

TESIS DOCTORAL

Tratamiento semántico de atributos textuales en un Modelo Relacional Orientado a Objetos: Implementación en Software Libre

Sandro Martínez Folgoso

Granada, 2008

Editor: Editorial de la Universidad de Granada
Autor: Sandro Martínez Folgoso
D.L.: GR.1788-2008
ISBN: 978-84-691-5650-6



Tratamiento semántico de atributos textuales en un Modelo Relacional Orientado a Objetos: Implementación en Software Libre

memoria que presenta

Sandro Martínez Folgoso

para optar al grado de

Doctor en Informática

2008

DIRECTORES

Dra. María Amparo Vila Miranda

Dra. María José Martín Bautista

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
E INTELIGENCIA ARTIFICIAL

La memoria titulada " Tratamiento semántico de atributos textuales en un Modelo Relacional Orientado a Objetos: Implementación en Software Libre ", que presenta D. Sandro Martínez Folgoso para optar al grado de Doctor, ha sido realizada en el Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada bajo la dirección de las Doctoras Dra. María Amparo Vila Miranda y Dra. María José Martín Bautista.

Granada, 2008.

El Doctorando

Los Directores

D. Sandro Martínez Folgoso

Dra. María Amparo Vila Miranda

Dra. María José Martín Bautista

Dedicatoria

Finalmente ha llegado el momento de dedicar este trabajo a las personas más importantes de mi vida: mi esposa Eliza y toda nuestra familia. Al hacerlo, pienso en todos los momentos buenos y malos que hemos tenido que pasar, para que yo consiguiera este grado y coronar así mi vida, completamente dedicada al estudio. Hoy creo, que llegando a la defensa de este trabajo les puedo decir a todos: ¡Valió la pena!

A mi adorada esposa Eliza, que con su dulzura y amor incondicional ha sido mi mejor estímulo todos estos años de duro bregar. A ella que ha sido la fuente de inspiración de todos mis actos y motivo de mi felicidad infinita.

A nuestra Claudita que simboliza el primero de nuestros hijos, a quienes estarán dedicado el resto de nuestros días.

A mis padres y hermanos, por su sacrificio, amor y entrega constante en estos años, quienes sin duda alguna, son mi mejor ejemplo en la vida.

A todos mis abuelos, los que están y los que no, por ser la raíz de todo lo que somos a base de mucho sacrificio y amor.

Al resto de esa familia hermosa que me ha querido y apoyado todo el tiempo. De forma especial a Aleida, Jorge, Elia, tía, Gustavito, Carlos y Bobe.

A mis suegros Clari y Fernan, por ser personas excepcionalmente nobles y abnegadas.

A mis hermanos de la vida: Erich, Sutil y Yordanis, y sus esposas y familiares.

Agradecimientos

Nuestro José Martí dijo " Toda la gloria del mundo cabe en un grano de maíz ". Este principio ha regido siempre mi vida. Por ello, para nada considero como mío propio el obtener este resultado, el que sin duda alguna, es el más grande en mi vida intelectual. Vale más agradecer a todas y cada una de las personas que me han ayudado a conseguirlo. Por un problema elemental de espacio no los podría mencionar a todos, pero sepan que TODOS están en mi mente y mi corazón, en este momento de especial alegría.

Esta carrera comenzó con un primer correo que le escribí desde Cuba a Amparo, donde le prometía que estaría entre uno de los que llegaría a la meta. El trabajo ha sido duro, pero gracias a muchas personas hoy puedo escribir estas palabras.

Quiero agradecer de la manera más especial posible a Amparo, quien más que directora de tesis, ha sido un ejemplo a seguir. Le agradezco por su amistad incondicional y por su apoyo total en todas las oportunidades que he venido a Granada. También por tener la solución precisa siempre, por inculcarme su espíritu de sacrificio y trabajo. Sin duda alguna el haber tenido el privilegio de trabajar a su lado, ha marcado en mí un antes y un después. Ella me ha cambiado la vida para siempre.

De forma especial quiero agradecer a María José, mi otra directora de tesis que me ha apoyado de forma incondicional. Es ella una excelente profesional y amiga de la que he aprendido muchísimo también. Gracias por estar siempre ahí en el momento preciso, gracias por ayudarme a crecer como investigador. Gracias por ser tan buena y comprensiva.

Agradecer también de forma especial a Miguel Delgado, por su amistad, franqueza y filosofía de la vida, quien es para mí una persona excepcional.

Tampoco estaría hoy aquí de no ser por el profesor Verdegay que generosamente extendió a Cuba esta acción formativa, gracias por su amistad y para él todo nuestro respeto y admiración. De igual forma a Rosa por todos sus consejos y buenