a ti te debo el hombre que soy.

Por ser mi motivo de inspiración y guía en la vida.

A mi abuelo:

Por su sacrificio al cuidarme como un hijo.

Por su apoyo y preocupación constante.

Aunque no está ya físicamente es un motivo de inspiración en mi vida.

A mi papá:

Por sus consejos y enseñanzas tan valiosos que me guían en la vida.

Por todo su apoyo como padre.

Por brindarme su sabiduría y conocimiento.

A mi familia:

Por todo su apoyo, han sido muy importantes para mí, porque sin ustedes no fuera lo que soy hoy.

A mi novia:

Por brindarme todo su apoyo, cariño y amor.

Por su ayuda constante e incondicional.

Por estar a mi lado en los buenos y malos momentos.

Agradecimientos

- *A mi mamá y abuela por ser mis motivos de inspiración y por todo el sacrificio que han realizado en pos de que me convierta en un buen hombre.*
- *A mis tutoras Lisandra y Luisa, por toda su ayuda y sobre todo su comprensión, además de sus constantes revisiones y sugerencias.*
- *A mi consultante Amed por su constante ayuda.*
- *A mi familia materna por su ayuda y apoyo constante en especial a mi abuelo Luis, a mis tías Isa, Tania, Barbarita y Milady, a mis tíos y mis primos.*
- *A mi papá y familia paterna por su ayuda incondicional y por todo su cariño.*
- *A mi novia y su familia por brindarme su cariño y apoyo incondicional.*
- *A mis amigos del pre y de la universidad por ayudarme y brindarme su apoyo cuando más lo necesité.*
- *A mis hermanos que aunque no vivan conmigo sé que están ahí.*
- *A todos mis compañeros de trabajo que me han apoyado y apoyan constantemente y contribuyen en mi superación profesional.*
- *A los trabajadores del Departamento de Ciencias de la Información que me brindaron sus consejos y ayuda incondicional.*
- *A todos aquellos que no he mencionado y de una forma u otra me dieron su ayuda.*

Resumen

La incompletitud de los metadatos constituye uno de los principales problemas que afecta la búsqueda y recuperación de información debido a la ausencia de elementos básicos. Hasta el momento, en las universidades cubanas no existe un equipo de especialistas para el control de la calidad de los metadatos, específicamente no se conoce el grado de completitud que presentan estos catálogos de registros bibliográficos con formato MARC 21, ni se ofrece una vía de solución automatizada para dicho problema. Por consiguiente, el objetivo principal del presente trabajo es elaborar un procedimiento para la mejora de la completitud de registros bibliográficos con formato MARC 21 mediante la modificación de un algoritmo de detección de duplicados y la identificación de las características que pueden variar en las métricas de completitud de metadatos. Para la medición de la completitud se identificaron como las características que varían para este formato la cantidad de campos, el grado de importancia de cada uno de estos y cómo determinar si está completo o no. Además, se modificó un algoritmo para la detección de registros duplicados para reducir el espacio de búsqueda. Como resultado, se elaboró un procedimiento que permite la mejora de los valores de completitud resultantes a partir de la integración de los elementos duplicados y se implementó una herramienta extensible que incorpora las métricas y el procedimiento propuesto. El presente trabajo, a pesar de basarse en una sola dimensión, incluye las fases de medición y mejora presentes en las metodologías de calidad de datos.

Palabras claves: procedimiento para la mejora, detección de duplicados, completitud de registros bibliográficos, formato MARC 21, integración

Abstract

The lack of completeness of the metadata constitutes one of the main problems that affects the search and information retrieval due to the absence of basic elements. So far, in the Cuban universities does not exist a team of specialists for the control of the metadata quality. More specifically, the degree of completeness of bibliographic record catalogs in MARC 21 format is unknown and there not exists an automated solution for this problem. Therefore, the main objective of this investigation is to elaborate a procedure for the improvement of the completeness of bibliographic records in MARC 21 format by mean of modification of an algorithm for duplicates detection and the identification of the characteristics that can variate in the metrics of completeness of metadata. To the completeness measurement were identified as the characteristics that change for this format the quantity of fields, the degree of importance of every one of these and how to determine if it is complete or not. Furthermore, an algorithm for the detection of duplicated records to reduce the space of search was modified. As a result, there was elaborated a procedure for the improvement of the values of completeness resulting from the integration of duplicated elements and an extensible tool that incorporates the metrics and the proposed procedure. The present work, in spite of being based on a single dimension, includes the phases of measurement and improvement that belong to the methodologies of data quality.

Keywords: procedure for the improvement, duplicate detection, bibliographic records completeness, MARC 21 format, integration

Tabla de Contenidos

Lista de Figuras

Lista de Tablas

Introducción

La mayoría de las actividades que se realizan en las organizaciones implican el uso de datos para la toma de decisiones. Si estos no son correctos pueden tomarse decisiones equivocadas con la consiguiente pérdida de recursos (Amón *et al.*, 2012). Es por ello que una buena calidad en los datos es crucial para el éxito y buen desempeño de una organización.

Las bibliotecas y centros de documentación no están ajenos a esta problemática. La calidad de datos, específicamente, la calidad de metadatos, es una característica que está directamente asociada con el valor y la efectividad de las bibliotecas digitales (Tani *et al.*, 2013). Una baja calidad de metadatos, puede afectar la operación e interoperabilidad de los repositorios digitales en los cuales están almacenados (Barton *et al.*, 2003; Ochoa y Duval, 2006a). Además, la recuperación de la información digital en las bibliotecas no será efectiva (Beall, 2006; Tani *et al.*, 2013), lo que dificulta el acceso a los objetos en línea y provoca la invisibilidad de uno o varios registros dentro de un repositorio (Barton *et al.*, 2003; Ochoa y Duval, 2008).

El sistema para la Automatización de Bibliotecas y Centros de Documentación (ABCD) está dirigido a la gestión integrada de los procesos de bibliotecas y de la operación automatizada en línea así como fuera de línea (Smet y Spinak, 2009). Este sistema se ha adoptado en diferentes universidades cubanas con el objetivo de digitalizar sus servicios bibliotecarios. Dentro de estas universidades se destacan cinco que forman parte del proyecto Red-TIC: Universidad de Pinar del Río, Universidad de Holguín, Universidad de Camagüey, Universidad de Ciencias Informáticas (UCI) y Universidad Central "Marta Abreu" de Las Villas (UCLV).

Uno de los módulos que contiene el sistema ABCD es el de catalogación de registros bibliográficos. Por orientaciones del Ministerio de Educación Superior (MES) el formato utilizado en las universidades cubanas para la catalogación de estos registros es MARC 21 (acrónimo del ingés *Machine Readable Cataloging*). Los registros bibliográficos con este formato contienen los elementos de datos esenciales que se necesitan para crear

descripciones bibliográficas de información de los ítems. Estos registros deben incluir campos que "proporcionen información suficiente para identificar un elemento bibliográfico y generar una descripción bibliográfica básica"[1].

En (Andreu-Alvarez, 2015) se realizó un análisis estadístico descriptivo del cuestionario aplicado a 16 especialistas bibliotecarios sobre la calidad de datos en el sistema ABCD utilizado en la UCLV. En su diseño se aplicaron las tres primeras fases correspondientes a la metodología TDQM (acrónimo del inglés *Total Data Quality Management*) (Wang, 1998). Uno de los resultados de este análisis fue que los metadatos pertenecientes a los registros bibliográficos del módulo de catalogación presentaban incompletitud. Además, se concluye que la incompletitud es uno de los principales problemas de calidad de datos presentes en el sistema. En trabajos anteriores se ha medido la completitud de registros bibliográficos con formato MARC 21 utilizando tecnologías *big data* (León-Hernández, 2015) y se ha propuesto una herramienta que realiza la medición y detecta registros duplicados (Guevara-Torres, 2016). Sin embargo, estos trabajos no tienen en cuenta los niveles de completitud del formato MARC 21 ni la mejoran.

Por otro lado, en (García-Mendoza, Díaz-de-la-Paz, González-González, Nuñez-Arcia y Leiva-Mederos, 2016) se mide la completitud de dos repositorios pertenecientes a las universidades de Cambridge[2] y Michigan[3] respectivamente, donde el nivel de completitud obtenido en ambas base de datos fue bajo.

Según Batini *et al.* (2009) y Batini y Scannapieco (2016) se pueden aplicar técnicas como la detección de duplicados e integración de datos y esquema para mejorar la completitud. La primera de estas técnicas detecta información redundante mientras que la segunda

[1] http://www.loc.gov/marc/bibliographic/litespa/elbdspa.html
[2] http://data.lib.cam.ac.uk/data/cambridge.mrc.gz
[3] http://www.lib.umich.edu/files/open-access-marc/umich_created_20140827.marc.gz

permite eliminar esa información integrándola en un solo registro. Esta integración contribuye a la mejora de la completitud de un registro.

Por lo anterior, se hace necesario la medición y mejora de la completitud en registros bibliográficos con formato MARC 21. Razones por las que se proponen los siguientes objetivos.

Objetivo general

Elaborar un procedimiento para la mejora de la completitud de registros bibliográficos con formato MARC 21 mediante la modificación de un algoritmo de detección de duplicados y la identificación de las características que pueden variar en las métricas de completitud de metadatos.

Objetivos específicos

1. Identificar las características de las métricas propuestas en la literatura para la medición de la completitud de metadatos que pueden variar en el contexto de los registros bibliográficos con formato MARC 21.
2. Modificar un algoritmo para la detección de elementos duplicados expuesto en la literatura de manera que se reduzca el espacio de búsqueda en los catálogos de registros bibliográficos con formato MARC 21.
3. Establecer reglas que permitan la integración de elementos duplicados en un catálogo con registros unificados cuya completitud resultante sea mayor o igual que la de dichos elementos por separado.
4. Implementar las métricas y los algoritmos para la integración de registros bibliográficos duplicados en una herramienta computacional extensible.

Las **preguntas de investigación** planteadas son:

1. ¿Qué características de las métricas propuestas en la literatura para la medición de la completitud de metadatos pueden variar al aplicarse en registros bibliográficos con formato MARC 21?

2. ¿Qué modificaciones se le deben realizar al algoritmo expuesto en la literatura para detectar elementos duplicados en los catálogos de registros bibliográficos de manera que se reduzca el espacio de búsqueda?
3. ¿Qué reglas se deben establecer en la confección de un procedimiento que permita la integración de un grupo de elementos duplicados en un registro unificado cuya completitud resultante sea mayor o igual que la de dichos elementos por separado?
4. ¿Qué patrones de diseño se pueden utilizar en la implementación de una herramienta computacional que incluya las métricas y algoritmos para la integración de manera que sea fácil su mantenimiento y sea resistente al cambio?

Justificación de la investigación

La investigación posee utilidad teórica porque se brinda un procedimiento para la mejora de la completitud en registros bibliográficos con formato MARC 21. Además, presenta un valor práctico porque sirve de apoyo y facilita el trabajo de los catalogadores mediante la medición de la completitud en registros bibliográficos con formato MARC 21 por los tres niveles de completitud y por cada clase de material. Además, brinda la posibilidad de obtener registros unificados más completos a partir de la integración de los registros que se detecten como duplicados, lo cual constituye una mejora de la calidad de los metadatos almacenados en los catálogos. Lo anterior cobra mayor relevancia al no existir, hasta el momento, en las universidades cubanas un equipo de especialistas para el control de la calidad de la catalogación y ser la incompletitud uno de los principales problemas que se presentan en las bibliotecas y centros de documentación del país. Conjuntamente, no se conoce de la existencia de una herramienta que realice la medición de esta dimensión y proponga mejoras para este formato.

Viabilidad

Para llevar a cabo la presente investigación se utiliza el lenguaje de programación Java, la base de datos llave/valor MapDB, la estructura de datos M-Tree y varias bibliotecas para el trabajo con el formato MARC 21 y la visualización de gráficos. Además, en el laboratorio de Tecnologías de Programación y Sistemas de Información perteneciente al

Centro de Investigaciones de la Informática (CII), ubicado en la UCLV se cuenta con los medios técnicos, software y personal necesario para desarrollar este trabajo.

El presente trabajo se encuentra estructurado de la siguiente manera:

En el Capítulo 1 se ofrece una perspectiva sobre la calidad de los metadatos. Se muestran las etapas del proceso de detección de duplicados y elementos del formato MARC 21. Además, se presentan las características principales del proceso de integración de datos.

En el Capítulo 2 se describe el proceso de medición de la completitud de registros bibliográficos con formato MARC 21. Se modifica un algoritmo expuesto en la literatura para la detección de registros bibliográficos duplicados. También, se propone un algoritmo para la integración de registros bibliográficos con este formato. Por último, se exponen los principales diagramas que se utilizan en el desarrollo de la herramienta propuesta.

En el Capítulo 3 se realizan pruebas unitarias en dos casos de uso de la herramienta propuesta. Además, se mide la completitud en dos bases de datos reales y se evalúan los algoritmos para la detección de registros duplicados e integración.

Este documento culmina con las conclusiones, recomendaciones, referencias bibliográficas y anexos.

Capítulo 1 Aspectos teóricos sobre la medición y mejora de la dimensión de calidad completitud en registros bibliográficos con formato MARC 21

En el presente capítulo se adopta una perspectiva sobre la calidad de los metadatos. Se expone la dimensión completitud presente en varios marcos de trabajo así como las métricas empleadas como parte de la fase de medición de la calidad de los metadatos. También, se presentan características del formato MARC 21 para datos bibliográficos. Además, se explican las etapas que componen el proceso de detección de duplicados y se mencionan varios algoritmos usualmente aplicados para la detección de duplicados en datos bibliográficos. Por último, se ofrece una panorámica general sobre el proceso de integración de datos.

1.1 Calidad de datos

El término calidad de datos es un concepto multidimensional (Ballou y Pazer, 1985; Wand y Wang, 1996; Wang y Strong, 1996). Según Bobrowski *et al.* (1999) es difícil brindar una "definición universal de lo que significa calidad". Aunque no existe una definición única del término, una de las más aceptadas en la literatura es "datos adecuados para el uso" de los consumidores de datos dada por Juran y Gryna Jr (1980) y retomada por Wang y Strong (1996). Son varios los autores que respaldan esta definición (Cappiello *et al.*, 2004; Chapman, 2005; Shankaranarayanan y Cai, 2006; Batini *et al.*, 2009; Sadiq *et al.*, 2011; Moges *et al.*, 2013; Yeganeh *et al.*, 2014; Díaz-de-la-Paz, García-Mendoza, Andreu-Alvarez, *et al.*, 2015; García-Mendoza *et al.*, 2015). Este concepto se refiere a que un dato posee calidad si este satisface las necesidades del cliente en un contexto específico de acuerdo a determinados criterios (Wang y Strong, 1996; Pipino *et al.*, 2002; Cappiello *et al.*, 2006; Lee *et al.*, 2006; Moges *et al.*, 2013). Además, esta definición resalta la importancia del punto de vista del consumidor porque es éste quien debe evaluar si los datos son adecuados o no para su uso (Wang y Strong, 1996). Por ejemplo, un dato puede presentar baja calidad para un uso determinado mientras que para otro puede tener una calidad adecuada.

En (Batini y Scannapieco, 2016) se plantea que la calidad de datos se refiere a las dimensiones, modelos y técnicas relacionadas con datos estructurados. Por otro lado,

cuando se trabaja con datos semiestructurados y no estructurados se utiliza el término de calidad de información. En este trabajo no es objetivo discutir sobre esta terminología por lo de ahora en adelante se utiliza el término de calidad de datos en general.

1.1.1 Metodologías para evaluar y mejorar la calidad de datos

Debido a la diversidad de técnicas existentes para la evaluación y mejora de la calidad de datos se han definido metodologías que ayudan a la selección, personalización y aplicación de las mismas. En (Batini *et al.*, 2009) se plantea que una metodología de calidad de datos es "un conjunto de directrices y técnicas que, a partir de una información de entrada descrita en un contexto dado, define un proceso racional para evaluar y mejorar la calidad de los datos".

De manera general, según Batini *et al.* (2009) y Batini y Scannapieco (2016) una metodología de calidad de datos debe estar compuesta de las siguientes fases:

- Reconstrucción de estado.
- Evaluación/medición.
- Mejora.

Fase de reconstrucción de estado

La fase de reconstrucción de estado está dirigida a coleccionar información contextual en los procesos organizativos y servicios, recogida de datos y métodos administrativos relacionados, problemas de calidad y costos correspondientes (Batini *et al.*, 2009). En caso de haberse realizado análisis previos y de estar disponible la información esta fase puede omitirse.

Fase de evaluación/medición

La fase de evaluación/medición se encarga de medir la calidad de las colecciones o bases de datos a través de dimensiones de calidad (Batini *et al.*, 2009) y evalúa los resultados para establecer un diagnóstico de calidad. En esta fase se debe realizar un análisis de requisitos de calidad de datos a partir de la opinión de los usuarios y administradores con el objetivo de identificar los problemas existentes. Además, se deben identificar las áreas críticas, modelar el proceso de generación y actualización de datos y analizar estos últimos

en cuanto a estructura y rendimiento (Batini *et al.*, 2009; Batini y Scannapieco, 2016). En el proceso de medición se deben seleccionar las dimensiones de calidad afectadas por problemas identificados en la fase de reconstrucción de estado y sus métricas correspondientes. Por otro lado, la evaluación es usada para establecer referencias, con el propósito de permitir un diagnóstico de calidad.

Fase de Mejora

Esta fase incluye la mejora en la gestión de la calidad a través de nuevas reglas, mejora en el monitoreo, rediseño de los procesos y la definición de puntos de control en los procesos de generación de datos (Batini *et al.*, 2009; Batini y Scannapieco, 2016). Además, se debe realizar una selección de pasos, estrategias y técnicas para mejorar la calidad de datos, los cuáles están agrupados en las categorías guiada a procesos y guiada a datos.

En la categoría guiada a procesos existen dos técnicas fundamentales:

- Control de procesos: Consiste en insertar puntos y procedimientos de control en la creación o modificación de datos.
- Rediseño de procesos: Consiste en rediseñar los procesos e introducir nuevas actividades con el objetivo de eliminar las causas que provocan una pobre calidad de datos y de esa manera aumentar la misma.

En el caso de la categoría guiada a datos son varias las técnicas:

- Adquisición de nuevos datos: Se sustituyen los datos que presentan problemas con nuevos datos de alta calidad adquiridos.
- Estandarización o normalización: Se reemplazan los valores que no cumplen con un estándar determinado con valores que lo cumplen.
- Detección de duplicados: Se identifican representaciones de datos que pueden ser la misma entidad en el mundo real.
- Integración de datos y esquema: Se define una vista unificada de los datos procedentes de varias fuentes de datos.
- Localización y corrección del error: Se localizan y corrigen errores de calidad de datos detectados.

Tabla 1: Metodologías para evaluar y mejorar la calidad de datos. Fuente: (Batini *et al.*, 2009; Batini y Scannapieco, 2016).

Acrónimo	Nombre	Referencia
TDQM	*Total Data Quality Management*	(Wang, 1998)
DWQ	*The Datawarehouse Quality Methodology*	(Jeusfeld *et al.*, 1998)
TIQM	*Total Information Quality Management*	(English, 1999)
AIMQ	*A methodology for information quality assessment*	(Lee *et al.*, 2002)
CIHI	*Canadian Institute for Health Information methodology*	(Long y Seko, 2005)
DQA	*Data Quality Assessment*	(Pipino *et al.*, 2002)
IQM	*Information Quality Measurement*	(Eppler y Muenzenmayer, 2002)
ISTAT	*ISTAT methodology*	(Falorsi *et al.*, 2003)
AMEQ	*Activity-based Measuring and Evaluating of product information Quality methodology*	(Su y Jin, 2004, 2006)
COLDQ	*Loshin Methodology (Cost-effect Of Low Data Quality)*	(Loshin, 2001)
DaQuinCIS	*Data Quality in Cooperative Information Systems*	(Scannapieco *et al.*, 2004)
QAFD	*Methodology for the Quality Assessment of Financial Data*	(De Amicis y Batini, 2004)
CDQ	*Comprehensive methodology for Data Quality management*	(Batini y Scannapieco, 2006)

En (Batini *et al.*, 2009) se muestra una comparación entre las metodologías presentadas en la Tabla 1. Esta comparación se realiza en base a las dimensiones de calidad de datos, fases, pasos, técnicas y estrategias que incluye cada una. Además, se analizan los tipos de datos hacia los que están dirigidas. En la presente investigación se siguen las fases expuestas anteriormente debido a que la mayoría de las metodologías de calidad de datos de una manera u otra las incluyen.

1.1.2 Dimensiones de calidad de datos

En (Wang y Strong, 1996) se define el término "dimensión de calidad de datos" como un conjunto de atributos que representan un solo aspecto de calidad. Según Tayi y Ballou (1998) los datos son descritos o analizados de una mejor manera a través de múltiples dimensiones, las cuales se agrupan en marcos de trabajo. Estas dimensiones varían de un marco de trabajo a otro y dependen del contexto en que se estén analizando (Moges *et al.*, 2013). Uno de los marcos más citados en la literatura es el propuesto en (Wang y Strong, 1996). Este marco de trabajo es jerárquico y genérico, por lo que puede aplicarse a varios contextos. Además, agrupa 15 dimensiones en cuatro categorías: intrínseca, contextual,

representacional y accesibilidad (ver Figura 1).

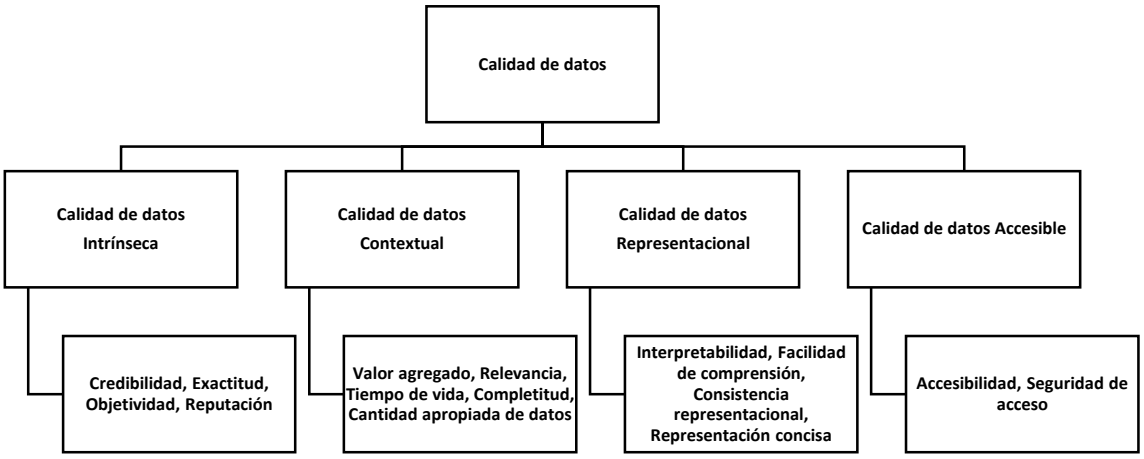


Figura 1: Esquema conceptual de calidad de datos. Fuente: (Wang y Strong, 1996).

1.2 Calidad de metadatos

El término calidad de metadatos, al igual que el de calidad de datos, es difícil de definir (Bruce y Hillmann, 2004). Hasta el momento no se ha alcanzado ningún consenso en su definición excepto por su carácter multidimensional (Tani *et al.*, 2013). Según Madnick *et al.* (2009), la calidad de metadatos puede definirse de manera similar a la calidad de datos, como "datos adecuados para el uso" de los consumidores de datos (Juran y Gryna Jr, 1980; Wang y Strong, 1996; Cappiello *et al.*, 2004; Chapman, 2005; Shankaranarayanan y Cai, 2006; Batini *et al.*, 2009; Sadiq *et al.*, 2011; Moges *et al.*, 2013; Yeganeh *et al.*, 2014; Díaz-de-la-Paz, García-Mendoza, Andreu-Alvarez, *et al.*, 2015; García-Mendoza *et al.*, 2015). Según Day *et al.* (2004) los metadatos presentan alta calidad cuando "respaldan los requerimientos funcionales del sistema para el que fueron diseñados". Al igual que la calidad de datos, la calidad de metadatos se mide de acuerdo a dimensiones de interés y estas varían de un marco de trabajo a otro.

1.2.1 Marcos de trabajo

En la literatura se han propuesto varios marcos de trabajo para la calidad de metadatos (Ochoa y Duval, 2006a), entre los que se destacan los propuestos en (Moen *et al.*, 1997), (Gasser y Stvilia, 2001) y (Bruce y Hillmann, 2004).

En (Moen *et al.*, 1997) se propone un marco de trabajo con 23 parámetros de calidad que se utilizan para evaluar la calidad de los registros de metadatos de 42 agencias de Estados Unidos que implementan el Servicio Localizador de Información del Gobierno (GILS por sus siglas en inglés).

Por otra parte, en (Gasser y Stvilia, 2001) se proponen 22 dimensiones de calidad agrupadas en tres categorías: intrínseca, relacional y reputacional. Estas dimensiones surgen del análisis de los parámetros definidos en (Moen *et al.*, 1997) y de 32 dimensiones de calidad presentes en la literatura relacionada con la calidad de información.

Además, en (Bruce y Hillmann, 2004) se concentran los parámetros planteados por Gasser y Stvilia (2001) en siete dimensiones generales o independientes del dominio con el objetivo de mejorar su aplicabilidad (ver Figura 2) (Shreeves *et al.*, 2005; Ochoa y Duval, 2006a, 2006c, 2008, 2009). Estas dimensiones son completitud, exactitud, conformidad a las expectativas, consistencia lógica, accesibilidad, tiempo de vida y procedencia. En este marco de trabajo se describe cada dimensión y se definen tres niveles de calidad de metadatos: estructura semántica o esquema, estructura sintáctica y valor del dato. Sin embargo, no brinda la definición formal, ni las métricas para cada una de estas dimensiones (Tani *et al.*, 2013).

En (Ochoa y Duval, 2006a, 2009) se proponen una o más métricas por cada dimensión del marco de trabajo propuesto en (Bruce y Hillmann, 2004) lo cual lo complementa. Además, en (Ochoa y Duval, 2009) se plantea que estas métricas no son un conjunto bien definido pero que deberían verse como "un primer paso hacia una evaluación automatizada de la calidad de metadatos". No obstante, según Tani *et al.* (2013) estas métricas tienen las siguientes características:

- Son fáciles de implementar y pueden servir para una gran variedad de repositorios digitales.
- Son estándares aunque los parámetros necesarios para la inicialización son dependientes del contexto.

- Están principalmente diseñadas para trabajar sobre metadatos en forma de texto y números. Para metadatos que contengan información alfanumérica se necesitan nuevas aproximaciones.
- Son principalmente concebidas para instancias de metadatos conformadas con una relativa estabilidad en cuanto al esquema.
- La normalización de la métrica no siempre es posible.
- La mezcla de los parámetros de calidad es la calidad general de la instancia de los metadatos, aunque no se hace ninguna propuesta en relación a la forma de mezclarlos.

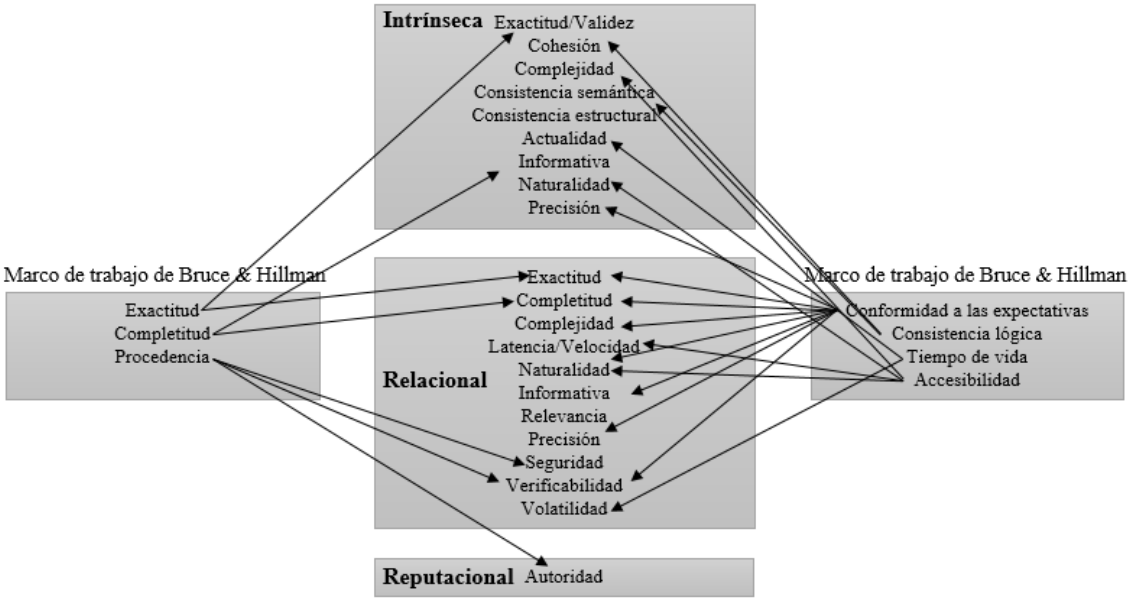


Figura 2: Relación entre las dimensiones de los marcos de trabajo propuestos en (Gasser y Stvilia, 2001) y (Bruce y Hillmann, 2004). Fuente: (Shreeves *et al.*, 2005).

1.2.2 Dimensión de calidad completitud

Dentro de las dimensiones de calidad propuestas en (Bruce y Hillmann, 2004), la completitud constituye una de las más utilizadas (Scannapieco y Catarci, 2002; Shreeves *et al.*, 2005; Park, 2005; Shankaranarayanan y Cai, 2006; Jackson *et al.*, 2008; Nichols *et al.*, 2008; Man *et al.*, 2010; Haug y Arlbjørn, 2011; Sivogolovko, 2011; Mendes *et al.*, 2012; Liaw *et al.*, 2013; Reynolds *et al.*, 2013) y en la cual se centran la mayoría de los autores (Scannapieco y Catarci, 2002). Además, la completitud puede afectar la búsqueda y la recuperación de información de registros bibliográficos en repositorios digitales o

catálogos en línea debido a la ausencia de elementos básicos (Bruce y Hillmann, 2004; Nichols *et al.*, 2008; Ochoa y Duval, 2008).

Esta dimensión se encuentra en la categoría contextual según el marco de trabajo propuesto en (Wang y Strong, 1996). Además, de manera general se asocia con la presencia de campos incompletos en un registro (Berti-Equille, 2007; Bellini y Nesi, 2013). Sin embargo, en (Moges *et al.*, 2013) se plantea que las dimensiones de calidad de datos dependen del contexto en que se estén analizando. Debido a esto, existen varias definiciones de la dimensión completitud de acuerdo al contexto en el cual es aplicada (Wand y Wang, 1996; Liu y Chi, 2002; Pipino *et al.*, 2002; Batini *et al.*, 2009; Sivogolovko, 2011).

En el contexto de las bases de datos relacionales, en (Batini *et al.*, 2009) se define como "el grado en que una colección de datos dada incluye datos que describen el conjunto correspondiente de objetos del mundo real". En esta área la completitud a menudo se relaciona con el significado de los valores nulos (Batini y Scannapieco, 2006, 2016; Batini *et al.*, 2009). En el caso de esta última aproximación mencionan tres tipos de valores nulos: el valor no existe, el valor existe pero es desconocido y el valor no se conoce si existe (Batini y Scannapieco, 2006, 2016).

En el contexto de los sistemas de información, en (Wand y Wang, 1996) se define la completitud como "la habilidad de un sistema de información para representar cada estado significativo de un sistema del mundo real".

Por otra parte, en el entorno de los almacenes de datos se define la completitud como "el porciento de información del mundo real introducida en fuentes de datos y/o almacenes de datos" (Jarke *et al.*, 1995).

Sin embargo, en el caso de los metadatos en (Ochoa y Duval, 2006a, 2006c, 2009) se define como "el grado en el cual un registro de metadatos almacena toda la información necesaria para tener una representación global del objeto descrito". Esta información necesaria varía en dependencia del dominio de aplicación y es importante medirla con el objetivo de determinar el nivel de completitud que presenta un determinado registro.

De manera general, las definiciones anteriores se refieren a la completitud como el grado de información almacenada con respecto al total. En el presente trabajo se utiliza la definición dada en el contexto de los metadatos debido a que se trabaja con un formato que constituye un modelo de metadatos.

1.2.3 Métricas para la dimensión completitud en metadatos

La forma de medir la completitud (C) a su vez puede variar en dependencia del contexto en que se aplique. En el área de las bases de datos relacionales en (Batini y Scannapieco, 2006) se define la métrica de la Ecuación 1 donde *r* es una relación, |*r*| es la cantidad de valores almacenados en la relación exceptuando los valores nulos y *ref(r)* es la cantidad real de valores en dicha relación, incluyendo los nulos.

$$C = \frac{|r|}{|ref(r)|} \qquad \text{Ecuación 1}$$

En el caso de los metadatos en (Ochoa y Duval, 2006a, 2006c, 2009) se proponen dos métricas para medir esta dimensión de calidad. La primera de ellas se muestra en la Ecuación 2, donde *N* representa el total de campos y $P(i)$ toma valor uno si el i-ésimo campo está completo y cero en otro caso.

$$C = \frac{\sum_{i=1}^{N} P(i)}{N} \qquad \text{Ecuación 2}$$

La segunda métrica propuesta en (Ochoa y Duval, 2006a, 2006c, 2009) introduce un factor de peso o grado de importancia que refleja la significación del i-ésimo campo. Esta métrica se muestra en la Ecuación 3, donde *N* representa el total de campos, α_i significa el factor de peso o grado de importancia del campo i-ésimo y $P(i)$ toma valor uno si el i-ésimo campo está completo o cero en otro caso. La Ecuación 2 es un caso particular de la Ecuación 3 cuando todos los campos tienen un grado de importancia igual a uno.

$$C = \frac{\sum_{i=1}^{N} \alpha_i * P(i)}{\sum_{i=1}^{N} \alpha_i} \qquad \text{Ecuación 3}$$

En ambas métricas cuando existe más de una instancia de un campo, se considera completo el campo si al menos una de sus instancias es completa. Estas métricas pueden tomar como máximo el valor uno (cuando todos los campos están completos) y como mínimo el valor cero (cuando ningún campo está completo) (Ochoa y Duval, 2006a, 2006c, 2009). Además, cuando se mide la completitud de un registro con estas métricas se calcula cuán completos están los datos almacenados con respecto al formato.

1.3 Formato MARC 21 para datos bibliográficos

El formato MARC 21 es un modelo de metadatos y constituye una norma utilizada para la representación e intercambio de datos bibliográficos, de autoridad, de existencias, de clasificación y de información de interés para la comunidad[1]. Este formato está formado por tres componentes principales[4]: cabecera, directorios y campos variables (ver Figura 3).

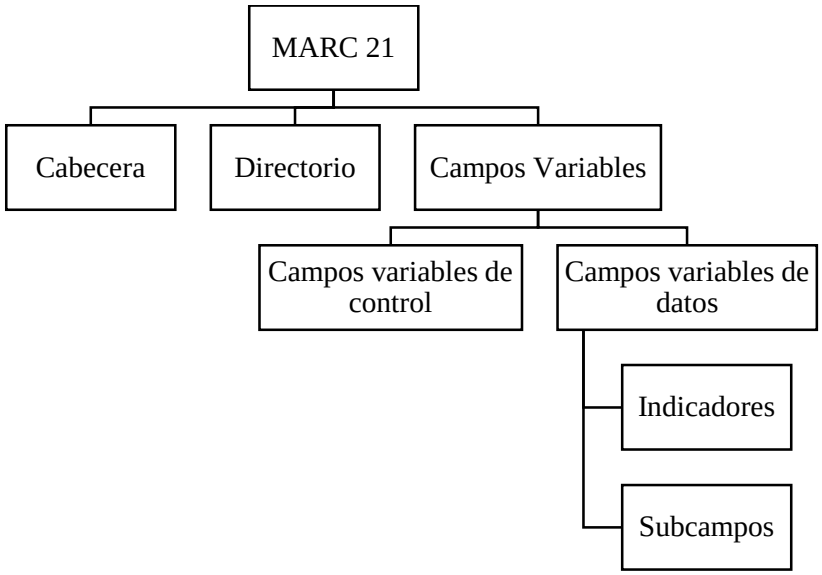


Figura 3: Estructura de un registro MARC 21. Fuente: (García-Mendoza, Díaz-de-la-Paz, González-González, Nuñez-Arcia y Leiva-Mederos, 2016).

[4] http://www.loc.gov/marc/bibliographic/bdintro.html

Las características principales de los elementos que conforman la estructura[4] del formato MARC 21 son:

- Cabecera: campo fijo que comprende las primeras 24 posiciones (00-23) de cada registro y suministra información para el procesamiento del mismo. Es el primer campo de un registro MARC 21.
- Directorio: aparece inmediatamente a continuación de la cabecera en la posición 24 y consiste en una serie de entradas de longitud fija (12 caracteres) que indican la etiqueta, la longitud y la posición del carácter inicial de cada campo variable.
    - 00-02 (Etiqueta): tres caracteres numéricos que identifican un campo asociado (ver Tabla 2).
    - 03-06 (Longitud de campo): cuatro caracteres numéricos que indican la longitud del campo, incluyendo los indicadores, los códigos de subcampos, los datos y el elemento que indica el final del campo. El número se justifica a la derecha y las posiciones no utilizadas toman valor cero.
    - 07-11 (Posición del carácter inicial): cinco caracteres numéricos que indican la posición del carácter inicial del campo en relación al inicio de los datos de la cabecera del registro (caracteres de la posición 12 a la 16). El número se justifica a la derecha y las posiciones no utilizadas adquieren el valor cero.
- Campos variables: son los encargados de organizar los datos de un registro con formato MARC 21. Cada campo es identificado por una etiqueta numérica de tres caracteres que se almacena en la entrada del directorio para el campo correspondiente (ver Figura 3). Existen campos que pueden tener varias instancias y se conocen como campos repetibles. Por el contrario, los que solo pueden tener una instancia son conocidos como campos no repetibles.
    - Campos variables de control: la etiqueta de estos campos tienen la forma 00X. Los campos variables de control son estructuralmente diferentes a los campos variables de datos debido a que no contienen indicadores ni

subcampos. Estos campos contienen un elemento individual de datos o una serie de elementos de longitud fija identificados por su posición relativa.

- Campos variables de datos: contienen dos posiciones para los dos indicadores al principio de cada campo, una para cada indicador. Las restantes posiciones del campo contienen varios subcampos. Estos subcampos tienen al inicio un código de dos caracteres (un delimitador y un carácter numérico o alfabético en minúscula) y a continuación los datos. Los subcampos pueden ser repetibles (varias instancias) y no repetibles (una instancia) con respecto a un campo determinado.

Tabla 2: Función de cada grupo de etiquetas[5].

Grupo de etiquetas	Función
0xx	Números de control bibliográfico e información codificada.
1xx	Entradas principales.
2xx	Información sobre títulos, edición. sello editorial, etc.
3xx	Descripción física, etc.
4xx	Menciones de series.
5xx	Notas.
6xx	Entradas de acceso por tema.
7xx	Entradas agregadas que no son de tema, ni de series ni campos de enlace.
8xx	Entradas agregadas de series y existencias
9xx	Campos para uso local.

En el caso del formato MARC 21 para datos bibliográficos existen ocho clases de materiales[5]: libro, archivo de computadora, partitura, mapa, publicación seriada, material mixto, material visual y grabación sonora. Estas clases de materiales se determinan a través de las posiciones seis y siete de la cabecera de un registro (ver Tabla 3).

La posición seis indica el tipo de material y los valores permitidos se aprecian en la columna “Código de tipo” de la Tabla 3.

[5] https://www.oclc.org/bibformats/es/introduction.html

Tabla 3: Clases de material del formato MARC 21 para datos bibliográficos[5].

CM	Código de tipo		Nivel bibliográfico
L	a	Material lingüístico	a, c, d, m
	t	Material lingüístico manuscrito	a, c, d, m
Ps	a	Material lingüístico	b, i, s
Mv	g	Medio proyectado	a, b, c, d, i, m, s
	k	Gráfico bidimensional no proyectable	a, b, c, d, i, m, s
	r	Artefactos y objetos naturales tridimensionales	a, b, c, d, i, m, s
	o	Kits	a, b, c, d, i, m, s
Mm	p	Material mixto	c, d
M	e	Material cartográfico	a, b, c, d, i, m, s
	f	Mapa manuscrito	a, c, d, m
P	c	Música impresa	a, b, c, d, i, m, s
	d	Música manuscrita	a, c, d, m
Gs	i	Grabación sonora no musical	a, b, c, d, i, m, s
	j	Grabación sonora musical	a, b, c, d, i, m, s
A	m	Archivo de computadora	a, b, c, d, i, m, s

CM: Clase de material **L:** Libro. **A:** Archivo de computadora. **P:** Partitura. **M:** Mapa. **Ps:** Publicación seriada. **Mm:** Material mixto. **Gs:** Grabación sonora. **Mv**: Material visual.

Por otro lado, la posición siete representa el nivel bibliográfico. Los valores que puede tomar se muestran en la Tabla 4. Si existe en la cabecera algún valor que no coincida con los mencionados anteriormente, se considera un error de catalogación y pasan a la clase de material "Desconocida" para luego ser tratados. Dicha clase es ficticia y solamente se emplea para no perder información.

Tabla 4: Niveles bibliográficos que pueden aparecer en la posición siete de la cabecera[5].

Código	Nivel bibliográfico
a	Componente, monografía.
b	Componente, publicación seriada.
c	Colección.
d	Subunidad.
i	Recurso integrado.
m	Monografía.
s	Publicación seriada.

1.3.1 Formato MARCXML

El formato MARCXML es una representación XML (*eXtensible Markup Language*) del formato MARC 21 clásico[6] (ver Figura 4). En este formato no es necesario almacenar la

[6] https://www.loc.gov/marc/marcxml.html

longitud del campo ni la posición de inicio de cada campo variable. El valor de la cabecera y los campos variables de control se almacenan como cadenas. Los campos variables de datos son elementos y la etiqueta e indicadores atributos de ese elemento. Los subcampos asociados a estos campos variables de datos son tratados como subelementos del mismo. Una de las ventajas de utilizar XML para representar registros en el formato MARC 21 es su fácil manipulación en la construcción de herramientas[6].

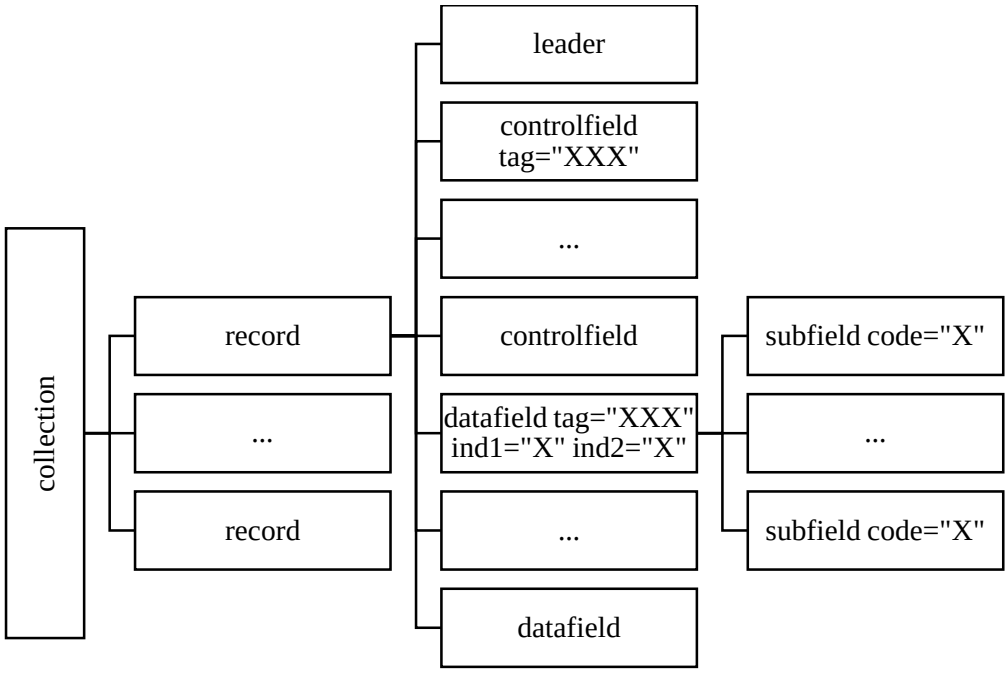


Figura 4: Estructura del formato MARCXML. Fuente: (Borel y Krause, 2009).

Además del formato MARXML, existen otras iniciativas como la descripción del formato MARC 21 en el formato JSON[7] (*JavaScript Object Notation*).

1.4 Mejora de la completitud

Una de las técnicas que se pueden aplicar en la fase de mejora es la detección de duplicados, la cual tiene varios usos. Uno de ellos es el descubrimiento de información. Por ejemplo, una vez que un registro ha sido utilizado por un usuario se le pueden proponer

[7] https://github.com/thisismattmiller/marc-json-schema

otros similares (Borel y Krause, 2009). Además, la detección de registros duplicados y su posterior mezcla puede contribuir en la mejora de la completitud del registro y por consiguiente de la calidad de datos.

1.4.1 Detección de duplicados

Uno de los problemas que pueden existir en los datos almacenados es la existencia de varios objetos que representan una misma entidad del mundo real (Amón *et al.*, 2012). La tarea de detectar registros duplicados en una misma o en diferentes bases de datos fue identificada inicialmente en (Newcombe *et al.*, 1959). Esta tarea se conoce de varias formas (ver Tabla 5).

Tabla 5: Formas para referirse a la detección de duplicados.

Nombre	Referencias
Detección de duplicados	(Monge y Elkan, 1997; Sarawagi y Bhamidipaty, 2002; Bilenko y Mooney, 2003; Elmagarmid *et al.*, 2007; Hussain *et al.*, 2013; Leitao *et al.*, 2013; Dong *et al.*, 2014; Heise *et al.*, 2014; Papenbrock *et al.*, 2015)
Depurar y mezclar	(Hernández y Stolfo, 1995)
Eliminación de datos duplicados	(Sarawagi y Bhamidipaty, 2002; Kejriwal y Miranker, 2013)
Correspondencia de objetos	(Thor y Rahm, 2007)
Identificación de instancias	(Wang y Madnick, 1989)
Identificación de objetos	(Batini y Scannapieco, 2016)
Estabilización de la base de datos	(Cohen *et al.*, 2000)
Reconciliación de referencias	(Dong *et al.*, 2005)
Vinculación de registros	(Fellegi y Sunter, 1969; Wang y Madnick, 1989; Winkler, 1999; Brizan y Tansel, 2006; Berti-Equille, 2007; Christen, 2008a; Gruenheid *et al.*, 2014; Steorts *et al.*, 2014)
Resolución de entidades	(Bhattacharya y Getoor, 2005; Brizan y Tansel, 2006; Kenig y Gal, 2009, 2013; Papadakis, Ioannou, Niederée y Fankhauser, 2011; Papadakis, Ioannou, Niederée, Palpana, *et al.*, 2011; Papadakis, Ioannou, Niederée, Palpanas, *et al.*, 2011; Shu *et al.*, 2011; Getoor y Machanavajjhala, 2012; Papadakis *et al.*, 2012, 2013, 2014; Whang y Marmaros, 2013; Fisher *et al.*, 2015; Papadakis y Palpanas, 2016)
Emparejamiento de entidades	(Köpcke y Rahm, 2010)
Emparejamiento de datos	(Christen, 2012b; Christen *et al.*, 2015; Papadakis y Palpanas, 2016)

Según Christen (2012b) cuando el proceso de búsqueda se realiza en una sola base de datos se conoce como detección de registros duplicados. En este caso cada registro necesita compararse con los demás para determinar si representan la misma entidad o no. El total de comparaciones potenciales entre pares de registros es $|A| \times (|A| - 1)/2$, donde

$|A|$ es el número de registros de la base de datos (Christen, 2008a). Por el contrario, cuando el proceso de búsqueda se ejecuta en dos o más bases de datos se conoce como vinculación de registros (Christen, 2012b). En este proceso todos los registros de una base de datos se deben comparar con los registros de la otra (Christen, 2008a). Para el caso de dos bases de datos *A* y *B,* cada registro de *A* debe ser comparado con todos los registros de *B*. El número de comparaciones de pares de registros es $|A| \times |B|$, donde $|A|$ y $|B|$ representan la cantidad de registros de *A* y *B* respectivamente (Christen, 2008a). En este trabajo se utiliza el término "detección de duplicados" para referirse a los dos casos.

El principal problema de rendimiento en la detección de duplicados es la costosa comparación detallada entre los valores de cada campo (o atributo) de los registros (Baxter *et al.*, 2003; Christen y Goiser, 2007; Christen, 2008a). La problemática anterior adquiere mayor relevancia en la medida que el volumen de los datos aumenta lo que hace casi imposible la comparación de todos los pares de registros (Christen, 2008a, 2012b). Además, si se asume que no hay registros duplicados en una misma base de datos (un registro de *A* solo puede enlazarse con uno de *B* y viceversa), el número máximo de duplicados es el $\min(|A|, |B|)$. De esta manera, al trabajar con grandes bases de datos la complejidad computacional aumenta cuadráticamente mientras que el número de duplicados crece de manera lineal (Bilenko *et al.*, 2006; Christen, 2008a, 2012b). Teniendo *d* bases de datos de *n* registros la complejidad computacional utilizando fuerza bruta es $O(n^d)$ (Steorts *et al.*, 2014). Lo anterior también se aplica a una sola base de datos, en este caso, el número máximo de duplicados es siempre menor que el total de registros de la misma.

En general el proceso de detección de duplicados consta de las cinco etapas siguientes: preprocesamiento de datos, indexado o bloqueo, comparación de pares de registros, clasificación y evaluación (Christen, 2012b; Batini y Scannapieco, 2016) (ver Figura 5).

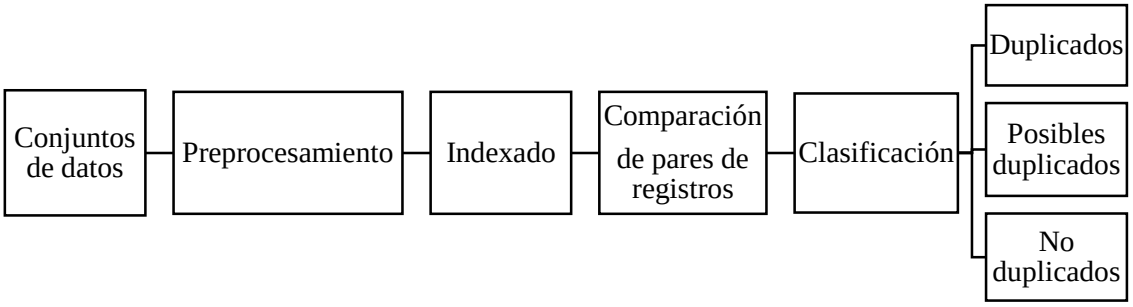


Figura 5: Procedimiento general de la detección de duplicados. Fuente: (Christen, 2012b; Batini y Scannapieco, 2016).

1.4.1.1 Etapa de preprocesamiento

Esta etapa es la primera en el proceso de detección de duplicados. En ella se limpian y estandarizan los datos de las fuentes, los cuales según Herzog *et al.* (2007) son pasos cruciales en este proceso y de manera general en la gestión de la información (Nuñez-Arcia *et al.*, 2016). El principal objetivo de esta etapa es garantizar que los campos o atributos que se comparen tengan la misma estructura y su contenido siga el mismo formato (Christen, 2012b).

Según Christen (2012b) esta etapa se divide en cuatro pasos:

- Eliminar caracteres y palabras no deseadas: este paso se corresponde con una limpieza inicial donde se eliminan caracteres tales como comas, puntos y comas y dos puntos. Además, en varias ocasiones se eliminan las palabras vacías o que no brindan información de relevancia para el proceso (*stopwords*), tales como artículos, conjunciones y preposiciones.
- Expandir abreviaturas y corregir faltas de ortografía: en este paso generalmente se tienen tablas auxiliares para buscar variaciones de nombres, apodos, faltas de ortografía comunes, las cuales se sustituyen por un valor correcto. Este paso es crucial para mejorar la calidad de los datos a comparar.
- Segmentos de atributos bien definidos y atributos de salida consistentes: este paso se encarga de los atributos que contienen varios segmentos de información como el campo dirección postal, en el cual aparecen calle, entrecalle, número, entre otros. Es recomendable separar estos segmentos en nuevos campos.

- Verificar que los valores de los atributos estén correctos: en este paso se comprueba con una fuente confiable si los valores de los atributos son correctos. Por ejemplo, una base de datos externa con todas las direcciones válidas de una región determinada.

Para dar solución a lo que se plantea en los pasos anteriores se pueden utilizar procesos de Extracción, Transformación y Carga (ETL), los cuales tienen entre sus funcionalidades la preparación y limpieza de los datos (Vassiliadis *et al.*, 2003; Jörg y Deßloch, 2009; Vassiliadis, 2009; Karagiannis *et al.*, 2013; Díaz-de-la-Paz, García-Mendoza, López-Porrero, *et al.*, 2015). En (García-Mendoza *et al.*, 2015) se propone una guía para la detección de errores utilizando procesos ETL.

1.4.1.2 Etapa de indexado

El principal objetivo de esta etapa es reducir tanto como sea posible el número de pares de registros que son comparados en detalle (Baxter *et al.*, 2003; Herzog *et al.*, 2007; Christen, 2012b; Steorts *et al.*, 2014) y de esta manera disminuir el costo computacional. Para reducir este número se utilizan generalmente técnicas de bloqueo o indexado (Elmagarmid *et al.*, 2007; Christen, 2012a, 2012b; Draisbach *et al.*, 2012; Papadakis *et al.*, 2013, 2016; Fisher *et al.*, 2015). Estas técnicas forman bloques o grupos de registros de manera que los registros que pertenecen a un mismo bloque tienen una alta probabilidad de ser similares (Kenig y Gal, 2009; Papadakis *et al.*, 2016).

Según Christen (2012a, 2012b) el proceso de indexado puede verse como un paso de filtrado o búsqueda y se basa mayoritariamente en una estructura de datos que une registros similares. En (Steorts *et al.*, 2014) se plantea que los métodos de detección de duplicados aplicados a bloques reducen las comparaciones hasta un $O(Bn_{max}^{d})$, siendo *d* la cantidad de bases de datos y n_{max} el tamaño máximo de los *B* bloques.

En la Figura 6 se muestran varias técnicas de indexado usualmente empleadas en la detección de duplicados (Baxter *et al.*, 2003; Bilenko *et al.*, 2006; Christen, 2012a, 2012b).

Una de las técnicas más conocidas es la del bloqueo tradicional. Este procedimiento se ha utilizado desde los años 60 del pasado siglo (Fellegi y Sunter, 1969). El objetivo de esta

técnica es la formación de bloques disjuntos (mutuamente excluyentes) de registros de acuerdo a cierto criterio de bloqueo (conocido como llave de bloqueo) (Baxter *et al.*, 2003; Elmagarmid *et al.*, 2007; Christen, 2012a; Karthigha y Anand, 2013; Papadakis *et al.*, 2013, 2016; Steorts *et al.*, 2014). El número de pares de registros que son generados mediante esta técnica depende de la frecuencia de distribución de la llave de bloqueo (Christen, 2012a). Si los bloques contienen un gran número de registros es posible que se generen pares de registros a comparar de manera innecesaria, lo cual es ineficiente (Baxter *et al.*, 2003). Por otro lado, si los bloques son muy pequeños se pueden perder pares de registros similares.

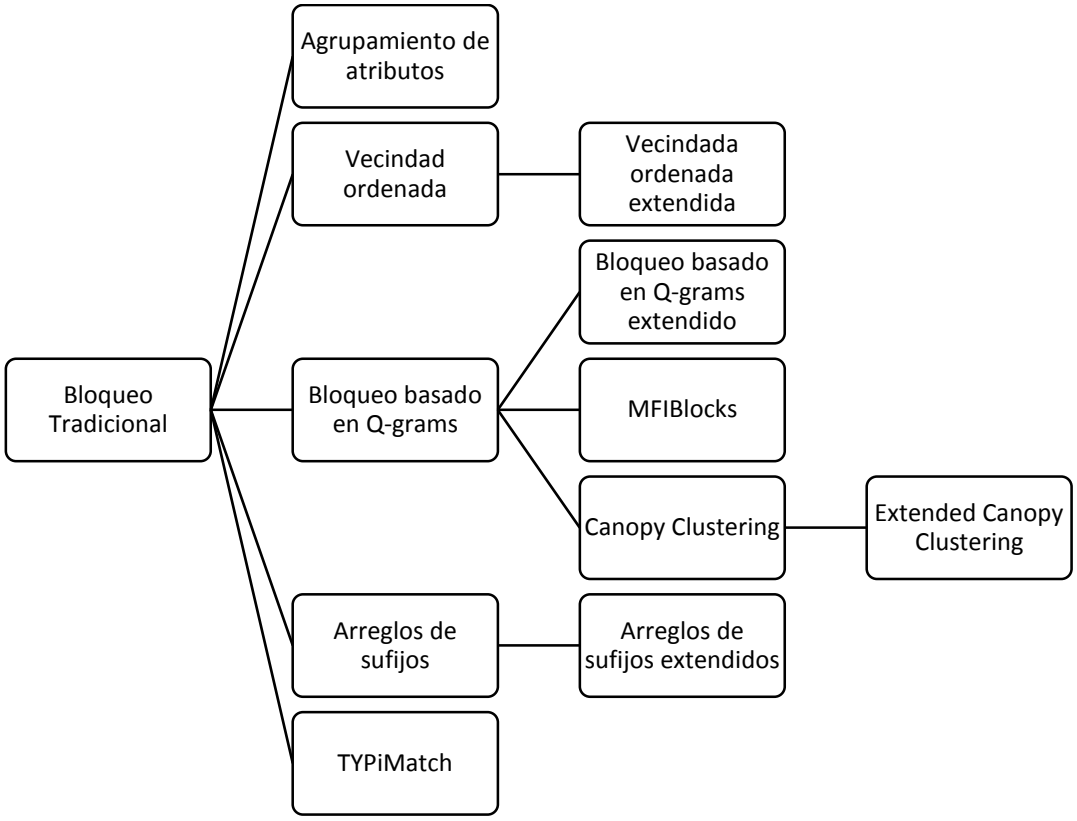


Figura 6: Técnicas de indexado. Fuente: (Papadakis *et al.*, 2016).

La llave de bloqueo se forma con los atributos de los registros. Con el objetivo de lograr una buena exactitud se deben escoger atributos que no sean tan propensos a errores (Baxter *et al.*, 2003). Además, se pueden utilizar múltiples llaves de bloqueo, para lo cual se deben realizar varias iteraciones utilizando en cada una de ellas una llave de bloqueo. Por ejemplo, si se tienen dos llaves de bloqueo (el año y las iniciales del autor), en una primera

iteración se realiza el bloqueo por año. Luego, en una segunda iteración se realiza el bloqueo, dentro de cada bloque creado en la iteración anterior, por las iniciales del autor.

1.4.1.3 Etapa de comparación de pares de registros

En esta etapa se ejecuta una comparación detallada de todos los pares de registros que pertenecen a un mismo bloque. Esta comparación se efectúa entre uno o varios atributos de cada registro. Según Christen (2012b) es conveniente incluir atributos que no pertenecen al criterio de bloqueo y que se encuentren disponibles.

Los valores de similitud generalmente se normalizan, donde el valor uno indica una correspondencia completa y el cero lo contrario (Christen, 2008b, 2012b). Los atributos a comparar pueden ser de varios tipos por lo que es necesario escoger una función de similitud de acuerdo al tipo de dato. Para atributos como nombres, que son de tipo cadena, existen varias funciones como las basadas en caracteres, en *tokens* (una palabra) o en la fonética (Bilenko *et al.*, 2003; Cohen *et al.*, 2003b; Christen, 2006, 2012b; Elmagarmid *et al.*, 2007; Herzog *et al.*, 2007; Bharambe *et al.*, 2012). En el caso de atributos de tipo numérico o fecha en (Christen, 2012b) se proponen funciones de similitud. La relación que existe entre la similitud y la distancia entre dos elementos es la siguiente: *similitud = 1 - distancia*.

El resultado de la comparación del conjunto de atributos de cada par de registros da como resultado un vector de similitudes conocido como vector de comparación (Christen, 2012b).

Métricas basadas en caracteres

Las métricas de similitud basadas en caracteres están diseñadas para maniobrar errores tipográficos (Elmagarmid *et al.*, 2007). Estas métricas consideran cada cadena como una secuencia ininterrumpida de caracteres (Amón y Jiménez, 2010). Dentro de este conjunto destaca la distancia de edición denominada Levenshtein (Levenshtein, 1966). Dicha distancia calcula el número mínimo de operaciones de edición (inserción, eliminación o reemplazo de un carácter) que se necesitan para transformar una cadena s_1 en otra s_2 (Levenshtein, 1966). En el modelo original (Levenshtein, 1966) todas las operaciones de edición tienen costo unitario aunque autores como (Hall y Dowling, 1980; Zhu y Ungar,

2000; Bilenko y Mooney, 2003; Cohen *et al.*, 2003a; Yancey, 2004) utilizan diferentes costos paras las distintas operaciones. En (Damerau, 1964) se añade la trasposición de dos caracteres adyacentes como otra operación permitida. Esta distancia se conoce como Damerau-Levenshtein. Otras distancias basadas en caracteres son la distancia de brecha afín (Waterman *et al.*, 1976), la distancia de Smith-Waterman (extensión de las distancias de edición y brecha afín) (Smith y Waterman, 1981), la distancia de Jaro (Jaro, 1978), la distancia de Jaro-Winkler (Winkler, 1990), la distancia basada en *q-grams* (Ullmann, 1977) y la distancia de edición utilizando el teclado (EDUK) (López-Porrero, 2011).

Métricas basadas en tokens

Las métricas basadas en caracteres no funcionan bien cuando el orden de las palabras cambia. Con el objetivo de solucionar este problema surgen las métricas basadas en *tokens* (Elmagarmid *et al.*, 2007).

Una de las métricas basadas en *tokens* es la función de similitud de Monge-Elkan (Monge y Elkan, 1996). Esta métrica divide cada cadena en *tokens*. Para cada *token* de una primera cadena existe otro perteneciente a la segunda de máxima similitud. Entonces la similitud de Monge-Elkan entre dos cadenas es la similitud máxima promedio entre una pareja de *tokens*. Otras métricas basadas en *tokens* es la similitud coseno TF-IDF (Cohen *et al.*, 2003a) y la generalización de la distancia EDUK (López-Porrero, 2011).

Métricas basadas en la fonética

Estas funciones se enfocan en la similitud fonética entre las cadenas, es decir, similitud en cuanto a su pronunciación. En (Elmagarmid *et al.*, 2007) se expone el siguiente ejemplo, la palabra *Kageonne* es fonéticamente similar (en inglés) a *cajun* a pesar de que sus representaciones como cadena son muy diferentes. Algunas de estas funciones son *Soundex* (Russell, 1918, 1922), *New York State Identification and Intelligence System* (NYSIIS) (Taft, 1970), *Metaphone* (Philips, 1990) y *Double Metaphone* (Philips, 2000).

1.4.1.4 Etapa de clasificación de pares de registros

En esta etapa los pares de registros comparados en la fase anterior son clasificados en base al vector de comparación o la suma de las similitudes (Christen, 2012b). Una de las formas de clasificación existente es la basada en reglas. Estas reglas permiten clasificar los pares de registros en cada clase (Hernández y Stolfo, 1998; Cohen, 2000; Naumann y Herschel, 2010; Bharambe *et al.*, 2012) y se aplican al vector de comparación. Las reglas están compuestas por los valores de similitud combinados con conjunciones, disyunciones y negaciones. La forma general de cada regla es $P \rightarrow C$, donde P es un predicado que se aplica sobre el vector de comparación del par (*ri*, *rj*) y *C* es la clase a la cual pertenece. El predicado P es una expresión booleana y tiene la siguiente forma $P = (term_{1,1} \vee term_{1,2} \vee \ldots) \wedge \ldots \wedge (term_{n,1} \vee term_{n,2} \vee \ldots)$.

Existen otras formas de clasificación tales como las basadas en umbrales, probabilística, supervisada y colectiva (Christen, 2012b).

1.4.1.5 Etapa de evaluación

En esta etapa se debe evaluar la calidad de la clasificación (Christen, 2012b; Batini y Scannapieco, 2016). Para evaluar esta calidad se le debe asignar a cada par de registros comparado y clasificado una de estas cuatro categorías (Christen y Goiser, 2007):

- Verdaderos positivos (VP): El par de registros ha sido clasificado como duplicado y realmente lo es.
- Falsos positivos (FP): El par de registros ha sido clasificado como duplicado y realmente no lo es.
- Verdaderos negativos (VN): El par de registros no ha sido clasificado como duplicado y realmente no lo es.
- Falsos negativos (FN): El par de registros no ha sido clasificado como duplicado y realmente lo es.

Basado en las categorías anteriores existen varias medidas de calidad que pueden calcularse (Christen y Goiser, 2007). Es necesario señalar que la cantidad de verdaderos negativos en la detección de duplicados es un número muy elevado con respecto a las

demás categorías. Esto se debe a que el total de pares que realmente no son duplicados es mayor que los que lo son (Christen y Goiser, 2007; Christen, 2012b), lo cual provoca que las medidas que involucren esta categoría no deban ser utilizadas para medir la calidad de la clasificación (Christen y Goiser, 2007). Las principales medidas que suelen utilizarse en los problemas de detección de duplicados son:

- Precisión: Esta medida se utiliza frecuentemente en la recuperación de información (Christen, 2012b) para evaluar la calidad de la búsqueda (ver Ecuación 4). La precisión indica la proporción de cuántos de los registros clasificados como duplicados (*VP*+*FP*) han sido correctamente clasificados (*VP*).

$$Precisión = \frac{VP}{VP + FP} \qquad \text{Ecuación 4}$$

- Sensibilidad o *recall*: Esta medida es la segunda más utilizada en la recuperación de información (Christen, 2012b) (ver Ecuación 5). La sensibilidad indica la proporción de cuántos de los registros que realmente son duplicados (*TP* + *FN*) han sido clasificados correctamente (*VP*).

$$Sensibilidad = \frac{VP}{VP + FN} \qquad \text{Ecuación 5}$$

- *F-measure*: Esta medida calcula la media armónica entre precisión y sensibilidad. Si su valor es alto es porque la precisión y la sensibilidad tienen valores altos o viceversa (ver Ecuación 6).

$$F - measure = 2 * \frac{Precisión * Sensibilidad}{Precisión + Sensibilidad} \qquad \text{Ecuación 6}$$

Luego de detectar los registros duplicados debe realizarse alguna operación con ellos. En la siguiente sección se expone la mezcla o integración de datos como una de las posibles variantes a aplicar.

1.4.2 Integración de datos

Las prácticas principales según Toney (1992) ante registros duplicados son:

- Un registro es seleccionado como maestro y los otros se borran.
- Todos los registros se mantienen, pero se agrupan alrededor de un registro maestro.
- Un registro es seleccionado como maestro y todos los campos de los otros registros que no coincidan se agregan al maestro (integración).

Haciendo referencia a esta última práctica, se tiene que la integración de datos tienen como objetivo combinar datos de diversas fuentes y presentarle al usuario una vista unificada de estos datos (Lenzerini, 2002). Según Batini y Scannapieco (2016) existen dos acercamientos en cuanto al lugar real donde los datos son almacenados por las fuentes para su posterior integración:

- Integración de datos virtual: La vista unificada es virtual y los datos solo se almacenan en sus respectivas fuentes.
- Integración de datos materializada: La vista unificada es materializada, por ejemplo, en un almacén de datos.

Por otra parte, en el proceso de integración de datos se pueden presentar los siguientes conflictos:

- Conflictos a nivel de esquema: Se originan debido a la heterogeneidad de los esquemas. Ejemplos de estos conflictos son los semánticos, de descripción y de estructura (Batini y Scannapieco, 2016).
- Conflictos a nivel de instancia: Se originan debido a las heterogeneidades a nivel de instancia. Los principales conflictos a nivel de instancia son de atributos o de llaves, este último también conocido como de entidad o tupla (Batini y Scannapieco, 2016). Los conflictos de atributos ocurren cuando dos elementos presentan el mismo esquema pero el valor almacenado en determinado atributo defiere en ambos. En el caso de los conflictos de llaves se refieren a dos elementos con el mismo esquema e iguales valores en todos los atributos excepto en aquel que lo identifica (ver Figura 7).

Según Christen (2012b), a pesar de que la mezcla de registros duplicados no se considera como una etapa dentro del proceso de detección de duplicados, el proyecto nombrado *Stanford Entity Resolution Framework*[8] (SERF) (Benjelloun *et al.*, 2006) indica lo conveniente que es su implementación. Como parte de este proyecto se han propuesto algoritmos como *D-Swoosh* (Benjelloun *et al.*, 2007), *G-Swoosh* (Benjelloun *et al.*, 2009), *R-Swoosh* (Benjelloun *et al.*, 2009) y *F-Swoosh* (Benjelloun *et al.*, 2009) que incluyen la mezcla como una de sus etapas.

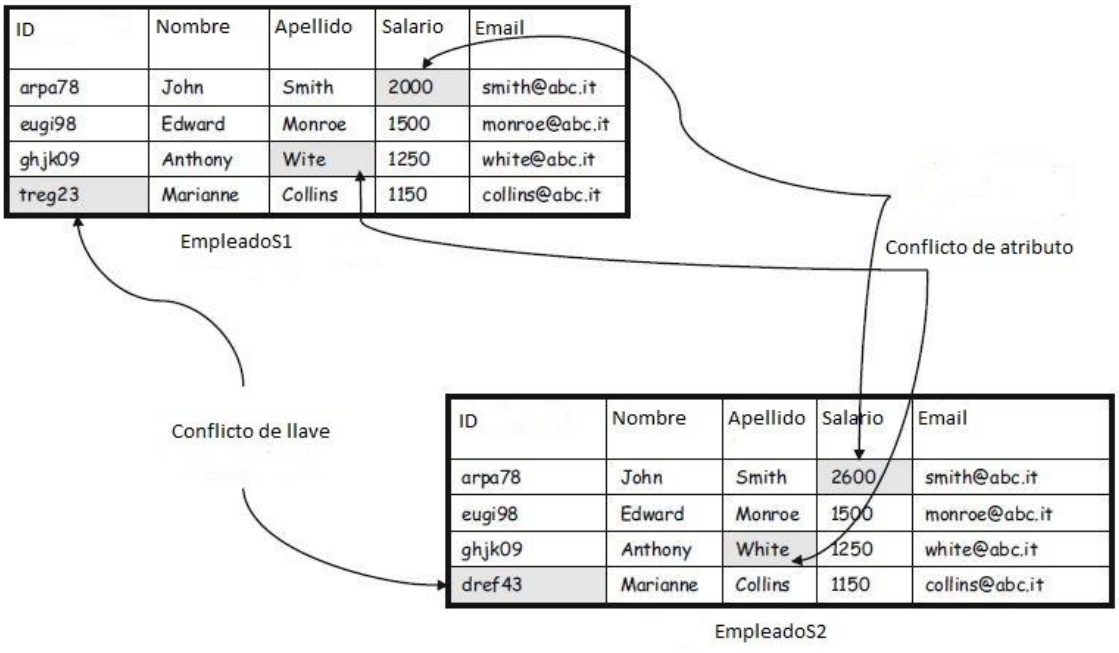


Figura 7: Ejemplo de conflictos de atributo y de llave. Fuente (Batini y Scannapieco, 2016).

1.5 Detección de duplicados en bases de datos bibliográficas

En el contexto de las bases de datos bibliográficas un registro duplicado puede definirse según Sitas y Kapidakis (2008) como dos o más registros que representan el mismo documento (cualquier recurso de información). Entre los factores que propician la

[8] http://infolab.stanford.edu/serf/

existencia de registros duplicados en bases de datos bibliográficas se encuentran la política de catalogación cooperativa (Oliveira *et al.*, 2010), inconsistencias y manejo descuidado en la catalogación y errores en la sintaxis de formatos como MARC 21 (Sitas y Kapidakis, 2008).

Por otra parte, según Sitas y Kapidakis (2008) la existencia de registros duplicados pueden causar los siguientes problemas:

- Bajo control de la calidad.
- Sobrecarga de la información del usuario, debido a la cantidad de documentos que son mostrados y que representan a la misma entidad del mundo real.
- Disminución de la eficiencia del sistema debido al incremento de la cantidad de registros.
- Baja productividad en la catalogación debido al tiempo consumido en la identificación de registros duplicados y la limpieza de la base de datos.
- Incremento del costo del mantenimiento de la base de datos.

Debido a la importancia de las bases de datos bibliográficas para instituciones académicas y de negocios (Fisher *et al.*, 2013) lo ideal es tener un único registro bibliográfico por cada entidad bibliográfica (Sitas y Kapidakis, 2008). Con este objetivo se han propuesto varios algoritmos para la detección de duplicados en datos bibliográficos (Hylton, 1996; Veloso de Melo y de Andrade Lopes, 2005; Borel y Krause, 2009; Borges, de Carvalho, *et al.*, 2011; Borges, Karim Becker, *et al.*, 2011; Borges, Karin Becker, *et al.*, 2011; Jiang *et al.*, 2013; Turenne, 2015).

En (Borges, Karim Becker, *et al.*, 2011; Borges, Karin Becker, *et al.*, 2011) se propone un algoritmo para la detección de duplicados utilizando clasificación. Solo los campos que representan a los autores, título y año de publicación son considerados porque son los mayormente utilizados en la búsqueda y recuperación de información. En la fase de preprocesamiento se realizan las siguientes transformaciones:

- Se convierte el año en un número entero válido en el dominio.
- Se remueven instancias con ruido como las que no tienen autores o año de publicación válido.

- Se eliminan las comillas y los acentos.
- Se convierte todo a minúscula.

Luego de la fase de preprocesamiento, las referencias se combinan para formar pares que se convierten en nuevos registros. Después se calcula la similitud entre cada uno de los campos de cada par de registros y el resultado se añade al registro como un nuevo campo. Por último, si ambos registros son iguales se pone en el atributo "*duplicated pair*" el valor "*yes"*, en caso contrario el valor "*no"* (ver Figura 8).

Los nuevos registros se convierten en la entrada de un algoritmo de clasificación para su entrenamiento. Los algoritmos de clasificación que mejores resultados obtienen según Borges et al. (2011) son *Naïve Bayes* (basado en el teorema de Bayes), RIPPER (basado en reglas) y C4.5 (basado en árboles de decisión). La principal dificultad de este algoritmo es el gran número de pares de registros generados, lo cual no es viable para bases de datos muy grandes (Borges, Karim Becker, *et al.*, 2011; Borges, Karin Becker, *et al.*, 2011). Además, en muchos problemas reales no se conoce a priori la cantidad de elementos duplicados ni los pares de registros que representan a la misma entidad.

	New metadata field	Distance or similarity function
1	authors number$_i$	numbers of authors from reference *i*
2	authors number$_j$	numbers of authors from reference *j*
3	authors number diff	absolute difference between the numbers of authors ($\|authors\ number_i - authors\ number_i\|$)
4	year diff	absolute difference between the publication years ($\|year_i - year_j\|$)
5	authors diff	difference between the authors according to the algorithm *NameMatch* (Borges et al., 2011b)
6	authors sim	similarity between the authors based on the normalized *authors diff*
7	title diff	edit-distance between $title_i$ and $title_j$
8	title sim	similarity between titles based on the normalized difference *title diff*
9	duplicated pair	binary class (*yes* if $class_i = class_j$ or *no* otherwise)

Figura 8: Nuevo registro formado por un par de registros combinados (Borges, Karim Becker, *et al.*, 2011; Borges, Karin Becker, *et al.*, 2011).

Otro algoritmo empleado para la detección de duplicados en datos bibliográficos se propone en (Jiang *et al.*, 2013). Este algoritmo está basado en reglas que utilizan múltiples atributos, para lo cual se tienen en cuenta las siguientes consideraciones:

- Inicialmente se identifican duplicados basados en los atributos PMID (acrónimo del inglés *PubMed Identifier*) y DOI (acrónimo del inglés *Digital Object Identifier*).

- Al conjunto de registros restantes, después de aplicar el paso anterior, se le realiza un proceso de unión natural, en este caso unión basada en un valor *hash*. El atributo escogido es el año.
- Se normalizan los registros.
- Una combinación de dos atributos no decisivos puede identificar unívocamente a un registro. Por ejemplo, ISSN (acrónimo del inglés *International Standard Serial Number*) más título de la revista, título de la revista más título del registro, o título de la revista más información de la página. Cada par de atributos se colocan en una regla que decide si dos registros son el mismo.
- Para los registros restantes se consideran otros atributos.

En este algoritmo se utiliza la función *Longest-Common-Subsequence* (LCS) (Masek y Paterson, 1980) para la comparación entre dos cadenas y considerándose similares cuando la similitud sobrepasa el valor 0.8.

Por otra parte, en (Veloso de Melo y de Andrade Lopes, 2005) se propone un algoritmo para identificar referencias bibliográficas duplicadas. Este algoritmo se divide en dos etapas: preprocesamiento e identificación.

En la etapa de preprocesamiento se extraen los autores, el título y el año de publicación de cada referencia y se crea un registro que contiene como identificador de la referencia (ID) los campos correspondientes a los autores, título, año de publicación y una llave. En caso que la referencia no contenga el año de publicación se asume el valor 9999 para evitar pérdida de información.

Año	Llave	Autores	Titulo
1975	BCEMRCFIRS	E. Rosch and C.B. Mervis	Family resemblance studies in the internal structure of categories.
1975	BCEMRCFRS	Rosch, E., Mervis, C. B	Family resemblances: studies in the structure of categories.
1975	DESBPS	E. D. Sacerdoti	A structure for plans and behavior.
1975	EMRCFPRS	Rosch, E., Mervis	CB.: Family resemblances: studies in the structure of categories.

Figura 9: Salida de la fase de preprocesamiento (Veloso de Melo y de Andrade Lopes, 2005).

La llave se crea tomando el primer carácter de cada *token* presente en los autores y el título, que no sean palabras vacías, dígitos o símbolos delimitadores. Luego, estas iniciales se ordenan alfabéticamente y se eliminan los caracteres repetidos (ver Figura 9).

Estos autores utilizan una estructura de datos métrica o Método de Acceso Métrico (MAM) con el objetivo de reducir la cantidad de comparaciones. En este caso seleccionan Slim-Tree (Traina Jr. *et al.*, 2000), el cual constituye una extensión de M-Tree (ver Anexo 1) (Ciaccia, Patella y Zezula, 1997) y tiene entre sus ventajas que su estructura es dinámica y balanceada (Ciaccia, Patella y Zezula, 1997; Dohnal, 2004). Estas estructuras indexan los objetos en base a sus distancias relativas, las cuales deben ser definidas al crearlas. Un aspecto importante de estas estructuras es que permiten dos tipos de búsqueda: *k* vecinos más cercanos y consulta de rango. En ambas se pasa como primer parámetro el elemento a buscar. El segundo parámetro es diferente en cada tipo de búsqueda. En el caso de los *k* vecinos más cercanos se pasa la cantidad de elementos a recuperar (*k*) y como resultado se obtienen los *k* elementos más cercanos al elemento pasado en el primer parámetro. Por otro lado, en la consulta de rango se pasa un umbral *d*. Todos los elementos cuya distancia respecto al elemento pasado por parámetro sea menor o igual que *d* es recuperado en la búsqueda.

En la etapa de identificación (ver Figura 10) se emplean dos estructuras: una tabla que contiene todos los registros preprocesados y un arreglo de Slim-Tree. Cada Slim-Tree contiene los registros de un año determinado, es decir, se realiza un bloqueo tradicional utilizando como llave el año. En cada Slim-Tree se insertan solamente las llaves no duplicadas con sus respectivos enlaces al registro en la tabla. En ese trabajo se proponen los valores *k*=4 y como umbral de similitud 0.6 o en caso de ser distancia 0.4, para ambas búsquedas.

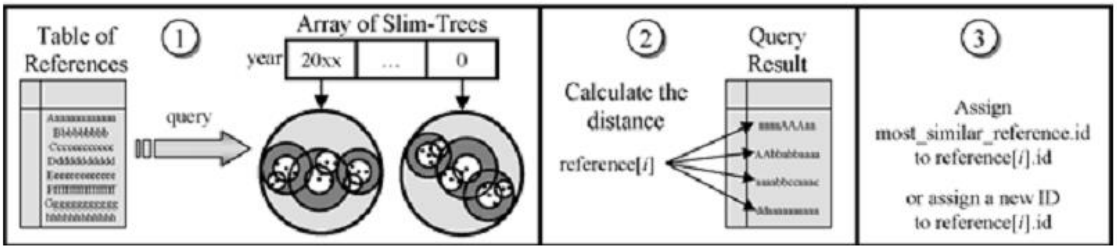


Figura 10: Etapa de identificación (Veloso de Melo y de Andrade Lopes, 2005).

En el caso específico de MARC 21, en (Borel y Krause, 2009) se propone una herramienta escrita en Python denominada MarcXimil que, entre otras funcionalidades, busca registros similares en este formato. La velocidad de esta herramienta, según los propios autores, no es buena para colecciones de mediano y gran tamaño. Su rendimiento es lento en colecciones que superen los 10 000 registros.

Además de los enunciados anteriormente, existen otros algoritmos de detección de duplicados para bases de datos bibliográficas. En (Sitas y Kapidakis, 2008) se realiza un análisis de diez algoritmos, de los cuales cinco de ellos se mantienen activos en la actualidad (ver Tabla 6). Además, cuatro de los diez utilizan la comparación exacta entre los valores de los campos, lo que limita la cuando existen errores en la entrada de los datos (por ejemplo, tipográficos). Por otra parte, la mitad utiliza la mezcla para el manejo de los registros duplicados.

Tabla 6: Comparación entre diez algoritmos de detección de duplicados. Fuente (Sitas y Kapidakis, 2008).

Algoritmo	Activo	Comparación exacta	Manejo de duplicados	
			Eliminación	Mezcla
ALEPH-ULM		X		X
ILCSO	X		X	
Greek Union Catalog	X	X		X
OAK			*	*
MDBUPD		X	X	
IUCS		X	X	
OCLC (Hickey and Rypka)			*	*
DDR	X			X
COPAC	X			X
MELVYL	X			X

X: Presenta la característica. ***** Información no disponible.

1.6 Conclusiones parciales

La calidad de los metadatos es un aspecto importante en el sistema para el que fueron diseñados, especialmente para catálogos que utilicen el formato MARC 21. Particularmente, la completitud constituye una de las principales dimensiones de calidad. Para su medición en el formato MARC 21 para datos bibliográficos las métricas propuestas por Ochoa y Duval se vislumbran como las más convenientes por su fácil adaptación. Por otra parte, la detección de registros duplicados y su posterior integración o mezcla pueden contribuir en la mejora de la calidad, específicamente de la completitud

de los mismos. En este caso, la utilización de varias llaves de bloqueo y de una estructrura de datos perteneciente a la familia de los M-Tree permiten a la reducción del espacio de búsqueda.

Capítulo 2 Métodos empleados para la gestión de la completitud en registros bibliográficos con formato MARC 21.

En este capítulo se identifican las características de las métricas propuestas por Ochoa y Duval que varían para realizar la medición de la completitud en el formato MARC 21. Para ello, se define la cantidad de campos, el grado de importancia de estos y cómo determinar si un campo está completo. Se argumentan las modificaciones realizadas a un algoritmo propuesto en la literatura para la detección de registros bibliográficos duplicados. Además, se elabora un procedimiento para la mejora de la completitud en registros con formato MARC 21. También, se propone un algoritmo para la integración de registros duplicados con este formato que se incluye en el procedimiento elaborado. Por último, se muestran los diagramas creados durante las fases de análisis y diseño del software propuesto y se ilustran patrones de diseño empleados.

2.1 Medición de la completitud en registros bibliográficos con formato MARC 21

En la fase de medición de la completitud se deben seleccionar las métricas a implementar en dependencia del dominio donde se apliquen. Debido a que el formato MARC 21 es un modelo de metadatos, en el presente trabajo se utilizan las métricas propuestas por (Ochoa y Duval, 2006a, 2006c, 2009) y expuestas en el epígrafe 1.2.3. Estas son fáciles de implementar en una amplia variedad de formatos de metadatos porque, a pesar de que su configuración inicial es dependiente del contexto (Tani *et al.*, 2013), han sido implementadas para formatos de metadatos como *Learning Object Metadata* (Ochoa y Duval, 2006a, 2006b, 2008). La configuración inicial de dichas métricas puede variar en base a los siguientes aspectos:

- La definición de completitud de un campo (ver Ecuación 2 y Ecuación 3).
- La cantidad de campos totales a medir (ver Ecuación 2 y Ecuación 3).
- El grado de importancia que se le atribuye a cada campo (ver Ecuación 3).

En la literatura revisada no se reportan implementadas estas métricas para el caso del formato MARC 21, por lo que debe definirse la configuración inicial de las mismas para este contexto.

2.1.1 Definición de completitud de un campo

En el presente trabajo se plantean las siguientes consideraciones para determinar si un campo perteneciente al formato MARC 21 está completo:

- Una instancia de un campo variable de un registro en formato MARC 21 está completa si:
    - su valor no es la cadena vacía, para el caso de los campos variables de control.
    - si al menos está presente el subcampo "a" y su valor no es la cadena vacía, para los campos variables de datos.
- Un campo variable está completo si al menos una de sus instancias está completa (Ochoa y Duval, 2006a, 2006c, 2009).

En la Figura 11 se muestran dos instancias del campo variable de datos repetible con etiqueta 300. Según las consideraciones anteriores la primera instancia está completa debido a que presenta el subcampo "a" y su valor no es la cadena vacía. Por el contrario, la segunda instancia no está completa. Por tanto, el campo 300 está completo porque al menos una de sus instancias lo está.

```
300 $a500 p. ;
300 $c22 cm.
```

Figura 11: Completitud de un campo con múltiples instancias.

2.1.2 Cantidad de campos totales a medir

La cantidad de campos totales a medir que se utilizan en la Ecuación 2 (todos los campos tienen igual grado de importancia) se toman a partir de tres niveles: mínimo[9], completo[10]

[9] https://www.loc.gov/marc/bibliographic/bdapndxc.html.
[10]https://www.loc.gov/marc/bibliographic/examples.html

y general[11]. Esta clasificación se corresponde con los ejemplos expuestos en la página oficial de la Biblioteca del Congreso para el formato MARC 21.

En (García-Mendoza, Díaz-de-la-Paz, González-González, Nuñez-Arcia, Leiva-Mederos, *et al.*, 2016) se definen los campos variables por clase de material que debe tener un registro para que este se corresponda con el nivel mínimo o completo. Para el caso del nivel general son todos aquellos que se utilizan en la catalogación de registros bibliográficos. El total de campos a medir debe corresponderse con la clase de material y el nivel que se esté utilizando.

2.1.2.1 Nivel mínimo

En la página de la Biblioteca del Congreso destinada al formato MARC 21 se exponen ejemplos con los campos que presentan el nivel mínimo para las siguientes seis de las ocho clases de materiales existentes[9]: libro, archivo de computadora, partitura, mapa, publicación seriada y material mixto. En (García-Mendoza, Díaz-de-la-Paz, González-González, Nuñez-Arcia, Leiva-Mederos, *et al.*, 2016) se utilizan estos campos para la medición de la completitud.

El dominio de aplicación inmediato de la presente investigación son los catálogos cubanos con formato MARC 21. Para lo cual se refinan dichos campos (ver Tabla 7) teniendo en cuenta la opinión del Dr. Amed Abel Leiva Mederos[12, 13] y de otros especialistas en catalogación de la UCLV. Con respecto a lo planteado en (García-Mendoza, Díaz-de-la-Paz, González-González, Nuñez-Arcia, Leiva-Mederos, *et al.*, 2016) se eliminan de este nivel los campos con etiquetas 007 y 040 en todas las clases de material porque no se utilizan en nuestro país. Además se realiza lo siguiente:

[11] https://www.loc.gov/marc/bibliographic/ecbdlist.html
[12] https://www.researchgate.net/profile/Amed_Mederos
[13] https://scholar.google.com.cu/citations?user=5a5yzPQAAAAJ&hl=es

- Se añade el campo 222 a la clase de material "publicación seriada" con la restricción de que debe estar completo este campo o el 245.
- Se añade a la clase "mapa" el campo 100.
- En el caso de las clases "material visual" y "grabación sonora" se toman los campos que deben tener todos los registros de manera obligatoria al no presentarse ejemplos de estos.

Tabla 7: Campos necesarios por clase de material para el nivel mínimo.

CM	Campos necesarios	N
L	001, 003, 005, 008, 100, 245, 260, 300	8
A	001, 003, 005, 008, 245, 256, 260, 300, 538	9
P	001, 003, 005, 008, 028, 100, 240, 245, 260, 300, 650	11
M	001, 003, 005, 008, 034, 052, 100, 110, 245, 255, 260, 300, 500, 650	14
Ps	001, 003, 005, 008, 245 (222), 260, 300, 310, 500, 710	10
Mm	001, 003, 005, 008, 100, 245, 300, 545	8
Gs	001, 003, 005, 008, 245, 300	6
Mv	001, 003, 005, 008, 245, 300	6

CM: Clase de material **L:** Libro. **A:** Archivo de computadora. **P:** Partitura. **M:** Mapa. **Ps:** Publicación seriada. **Mm:** Material mixto. **Gs:** Grabación sonora. **Mv**: Material visual. **N**: Cantidad de campos totales a medir

2.1.2.2 Nivel completo

De manera similar al nivel mínimo en la página de la Biblioteca del Congreso se exponen ejemplos de registros que presentan el nivel completo para las siguientes siete de las ocho clases de materiales existentes[10]: libro, archivo de computadora, mapa, publicación seriada, material mixto, grabación sonora y material visual. En este nivel también se eliminan de todas las clases de material los campos 007 y 040 como resultado del refinamiento de lo planteado en (García-Mendoza, Díaz-de-la-Paz, González-González, Nuñez-Arcia, Leiva-Mederos, *et al.*, 2016). Además, se realiza lo siguiente:

- Se eliminan los campos 050 y 246 de la clase "libro".
- Se eliminan los campos 362 y 780 de la clase "publicación seriada".
- Se eliminan los campos 010 y 035 de la clase "material mixto".
- Se eliminan los campos 043, 045, 047, 048 y 050 de la clase "grabación sonora".
- Se añade el campo 256 a la clase "archivo de computadora".
- En el caso de la clase "material visual" se toman los campos necesarios comunes a todos los ejemplos.

- En el caso de la clase de material "partitura" clase se consulta con los especialistas al no tener ejemplos publicados en la Biblioteca del Congreso sobre este nivel.

Tabla 8: Campos necesarios por clase de material para nivel completo.

CM	Campos necesarios	N
L	001, 003, 005, 008, 020, 082, 100, 245, 260, 300, 500, 650	12
A	001, 003, 005, 008, 100, 245, 250, 256, 260, 300, 500, 520, 538, 710, 753	15
P	001, 003, 005, 008, 028, 100, 240, 245, 260, 300, 650, 710	12
M	001, 003, 005, 008, 034, 052, 100, 110, 245, 246, 255, 260, 300, 500, 650, 700, 710, 730	18
Ps	001, 003, 005, 008, 010, 022, 035, 042, 043, 050, 082, 210, 222, 245, 246, 260, 300, 310, 500, 650, 710, 850	22
Mm	001, 003, 005, 008, 041, 100, 245, 300, 351, 506, 520, 524, 530, 541, 544, 545, 546, 555, 600, 610, 650, 651, 655, 656, 852	25
Gs	001, 003, 005, 008, 028, 100, 245, 260, 300, 500, 505, 511, 650, 700	14
Mv	001, 003, 005, 008, 245, 300, 500, 520, 650	9

CM: Clase de material **L:** Libro. **A:** Archivo de computadora. **P:** Partitura. **M:** Mapa. **Ps:** Publicación seriada. **Mm:** Material mixto. **Gs:** Grabación sonora. **Mv**: Material visual. **N**: Cantidad de campos totales a medir

2.1.2.3 Nivel general

En este nivel también se eliminan los campos 007 y 040 por no utilizarse en los catálogos cubanos. El total de campos a medir en este nivel es 272 campos. Otro elemento es que no se realiza ninguna distinción entre clases de materiales.

2.1.3 Determinación del grado de importancia

En la medición de la completitud utilizando la Ecuación 3 es necesario calcular primero los grados de importancia (también conocidos como pesos) de cada campo. En el presente trabajo se calculan los mismos como la suma de las incidencias (o frecuencia de aparición) de cada campo en un conjunto de registros con igual clase de material.

La incidencia de un campo en un registro consiste en la ausencia o presencia de un campo dentro del mismo (Mayernik, 2010). A la definición anterior se le añade que el campo además de estar presente, tiene que estar completo (ver epígrafe 2.1.1) para que este incida en un registro.

Es necesario señalar que se le determina el grado de importancia solo a los 272 campos del nivel general. Los restantes campos tienen como grado el valor 0.

Para la obtención del grado de importancia de un campo determinado, se propone como parte de esta investigación la Ecuación 7:

$$\mathrm{I(i)} = \sum_{j=1}^{N} R(j) \qquad \text{Ecuación 7}$$

Donde N es la cantidad de registros, $R(j)$ toma valor uno si el campo i está presente y completo en el registro j y cero en otro caso.

Por ejemplo, se tiene una colección de cinco registros con formato MARC 21. De ellos tres registros pertenecen a la clase de material "libro" (R1, R2, R3) y dos a la clase "material mixto" (R4, R5). A estos registros se les calcula la completitud variando el valor total de campos (N) de acuerdo a los tres niveles de completitud mencionados anteriormente (mínimo, completo y general). Además, se le añade una última columna a dicha tabla donde se calcula la completitud utilizando la métrica pesada tomando como grado de importancia de cada campo su grado de incidencia dentro de la clase de material a la que pertenece el registro.

Tabla 9: Ejemplo del cálculo de la completitud.

	Campos completos que contiene el registro	CM	CC	CNM	CNC	CNG	CGI
R1	001, 003, 005, 008, 100, 245, 260, 500, 650	L	9	0.875	0.692	0.033	0.800
R2	001, 003, 005, 008, 020, 041, 082, 245, 260, 700	L	10	0.750	0.615	0.036	0.767
R3	001, 003, 005, 008, 020, 100, 245, 260, 300, 500, 650	L	11	1.000	0.846	0.040	0.9
R4	001, 003, 005, 008, 100, 245, 300, 600, 650	Mm	9	0.875	0.346	0.033	0.850
R5	001, 003, 005, 008, 041, 100, 245, 300, 524, 545, 650	Mm	11	1.000	0.423	0.040	0.95

CM: Clase de material. **L:** Libro. **Mm:** Material mixto. **CC:** Total de campos completos. **CNM:** Completitud nivel mínimo. **CNC**: Completitud nivel completo **CNG:** Completitud nivel general. **CGI:** Completitud con diferentes grados de importancia.

En los resultados expuestos en la Tabla 9 se aprecia que a pesar de que el registro R1 tiene menor cantidad de campos que R2, su completitud basada en diferentes grados de importancia para los campos es mayor. Esto se debe a que al registro R2 además del campo 300 (que también le falta a R1) carece de los campos 100 (autor principal), 500 (nota al autor) y 650 (materia). Los demás registros que pertenecen a esa clase (R1 y R3) los tienen. Además, se observa cómo el grado de importancia de un campo se calcula para

cada clase de material. Por ejemplo, el campo 041 tiene un grado de importancia de uno en las dos clases porque incide solamente en un registro de cada una de ellas (R2y R5).

En la medida en que el volumen de los metadatos aumenta, el cálculo de la completitud se ralentiza, por lo que se hace necesario utilizar otras tecnologías que agilicen dicho proceso.

2.1.4 Medición de la completitud en grandes volúmenes de metadatos

Una de las tecnologías utilizadas en la actualidad para el procesamiento de grandes volúmenes de datos es el modelo de programación *MapReduce* (Liu *et al.*, 2012), el cual da soporte a la computación distribuida en grupos de computadoras cuyas prestaciones pueden ser bajas (Dean y Ghemawat, 2008). Este modelo permite ejecutar un programa de manera distribuida encargándose de las fallas y manejando la comunicación requerida entre el grupo de computadoras. Además, ha sido utilizado para reducir el tiempo de ejecución de varios algoritmos (McNeill *et al.*, 2012). Su núcleo son las funciones *map* y *reduce*, que están inspiradas en las funciones de igual nombre presentes en el lenguaje *Lisp* y otros lenguajes funcionales (Dean y Ghemawat, 2008). Estas funciones se describen a continuación:

- *map*: La función *map*, creada por el programador, se aplica a todos los pares $<k_1, v_1>$ de entrada. Para cada par devuelve una lista de cero o más pares de llave/valor intermedios *(lista* $(<k_2, v_2>)$*)* (Figura 12). Luego los valores (v_2) de cada par $<k_2, v_2>$ devuelto por la función *map* son reorganizados por su llave (k_2). De esta manera los valores con las mismas llaves (k_2) son unidos en un par que tiene la forma $<k_2, lista(v_2)>$. Este último par se convierte en la entrada de la función *reduce*.

$$map(k_1, v_1) \longrightarrow lista(<k_2, v_2>)$$

Figura 12: Vista lógica de la función map. Fuente: (Holmes, 2012).

- *reduce*: La función *reduce*, creada por el programador, se invoca una vez para cada llave distinta en los pares $<k_2, lista(v_2)>$. Para cada par la función *reduce* tiene como salida cero o más pares $< k_3, v_3 >$ (Figura 13). La lista de valores asociados a una llave determinada se suministra a la función *reduce* mediante un iterador.

$$reduce(k_2, lista\ (v_2)) \longrightarrow lista(<k_3, v_3>)$$

Figura 13: Vista lógica de la función reduce. Fuente: (Holmes, 2012).

En cada uno de los algoritmos que se presentan en este trabajo que utilizan *MapReduce*, la función *map* recibe como *llave* (k_1), un número entero que representa la posición del registro dentro de la colección y como *valor* (v_1) el registro en sí. Además, se utiliza en cada una de estas funciones un conjunto nombrado *completos*, para indicar las etiquetas de los campos donde al menos una de sus instancias está completa y una función denominada *esCompleta*, que determina si una instancia dada de un campo está completa o no siguiendo las consideraciones del epígrafe 2.1.1.

Algoritmo 1: Función **map** para medir la completitud utilizando la Ecuación 2

```
Entrada: Posición del registro dentro del conjunto (llave)
Entrada: Registro (valor)
Salida: Un par con el identificador del registro y el valor de la
completitud
función map (llave, valor)
1:   completos<Etiqueta> ← conjunto vacío
2:   lista_De_Instancias<Instancia> ← Obtener las instancias de valor
3:   que pertenezcan a los campos del nivel analizado
4:   Para cada instancia de lista_De_Instancias Hacer
5:     etiqueta ← Etiqueta de instancia
6:     Si completos no contiene etiqueta y esCompleta(instancia)
7:     Entonces
8:       Añadir etiqueta a completos
9:     Fin Si
10:  Fin Para
11:  completitud ← tamaño de completos / total_de_campos
12:  escribir(llave, completitud)
```

En la medición de la completitud utilizando la Ecuación 2 se crea un trabajo *MapReduce* que solo contiene la función *map*, como se muestra en el Algoritmo 1. En este debe sustituirse el término *total_de_campos* por el total de campos que tiene la clase de material a la que pertenece el registro de metadatos en el nivel de completitud que se esté analizando. Para cada instancia de un campo perteneciente al registro de entrada se verifica si no existe ninguna instancia analizada anteriormente que esté completa con la misma etiqueta y que la instancia esté completa con la función *esCompleta*. De cumplirse lo anterior se añade el identificador del campo al conjunto *completos*. Por último, se escribe un par <*llave*, *valor*> cuya *llave* es el identificador del registro y el *valor* la razón

entre los campos completos (tamaño del conjunto *completos*) y el total de campos, es decir, la completitud.

Para medir la completitud utilizando la Ecuación 3 es necesario calcular primero el grado de importancia de cada campo y la suma de los mismos por cada clase de material. Una vez calculados los valores anteriores se ejecuta el Algoritmo 2 que utiliza además del conjunto *completos* y la función *esCompleta* una tabla *hash* denominada *grados* y la variable entera *total_De_Grados* que representan el grado de importancia de cada campo y la suma de los mismos, respectivamente. Esta función solo se diferencia de la función *map* descrita en el Algoritmo 1 en la línea 8 donde se actualiza la variable *completitud* a partir de la razón entre el grado de importancia del campo que representa cada instancia completa y la sumatoria total de los grados (*total_De_Grados*).

Algoritmo 2: Función **map** para medir la completitud utilizando la Ecuación 3

```
Entrada: Posición del registro dentro del conjunto (llave)
Entrada: Registro (valor)
Salida: Un par con el identificador del registro y el valor de la
completitud
función map (llave, valor)
1:   completos<Etiqueta> ← conjunto vacío
2:   lista_De_Instancias<Instancia> ← Obtener las instancias de valor
3:   que pertenezcan a los campos del nivel analizado
4:   completitud ← 0
5:   Para cada instancia de lista_De_Instancias Hacer
6:     etiqueta ← Etiqueta de instancia
7:     Si completos no contiene etiqueta y esCompleta(instancia)
8:     Entonces
9:       Añadir etiqueta a completos
10:      fracción ← grado de etiqueta / total_De_Grados
11:      completitud ← completitud + fracción
12:    Fin Si
13:  Fin Para
14:  escribir(llave, completitud)
```

Una vez realizada la medición se debe aplicar la fase de mejora. Esta es la siguiente fase de las metodologías de calidad de datos e incluye entre sus principales técnicas la detección de duplicados e integración de datos (Batini *et al.*, 2009).

2.2 Detección de registros duplicados con formato MARC 21

Una de las técnicas que se puede utilizar en la mejora de la calidad de datos es la detección de duplicados (Batini *et al.*, 2009). En el presente trabajo se propone una modificación al

algoritmo propuesto en (Veloso de Melo y de Andrade Lopes, 2005) para la detección de referencias bibliográficas duplicadas. Este algoritmo presenta entre sus ventajas con respecto a los demás explicados en el epígrafe 1.5 la reducción del espacio de búsqueda y de la cantidad de comparaciones, además no necesita conocer a priori un conjunto de entrenamiento. Otra ventaja es la utilización de una estructura de datos métrica que permite indexar los objetos y realizar dos tipos de búsquedas. También, en esta estructura de datos es posible insertar los objetos una única vez y luego realizar búsquedas solamente, lo que reduce la complejidad computacional a un $O(\log n)$.

Los campos utilizados por el algoritmo propuesto en (Veloso de Melo y de Andrade Lopes, 2005) incluyen el año, el autor y el título de cada referencia por lo que es posible su adaptación al formato MARC 21. Además, estos campos son utilizados por otros algoritmos como el propuesto en (Borges, Karim Becker, *et al.*, 2011; Borges, Karin Becker, *et al.*, 2011). En el presente trabajo se añaden los campos "ISBN" (acrónimo del ingés *International Standard Book Number*) y la "clase de material" de cada registro (ver Tabla 10). Esto se debe a que el 90% de los algoritmos expuestos en (Sitas y Kapidakis, 2008) utilizan los campos año, autor y título y el 60 % utiliza el ISBN.

Tabla 10: Campos del formato MARC 21 utilizados en la detección de duplicados.

Etiqueta	Subcampos	Descripción
Cabecera	6, 7	Clase de material.
020	a	ISBN.
100	a	Autor principal.
110	a	Autor corporativo (Este campo se toma en caso que el 100 esté vacío).
245	a, b	Título.
260	c	Año de publicación.

A continuación se describe el algoritmo con las modificaciones realizadas a través de las etapas que conforman el proceso de detección de duplicados (ver epígrafe 1.4.1).

2.2.1 Etapa de preprocesamiento

En esta etapa se realiza un preprocesamiento de los datos presentes en cada uno de los campos seleccionados. A estos campos se les aplica un grupo de transformaciones, como se muestra en la Tabla 11. En el algoritmo propuesto esta etapa se realiza una única vez al insertar un registro en las estructuras de cada bloque.

Luego de aplicar estas transformaciones a los campos seleccionados se crea un nuevo campo con las iniciales resultantes del autor y del título ordenadas alfabéticamente por separado y posteriormente concatenadas (ver Figura 9). Por ejemplo, si se tiene un registro que posee como autor "García Mendoza, Juan Luis" y título "Medición de la completitud", entonces las iniciales ordenadas resultantes son "GJLM" y "CM" respectivamente, en este último caso se eliminan las palabras vacías "de" y "la". El nuevo campo a añadir al registro es "GJLMCM".

Tabla 11: Transformaciones realizadas a los datos.

Campos	Transformaciones
Autores y Título	• Eliminar palabras vacías[14*]. Las palabras vacías se eliminan de acuerdo al idioma del texto. Solo se tienen en cuenta los idiomas español, inglés, francés, italiano y portugués. • Eliminar espacios al inicio y al final. • Eliminar caracteres como dos puntos, puntos, puntos y comas, comas, guión, entre otros. • Convertir todo a minúscula.
Año de publicación	• Eliminar cualquier carácter que no sea dígito, guión o ?. • Si existen más de cuatro dígitos se extraen los cuatro más a la izquierda. • Si existe un intervalo de la forma xxxx-xxxx, donde x es un dígito, se extrae el primer año. • Si existe un intervalo de la forma xxxx-, donde x es un dígito, se elimina el guión • En otro caso se asigna el valor -1.

*En el caso del autor no se eliminan las palabras vacías que sean una sola letra debido a que es común el uso de iniciales en los nombres a modo de abreviaturas.

2.2.2 Etapa de indexado

Según Christen (2012a) el factor más importante para la eficiencia y exactitud de las técnicas de bloqueo es la definición de la llave. Se plantea que ante la no existencia de conjuntos de entrenamiento en aplicaciones reales, el conocimiento del dominio de aplicación puede influir en la definición de la llave. Por otro lado, en (Winkler *et al.*, 2010;

[14] IR Multilingual Resources at UniNE http://members.unine.ch/jacques.savoy/clef/

Christen, 2012b) se recomienda definir varias llaves de bloqueo y realizar varias pasadas utilizando en cada una de ellas una llave determinada.

En el algoritmo propuesto en (Veloso de Melo y de Andrade Lopes, 2005) se utilizan dos llaves de bloqueo: el año y la llave compuesta por las iniciales ordenadas del autor y título. En la presente investigación se propone una modificación a las llaves de bloqueo anteriores, combinándose la clase de material con el año de publicación del registro bibliográfico y se mantienen las iniciales. Esta modificación permite una mayor reducción con respecto a las llaves anteriores. Por ejemplo, se tienen 500 registros en un año determinado, de ellos 350 son libros y 150 son mapas. Si se utiliza como primera llave solamente el año, el espacio de búsqueda son los 500 registros pertenecientes a ese año. Por el contrario, de emplearse la llave "clase de material-año", ese espacio de búsqueda se reduce a 350 y 150 registros. Por otra parte, la llave "clase de material-año" tiene como desventaja que el registro debe tener la clase de material y el año correctamente. La llave de bloqueo compuesta por las iniciales no presenta este problema porque la comparación que se utiliza es aproximada.

Otra diferencia con respecto a lo planteado en (Veloso de Melo y de Andrade Lopes, 2005) es que se tiene una tabla con todos los registros por cada combinación de la clase de material y el año en vez de una tabla general. De manera similar que en (Veloso de Melo y de Andrade Lopes, 2005), se adopta una estructura de datos métrica por cada combinación (ver Figura 14). En este caso se escoge M-Tree por cada combinación porque permite un indexado dinámico y balanceado, y el almacenamiento en disco (Dohnal, 2004). Además, la estructura utilizada en (Veloso de Melo y de Andrade Lopes, 2005) es una extensión de M-Tree.

El procedimiento de bloqueo o indexado para cada registro es el siguiente:

1. Se obtiene la clase de material y el año a concatenar.
2. Se inserta el registro en la tabla asociada a la llave de bloqueo "clase de material-año".
3. Se inserta la llave de bloqueo compuesta por las iniciales en la estructura métrica M-Tree si no existe ninguna igual a ella.

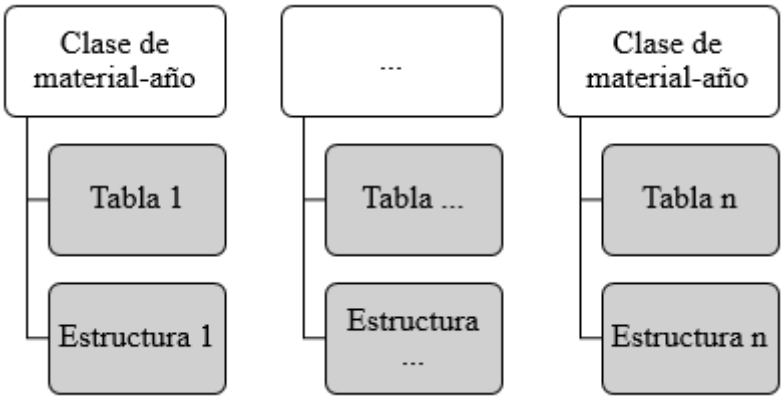


Figura 14: Estructura de bloqueo por cada llave "clase de material-año".

Al igual que la fase de preprocesamiento, el indexado se realiza una única vez al indexar un registro nuevo. Cuando concluye esta etapa almacenan estas estructuras en una base de datos para posteriores búsquedas.

Luego de preprocesar el año de publicación de un registro se pueden presentar los siguientes patrones[15], los cuales son considerados en la solución propuesta:

- 1982: Se utilizan las estructuras asociadas a la llave "clase de material-año".
- 1982? (año probable): Se utilizan las estructuras asociadas a la llave "clase de material-año probable".
- 189- (década segura): Se utilizan las estructuras asociadas a las llaves clase de material concatenada con todos los años pertenecientes a la década segura.
- 189-? (década probable): Se utilizan las estructuras asociadas a las llaves clase de material concatenada con todos los años pertenecientes a la década probable.
- 18-- (siglo seguro): Se utilizan las estructuras asociadas a las llaves clase de material concatenada con todos los años pertenecientes al siglo seguro.

[15] Norma Cubana 154: 2002. Descripción Bibliográfica de Libros y Folletos

- 18--? (siglo probable): Se utilizan las estructuras asociadas a las llaves clase de material concatenada con todos los años pertenecientes al siglo probable.
- -1: Este patrón se asigna cuando no existe el año o si existe no se corresponde con ninguno de los patrones anteriores. En este caso se utilizan las estructuras asociadas a las llaves clase de material concatenada con todos los años existentes en la colección de registros.

2.2.3 Comparación de pares de registros

En esta etapa se realiza la comparación entre todos los pares de registros que pertenecen a un mismo bloque. Esta comparación se realiza cada vez que se efectúa una inserción en la estructura de datos M-Tree. En dependencia de los objetos almacenados en la estructura se debe definir o utilizar una función de distancia. Debido a que los objetos que se almacenan son cadenas se deben utilizar funciones definidas para este tipo de datos. Si a lo anterior se le añade que estas cadenas están formadas por un solo *token* que contiene las iniciales de los autores y del título, la función a emplear debe ser basada en caracteres. En este trabajo se selecciona la distancia de edición de Levenshtein clásica (Levenshtein, 1966) por ser una de las más utilizadas en la literatura. En investigaciones posteriores debe evaluarse la utilización de otras funciones.

2.2.4 Clasificación de pares de registros

En esta etapa se realiza la clasificación sobre los elementos devueltos en cada consulta sobre la estructura M-Tree. Un elemento a considerar en esta etapa es la cantidad de bases de datos. En el caso de que el total sea uno, se realiza la clasificación entre los registros pertenecientes al mismo bloque dentro de la base de datos. Por el contrario, cuando la búsqueda se realiza entre dos o más bases de datos la clasificación se realiza de la siguiente manera:

- Entre el total de bases de datos se debe seleccionar una como principal.
- Luego, por cada registro de la base de datos principal se obtiene la llave “clase de material-año” del bloque al que pertenece y se busca en las demás bases de datos el bloque que se corresponde con esa llave.

- Si existe el bloque en al menos una base de datos se consulta la estructura de datos M-Tree del mismo con la iniciales del registro de la base de datos principal. La clasificación se realiza entre el registro buscado y los devueltos por la consulta.
- Si no existe ningún bloque, se analiza el próximo registro y se repite el proceso.

De esta forma se garantiza lo planteado en el epígrafe 1.4.1 al buscar duplicados en una base de datos o en varias.

El algoritmo original (Veloso de Melo y de Andrade Lopes, 2005) realiza la clasificación en base al autor y título de cada referencia devuelta por la consulta. La propuesta que se realiza en la presente investigación es utilizar la clasificación basada en reglas (Hernández y Stolfo, 1998; Cohen, 2000; Naumann y Herschel, 2010; Bharambe *et al.*, 2012) teniendo en cuenta algunos elementos expuestos en (Jiang *et al.*, 2013). Esto permite extender las reglas de manera sencilla e incluso definirlas para cada clase de material. Las reglas que se proponen son para todas las clases de materiales. Estas reglas se describen a continuación:

- Si el ISBN existe en ambos registros y es el mismo, entonces son duplicados.
- En caso contrario, si las similitudes entre los autores y los títulos es mayor que 0.75, son duplicados. Este valor se utilizan en (Veloso de Melo y de Andrade Lopes, 2005).
- En otro caso, no son duplicados.

Las reglas descritas anteriormente se pueden apreciar entre las líneas 12 y 20 del Algoritmo 3 que busca duplicados en una sola base de datos. Para el cálculo de la similitud se utiliza la métrica basada en *tokens* de Monge-Elkan (Monge y Elkan, 1996) de conjunto con la de Smith-Waterman (Smith y Waterman, 1981). Esta última se utiliza para calcular la similitud entre dos *tokens*.

El Algoritmo 3 recibe como entrada una tabla y una estructura M-Tree ambas pertenecientes a una llave de bloqueo “clase de material-año” determinada y un parámetro que puede ser los *k* vecinos más cercanos a recuperar o un umbral de distancia. Este último

parámetro es necesario para realizar la consulta sobre el M-Tree. Como resultado devuelve una tabla *hash* donde la llave es el elemento que se busca en la estructura y el valor es la lista de los posibles duplicados.

Algoritmo 3: Buscar registros bibliográficos con formato MARC 21 que sean duplicados en una clase de material y un año determinado en una sola base de datos

```
Entrada: Tabla (tabla)
Entrada: Árbol M-Tree (mtree)
Entrada: Parámetro para la consulta (param)
Salida: Posibles registros duplicados
función obtener_Posibles_Duplicados (tabla, mtree, param)
1:   tabla_Hash<Registro, Lista<Registro>> ← null
2:   Para cada fila de tabla Hacer
3:     llave ← Obtener llave de fila
4:     resultado ← Consulta (k vecinos más cercanos o de rango) sobre
5:     mtree con llave y param
6:     bloque<Registro> ← null
7:     duplicados<Registro> ← null
8:     Si el tamaño de resultado > 0 Entonces
9:       Añadir a bloque todos los registros de tabla que tengan como
10:      llave una que esté contenida en resultado
11:      Para cada registro de bloque Hacer
12:        Si fila.ISBN != null y registro.ISBN ¡= null y
13:           fila.ISBN = registro.ISBN Entonces
14:          Añadir registro a duplicados
15:        En caso contrario
16:          Si Similitud entre fila.Autor y registro.Autor > 0.75 y
17:          Similitud entre fila.Título y registro.Título > 0.75
18:          Entonces
19:             Añadir registro a duplicados
20:        Fin Si
21:      Fin Para
22:    Fin Si
23:    Si el tamaño de duplicados > 0 Entonces
24:      Añadir <llave, duplicados> a tabla_Hash
25:    Fin Si
26:  Fin Para
27:  Retornar tabla_Hash
```

Las clasificaciones para cada par de registros son “posibles duplicados” y “no duplicados”. El especialista es el responsable de seleccionar entre los posibles duplicados los que realmente lo son.

La complejidad computacional de esta etapa es $O(Bn_{max}d\log m_{max})$, donde n_{max} es el tamaño máximo de los *B* bloques pertenecientes a la colección principal, *d* es la cantidad de bases de datos y m_{max} el tamaño máximo del total de bloques incluyendo todas las bases de datos. Esta complejidad representa las consultas sobre cada M-Tree.

2.2.5 Evaluación

La fase de evaluación se realiza en el próximo capítulo para varios conjuntos de datos sintéticos creados que contienen registros bibliográficos con formato MARC 21.

La complejidad del algoritmo debe verse en dos momentos: construcción y búsqueda. Al momento de construcción pertenecen las etapas de preprocesamiento, indexado y comparación, en las cuales se deben recorrer todos los registros de la colección, preprocesarlos e indexarlos. En la etapa de indexado, entre otras acciones, se debe construir la estructura M-Tree que incluye la etapa de comparación y presenta una complejidad computacional alta que se discute en (Dohnal, 2004).

Por otro lado, la búsqueda incluye la etapa de clasificación. La complejidad de esta etapa es $O(Bn_{max} d \log m_{max})$ que se discute en el epígrafe 2.2.4. La etapa de evaluación no se incluye en ninguno de los dos momentos porque se utiliza para evaluar la calidad de la clasificación y no es necesario realizarla siempre.

2.3 Procedimiento para la mejora de la completitud

La integración o mezcla de datos es otra de las técnicas que contribuyen a la mejora de la calidad de datos (Batini *et al.*, 2009). Según Menestrina *et al.* (2006) y Benjelloun *et al.* (2007, 2009) incluir este proceso en la detección de duplicados es conveniente. En la presente investigación se propone un algoritmo que integra un conjunto de registros bibliográficos duplicados con formato MARC 21 en uno más completo. De manera general el procedimiento para la integración es el siguiente:

- **Paso 1:** Detectar posibles registros duplicados.
- **Paso 2:** De cada conjunto de posibles registros duplicados, seleccionar los que realmente lo son. El especialista debe revisar si realmente los registros se refieren a la misma entidad del mundo real.
- **Paso 3:** Seleccionar dentro del conjunto de duplicados seleccionados en el paso anterior el registro principal en base a:
    - Colección a la que pertenece: El registro principal debe pertenecer a la colección principal.

  - Valores de completitud (en un futuro nivel de calidad): Cuando se tienen varios registros que pertenecen a la colección principal se comparan los valores de completitud y el que tenga mayor valor es el registro principal. La comparación se realiza primero en el nivel mínimo. De tener valores iguales se analiza el nivel completo, luego el general y por último la completitud con diferentes grados de importancia para cada campo. En el caso de que todos los valores sean iguales se escoge el primero.
  - Criterio del experto: El experto puede decidir en base a su criterio cuál es el registro principal. La única restricción para elegir este registro es que debe pertenecer a la colección principal.
- **Paso 4:** Integrar el registro principal con los registros restantes del conjunto de duplicados para obtener uno solo (ver Algoritmo 4).
- **Paso 5:** El experto debe verificar que el registro integrado, también denominado registro unificado, esté correcto.
- **Paso 6:** La integración de datos se materializa eliminando de la colección principal todos los registros que pertenezcan a la misma y añadiendo el registro integrado. Con los registros que pertenecen a otras colecciones no se realiza ninguna acción.

En el paso 4 pueden ocurrir conflictos a nivel de esquema y de instancia. Los conflictos a nivel de esquema (Batini y Scannapieco, 2016) están dados porque todos los registros no contienen los mismos campos ni la misma cantidad de instancias por cada uno de ellos. Además, existen campos que no son repetibles, es decir, solo puede existir una instancia de ellos en un registro. De igual manera, pero a nivel de campos, todas las instancias no tienen la misma cantidad de subcampos y existen muchos de ellos que no son repetibles.

Por otra parte, se pueden presentar a nivel de instancia (Batini y Scannapieco, 2016) los conflictos de llaves y de atributos. En el caso de los conflictos de llaves, éstos se solucionan tomando el campo de control 001 del registro seleccionado como principal. Igualmente sucede con los restantes campos de control, que por su importancia se decide tomar los del principal. En cuanto a los conflictos de atributos ocurren cuando existen diversas instancias de un campo no repetible y varias de ellas tienen subcampos no repetibles con distintos valores (ver Figura 15). Debido a lo anterior se debe decidir cuál

de ellas es la correcta. Este conflicto de atributos está influenciado por la entrada manual de datos y el desconocimiento o incorrecta aplicación de las Reglas de Catalogación Angloamericanas (RCA) por parte de los especialistas bibliotecarios (Andreu-Alvarez, 2015). En la Figura 15 se muestra un ejemplo de este último conflicto. En ella se tienen dos fragmentos de registros duplicados que contienen una instancia del campo variable no repetible 245. Cada una de estas instancias incluye el subcampo "a" que no es repetible para este campo y tienen valores distintos. En la integración se debe escoger cuál de estos dos subcampos va a estar presente en la instancia del campo 245 que se añade al registro integrado. En esta figura también se puede observar errores en la entrada de los datos.

Registro 1
245 $aJosé Carlo Mariátegui y el proceso de la literatura en Perú$cAntonio Bermejos Santos
Registro 2
245 $aJosé Carlos Mariátegui y el proceso de la literatura en el Perú$cAntonio Bermejo Santos

Figura 15: Ejemplo de conflicto de atributos en dos registros bibliográficos con formato MARC 21.

El Algoritmo 4 se utiliza en la integración del conjunto de registros con el principal, es decir, en el paso 4. En este algoritmo se incluyen las funciones *mezclarInstanciasNoRepetibles* y *mezclarInstanciasRepetibles*, que se muestran en el Anexo 2, para la mezcla de instancias que pertenecen a campos no repetibles y repetibles respectivamente. Estas funciones utilizan varias reglas para resolver los conflictos anteriores.

Para resolver los conflictos a nivel de esquema y de instancia planteados anteriormente se sigue en el paso 4 (representado mediante el Algoritmo 4) un conjunto de reglas. Es necesario tener en cuenta las siguientes consideraciones antes de aplicar estas reglas:

- El "total de instancias" se refiere a las existentes en el registro principal y el conjunto de registros duplicados como un todo.
- Antes de aplicar estas reglas se eliminan instancias de subcampos con el mismo valor.
- Se eliminan instancias de campos que no contengan ningún subcampo.

```
Algoritmo 4: Integración de registros bibliográficos con formato MARC 21.
Entrada: Registro bibliográfico principal con formato MARC 21 (principal)
Entrada: Lista de registros bibliográficos con formato MARC 21 (lista)
Salida: Registro bibliográfico integrado
función integrar_Registros (principal, lista<Registro>)
1:   integrado ← null
2:   Añadir a integrado la cabecera de principal
3:   Para cada instancia de cada campo de control de principal Hacer
4:     Añadir instancia a integrado
5:   Fin Para
6:   Para cada etiqueta de campo variable de datos del formato Hacer
7:     lista_Instancias<Instancia> ← Obtener instancias con esa
8:     etiqueta que están en integrado
9:     Para cada registro de lista Hacer
10:      Añadir a lista_Instancias instancias con esa etiqueta que
11:      están en registro
12:    Fin Para
13:    Si la etiqueta pertenece a un campo variable de datos no
14:    repetible Entonces
15:      instancia ← mezclarInstanciasNoRepetibles(lista_Instancias)
16:      (ver Anexo 2)
17:      Añadir instancia a integrado
18:    En caso contrario
19:      aux<Instancia>                                          ←
20:      mezclarInstanciasRepetibles(lista_Instancias) (ver Anexo 2)
21:      Para cada instancia de aux Hacer
22:        Añadir instancia a integrado
23:      Fin Para
24:    Fin Si
25:  Fin Para
26:  Retornar integrado
```

Las reglas presentes en el Algoritmo 4 son las siguientes:

- Si el campo es variable de control.

  Regla 1: Se añaden las instancias del registro principal al integrado.

- Si el campo es variable de datos no repetible.

  Regla 2: Si existe una sola instancia del campo se añade al registro integrado.

  Regla 3: Si existe más de una instancia del campo, se mezclan los subcampos y se obtiene solo una, la cual se añade al registro integrado. La mezcla de subcampos se realiza de la siguiente manera:

  Regla 3.1: Si el subcampo no es repetible y existe una sola instancia del mismo, se agrega a la instancia del campo a añadir al registro integrado.

Regla 3.2: Si el subcampo no es repetible y existe más de una instancia de este se debe decidir cuál de ellas se agrega a la instancia del campo a añadir al registro integrado. En este trabajo esta decisión la realiza un experto.

Regla 3.3: Si el subcampo es repetible se agregan todas las instancias de este a la instancia del campo a añadir al registro integrado.

- Si el campo es variable de datos repetible.

Regla 4: Si existe una sola instancia del campo se añade al registro integrado.

Regla 5: Si existe más de una instancia del campo, se mezclan. Las instancias resultantes de la mezcla se añaden al registro integrado. Para efectuar la mezcla primero se buscan instancias de campos que se puedan combinar con las otras que no pueden ser mezcladas. Las instancias que se pueden mezclar solo deben contener instancias de subcampos repetibles y en caso de existir una instancia de un subcampo no repetible, esta debe ser única entre el conjunto de instancias del campo.

Regla 5.1: Si el total de instancias del campo que se pueden mezclar es igual al total de instancias del campo, entonces se forma una sola que contiene todas las instancias de los subcampos presentes en las demás. Esto quiere decir que todas las instancias se pueden mezclar.

Regla 5.2: Si el total de instancias del campo a mezclar es cero entonces todas las instancias del campo se añaden al registro integrado. Ninguna instancia se puede mezclar.

Regla 5.3: Si no ocurre ninguno de los dos casos anteriores entonces se mezclan todas las instancias de los subcampos pertenecientes a las instancias del campo que se pueden mezclar con las que no. Luego se añaden al registro integrado las instancias del campo que no se pueden mezclar con las nuevas instancias de los subcampos añadidas.

El procedimiento para la integración descrita de conjunto con estas reglas pertenecientes al cuarto paso del mismo contribuyen a la mejora de la completitud debido a que la completitud del registro integrado que se obtiene como resultado es mayor o igual que la completitud máxima de los registros que se desean integrar.

2.4 Herramienta para la medición y mejora de la completitud en registros bibliográficos con formato MARC 21

Con el objetivo de materializar lo expuesto anteriormente se diseña una herramienta computacional donde, a partir de una colección de registros de entrada, se realiza la medición y mejora de la completitud de los mismos. La medición se lleva a cabo utilizando la Ecuación 2 y la Ecuación 3 con la configuración inicial descrita en el epígrafe 2.1. Por otra parte, la mejora ocurre a través de la detección de registros duplicados y su posterior integración. En este caso, la presencia del especialista es vital para garantizar la veracidad de las soluciones brindadas. A partir del nombre del formato y la dimensión de calidad completitud se decide nombrar a la herramienta como MARComp.

En el presente epígrafe se describe la herramienta MARComp mediante diagramas de casos de uso, de clases y de componentes. En cada uno de estos diagramas se brinda una explicación de sus elementos.

2.4.1 Diagrama de casos de uso

Los diagramas de casos de uso contienen actores, casos de uso y las relaciones entre estos. Estos diagramas proporcionan una visión general de todo o parte de los requisitos funcionales de un sistema (Ambler, 2005). Además, ayudan en la comunicación entre el cliente, los usuarios y los desarrolladores (Jacobson *et al.*, 2000). Uno de los principales elementos de estos diagramas son los casos de uso, los cuales dirigen el proceso de desarrollo en su totalidad (Jacobson *et al.*, 2000). Actividades como el análisis, diseño y prueba de software se llevan a cabo partiendo de los casos de uso. En la Figura 16 se muestra el diagrama de casos de uso de la herramienta propuesta.

El software presenta un único actor (*Catalogador*) que interactúa con los cinco casos de uso representados en la Figura 16. Este actor es el encargado de importar la colección, medir la completitud de los registros bibliográficos que se encuentran en la misma,

detectar duplicados, mezclarlos y finalmente exportar la colección con los cambios realizados.

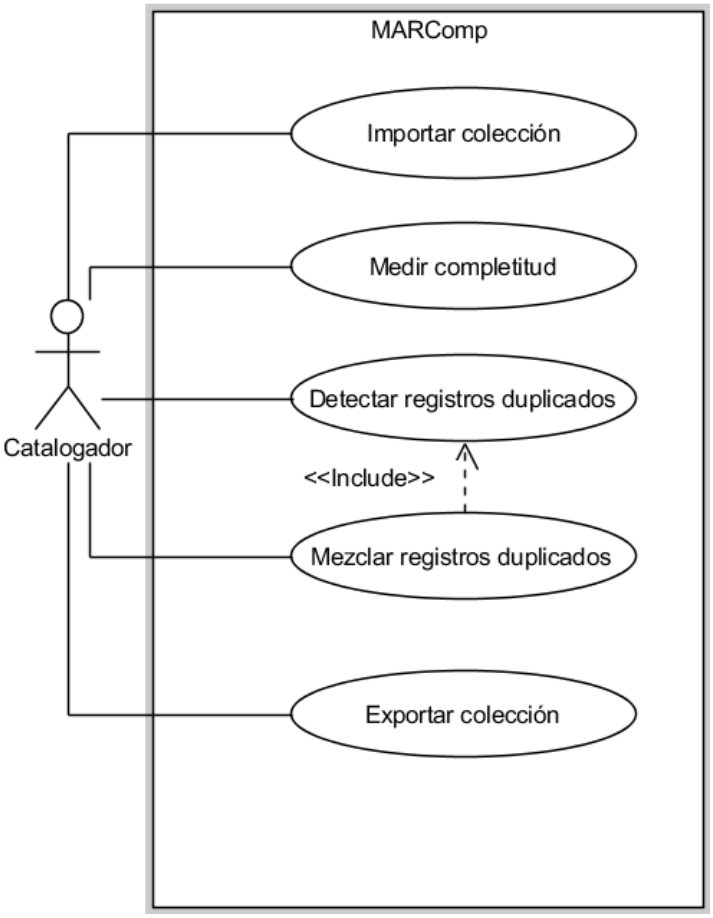


Figura 16: Diagrama de casos de uso de la herramienta MARComp.

A continuación se describen los casos de uso de la herramienta MARComp:

- Importar colección: El caso de uso se inicia cuando el catalogador desea importar una colección de registros bibliográficos con formato MARC 21 con el objetivo de medir y corregir la completitud de los mismos. Esta colección puede tener extensión *.mrc*, *.xml* o *.json* que se corresponden con la representación tradicional, en XML y en JSON del formato MARC 21 respectivamente.
- Medir completitud: El caso de uso se inicia en el propio proceso de importación. Cada vez que se importa un registro de la colección se le mide la completitud en base a las diferentes métricas definidas.

- Detectar registros duplicados: El caso de uso se inicia cuando el catalogador desea detectar registros que representan la misma entidad bibliográfica en el mundo real.
- Mezclar registros duplicados: El caso de uso se inicia cuando el catalogador detecta registros que representan la misma entidad bibliográfica en el mundo real y desea mezclarlos. Es necesario realizar primero la detección de registros duplicados.
- Exportar colección: El caso de uso se inicia cuando el catalogador desea exportar una colección de registros bibliográficos con formato MARC 21. Esta colección puede tener extensión *.mrc*, *.xml* o *.json*.

2.4.2 Diagrama de clases

Los diagramas de clases muestran las clases de un sistema con sus operaciones, atributos y sus interrelaciones (Ambler, 2005). Estos diagramas describen la realización de los casos de usos (Jacobson *et al.*, 2000). Además, sirven de abstracción en la implementación del sistema y son utilizados como una entrada fundamental en las actividades de implementación. A continuación se describen varios diagramas de clases que se utilizan en la realización de los casos de uso que conforman la herramienta MARComp.

Uno de los casos de usos fundamentales es la importación de colecciones. Estas colecciones pueden tener miles de registros. Debido a esto y a las operaciones que se deben realizar sobre cada registro en el momento de su importación, se decide buscar alternativas para agilizar el tiempo de ejecución. Las alternativas empleadas en la presente investigación son la utilización de multihilos y la implementación del problema del productor-consumidor (Silberschatz *et al.*, 2005). Estas alternativas se ven reflejadas en la Figura 17.

En este caso se pueden tener uno o varios objetos (hilos) de la clase *ReaderThreads* (productores), uno o varios objetos (hilos) de la clase *WriterThreads* (consumidores) y un objeto de la clase *ContextThread* que es compartido por las instancias anteriores. Además, se utiliza un objeto de la clase *BlockingQueue* (representa la memoria compartida) del paquete *java.util.concurrent* de la biblioteca estándar de Java. Esta clase permite

implementar el problema del productor-consumidor con memoria compartida limitada. Las clases *ReaderThreads* y *WriterThreads* tienen una referencia a este último objeto.

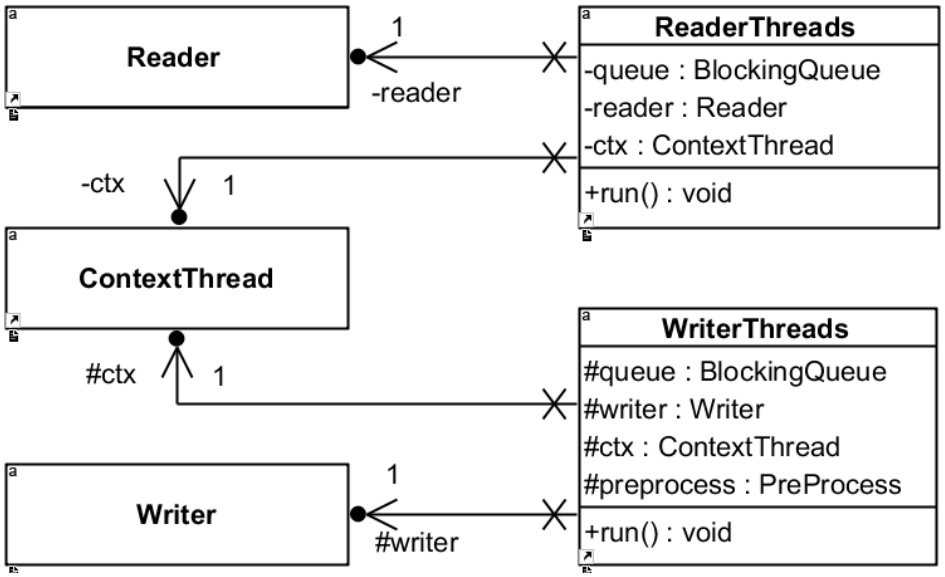


Figura 17: Diagrama de clases para importar colecciones.

Estos objetos funcionan de la siguiente manera:

- Los objetos (hilos) de la clase *ReaderThreads* colocan en la cola (*BlockingQueue*) los registros que leen mediante la clase *Reader*. Si la cola está llena, se bloquean todos los objetos hasta que exista espacio en la cola para seguir colocando. Esto se logra con el método *put*. Cuando no existen más registros a importar, se le notifica a los objetos de la clase *WriterThreads* que se terminó de colocar elementos en la cola mediante el objeto de la clase *ContextThread*.
- Los objetos (hilos) de la clase *WriterThreads* toman los registros de la cola (*BlockingQueue*) mientras existan elementos. Estos registros son preprocesados, se le calcula la completitud y se guarda en la base de datos seleccionada utilizando la clase *Writer*. Estos objetos terminan su ejecución cuando los objetos que colocan los registros en la cola notifican que terminaron mediante *ContextThread* y la cola esté vacía.

En la Figura 18 se muestra un diagrama de clases que se utiliza en la realización del caso de uso Medir completitud. En el diseño de este diagrama se tiene en cuenta que la herramienta se pueda extender a otros formatos de metadatos. Para resolver esta necesidad

se aplica el patrón estructural *Bridge.* Este patrón coloca en dos jerarquías diferentes la abstracción y la implementación (Gamma *et al.*, 1993; Freeman *et al.*, 2004), las cuales pueden variar independientemente. Además, es útil cuando se quiere hacer frente a una proliferación de clases tal que el uso de la herencia no es recomendable por su inflexibilidad y cuando se quiere extender tanto abstracción como implementación por herencia. Los usos anteriores permiten que la extensibilidad sea una ventaja que ofrece este patrón.

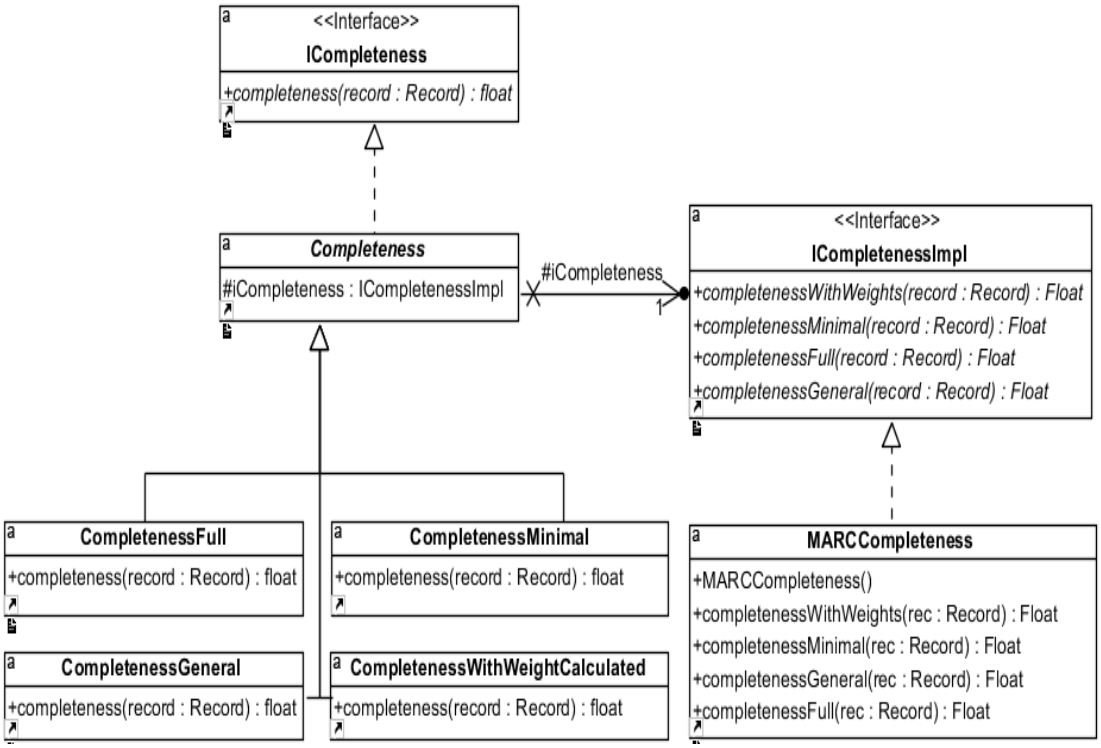


Figura 18: Diagrama de clases asociado a la medición de la completitud.

En esta figura (Figura 18) se tienen las dos jerarquías del patrón *Bridge*, la de abstracción (*Completeness*) y la de implementación (*ICompletenessImpl).* En la jerarquía de implementación se tiene la clase *MARCCompleteness* que implementa las métricas para la medición de la completitud en registros con formato MARC 21. En caso de adicionar un nuevo formato de metadatos a la herramienta basta con implementar la interfaz *ICompletenessImpl* manteniéndose la abstracción sin cambios.

En la jerarquía de abstracción se tiene la clase *Completeness* que implementa la interfaz *ICompleteness* y tiene como atributo un objeto declarado con el tipo de la interfaz

ICompletenessImpl nombrado *iCompleteness*. Mediante el objeto anterior se accede a la jerarquía de implementación. Además, en la abstracción existen cuatro clases que representan los cuatro tipos de completitud que se miden. En cada caso se invoca en el método *completeness* al método de la implementación que se corresponde con el tipo de completitud a medir. Por ejemplo, en el método *completeness* de la clase *CompletenessMinimal*, se invoca el método *completenessMinimal* de la jerarquía de implementación mediante el atributo *iCompleteness*.

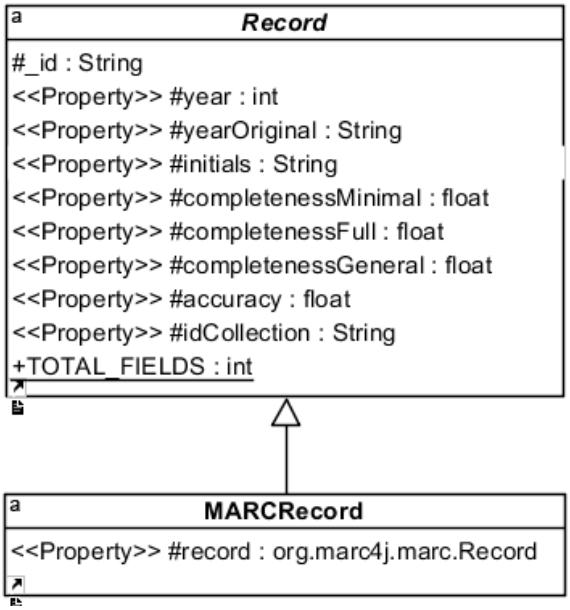


Figura 19: Diagrama de clases para los diferentes formatos de registros.

Por otra parte, se crea una jerarquía para los distintos formatos de metadatos que pueden presentar los registros (ver Figura 19). Hasta el momento el único formato incorporado a la herramienta es MARC 21. Por la complejidad de este formato se decide utilizar la biblioteca MARC4J[16] que ofrece varias clases para la representación de dicho formato. Debido a que estas clases no contienen todos los elementos necesarios para la representación de un registro en la herramienta MARComp se emplea el patrón *Adapter*

[16] http://svn.k-int.com/default/components/marc4j/tutorial.html

(Gamma *et al.*, 1993; Freeman *et al.*, 2004). El uso de este patrón se evidencia en la clase *MARCRecord*, que representa al formato MARC 21 en la herramienta MARComp. Dicha clase tiene un atributo nombrado *record* declarado con el tipo de la interfaz *Record* existente en la biblioteca MARC4J. Además, implementa esta propia interfaz que contiene los métodos necesarios para operar con este formato. De esa manera cada vez que se invoca un método de la clase *MARCRecord* que está relacionado con la interfaz *Record* de MARC4J, realmente lo que se hace es invocar al método de igual nombre mediante el objeto *record*.

2.4.3 Diagrama de componentes

Los diagramas de componentes muestran las dependencias entre componentes del software (Ambler, 2005). Estos diagramas se crean con el objetivo de modelar la configuración del diseño de bajo nivel del sistema, la infraestructura técnica y la arquitectura negocio/dominio.

A continuación se describen los componentes que se muestran en la Figura 20.

- MARC4J[16]: Es una biblioteca de fácil uso para el trabajo con registros MARC 21 en Java. Esta biblioteca contiene varias clases para la lectura y escritura de registros con este formato de metadatos. Además permite la lectura y escritura de estos registros representados en XML, JSON o mrc. Por otra parte, ofrece un modelo para la edición en memoria de registros MARC 21.
- MapDB[17]: Es un motor de base de datos embebido escrito en Java y de código abierto bajo la licencia Apache 2.0. Este motor provee mapas, conjuntos, listas, colas, almacenamiento fuera de la pila y en disco, entre otras características. Además, es una de las bases de datos más rápidas de Java con un rendimiento comparable a las colecciones del paquete *java.util* de la librería estándar de Java.

[17] http://www.mapdb.org/

Este motor de base de datos se utiliza para almacenar toda la información que maneja la herramienta MARComp.

- SimMetrics[18]: Es una biblioteca de código abierto escrita en Java que contiene funciones para calcular la distancia entre cadenas de texto (Pei, 2008). Entre estas funciones se encuentran Levenshtein, Smith-Waterman, Jaro, Jaro-Winkler, Monge-Elkan, entre otras. Estas funciones devuelven la similitud de dos cadenas dadas. La similitud puede obtenerse normalizada o no. En este trabajo se utiliza la primera variante donde un uno significa que son similares y un cero lo contrario.
- Language Detector[19]: Es una biblioteca escrita en Java bajo la licencia de Apache que detecta el idioma de un texto dado. Se utiliza para detectar el idioma de los campos autor y título y dependencia de éste se eliminan las palabras vacías. Detectan 71 lenguajes, aunque en este trabajo solo se utilizan español, inglés, francés, italiano y portugués.
- JFreeChart[20]: Es una biblioteca gratuita escrita en Java que permite mostrar gráficos de calidad de una manera sencilla. Esta biblioteca incluye un diseño flexible y fácil de extender y una API (acrónimo del inglés *Application Programming Interface*) consistente y bien documentada. Además, soporta varios tipos de salida, incluyendo componentes Swing y JavaFX, archivos de imagen (PNG y JPEG) y formatos de archivo de gráficos vectoriales (PDF, EPS y SVG). En MARComp se utiliza esta biblioteca para brindar información sobre las colecciones de datos.

[18] https://sourceforge.net/projects/simmetrics/
[19] https://github.com/optimaize/language-detector
[20] http://www.jfree.org/jfreechart/

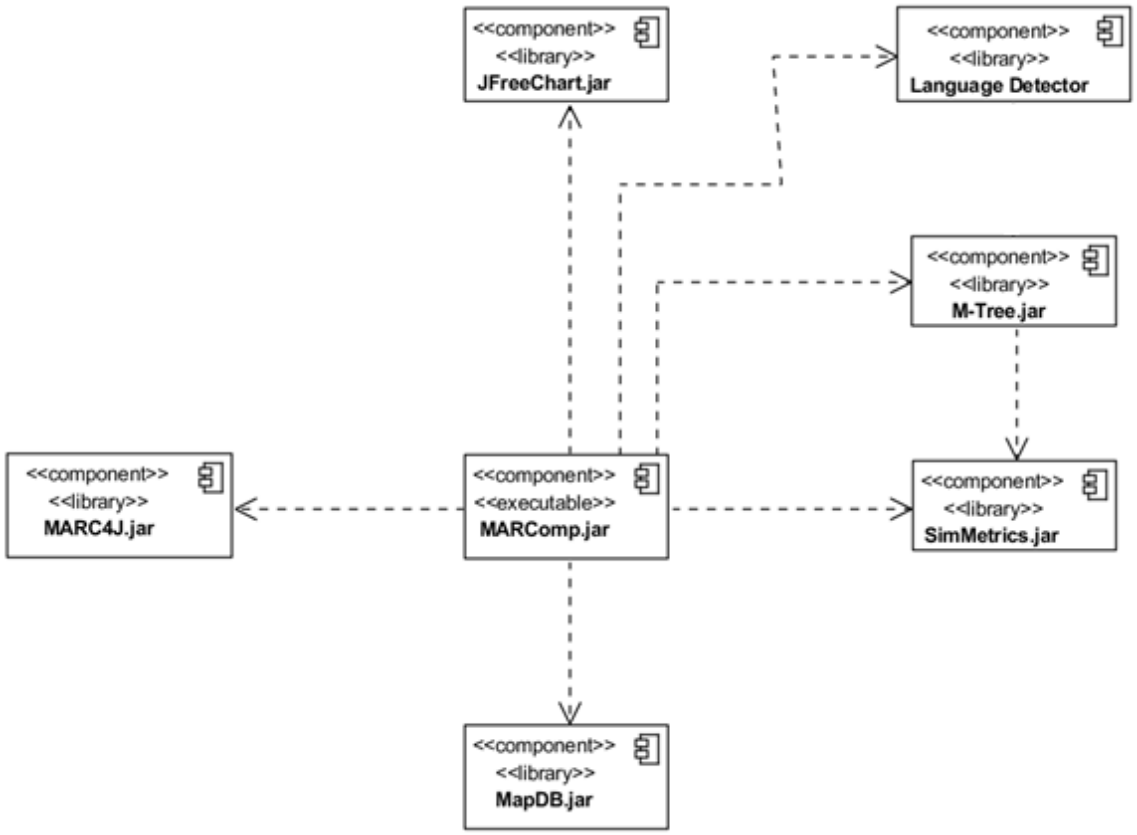


Figura 20: Diagrama de componentes de la herramienta MARComp.

2.5 Conclusiones parciales

En este capítulo se identificaron las características de las métricas propuestas por Ochoa y Duval que varían para el formato MARC 21, permitiendo de esta manera realizar la medición de la completitud en diferentes niveles de catalogación y utilizando diferentes grados de importancia para cada campo. En el caso de estos últimos se definió una variante para su cálculo. Se propusieron dos algoritmos para la medición de esta dimensión de calidad utilizando *MapReduce* que permiten procesar grandes volúmenes de registros. Además, se modificó un algoritmo propuesto en la literatura para la detección de registros duplicados. Estas modificaciones fueron en la etapa de indexado y de clasificación. En la etapa de indexado se cambió la llave de bloqueo "año" por "clase de material-año", lo que permite una mayor reducción del espacio de búsqueda. También, se varió la forma de almacenar las estructuras de datos asociadas a esta fase. Por otra parte, se establecieron un conjunto de reglas en la etapa de clasificación para determinar si un par de registros

son posibles duplicados o no. Asimismo, se elaboró un procedimiento para la mejora de la completitud en registros bibliográficos con formato MARC 21 que incluye un algoritmo para la mezcla de registros duplicados con este formato MARC 21 y presenta un conjunto de reglas para eliminar los conflictos a nivel de esquema y de instancia. Por último, se diseñaron diagramas de clases asociados a varios casos de uso que se utilizaron en la construcción de la herramienta MARComp y se emplearon patrones de diseño que garantizan que la misma sea extensible y resistente al cambio. Además, se explicaron las partes de los diagramas de clases que se deben modificar en caso de incorporarle un nuevo formato de metadatos.

Capítulo 3 Evaluación de la medición y mejora de la completitud.

En el presente capítulo se realizan pruebas unitarias a los casos de uso importar colección y medir completitud de la herramienta MARComp utilizando dos bases de datos de la biblioteca MARC4J. Se mide la completitud con esta herramienta en dos bases de datos reales que pertenecen a los catálogos de dos universidades cubanas que contienen registros bibliográficos con formato MARC 21. Además, se evalúa el algoritmo propuesto en el capítulo anterior para la detección de registros duplicados empleando seis bases de datos sintéticas. También, se expone un caso de prueba para la integración de registros con este formato. Por último, se realiza una comparación entre la herramienta MarcXimil y la que se propone MARComp.

3.1 Bases de datos para evaluar la herramienta MARComp

Con el objetivo de evaluar los resultados de la herramienta MARComp se utilizan 10 bases de datos que contienen registros bibliográficos con formato MARC 21 representados en formato XML o en su forma tradicional (extensión *.mrc*). De estas bases de datos dos pertenecen a catálogos de dos universidades cubanas.

3.1.1 Bases de datos de entidades reales

En la medición de la completitud se utilizan dos bases de datos que pertenecen a entidades reales. Estas bases de datos pertenecen a los catálogos de la Universidad de Ciencias Informáticas (UCI) y la Universidad Central “Marta Abreu” de Las Villas (UCLV) y contienen 13807 y 18745 registros respectivamente. En lo adelante se denominan como BD_UCI y BD_UCLV.

En la Tabla 12 se muestra la cantidad de registros por clase de material en cada base de datos que brinda la herramienta MARComp. Es necesario destacar que la clase de material más representativa es libro. En BD_UCLV esta clase de material representa aproximadamente el 99.5% del total de registros catalogados mientras que en BD_UCI representa el 100%.

Tabla 12: Cantidad de registros bibliográficos por clase de material en cada base de datos.

CM	Bases de datos		Total
	BD_UCI	BD_UCLV	
L	18646	13807	32453
A	44	0	44
P	7	0	7
M	8	0	8
Ps	2	0	2
Mm	0	0	0
Gs	1	0	1
Mv	1	0	1
D	36	0	36
Total	18745	13807	32552

CM: Clase de material **L:** Libro. **A:** Archivo de computadora. **P:** Partitura. **M:** Mapa. **Ps:** Publicación seriada. **Mm:** Material mixto. **Gs:** Grabación sonora. **Mv**: Material visual. **D:** Clase desconocida.

3.1.2 Bases de datos sintéticas

Según Christen (2012b), los conjuntos de datos sintéticos se deben crear cuando los conjuntos disponibles presentan limitaciones en cuanto al contenido o a la representación del problema que se quiere evaluar.

En este trabajo se crean seis bases de datos sintéticas para evaluar la detección de posibles registros duplicados con formato MARC 21 utilizando el algoritmo expuesto en el epígrafe 2.2. Estas bases de datos se crean a partir de la selección aleatoria de registros de la base BD_UCLV de manera que un registro solo pertenezca una colección. Cada uno de estos conjuntos tiene como total de registros 500 (de Carvalho *et al.*, 2012) o 2000 (Borel y Krause, 2009). La cantidad de duplicados que se insertan son 150 cuando el total de la colección es 500 y 600 cuando el total es 2000, es decir, el 30% del total en ambos casos. Este porciento se considera representativo para poblaciones finitas. La cantidad máxima de duplicados por cada registro es uno.

A cada base de datos sintética se le nombra BD_S y seguidamente el número según el orden en que se crearon, por ejemplo BD_S1 corresponde a la base de datos sintética 1 y así sucesivamente. Las características de cada base de datos se describen a continuación:

- Bases de datos 1 (BD_S1): Presenta un total de 500 registros, de ellos 150 son duplicados. Estos registros duplicados se escogen al azar (muestreo aleatorio simple) y la modificación que se les realiza es la eliminación de una palabra del

título o del autor. Esta palabra se escoge de manera aleatoria y puede ser una palabra vacía.

- Bases de datos 2 (BD_S2): Presenta un total de 2000 registros, de ellos 600 son duplicados. Estos registros duplicados se escogen al azar y la modificación que se les realiza es la eliminación de una palabra del título o del autor. Esta palabra se escoge de manera aleatoria y puede ser una palabra vacía.
- Bases de datos 3 (BD_S3): Presenta un total de 500 registros, de ellos 150 son duplicados. Estos registros duplicados se escogen al azar y las modificaciones que se les realizan es la eliminación de una palabra del título y una del autor. Estas palabras se escogen de manera aleatoria y pueden ser palabras vacías.
- Bases de datos 4 (BD_S4): Presenta un total de 2000 registros, de ellos 600 son duplicados. Estos registros duplicados se escogen al azar y las modificaciones que se les realizan es la eliminación de una palabra del título y una del autor. Estas palabras se escogen de manera aleatoria y pueden ser palabras vacías.
- Bases de datos 5 (BD_S5): Presenta un total de 500 registros, de ellos 150 son duplicados. Estos registros duplicados se escogen al azar y las modificaciones que se les realizan es la eliminación de dos palabras del título y una del autor. Estas palabras se escogen de manera aleatoria y pueden ser palabras vacías.
- Bases de datos 6 (BD_S6): Presenta un total de 2000 registros, de ellos 600 son duplicados. Estos registros duplicados se escogen al azar y las modificaciones que se les realizan es la eliminación de dos palabras del título y una del autor. Estas palabras se escogen de manera aleatoria y pueden ser palabras vacías.

La cantidad de palabras a eliminar varía en cada caso para comprobar la resistencia al ruido del algoritmo durante la detección de posibles duplicados. Cada palabra que se elimina del autor o del título es una posible inicial que no aparece en la cadena que se inserta en la estructura M-Tree. Esto significa que mientras aumenta la cantidad de palabras que se eliminan debe disminuir la información recuperada correctamente de esta estructura. Se escoge como máximo el valor de tres debido a que si se eliminan más palabras se introduce demasiado ruido y esto no suele suceder en los casos reales. Además,

pueden existir registros donde el título y el autor presenten solamente una o dos palabras y no tiene sentido eliminarlas.

Además de estas seis bases de datos sintéticas, se utilizan otras dos que vienen junto con la biblioteca MARC4J y que a su vez se utilizan en pruebas realizadas a esta última. Estas bases de datos contienen dos registros. Estos registros son los mismos en cada base de datos variando solo su representación. En una están representados en formato XML (*chabon.xml*[21]) y en la otra en su forma tradicional (*chabon.mrc*[22]). En este trabajo las referidas bases de datos se nombran como BD_CHABON_XML y BD_CHABON_MRC respectivamente.

3.2 Pruebas unitarias

Según Gómez et al. (2013), la fase de prueba es importante en el desarrollo de un software debido a que en la misma se comprueba si éste satisface el conjunto de requisitos de los usuarios y clientes. Con el objetivo de comprobar el correcto funcionamiento de la herramienta MARComp, resulta necesario realizar un conjunto de pruebas unitarias como un primer paso de chequeo.

Las pruebas unitarias se diseñan para comprobar el correcto funcionamiento de pequeñas partes de código (Pressman, 2010). En el presente trabajo se diseñan pruebas unitarias para chequear la importación de registros bibliográficos con formato MARC 21 utilizando la base de datos BD_CHABON_XML. Además, se crean pruebas para la medición de la completitud empleando BD_CHABON_MRC.

3.2.1 Importación de registros bibliográficos

Este módulo es el encargado de la importación de registros bibliográficos con formato MARC 21. Para ello recibe ficheros con la extensión *.mrc*, *.xml* o *.json* que contienen registros representados en su forma tradicional, XML o JSON respectivamente. En este

[21] https://github.com/marc4j/marc4j/blob/master/test/resources/chabon.xml
[22] https://github.com/marc4j/marc4j/blob/master/test/resources/chabon.mrc

módulo se utiliza para la lectura de los registros la biblioteca MARC4J, la cual permite manipular estos ficheros. A pesar de que a dicha biblioteca se le realizan pruebas de lectura de registros, en este trabajo se decide realizar pruebas unitarias debido a que en este módulo, además de la lectura, se realiza la inserción en la base de datos embebida MapDB. Además, es necesario comprobar el funcionamiento de las clases creadas para estas funciones, para lo cual se definen dos casos de prueba.

```
@Test
public void importTestSuccess() {
    try {
        DataBaseConnection connection = new MapDBConnection("dbTest" + File.separator + "test.db");
        connection.connect();
        CollectionOperations collOp = new CollectionOperations((DataBaseConnection) connection);
        Collection collection = new Collection("test", "Colección de prueba", "Probando");
        collOp.insert(collection);
        RecordOperations recOp = RecordOperations.getRecordOperations(connection, collection, null);
        //BD_CHABON_XML = "chabon.xml";
        FileConnection connFile = new FileConnection(new File(URL_BASE + BD_CHABON_XML));
        RecordFormat rFormat = new RecordFormat(RecordFormat.MARC21_TYPE, RecordFormat.MARC21_NAME);
        Reader reader = Reader.createReader(connFile, rFormat);
        while (reader.hasNext()) {
            Record record = reader.next();
            record.setIdCollection(collection.getId_collection());
            recOp.insert(record);
        }
        collOp.update(collection);
        int total = (int) collOp.getByID("test").getTotalRecords();
        assertEquals(2, total);
    } catch (Exception ex) {
        ex.printStackTrace();
        fail();
    }
}
```

Figura 21: Método para probar el primer caso de la importación.

En el primer caso se selecciona BD_CHABON_XML. Luego, se leen los registros que contiene este fichero y se insertan en la base de datos embebida MapDB haciéndole corresponder una colección determinada. Por último, se comprueba que el total de registros almacenados en esta base de datos sean dos. El método que realiza la prueba se muestra en la Figura 21.

```
@Test
public void importTestFailed() {
    try {
        //BD_CHABON_ERROR = "chabon";
        FileConnection connFile = new FileConnection(new File(URL_BASE + BD_CHABON_ERROR));
        connFile.connect();
        RecordFormat rFormat = new RecordFormat(RecordFormat.MARC21_TYPE, RecordFormat.MARC21_NAME);
        Reader reader = Reader.createReader(connFile, rFormat);
        reader.open();
        fail("Error si no lanza excepción. Extensión inválida");
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

Figura 22: Método para probar el segundo caso de la importación.

En el segundo caso se selecciona un fichero sin extensión y se comprueba que se lance una excepción (ver Figura 22).

Como se puede apreciar en la Figura 23 el resultado fue satisfactorio en ambos casos. En el primer caso se importa un total de dos registros y en el segundo se lanza una excepción. Esto significa que las clases creadas para la importación de registros y su posterior inserción funcionan correctamente.

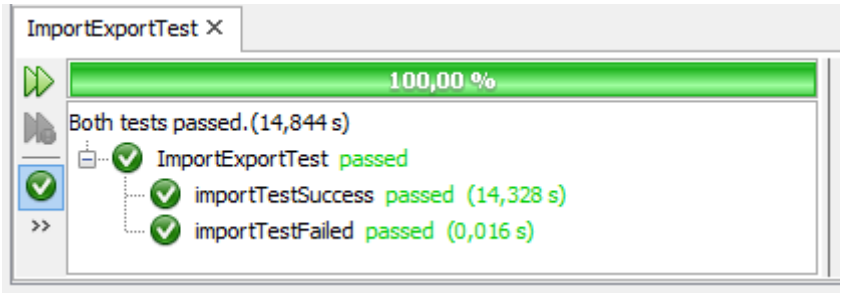


Figura 23: Resultado de las pruebas al módulo de importación.

3.2.2 Medición de la completitud

Para comprobar el correcto funcionamiento de la medición de la completitud se crean cuatro casos de prueba. Los primeros tres casos se refieren al cálculo de la completitud cuando todos los campos tienen igual grado de importancia. Se crea un caso para cada nivel de completitud y se utiliza BD_CHABON_MRC como base de datos. En la Figura 24 se muestra el método que realiza la comprobación para el nivel mínimo que se corresponde con el primer caso. El segundo y tercer caso son similares y se muestran en el Anexo 3.

```
@Test
public void minCompletenessTest() throws ConnectionNotEstablished { //BD_CHABON_MRC = "chabon.mrc"
    FileConnection connFile = new FileConnection(new File(URL_BASE + BD_CHABON_MRC));
    RecordFormat rFormat = new RecordFormat(RecordFormat.MARC21_TYPE, RecordFormat.MARC21_NAME);
    Reader reader = Reader.createReader(connFile, rFormat);
    while (reader.hasNext()) {
        Record record = reader.next();
        assertNotNull(record);
        Completeness comp = Completeness.createCompleteness(Metric.COMPLETENESS_MINIMAL, record.ge
        Float value = comp.completeness(record);
        Float expValue = 7 / 8.0f;
        assertEquals(expValue, value);
    }
}
```

Figura 24: Método para probar el cálculo de la completitud a nivel mínimo.

Por otra parte, el cuarto caso de prueba se refiere al valor de la completitud cuando el grado de importancia de cada campo es la incidencia del mismo dentro de su clase de

material. En este caso también se utiliza BD_CHABON_MRC debido a que los dos registros pertenecen a la misma clase y contienen dos campos diferentes cada uno. En la Tabla 13 se muestran los campos y la completitud que tiene cada registro perteneciente a esta base de datos.

Tabla 13: Completitud de los registros de BD_CHABON_MRC con diferentes grados de importancia para los campos.

Campo	Registro 1	Registro2	Incidencia
001	X	X	2
005	X	X	2
008	X	X	2
020	X	X	2
040	X	X	0
100	X	X	2
245	X	X	2
250		X	1
260	X	X	2
300	X	X	2
520		X	1
650	X	X	2
651	X		1
655	X		1
Total	(9*2+2*1+1*0)=20	(9*2+2*1+1*0)=20	22
Completitud	20/22	20/22	

En la Figura 25 se muestra el método que realiza la comprobación del cuarto caso. En este método se llama a la función *_import* que importa los registros de BD_CHABON_MRC y devuelve una conexión a la base de datos. Una vez que los registros han sido insertados se calcula la completitud.

```
@Test
public void weightedCompletenessTest() throws ConnectionNotEstablished, Exception {
    DataBaseConnection connection = _import();
    RecordFormat rFormat = new RecordFormat(RecordFormat.MARC21_TYPE, RecordFormat.MARC21_NAME);
    Completeness comp = Completeness.createCompleteness(Metric.COMPLETENESS_WITH_WEIGHT_CALCULATED, rFormat, connection);
    Collection collection = new Collection("test", "Colección de prueba", "Probando completitud con pesos calculados");
    RecordOperations recOp = RecordOperations.getRecordOperations(connection, collection,null);
    Iterator<Record> it = recOp.iterator();
    while (it.hasNext()) {
        Float expValue = 20.0f/22.0f;
        Float value = comp.completeness(it.next());
        assertEquals(expValue, value);
    }
}
```

Figura 25: Método para probar el cálculo de la completitud con diferentes grados de importancia para cada campo.

En la Figura 26 se aprecia que el resultado esperado coincidió con el calculado en los dos casos expuestos anteriormente y en los dos del Anexo 3.

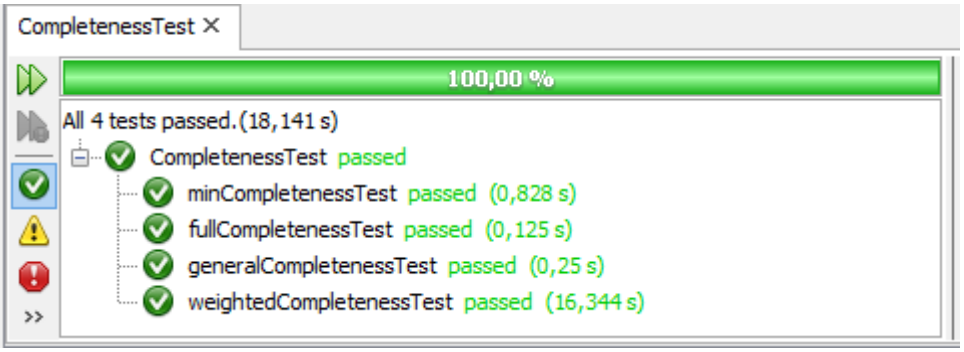


Figura 26: Resultado de las pruebas para el cálculo de la completitud.

3.3 Medición de la completitud en registros bibliográficos con formato MARC 21

En cada una de las bases de datos del epígrafe 3.1.1 se mide la completitud utilizando la Ecuación 2 para los tres niveles descritos en el epígrafe 2.1.2. Además de la ecuación anterior, también se utiliza la Ecuación 3 tomando como grado de importancia de cada campo la incidencia del mismo en cada clase de material dentro de una misma base de datos (ver epígrafe 2.1.3).

Tabla 14: Resultados de la medición de la completitud en BD_UCLV.

		L	A	P	M	Ps	Gs	Mv	Mm
CNM	Min	0.1250	0.3333	0.5455	0.3571	0.3000	0.5000	0.5000	-
	Max	0.7500	0.4444	0.5455	0.4286	0.4000	0.5000	0.5000	-
	Avg	0.5984	0.4369	0.5455	0.4196	0.3500	0.5000	0.5000	-
CNC	Min	0.0833	0.2000	0.5000	0.3333	0.2000	0.4286	0.3333	-
	Max	0.8333	0.3333	0.5000	0.4444	0.3000	0.4286	0.3333	-
	Avg	0.5633	0.2667	0.5000	0.3889	0.2500	0.4286	0.3333	-
CNG	Min	0.0037	0.0221	0.0257	0.0331	0.0221	0.0331	0.0184	-
	Max	0.0515	0.0294	0.0404	0.0441	0.0257	0.0331	0.0184	-
	Avg	0.0326	0.0261	0.0320	0.0368	0.0239	0.0331	0.0184	-
CGI	Min	0.1128	0.8462	0.8033	0.7750	0.8462	1.0000	1.0000	-
	Max	0.9713	0.9840	1.0000	1.0000	0.9231	1.0000	1.0000	-
	Avg	0.8351	0.9693	0.9063	0.8750	0.8846	1.0000	1.0000	-

L: Libros. **A:** Archivo de computadora. **P:** Partitura. **M:** Mapa. **Ps:** Publicación seriada. **Gs:** Grabación sonora. **Mv**: Material visual. **Mm**: Material mixto. **CNM:** Completitud nivel mínimo. **CNC:** Completitud nivel completo. **CNG:** Completitud nivel general. **CGI**: Completitud con grados de importancia **Min**: Valor mínimo. **Max**: Valor máximo. **Avg**: Promedio.

La Tabla 14 muestra los resultados de la medición de la completitud para la base de datos BD_UCLV. En esta se aprecia que cuando todos los campos tienen el mismo grado de importancia los mayores valores de completitud están a un nivel mínimo. Esto se debe fundamentalmente a que es menor la cantidad de campos totales que utiliza, por tanto el cociente debe dar un número mayor que en los restantes niveles. Sin embargo, a pesar de no tener definidos umbrales, se considera que los valores de completitud son bajos en

todos los niveles. Por ejemplo, en el nivel mínimo los campos totales para la clase libro son ocho y la media en BD_UCLV es 0.5984, lo que significa que aparecen solamente cinco campos completos de este nivel como promedio. Esta cantidad resulta baja teniendo en cuenta que se trata del nivel mínimo. En el caso de la utilización de diferentes grados de importancia para cada campo los valores resultan superiores, pero presentan como desventaja que no brindan información sobre la completitud de cada nivel. En este trabajo se recomienda como buena práctica analizar todos los valores de completitud debido a que cada uno ofrece una información diferente.

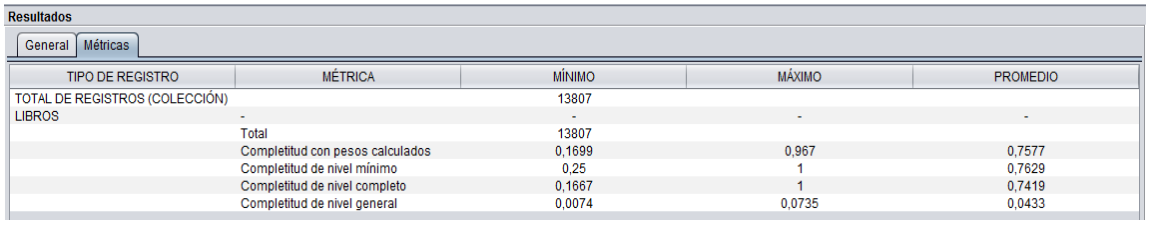
Tabla 15: Resultados de la medición de la completitud en BD_UCI.

		L	A	P	M	Ps	Gs	Mv	Mm
CNM	Min	0.2500	-	-	-	-	-	-	-
	Max	1.0000	-	-	-	-	-	-	-
	Avg	0.7629	-	-	-	-	-	-	-
CNC	Min	0.1667	-	-	-	-	-	-	-
	Max	1.0000	-	-	-	-	-	-	-
	Avg	0.7419	-	-	-	-	-	-	-
CNG	Min	0.0074	-	-	-	-	-	-	-
	Max	0.0735	-	-	-	-	-	-	-
	Avg	0.0433	-	-	-	-	-	-	-
CGI	Min	0.1699	-	-	-	-	-	-	-
	Max	0.9670	-	-	-	-	-	-	-
	Avg	0.7577	-	-	-	-	-	-	-

L: Libros. **A:** Archivo de computadora. **P:** Partitura. **M:** Mapa. **Ps:** Publicación seriada. **Gs:** Grabación sonora. **Mv**: Material visual. **Mm**: Material mixto. **CNM:** Completitud nivel mínimo. **CNC:** Completitud nivel completo. **CNG:** Completitud nivel general. **CGI**: Completitud con grados de importancia **Min**: Valor mínimo. **Max**: Valor máximo. **Avg**: Promedio.

Los resultados de la medición en BD_UCI se muestran en la Tabla 15. Al igual que en BD_UCLV los mejores resultados se obtienen en el nivel mínimo aunque la diferencia con respecto al completo con respecto al promedio es pequeña. En cuanto al uso de diferentes grados de importancia hay que destacar que el promedio es muy similar al de los niveles mínimo y completo lo que indica que los campos que se encuentran en estos niveles son los que más se utilizan por lo que se aprecia homogeneidad en la catalogación.

La herramienta MARComp facilita los resultados presentados en la Tabla 14 y la Tabla 15. En la Figura 27 se aprecian los resultados de la medición de la completitud para la base de datos BD_UCI.


Resultados

General | Métricas

TIPO DE REGISTRO	MÉTRICA	MÍNIMO	MÁXIMO	PROMEDIO
TOTAL DE REGISTROS (COLECCIÓN)		13807		
LIBROS	-	-	-	-
	Total	13807		
	Completitud con pesos calculados	0,1699	0,967	0,7577
	Completitud de nivel mínimo	0,25	1	0,7629
	Completitud de nivel completo	0,1667	1	0,7419
	Completitud de nivel general	0,0074	0,0735	0,0433

Figura 27: Resultados de la medición de la completitud utilizando MARComp.

3.4 Detección de duplicados

En este epígrafe se evalúa el algoritmo de detección de duplicados propuesto en el epígrafe 2.2. Para ello se utilizan las bases de datos sintéticas BD_S1, BD_S2, BD_S3, BD_S4, BD_S5 y BD_S6 expuestas en el epígrafe 3.1.2. Además, se utiliza solo la consulta de rango para obtener los elementos del M-Tree. El valor que se utiliza para esta consulta es 0.4, el cual coincide con el empleado en (Veloso de Melo y de Andrade Lopes, 2005).

La Tabla 16 muestra los valores VP, FP, FN, precisión, sensibilidad y *F-Measure* del algoritmo para cada una de estas bases de datos. Los resultados en cuanto a la precisión son iguales a 1.00 en todas las bases de datos, lo cual garantiza que todos los registros detectados como posibles duplicados realmente los son. En cuanto a la sensibilidad y *F-Measure* los valores para BD_S1 y BD_S2 son mayores o iguales que 0.98 y se consideran muy buenos estos resultados. La disminución mostrada a partir de BD_S3 es propiciada principalmente por la forma de construcción de las bases de datos. En BD_S3 y BD_S4 se eliminan dos posibles iniciales mientras que en BD_S5 y BD_S6 se eliminan tres. Esto afecta la recuperación de información de la estructura M-Tree cuando se tratan registros con un solo autor y títulos de una o dos palabras. Por ejemplo, si a un registro con las iniciales EFSB se le eliminan la F y la B, la distancia normalizada de Levenshtein entre estas dos cadenas es 0.5, la cual es superior al rango de 0.4 y por tanto esos dos registros no se encuentran en el mismo bloque y no se consideran posibles duplicados. Es necesario señalar que, aunque puede suceder, la mayoría de los errores que se presentan en el trabajo con cadenas en las bases de datos de catalogación son tipográficos. Esto propicia una buena recuperación en la consulta de rango y detectar la mayor cantidad posible de duplicados. No obstante, mientras menos sea la cantidad de iniciales resultantes del autor y título en dos registros duplicados y en uno de ellos falten iniciales, la efectividad del algoritmo propuesto disminuye.

Tabla 16: Resultados de precisión, sensibilidad y *F-Measure*.

Base de datos	VP	FP	FN	Precisión	Sensibilidad	F-Measure
BD_S1	147	0	3	1.00	0.98	0.99
BD_S2	588	0	12	1.00	0.98	0.99
BD_S3	139	0	11	1.00	0.93	0.96
BD_S4	534	0	66	1.00	0.89	0.94
BD_S5	133	0	17	1.00	0.89	0.94
BD_S6	534	0	66	1.00	0.89	0.94

En la Figura 28 se muestra la ventana que utiliza la herramienta MARComp para la detección de duplicados. Nótese que en el panel "Datos de los registros" se listan todos los elementos duplicados que se corresponden al registro seis seleccionado en el panel "Registros duplicados".

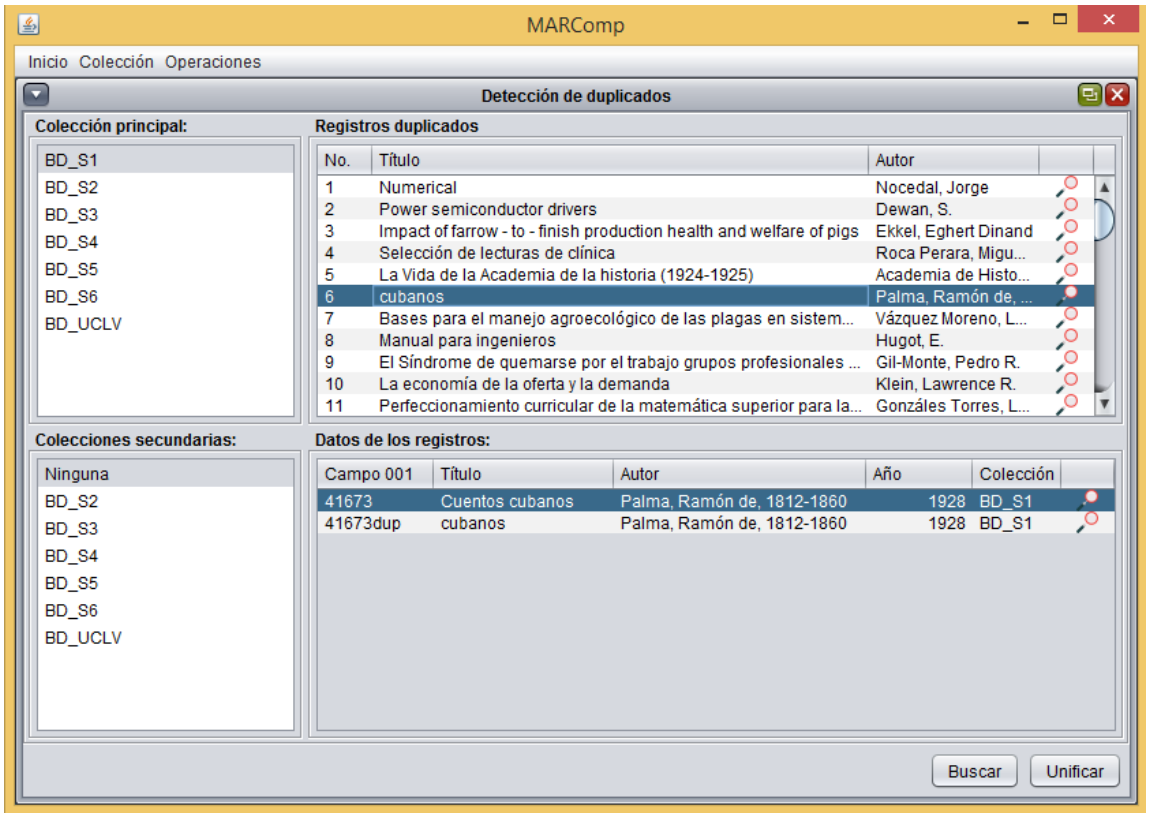


Figura 28: Resultado de la detección de duplicados en BD_S1 utilizando MARComp.

3.5 Evaluación del procedimiento para la mejora de la completitud

Una vez detectados los posibles registros duplicados se procede a la integración de los mismos en un registro unificado cuya completitud siempre es mayor o igual que la completitud máxima de los registros que se desean integrar. Para esto se utilizan los pasos descritos en el epígrafe 2.3.

A continuación, se ilustra el funcionamiento de estos pasos para mezclar registros bibliográficos con formato MARC 21 utilizando dos registros que son posibles duplicados (ver Figura 29). Los datos de estos registros no son reales debido a que se crearon para que representen la mayoría de los casos que se pueden encontrar durante la integración.

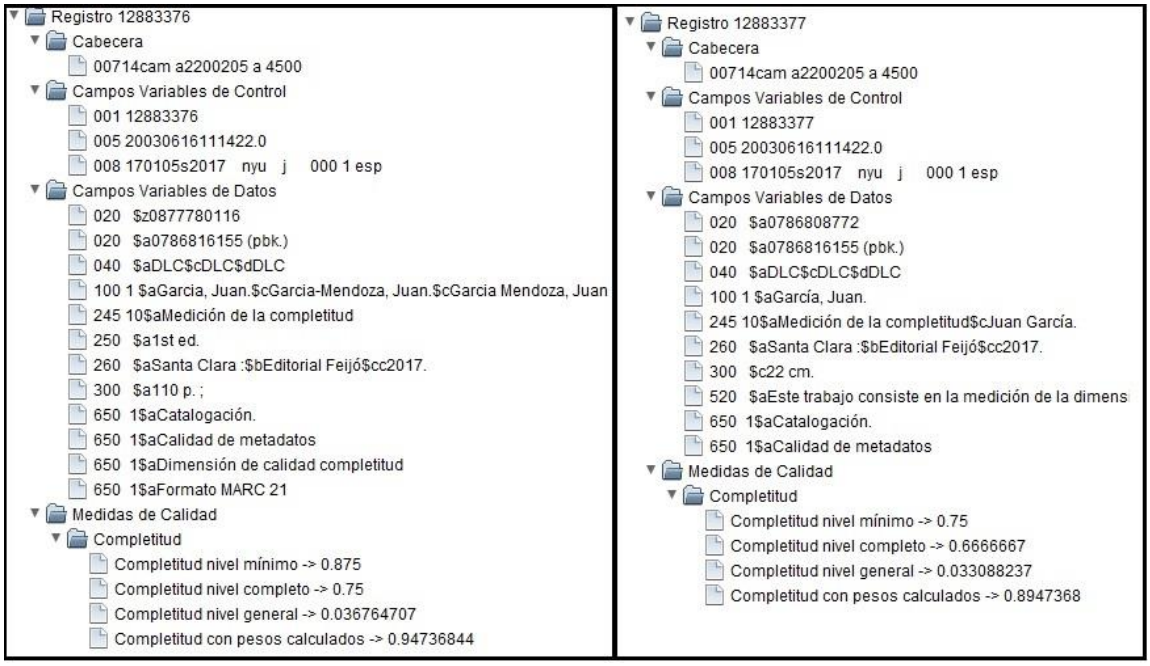


Figura 29: Dos registros que son posibles duplicados.

Paso 1. Detectar posibles duplicados

Este paso consiste en la detección de posibles duplicados. La herramienta MARComp utiliza en este paso la modificación del algoritmo de Veloso de Melo et al. (2005) descrita en el epígrafe 2.2. Como resultado se obtienen los registros de la Figura 29.

Paso 2. Seleccionar los verdaderos duplicados

En este paso se deben seleccionar de los posibles duplicados los que realmente lo son. En este caso se asume que los dos registros de la Figura 29 son duplicados.

Paso 3. Escoger el registro principal

En este paso se debe escoger el registro principal. En caso de que los posibles registros duplicados pertenezcan a la misma colección, se debe elegir el registro que posea mayor completitud respecto a los valores calculados anteriormente. En este caso se selecciona el registro que está a la izquierda en la Figura 29 por presentar mayor completitud en el nivel

mínimo. No obstante, se le brinda al experto la opción de poder cambiar la elección que se le sugiere si así lo desea.

Paso 4. Integración de los registros

En este paso se realiza la integración de los registros utilizando el Algoritmo 4. Este último emplea un conjunto de reglas para resolver los conflictos a nivel de esquema y de instancia. Para ilustrar el uso de estas reglas se escogen varios casos que se pueden presentar al unificar las instancias de los campos de los registros de la Figura 29. Antes de aplicar estas reglas se deben tener en cuenta las consideraciones expuestas en el epígrafe 2.3.

Caso 1: Si el campo es variable de control.

Regla 1: En este caso se encuentran las instancias de los campos variables de control 001, 005 y 008 que pertenecen al registro principal que se encuentra a la izquierda en la Figura 29. Estas instancias son añadidas al registro integrado. Con la instancia del campo 001 se resuelven los conflictos de llave.

En el caso de los campos variables en la Figura 30 se muestran las instancias a mezclar.

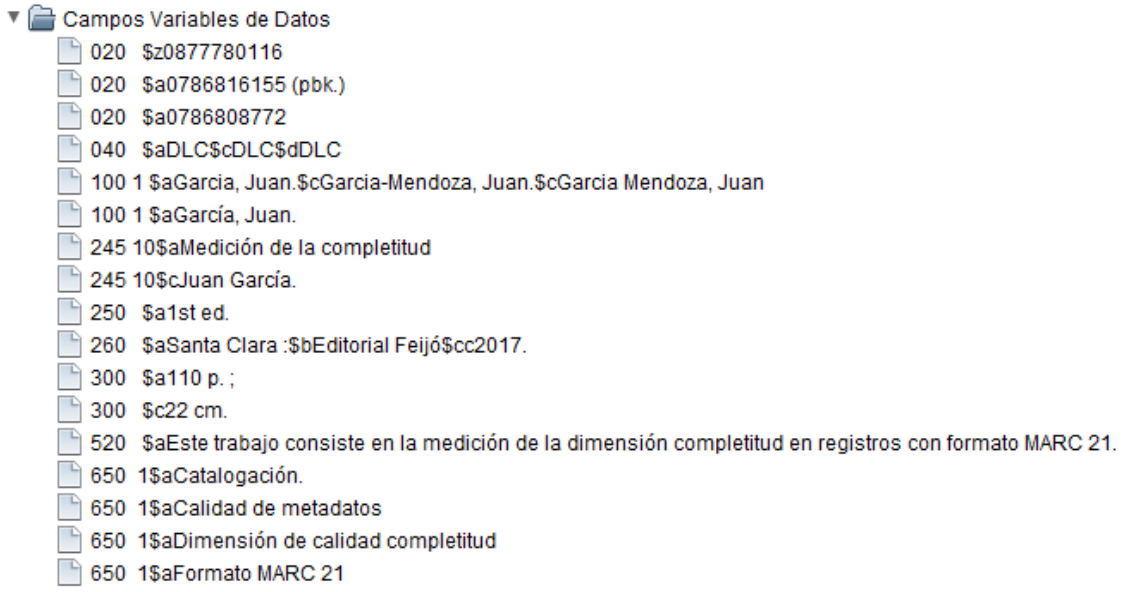


Figura 30: Conjunto de instancias a mezclar.

Caso 2: Si el campo es variable de datos no repetible.

Regla 2: En este caso se tiene la instancia del campo 040, la cual se añade al registro integrado.

Regla 3: Si existe más de una instancia del campo, se mezclan los subcampos y se obtiene solo una, la cual se añade al registro integrado. La mezcla de subcampos se realiza de la siguiente manera: En este caso se tienen los campos 100 y 245 que contienen más de una instancia.

Regla 3.1: En este caso se encuentra la instancia del subcampo "c" presente en una sola instancia del campo 245. A pesar de que esta instancia no es repetible, es única entre todas las instancias, por lo que puede añadirse a la instancia final del campo que se agrega al registro integrado.

Regla 3.2: En este caso se encuentran las instancias del subcampo "a" presentes en las dos instancias del campo 100. Debido a que este subcampo no es repetible y existen dos instancias se debe decidir cuál añadir a la instancia final del campo que se agrega al registro integrado. En este caso el experto debe decidir.

Regla 3.3: En este caso se encuentran las instancias del subcampo "c" presentes en una de las instancias del campo 100. Debido a que este subcampo es repetible se agregan todas sus instancias a la instancia final del campo que se agrega al registro integrado.

Caso 3: Si el campo es variable de datos repetible.

Regla 4: En este caso se encuentra la instancia del campo 250, la cual se añade al registro integrado.

Regla 5: En este caso se tienen los campos 020, 300 y 650 que contienen más de una instancia. A la izquierda de la Figura 31 se muestran las instancias de estos campos que pueden ser mezcladas, mientras que a la derecha se presentan las que no. Es necesario recordar que las instancias que se pueden mezclar solo deben contener instancias de subcampos repetibles y en caso de existir una instancia de un subcampo no repetible, esta debe ser única entre el conjunto de instancias del campo.

Regla 5.1: En este caso se encuentra el campo 300 debido a que todas las instancias de este campo pueden mezclarse para conformar una instancia final que se añade al registro integrado. Todas las instancias se pueden mezclar porque contienen instancias de subcampos que son repetibles.

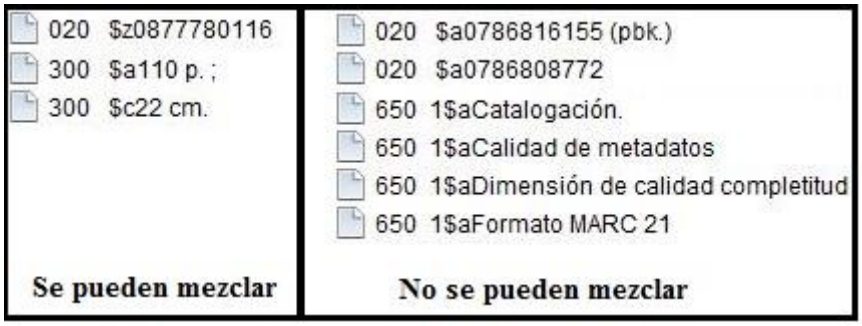


Figura 31: Instancias utilizadas en la mezcla de campos variables de datos.

Regla 5.2: En este caso se encuentra el campo 650 debido a que ninguna instancia se puede mezclar, por lo que todas se añaden al registro integrado. Esto sucede porque contienen instancias de subcampos no repetibles y no son únicas.

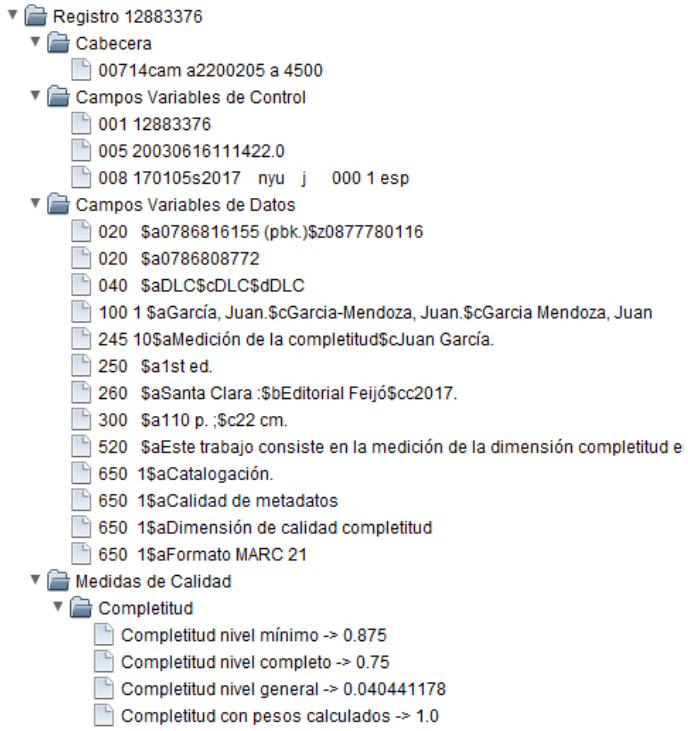


Figura 32: Registro integrado a partir de los dos registros expuestos en la Figura 29.

Regla 5.3: En este caso se encuentra el campo 020 que contiene una instancia que se puede mezclar con las otras dos. Luego de la mezcla se añaden las dos instancias que no se pueden mezclar con la información de las que se pueden mezclar al registro integrado.

El registro final integrado se muestra en la Figura 32 asumiendo que el registro principal se escoge en base a los niveles de completitud y el experto no interviene. En el caso del campo 100 se asume que el experto escoge la instancia del subcampo "a" en la cual el apellido "García" aparece con tilde. Se debe destacar el aumento de la completitud a nivel general, lo que confirma que la integración contribuye a la mejora de la completitud.

Paso 5. Verificar que el registro integrado esté correcto

En el paso 5 el experto debe verificar que el registro integrado esté correcto. Esta acción incluye la modificación o eliminación de instancias de campos y subcampos.

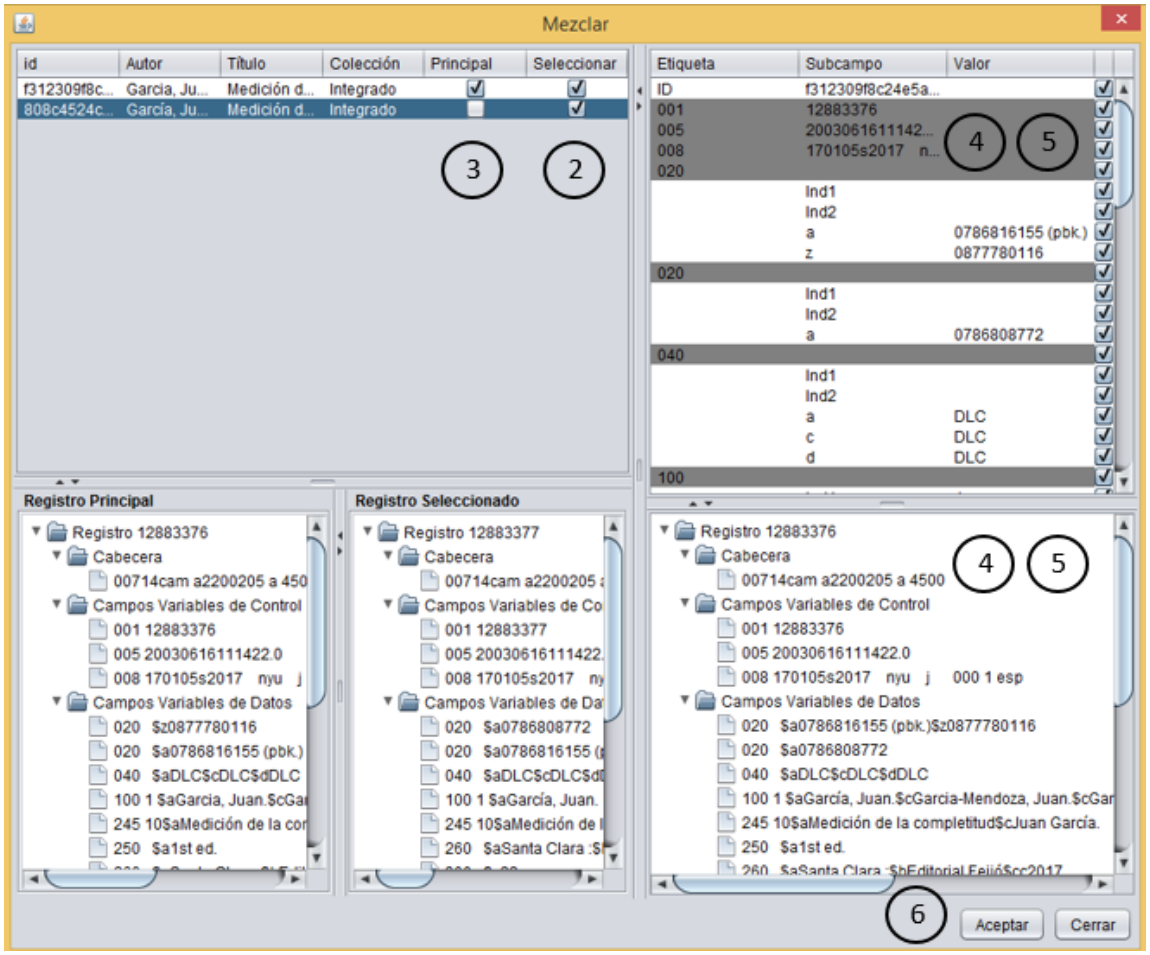


Figura 33: Integración de registros con la herramienta MARComp.

Paso 6. Actualizar la base de datos con el registro integrado

En el paso 6 se materializa la integración en la base de datos eliminando todos los registros duplicados e insertando el registro integrado en la colección.

En la Figura 33 se muestra como se reflejan los pasos 2, 3, 4, 5 y 6 en la herramienta MARComp. Para los pasos 4 y 5 la herramienta provee dos paneles. En el de la esquina superior derecha se modifican o eliminan instancias de campos y subcampos. Estas acciones se reflejan en la esquina inferior derecha, la cual ofrece una visión global del registro integrado.

3.6 Comparación entre MARComp y MarcXimil

MarcXimil es una herramienta que tiene como principal funcionalidad la búsqueda de registros duplicados con formato MARC 21 (Borel y Krause, 2009). En la Tabla 17 se realiza una comparación entre esta herramienta y MARComp.

La principal ventaja que tiene MARComp con respecto a MarcXimil es la posibilidad de medir la completitud y de integrar los registros duplicados una vez detectados. Además, permite una mayor variedad de formatos de entrada y presenta una interfaz gráfica de usuario amigable. Por otro lado, como desventaja, tiene solo una estrategia para la detección de duplicados (llaves de bloqueo "clase de material-año" e iniciales de autor y título y el campo ISBN). En este aspecto MarcXimil contiene varias estrategias que involucran más campos del formato MARC 21 para la detección de duplicados.

Tabla 17: Comparación entre las herramientas MarcXimil y MARComp.

Aspecto	MarcXimil	MARComp
Interfaz gráfica	No	Sí
Formato de entrada	MARCXML	MARC 21, MARCXML, JSON
Medición de la completitud	No	Sí
Detección de duplicados	Sí	Sí
Integración de registros	No	Sí
Estrategia para la detección de duplicados	Varias	Una
Información sobre la colección	Sí	Sí
Multiplataforma	Sí	Sí
Código abierto	Sí	Sí
Complejidad computacional en la búsqueda	$O(n^2)$	$O(n \log n)$

3.7 Conclusiones parciales

En este capítulo se realizaron pruebas unitarias a los casos de uso importar colección y medir completitud de la herramienta MARComp, las cuales fueron satisfactorias. Se midió la completitud con dicha herramienta en dos bases de datos que contienen registros bibliográficos con formato MARC 21 y que pertenecen a bibliotecas cubanas reales. Los resultados obtenidos fueron bajos en ambas bases de datos. Además, los resultados de las medidas precisión, sensibilidad y *F-Measure* al aplicar el algoritmo propuesto en el capítulo anterior para la detección de registros duplicados con formato MARC 21 utilizando seis bases de datos sintéticas fueron altos cuando existen pocas variaciones en las iniciales de los autores y del título. También, el procedimiento propuesto para la integración de registros duplicados con formato MARC 21 permitió la mejora de los valores de completitud del registro unificado con respecto a los que se integran. Esto se comprobó mediante un caso de prueba y la utilización de la herramienta MARComp. Finalmente, se comparó la herramienta MARComp con MarcXimil en cuanto a la detección de duplicados donde, a pesar de utilizar una sola estrategia, permite medir la completitud por varios niveles y clases de materiales y obtener un registro integrado más completo.

Conclusiones

- Se identificaron la cantidad de campos, el grado de importancia de cada uno de estos y cómo determinar si está completo o no como las características que varían en el contexto de los registros bibliográficos con formato MARC 21.
- Se modificó la etapa de indexado de un algoritmo expuesto en la literatura para detectar elementos duplicados lo que permite reducir el espacio de búsqueda en los catálogos de registros bibliográficos con formato MARC 21.
- Se establecieron reglas que permiten la integración de registros duplicados con formato MARC 21 en un registro unificado cuya completitud resultante es mayor o igual que la de dichos elementos por separado.
- Se implementó la herramienta MARComp utilizando patrones de diseño que permiten su extensibilidad e incluye las métricas y reglas utilizadas en el proceso de medición y mejora de la completitud.

Recomendaciones

- Estimar el grado de importancia ideal para cada campo independientemente de los conjuntos de datos a medir, con el objetivo de ofrecer un valor de completitud más certero.
- Extender la herramienta MARComp hacia otros formatos de metadatos.

Referencias Bibliográficas

Ambler, S. W. (2005) *The Elements of UML 2.0 Style*. New York: Cambridge University Press.

De Amicis, F. y Batini, C. (2004) «A methodology for data quality assessment on financial data», *Studies in Communication Sciences*, 4(2), pp. 115-137.

Amón, I. y Jiménez, C. (2010) «Funciones de similitud sobre cadenas de texto: una comparación basada en la naturaleza de los datos», *CONF-IRM Proceedings*.

Amón, I., Moreno, F. y Echeverri, J. (2012) «Algoritmo fonético para detección de cadenas de texto duplicadas en el idioma español», *Revista Ingenierías Universidad de Medellín*, 11(20), pp. 127-138.

Andreu-Alvarez, Y. (2015) *Análisis de la calidad de datos en fuentes de la suite ABCD*. Universidad Central «Marta Abreu» de Las Villas.

Ballou, D. P. y Pazer, H. L. (1985) «Modeling data and process quality in multi-input, multi-output information systems», *Management Science*, 31(2), pp. 150-162.

Barton, J., Currier, S. y Hey, J. (2003) «Building quality assurance into metadata creation: an analysis based on the learning objects and e-prints communities of practice.»

Batini, C., Cappiello, C., Francalanci, C. y Maurino, A. (2009) «Methodologies for data quality assessment and improvement», *ACM Computing Surveys*, 41(3), pp. 1-52. doi: 10.1145/1541880.1541883.

Batini, C. y Scannapieco, M. (2006) *Data Quality: Concepts, Methodologies and Techniques*. Springer-Verlag Berlin Heidelberg.

Batini, C. y Scannapieco, M. (2016) *Data and Information Quality*. Cham: Springer International Publishing (Data-Centric Systems and Applications). doi: 10.1007/978-3-319-24106-7.

Baxter, R., Christen, P. y Churches, T. (2003) «A Comparison of Fast Blocking Methods for Record Linkage», en *ACM KDD'03 Workshop on Data Cleaning, Record Linkage and Object Consolidation*. Washington DC, pp. 25-27.

Beall, J. (2006) «Metadata and data quality problems in the digital library», *Journal of Digital Information*, 6(3).

Bellini, E. y Nesi, P. (2013) «Metadata quality assessment tool for open access cultural heritage institutional repositories», en *Information Technologies for Performing Arts, Media Access, and Entertainment*. Springer Berlin Heidelberg, pp. 90-103.

Benjelloun, O., Garcia-Molina, H. y Gong, H. (2007) «D-swoosh: A family of algorithms for generic, distributed entity resolution», en *International Conference on Distributed Computing Systems (ICDCS'07)*.

Benjelloun, O., Garcia-Molina, H., Kawai, H., Larson, T. E., Menestrina, D., Su, Q., Thavisomboon, S. y Widom, J. (2006) *Generic entity resolution in the serf project, IEEE Computer Society Technical Committee on Data Engineering*.

Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Whang, S. E. y Widom, J.

(2009) «Swoosh: a generic approach to entity resolution», *The VLDB Journal*. Springer-Verlag, 18(1), pp. 255-276. doi: 10.1007/s00778-008-0098-x.

Berti-Equille, L. (2007) «Measuring and modelling data quality for quality-awareness in data mining», en *Studies in Computational Intelligence (SCI)*. Springer-Verlag Berlin Heidelberg, pp. 101-126.

Bharambe, D., Jain, S. y Jain, A. (2012) «A survey: detection of duplicate record», *International Journal of Emerging Technology and Advanced Engineering*, 2(11), pp. 298-307.

Bhattacharya, I. y Getoor, L. (2005) «Relational clustering for multi-type entity resolution», en *4th International Workshop on Multi-relational Data Mining (MRDM-2005)*. Chicago, Illinois, USA Workshop: ACM, pp. 3-12.

Bilenko, M., Kamath, B. y Mooney, R. J. (2006) «Adaptive blocking: Learning to scale up record linkage», en *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM'06)*. Hong Kong.

Bilenko, M. y Mooney, R. J. (2003) «Adaptive duplicate detection using learnable string similarity measures», en *Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining(KDD-2003)*. Washington, DC, USA: ACM, pp. 39-48.

Bilenko, M., Mooney, R. J., Cohen, W. W., Ravikumar, P. y Fienberg, S. E. (2003) «Adaptive name matching in information integration», *IEEE Intelligent Systems*, 18(5), pp. 16-23.

Bobrowski, M., Marré, M. y Yankelevich, D. (1999) *Measuring data quality, Report n.: 99-002*. Buenos Aires, Argentina.

Borel, A. y Krause, J. (2009) *Development of a flexible tool for the automatic comparison of bibliographic records. Application to sample collections*. Université de Genève.

Borges, E., Becker, K., Heuser, C. y Galante, R. (2011) «A classification-based approach for bibliographic metadata deduplication», en *IADIS International Conference WWW/Internet 2011*, pp. 221-228.

Borges, E., Becker, K., Heuser, C. y Galante, R. (2011) «An automatic approach for duplicate bibliographic metadata identification using classification», *Science Society*.

Borges, E., de Carvalho, M. y Galante, R. (2011) «An unsupervised heuristic-based approach for bibliographic metadata deduplication», *Information Processing & Management*, 47(5), pp. 706-718.

Brizan, D. y Tansel, A. (2006) «A. Survey of Entity Resolution and Record Linkage Methodologies», *Communications of the IIMA*, 6(3).

Bruce, T. R. y Hillmann, D. I. (2004) «The continuum of metadata quality: defining, expressing, exploiting», en Association, A. L. (ed.) *Metadata in Practice*. ALA Editions.

Cappiello, C., Ficiaro, P. y Pernici, B. (2006) «HIQM: a methodology for information quality monitoring, measurement, and improvement», en *International Conference on Conceptual Modeling*. Springer Berlin Heidelberg, pp. 339-351.

Cappiello, C., Francalanci, C. y Pernici, B. (2004) «Data quality assessment from the user's perspective», en *Proceedings of the 2004 Iternational Workshop on Information*

Quality in Information Systems. Maison de la Chimie, Paris, France, pp. 68-73.

de Carvalho, M. G., Laender, A. H. F., Gonzalves, M. A. y da Silva, A. S. (2012) «A genetic programming approach to record deduplication», *IEEE Transactions on Knowledge and Data Engineering*, 24(3), pp. 399-412.

Chapman, A. (2005) *Principles of Data Quality*.

Christen, P. (2006) «A comparison of personal name matching: Techniques and practical issues», en *Sixth IEEE International Conference on Data Mining Workshops*. IEEE, pp. 290-294.

Christen, P. (2008a) «Automatic record linkage using seeded nearest neighbour and support vector machine classification», en *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 151-159.

Christen, P. (2008b) «Febrl - an open source data cleaning, deduplication and record linkage system with a graphical user interface», en *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. Las Vegas, Nevada, USA: ACM, pp. 1065-1068.

Christen, P. (2012a) «A survey of indexing techniques for scalable record linkage and deduplication», *IEEE Transactions on Knowledge and Data Engineering*, 25(5), pp. 1537-1555.

Christen, P. (2012b) *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Editado por M. J. Carey y S. Ceri. Springer-Verlag Berlin Heidelberg. doi: 10.1007/978-3-642-31164-2.

Christen, P. y Goiser, K. (2007) «Quality and complexity measures for data linkage and deduplication», en F. Guillet and H. Hamilton (ed.) *Quality Measures in Data Mining*. Springer-Verlag Berlin Heidelberg.

Christen, P., Vatsalan, D. y Fu, Z. (2015) «Advanced Record Linkage Methods and Privacy Aspects for Population Reconstruction—A Survey and Case Studies», en *Population Reconstruction*. Springer International Publishing, pp. 87-110.

Ciaccia, P., Patella, M., Rabitti, F. y Zezula, P. (1997) «Indexing Metric Spaces with M-Tree», en *SEBD*, pp. 67-86.

Ciaccia, P., Patella, M. y Zezula, P. (1997) «M-tree: An Efficient Access Method for Similarity Search in Metric Spaces», en *23rd International Conference on Very Large Data Bases*. Athens, Greece: Morgan Kaufmann, pp. 426-435.

Cohen, W., Kautz, H. y McAllester, D. (2000) «Hardening soft information sources», en *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. Boston, Massachusetts, USA: ACM, pp. 255-259.

Cohen, W. W. (2000) «Data integration using similarity joins and a word-based information representation language», *ACM Transactions on Information Systems (TOIS)*, 18(3), pp. 288-321.

Cohen, W. W., Ravikumar, P. y Fienberg, S. E. (2003a) «A Comparison of String Distance Metrics for Name-Matching Tasks», *IIWeb*.

Cohen, W. W., Ravikumar, P. y Fienberg, S. E. (2003b) «A comparison of string metrics

for matching names and records», en *Kdd workshop on data cleaning and object consolidation*, pp. 73-78.

Damerau, F. (1964) «A technique for computer detection and correction of spelling errors», *Communications of the ACM*, 7(3), pp. 171-176.

Day, M., Guy, M. y Powell, A. (2004) «Improving the quality of metadata in Eprint archives», *Ariadne*, (38).

Dean, J. y Ghemawat, S. (2008) «MapReduce: Simplified Data Processing on Large Clusters», *Communications of the ACM*, 51(1), pp. 107-113. doi: 10.1145/1327452.1327492.

Díaz-de-la-Paz, L., García-Mendoza, J. L., Andreu-Alvarez, Y., López-Porrero, B., González-González, L. y Rodríguez-Morffi, A. (2015) *Calidad de datos*. Samuel Feijó.

Díaz-de-la-Paz, L., García-Mendoza, J. L., López-Porrero, B., González-González, L. y Lemahieu, W. (2015) «Técnicas para capturar cambios en los datos y mantener actualizado un almacén de datos», *Revista Cubana de Ciencias Informáticas*, 9(4), pp. 89-103.

Dohnal, V. (2004) *Indexing structures for searching in metric spaces*. Masaryk University.

Dong, X., Halevy, A. y Madhavan, J. (2005) «Reference Reconciliation in Complex Information Spaces», en *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*. Baltimore, Maryland, USA: ACM, pp. 85-96.

Dong, Y., Ling, P., Liu, Y. y Chu, Q. (2014) «TA-DRD: A Three-step Automatic Duplicate Record Detection», *Open Automation and Control Systems Journal*, 6, pp. 1277-1286.

Draisbach, U., Naumann, F., Szott, S. y Wonneberg, O. (2012) «Adaptive windows for duplicate detection», en *IEEE 28th International Conference on Data Engineering (ICDE)*. IEEE, pp. 1073-1083.

Elmagarmid, A. K., Ipeirotis, P. G. y Verykios, V. S. (2007) «Duplicate record detection: A survey», *IEEE Transactions on Knowledge and Data Engineering*. IEEE, 19(1).

English, L. (1999) *Improving data warehouse and business information quality: methods for reducing costs and increasing profits*. J. Wiley & Sons.

Eppler, M. y Muenzenmayer, P. (2002) «Measuring Information Quality in the Web Context: A Survey of State-of-the-Art Instruments and an Application Methodology.», en *Proceedings of the Seventh International Conference on Information Quality (ICIQ-02)*, pp. 187-196.

Falorsi, P., Pallara, S., Pavone, A., Alessandroni, A., Massella, E. y Scannapieco, M. (2003) «Improving the quality of toponymic data in the italian public administration», en *Proceedings of the ICDT*.

Fellegi, I. y Sunter, A. (1969) «A Theory for Record Linkage», *Journal of the American Statistical Association*, 64(328), pp. 1183-1210.

Fisher, J., Christen, P., Wang, Q. y Rahm, E. (2015) «A Clustering-Based Framework to Control Block Sizes for Entity Resolution», en *Proceedings of the 21th ACM SIGKDD*

International Conference on Knowledge Discovery and Data Mining. Sydney, NSW, Australia: ACM, pp. 279-288.

Fisher, J., Wang, Q., Wong, P. y Christen, P. (2013) «Data Cleaning and Matching of Institutions in Bibliographic Databases», en *Eleventh Australasian Data Mining Conference (AusDM 2013)*. Canberra, Australia.

Freeman, E., Freeman, E., Sierra, K. y Bates, B. (2004) *Head first. Design patterns*. O'Reilly.

Gamma, E., Helm, R., Johnson, R. y Vlissides, J. (1993) «Design Patterns: Abstraction and Reuse of Object-Oriented Design», en *European Conference on Object-Oriented Programming*. Springer-Verlag, pp. 406-433.

García-Mendoza, J. L., Díaz-de-la-Paz, L., González-González, L. y López-Porrero, B. (2015) «Detección de errores en los procesos ETL utilizando Pentaho Data Integration», en *Congreso Internacional COMPUMAT 2015*. La Habana, Cuba, p. 10.

García-Mendoza, J. L., Díaz-de-la-Paz, L., González-González, L., Nuñez-Arcia, Y. y Leiva-Mederos, A. (2016) «Herramienta CompMARC para la medición de la completitud de registros bibliográficos en formato MARC 21», *Revista Publicando*, 3(6), pp. 397-407.

García-Mendoza, J. L., Díaz-de-la-Paz, L., González-González, L., Nuñez-Arcia, Y., Leiva-Mederos, A. y Moreno-Montes-de-Oca, I. (2016) «Medición de la completitud de registros bibliográficos con formato MARC 21 de universidades cubanas», en *10ª Conferencia internacional de Ciencias Empresariales (CICE)*. Santa Clara, Cuba, p. 13.

Gasser, L. y Stvilia, B. (2001) «A new framework for information quality», *Urbana Champaign: University of Illinois at Urbana Champaign*.

Getoor, L. y Machanavajjhala, A. (2012) «Entity Resolution: Theory, Practice & Open Challenges», *Proceedings of the VLDB Endowment*. Istanbul, Turkey, 5(12), pp. 2018-2019.

Gómez, D., Jústiz, D. y Delgado, M. (2013) «Unit Tests of Software in a University Environment», *Computación y Sistemas*, 17(1), pp. 69-77.

Gruenheid, A., Dong, X. y Srivastava, D. (2014) «Incremental Record Linkage», *Proceedings of the VLDB Endowment*, 7(9), pp. 697-708.

Guevara-Torres, P. M. (2016) *Medición de la completitud y detección de duplicados en metadatos con formato MARC 21*. Universidad Central «Marta Abreu» de Las Villas.

Hall, P. y Dowling, G. (1980) «Approximate String Matching», *ACM Computing Surveys (CSUR)*, 12(4), pp. 381-402.

Haug, A. y Arlbjørn, J. S. (2011) «Barriers to master data quality», *Journal of Enterprise Information Managment*, 24(3), pp. 288-303.

Heise, A., Kasneci, G. y Naumann, F. (2014) «Estimating the number and sizes of fuzzy-duplicate clusters», en *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. ACM, pp. 959-968.

Hernández, M. y Stolfo, S. (1995) «The Merge/Purge Problem for Large Databases», en *ACM SIGMOD Record*. San Jose, CA, USA, pp. 127-138.

Hernández, M. y Stolfo, S. (1998) «Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem», *Data Mining and Knowledge Discovery*, 2, pp. 9-37.

Herzog, T. N., Scheuren, F. J. y Winkler, W. E. (2007) *Data quality and record linkage techniques*, *Data Quality and Record Linkage Techniques*. Springer Science+Business Media.

Holmes, A. (2012) *Hadoop in Practice*. Manning Publications Co.

Hussain, B., Hassanzadeh, O., Chiang, F., Lee, H. C. y Miller, R. J. (2013) *An Evaluation of Clustering Algorithms in Duplicate Detection*.

Hylton, J. A. (1996) *Identifying and merging related bibliographic records*. Massachusetts Institute of Technology.

Jackson, A. S., Han, M.-J., Groetsch, K., Mustafoff, M. y Cole, T. W. (2008) «Dublin Core metadata harvested through OAI-PMH», *Journal of Library Metadata*, 8(1), pp. 5-21. doi: 10.1300/J517v08n01_02.

Jacobson, I., Booch, G. y Rumbaugh, J. (2000) *El proceso unificado de desarrollo de software*. Madrid: Pearson Educación, S. A.

Jarke, M., Lenzerini, M., Vassiliou, Y. y Vassiliadis, P. (1995) *Fundamentals of Data Warehouses*. Springer Science & Business Media.

Jaro, M. (1978) *Unimatch: A record linkage system: Users manual*. Bureau of the Census.

Jeusfeld, M., Quix, C. y Jarke, M. (1998) «Design and analysis of quality information for data warehouses», en *International Conference on Conceptual Modeling*, pp. 349-362.

Jiang, Y., Lin, C., Meng, W., Yu, C., Cohen, A. M. y Smalheiser, N. R. (2013) «Rule-based deduplication of article records from bibliographic databases», *Database: The Journal of Biological Databases and Curation*. Oxford University Press, 2014. doi: 10.1093/database/bat086.

Jörg, T. y Deßloch, S. (2009) «Formalizing ETL Jobs for Incremental Loading of Data Warehouses», en *BTW*, pp. 327-346.

Juran, J. M. y Gryna Jr, F. M. (1980) *Quality Planning and Analysis: From Product Development Through Usage*, *McGraw-Hill*. New York, N.Y.

Karagiannis, A., Vassiliadis, P. y Simitsis, A. (2013) «Scheduling strategies for efficient ETL execution», *Information Systems*, 38(6), pp. 927-945. doi: 10.1016/j.is.2012.12.001.

Karthigha, M. y Anand, S. (2013) «A survey on removal of duplicate records in database», *Indian Journal of Science and Technology*, 6(4), pp. 4306-4311.

Kejriwal, M. y Miranker, D. (2013) «An Unsupervised Algorithm for Learning Blocking Schemes», en *IEEE 13th International Conference on Data Mining (ICDM)*. IEEE, pp. 340-349.

Kenig, B. y Gal, A. (2009) «Efficient Entity Resolution with MFIBlocks», en *Proceedings of the VLDB Endowment*.

Kenig, B. y Gal, A. (2013) «MFIBlocks: An effective blocking algorithm for entity resolution», *Information Systems*, 38(6), pp. 908-926.

Köpcke, H. y Rahm, E. (2010) «Frameworks for entity matching: A comparison», *Data*

& Knowledge Engineering, 69(2), pp. 197-210.

Lee, Y. W., Pipino, L. L., Funk, J. D. y Wang, R. Y. (2006) «Journey to Data Quality», *Cambridge, MA, USA: Massachussets Institute of Technology*.

Lee, Y. W., Strong, D. M., Kahn, B. K. y Wang, R. Y. (2002) «AIMQ: a methodology for information quality assessment», *Information & management*, 40(2002), pp. 133-146.

Leitao, L., Calado, P. y Herschel, M. (2013) «Efficient and effective duplicate detection in hierarchical data», *IEEE Transactions on Knowledge and Data Engineering*, 25(5), pp. 1028-1041.

Lenzerini, M. (2002) «Data Integration: A Theoretical Perspective», en *Symp. on Principles of Database Systems (PODS)*. Madison, Wisconsin, USA, pp. 233-246.

León-Hernández, D. (2015) *Medición de la dimensión completitud en la base de datos MARC de ABCD*. Universidad Central «Marta Abreu» de Las Villas.

Levenshtein, V. (1966) «Binary codes capable of correcting deletions, insertions and reversals», *Soviet Physics Doklady*, 10(8), pp. 707-710.

Liaw, S.-T., Rahimi, A., Ray, P., Taggart, J., Dennis, S., de Lusignan, S., Jalaludin, B., Yeo, A. E. T. y Talaei-Khoei, A. (2013) «Towards an ontology for data quality in integrated chronic disease management: a realist review of the literature», *International journal of medical informatics*, 82(1), pp. 10-24.

Liu, L. y Chi, L. (2002) «Evolutionary data quality», en *Proceedings of the 7th international conference on information quality (IQ)*.

Liu, X., Thomsen, C. y Pedersen, T. B. (2012) «MapReduce-based Dimensional ETL Made Easy», *Proceedings of the VLDB Endowment*, 5(12), pp. 1882-1885.

Long, J. y Seko, C. (2005) «A cyclic-hierarchical method for database data-quality evaluation and improvement», *Information quality*, 1, p. 52.

López-Porrero, B. (2011) *Limpieza de datos: Reemplazo de valores ausentes y estandarización*. Universidad Central «Marta Abreu» de Las Villas.

Loshin, D. (2001) *Enterprise knowledge management: The data quality approach*. Morgan Kaufmann.

Madnick, S. E., Wang, R. Y., Lee, Y. W. y Zhu, H. (2009) «Overview and Framework for Data and Information Quality Research», *ACM Journal of Data and Information Quality*, 1(1), p. 22.

Man, Y., Wei, L., Gang, H. y Juntao, G. (2010) «A Noval Data Quality Controlling and Assessing Model Based on Rules», en *Third International Symposium on Electronic Commerce and Security*, pp. 29-32.

Masek, W. J. y Paterson, M. S. (1980) «A faster algorithm computing string edit distances», *Journal of Computer and System Sciences*, 20(1), pp. 18-31. doi: 10.1016/0022-0000(80)90002-1.

Mayernik, M. (2010) «The distributions of MARC fields in bibliographic records: a power law analysis», *Library Resources and Technical Services*, 54(1), pp. 40-54.

McNeill, N., Kardes, H. y Borthwick, A. (2012) «Dynamic record blocking: efficient

linking of massive databases in mapreduce», en *Proceedings of the 9th International Workshop on Quality in Databases (QDB)*.

Mendes, P. N., Mühleisen, H. y Bizer, C. (2012) «Sieve: Linked Data Quality Assessment and Fusion», en *Proceedings of the 2012 Joint EDBT/ICDT Workshops*. Berlin, Germany: ACM, pp. 116-123.

Menestrina, D., Benjelloun, O. y Garcia-Molina, H. (2006) «Generic entity resolution with data confidences», en *First International VLDB Workshop on Clean Databases*. Seoul, South Korea.

Moen, W. E., Stewart, E. L. y McClure, C. R. (1997) «The role of content analysis in evaluating metadata for the us government information locator service (GILS): results from an exploratory study», en *IEEE Computer Society metadata conference*, pp. 1-14.

Moges, H.-T., Dejaeger, K., Lemahieu, W. y Baesens, B. (2013) «A multidimensional analysis of data quality for credit risk management: New insights and challenges», *Information & Management*, 50(1), pp. 43-58. doi: 10.1016/j.im.2012.10.001.

Monge, A. y Elkan, C. (1996) «The Field Matching Problem: Algorithms and Applications», en *Second International Conference on Knowledge Discovery and Data Mining*, pp. 267-270.

Monge, A. y Elkan, C. (1997) «An efficient domain-independent algorithm for detecting approximately duplicate database records.» Citeseer.

Naumann, F. y Herschel, M. (2010) *An Introduction to Duplicate Detection*, *Synthesis Lectures on Data Management*. Morgan & Claypool Publishers.

Newcombe, H., Kennedy, J., Axford, S. y James, A. (1959) «Automatic Linkage of Vital Records», *Science*, 30(3381), pp. 954-959.

Nichols, D. M., Chan, C.-H., Bainbridge, D., McKay, D. y Twidale, M. B. (2008) «A tool for metadata analysis.»

Nuñez-Arcia, Y., Díaz-de-la-Paz, L. y García-Mendoza, J. L. (2016) «Algoritmo para corregir anomalías a nivel de instancia en grandes volúmenes de datos utilizando MapReduce», *Revista Cubana de Ciencias Informáticas*, 10(3), pp. 105-118.

Ochoa, X. y Duval, E. (2006a) «Quality Metrics for learning object Metadata», en *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications*. AACE, pp. 1004-1011.

Ochoa, X. y Duval, E. (2006b) «Towards automatic evaluation of learning object metadata quality», en *Advances in Conceptual Modeling-Theory and Practice*. Springer, pp. 372-381.

Ochoa, X. y Duval, E. (2006c) «Towards Automatic Evaluation of Metadata Quality in Digital Repositories», *Lecture Notes in Computer Science*, 4231, pp. 372-381.

Ochoa, X. y Duval, E. (2008) *Learnometrics: Metrics for Learning Objects*. Katholieke Universiteit Leuven.

Ochoa, X. y Duval, E. (2009) «Automatic evaluation of metadata quality in digital repositories», *International Journal on Digital Libraries*, 10(2-3), pp. 67-91.

Oliveira, Z. de, Freitas, L. y Pavão, C. (2010) «Gerência de registros duplos em base de dados bibliográfica», en *Seminário Nacional de Bibliotecas Universitárias*. Rio de Janeiro.

Papadakis, G., Ioannou, E. y Niederée, C. (2012) «Beyond 100 million entities: large-scale blocking-based resolution for heterogeneous data», en *Proceedings of the fifth ACM international conference on Web search and data mining*. Seattle,Washingtion, USA: ACM, pp. 53-62.

Papadakis, G., Ioannou, E., Niederée, C. y Fankhauser (2011) «Efficient entity resolution for large heterogeneous information spaces», en *Proceedings of the fourth ACM international conference on Web search and data mining*. Hong Kong, China: ACM, pp. 535-544.

Papadakis, G., Ioannou, E., Niederée, C., Palpana, T. y Nejdl, W. (2011) «Eliminating the redundancy in blocking-based entity resolution methods», en *Proceedings of the 11th annual international ACM/IEEE joint conference on Digital libraries*. Ottawa, Ontario, Canada: ACM, pp. 85-94.

Papadakis, G., Ioannou, E., Niederée, C., Palpanas, T. y Nejdl, W. (2011) «To Compare or Not to Compare: Making Entity Resolution more Efficient», en *Proceedings of the International Workshop on Semantic Web Information Management*. ACM, p. 3.

Papadakis, G., Ioannou, E., Palpanas, T., Niederée, C. y Nejdl, W. (2013) «A blocking framework for entity resolution in highly heterogeneous information spaces», *IEEE Transactions on Knowledge and Data Engineering*, 25(12), pp. 2665-2682.

Papadakis, G. y Palpanas, T. (2016) «Blocking for large-scale Entity Resolution: Challenges, algorithms, and practical examples», en *IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, pp. 1436-1439.

Papadakis, G., Papastefanatos, G. y Koutriba, G. (2014) «Supervised meta-blocking», *Proceedings of the VLDB Endowment*. VLDB Endowment, 7(14), pp. 1929-1940.

Papadakis, G., Svirsky, J., Gal, A. y Palpanas, T. (2016) «Comparative analysis of approximate blocking techniques for entity resolution», *Proceedings of the VLDB Endowment*, 9(9), pp. 684-695.

Papenbrock, T., Heise, A. y Felix, N. (2015) «Progressive duplicate detection», *IEEE Transactions on Knowledge and Data Engineering*, 27(5), pp. 1316-1329.

Park, J.-R. (2005) «Semantic Interoperability across Digital Image Collections: A Pilot Study on Metadata Mapping», *CAIS/ACSI*, pp. 2-4.

Pei, S. O. A. (2008) *A Comparative Study of Record Matching Algorithms*, *European Master in Informatics (EuMI)*. RWTH Aachen, Germany and University of Edinburgh, Scotland.

Philips, L. (1990) «Hanging on the Metaphone», *Computer Language Magazine*, 7(12), pp. 39-44.

Philips, L. (2000) «The Double Metaphone Search Algorithm», *C/C++ Users Journal*, 18(5), pp. 38-43.

Pipino, L. L., Lee, Y. W. y Wang, R. Y. (2002) «Data quality assessment»,

Communications of the ACM, 45(4), pp. 211-218.

Pressman, R. S. (2010) *Software engineering : a practitioner's approach*. Seventh Ed. McGraw-Hill.

Reynolds, T., Painter, I. y Streichert, L. (2013) «Data Quality: A Systematic Review of the Biosurveillance Literature», *Online Journal of Public Health Informatics*, 5(1).

Russell, R. C. (1918) «Index.» United States: Google Patents.

Russell, R. C. (1922) «Index.» United States: Google Patents.

Sadiq, S., Yeganeh, N. y Indulska, M. (2011) «20 Years of Data Quality Research: Themes, Trends and Synergies», en *Proceedings of the Twenty-Second Australasian Database Conference*. Perth, Australia, pp. 153-162.

Sarawagi, S. y Bhamidipaty, A. (2002) «Interactive Deduplication using Active Learning», en *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. Edmonton, Alberta, Canada: ACM, pp. 269-278.

Scannapieco, M. y Catarci, T. (2002) «Data quality under a computer science perspective», *Archivi & Computer*, 2, pp. 1-15.

Scannapieco, M., Virgillito, A., Marchetti, C. y Mecella, M. (2004) «The DaQuinCIS architecture: a platform for exchanging and improving data quality in cooperative information systems», *Information systems*. Elsevier, 29(7), pp. 551-582.

Shankaranarayanan, G. y Cai, Y. (2006) «Supporting data quality management in decision-making», *Decision Support Systems*, 42(1), pp. 302-317.

Shreeves, S. L., Knutson, E. M., Stvilia, B., Palmer, C. L., Twidale, M. B. y Cole, T. W. (2005) «Is "quality" metadata "shareable" metadata? The Implications of Local Metadata Practices for Federated Collections», en *ACRL Twelfth National Conference*. Minneapolis, Minnesota, USA, pp. 223-2.

Shu, L., Chen, A., Xiong, M. y Meng, W. (2011) «Efficient spectral neighborhood blocking for entity resolution», en *IEEE 27th International Conference on Data Engineering (ICDE)*. IEEE, pp. 1067-1078.

Silberschatz, A., Galvin, P. B. y Gagne, G. (2005) *Operating Systems Concepts*. Seventh Ed. John Wiley & Sons. Inc.

Sitas, A. y Kapidakis, S. (2008) «Duplicate detection algorithms of bibliographic descriptions», *Library Hi Tech*, 26(2), pp. 287-301. doi: 10.1108/07378830810880379.

Sivogolovko, E. (2011) *Evaluation of impact of data quality on clustering with syntactic cluster validity methods*. Christian-Albrechts University.

Smet, E. de y Spinak, E. (2009) «The abc of ABCD: the Reference Manual.»

Smith, T. y Waterman, M. (1981) «Identification of common molecular subsequences», *Journal Molecular Biology*, 147(1), pp. 195-197.

Steorts, R., Ventura, S., Sadinle, M. y Fienberg, S. E. (2014) «A comparison of blocking methods for record linkage», en *International Conference on Privacy in Statistical Databases*. Springer International Publishing, pp. 253-268.

Su, Y. y Jin, Z. (2004) «A methodology for information quality assessment in the

designing and manufacturing processes of mechanical products», en *Proceedings of the 9th International Conference on Information Quality (ICIQ)*, pp. 447-465.

Su, Y. y Jin, Z. (2006) «A methodology for information quality assessment in the designing and manufacturing process of mechanical products», en *Information Quality Management: Theory and Applications*. Idea Group Publishing Hershey, pp. 190-220. doi: 10.4018/978-1-59904-024-0.ch009.

Taft, R. (1970) *Name search techniques: New york state identification and intelligence system, Technical Report Special Report*. Albany, N.Y.

Tani, A., Candela, L. y Castelli, D. (2013) «Dealing with metadata quality: The legacy of digital library efforts», *Information Processing & Management*, 49(2013), pp. 1194-1205.

Tayi, G. K. y Ballou, D. P. (1998) «Examining data quality», *Communications of the ACM*, 41(2), pp. 54-57.

Thor, A. y Rahm, E. (2007) «MOMA-A Mapping-based Object Matching System», en *3rd Biennial Conference on Innovative Data Systems Research (CIDR)*. Asilomar, California, USA, pp. 247-258.

Toney, S. R. (1992) «Cleanup and deduplication of an international bibliographic database», *Information Technology and Libraries*. American Library Association, Chicago, IL, ETATS-UNIS, 11(1), pp. 19-28.

Traina Jr., C., Traina, A., Seeger, B. y Faloutsos, C. (2000) «Slim-trees: High performance metric trees minimizing overlap between nodes», en *International Conference on Extending Database Technology*. Springer, pp. 51-65.

Turenne, N. (2015) «Duplicate Detection with Efficient Language Models for Automatic Bibliographic Heterogeneous Data Integration.»

Ullmann, J. (1977) «A binary n-gram technique for automatic correction of substitution, deletion, insertion and reversal errors in words», *The Computer Journal*, 20(2), pp. 141-147.

Vassiliadis, P. (2009) «A Survey of Extract–Transform–Load technology», *International Journal of Data Warehousing and Mining*, 5(3), pp. 1-27.

Vassiliadis, P., Simitsis, A., Georgantas, P. y Terrovitis, M. (2003) «A Framework for the Design of ETL Scenarios», en *15th Intl. Conference on Advanced Information Systems Engineering (CAiSE)*. Klagenfurt, Austria, pp. 520-535.

Veloso de Melo, V. y de Andrade Lopes, A. (2005) «Efficient identification of duplicate bibliographical references», en *Advances in logic based intelligent systems*, pp. 169-176.

Wand, Y. y Wang, R. Y. (1996) «Anchoring data quality dimensions in ontological foundations», *Communications of the ACM*, 39(11), pp. 86-95.

Wang, R. Y. (1998) «A product perspective on total data quality management», *Communications of the ACM*, 41(2), pp. 58-65.

Wang, R. Y. y Strong, D. M. (1996) «Beyond accuracy: What data quality means to data consumers», *Journal of Management Information Systems*, 12(4), pp. 5-34.

Wang, Y. y Madnick, S. E. (1989) «The Inter-Database Instance Identification Problem

in Integrating Autonomous Systems», en *Proceedings of the Fifth International Conference on Data Engineering*, pp. 46-55.

Waterman, M., Smith, T. y Beyer, W. (1976) «Some Biological Sequence Metrics», *Advances in Mathematics*, 20(3), pp. 367-387.

Whang, S. y Marmaros, D. (2013) «Pay-as-you-go entity resolution», *IEEE Transactions on Knowledge and Data Engineering*. IEEE, 25(5), pp. 1111-1124.

Winkler, W. E. (1990) «String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage», en *Section on Survey Research Methods*, pp. 354-359.

Winkler, W. E. (1999) «The state of record linkage and current research problems», en *Statistical Research Division, US Census Bureau*. Citeseer.

Winkler, W. E., Yancey, W. y Porter, E. (2010) «Fast record linkage of very large files in support of decennial and administrative records projects», en *Section on Survey Research Methods*, pp. 2120–2130.

Yancey, W. (2004) «An adaptive string comparator for record linkage», *Statistics*, p. 2.

Yeganeh, N., Sadiq, S. y Sharaf, M. (2014) «A framework for data quality aware query systems», *Information Systems*. Elsevier, 46, pp. 24-44.

Zhu, J. y Ungar, L. (2000) «String edit analysis for merging databases», *KDD workshop on text mining, held at ACM SIGKDD*.

Anexo 1 Estructura de datos métrica M-Tree

M-Tree es una estructura de datos métrica que permite un indexado dinámico y balanceado y el almacenamiento en disco (Dohnal, 2004). Esta estructura particiona objetos en base a sus distancias relativas medidas por una función de distancia *d* y los almacena en nodos de tamaño fijo que coinciden con las regiones del espacio métrico (Ciaccia, Patella y Zezula, 1997; Ciaccia, Patella, Rabitti, *et al.*, 1997).

M-Tree almacena todos los objetos indexados en las hojas a través de sus llaves o características. Los nodos hojas tienen la siguiente estructura:

$$entrada(O_j) = \left[O_j, oid(O_j), d\left(O_j, P(O_j)\right)\right]$$

Donde O_j son las características del objeto, $oid(O_j)$ el identificador del objeto y $d\left(O_j, P(O_j)\right)$ la distancia O_j y su padre $P(O_j)$.

Por otro lado, esta estructura almacena en los nodos intermedios objetos denominados de ruteo. La estructura de estos nodos intermedios es la siguiente:

$$entrada(O_r) = \left[O_r, ptr\left(T(O_r)\right), r(O_r), d\left(O_r, P(O_r)\right)\right]$$

Donde O_r son las características del objeto de ruteo, $r(O_r)$ el radio de cubrimiento ($r(O_r) > 0$), $ptr\left(T(O_r)\right)$ el puntero a la raíz del subárbol $T(O_r)$ (árbol de cubrimiento de O_r) y $d\left(O_r, P(O_r)\right)$ la distancia O_r y su padre $P(O_r)$.

Una de las propiedades que cumple esta estructura es que el radio de cubrimiento de un objeto de ruteo, O_r, debe satisfacer la desigualdad $d\left(O_r, P(O_r)\right) \leq r(O_r)$ para cada objeto O_j almacenado en el árbol de cobertura de O_r (Ciaccia, Patella, Rabitti, *et al.*, 1997).

Esta estructura permite dos tipos de búsqueda: *k* vecinos más cercanos y consulta de rango. En ambas búsquedas se pasan dos parámetros. El primero en ambas es el elemento a buscar mientras que el segundo varía en dependencia de la búsqueda. En el caso de los *k* vecinos más cercanos el segundo parámetro es la cantidad de elementos a recuperar (*k*) y como resultado se obtienen los *k* elementos más cercanos al elemento pasado en el primer

parámetro. En la consulta de rango el segundo parámetro es un umbral *d* y como resultado se obtienen todos los elementos cuya distancia respecto al elemento pasado en el primer parámetro sea menor o igual que *d*.

Anexo 2 Algoritmos para la mezcla de instancias pertenecientes a campos variables de datos repetibles y no repetibles.

Mezcla de varias instancias de un campo repetible en el formato MARC 21.

```
Entrada: Lista de instancias de un campo repetible (lista)
Salida:  Lista de instancias a añadir en el nuevo registro
función mezclarInstanciasRepetibles (lista<Instancia>)
1:    instancias ← null
2:    Si tamaño de lista es 1 Entonces
3:      Retornar lista
4:    Fin Si
5:    Eliminar subcampos con el mismo código y valor en todas las
6:    instancias de lista
7:    Eliminar de lista instancias que no tengan subcampos
8:    instancias_A_Mezclar<Instancia> ← Obtener instancias de la lista
9:    que pueden mezclarse (que tenga solamente subcampos repetibles
10:   y si tiene uno no repetible no existe en otra instancia)
11:   lista ← Eliminar de lista los elementos de instancias_A_Mezclar
12:   Si instancias_A_Mezclar está vacía Entonces
13:     Añadir a instancias todos los elementos de lista
14:   En caso contario Si lista está vacía Entonces
15:     instancia ← Primer elemento de instancias_A_Mezclar
16:     Para i=1; i < tamaño de instancias_A_Mezclar;  i++ Hacer
17:       Añadir a instancia todos los subcampos de la instancia i de
18:       instancias_A_Mezclar
19:     Fin Para
20:     Añadir instancia a instancias
21:   En caso contrario
22:     Para i=0; i < tamaño de lista; i++ Hacer
23:       instancia ← instancia i de lista
24:       Para j=0; j < tamaño de instancias_A_Mezclar; j++ Hacer
25:         aux ← instancia j de instancias_A_Mezclar
26:         subcampos_A_Mezclar ← Subcampos de la instancia aux
27:         Para z=0; z < tamaño de subcampos_A_Mezclar; z++ Hacer
28:           subcampo ← subcampo z de subcampos_A_Mezclar
29:           Si instancia no contiene al subcampo subcampo o subcampo
30:           es repetible Entonces
31:             Añadir a instancia el subcampo subcampo
32:             Eliminar el subcampo subcampo de la instancia aux
33:           Fin Si
34:         Fin Para
35:       Fin Para
36:       Añadir a instancia a instancias
37:     Fin Para
38:   Fin Si
39:   Retornar instancias
```

Mezcla de varias instancias de un campo no repetible en el formato MARC 21.

```
Entrada: Lista de instancias de un campo no repetible (lista)
Salida:  Instancia a añadir en el nuevo registro
función mezclarInstanciasNoRepetibles (lista<Instancia>)
1:    instancia ← null
2:    tablaHash<Código_de_Subcampo, Lista_de_Subcampos>
3:    Si tamaño de lista es 1 Entonces
4:      Retornar el único elemento de lista
5:    Fin Si
6:    Eliminar subcampos con el mismo código y valor en todas las
7:    instancias de lista
8:    Eliminar de lista instancias que no tengan subcampos
9:    instancia ← Instancia de lista con mayor cantidad de subcampos
10:   Eliminar instancia de lista
11:   Llenar la tabla hash tablaHash con todas las instancias de lista,
12:   de manera que cada entrada tenga como llave el código del subcampo
13:   y como valor una lista con todos los subcampos con ese código
14:   Para cada entrada de tablaHash Hacer
15:     código ← llave de entrada
16:     lista_De_Subcampos ←  valor de entrada
17:     Si tamaño de lista_De_Subcampos es 1 Entonces
18:       subcampo ← único elemento de lista_De_Subcampos
19:       Si instancia no contiene a subcampo Entonces
20:         Añadir a instancia el subcampo subcampo
21:       Fin Si
22:     En caso contario
23:       Si código es código de un subcampo repetible Entonces
24:         Añadir a instancia la lista de subcampos lista_De_Subcampos
25:       En caso contrario
26:         subcampo ← Subcampo de lista_De_Subcampos seleccionado por
27:         el usuario
28:         Añadir a instancia el subcampo subcampo
29:       Fin Si
30:     Fin Si
31:   Fin Para
32:   Retornar instancia
```

Anexo 3 Pruebas unitarias para la medición de la completitud.

Método para probar el cálculo de la completitud a nivel completo

```
@Test
public void fullCompletenessTest() throws ConnectionNotEstablished {//BD_CHABON_MRC = "chabon.mrc"
    FileConnection connFile = new FileConnection(new File(URL_BASE + BD_CHABON_MRC));
    RecordFormat rFormat = new RecordFormat(RecordFormat.MARC21_TYPE, RecordFormat.MARC21_NAME);
    Reader reader = Reader.createReader(connFile, rFormat);
    while (reader.hasNext()) {
        Record record = reader.next();
        assertNotNull(record);
        Completeness comp = Completeness.createCompleteness(Metric.COMPLETENESS_FULL, record.getRec
        Float value = comp.completeness(record);
        Float expValue = 9 / 12.0f;
        assertEquals(expValue, value);
    }
}
```

Método para probar el cálculo de la completitud a nivel general

```
@Test
public void generalCompletenessTest() throws ConnectionNotEstablished {//BD_CHABON_MRC = "chabon.mrc"
    FileConnection connFile = new FileConnection(new File(URL_BASE + BD_CHABON_MRC));
    RecordFormat rFormat = new RecordFormat(RecordFormat.MARC21_TYPE, RecordFormat.MARC21_NAME);
    Reader reader = Reader.createReader(connFile, rFormat);
    while (reader.hasNext()) {
        Record record = reader.next();
        assertNotNull(record);
        Completeness comp = Completeness.createCompleteness(Metric.COMPLETENESS_GENERAL, record.getRec
        Float value = comp.completeness(record);
        Float expValue = 11 / 272.0f;
       assertEquals(expValue, value);
    }
}
```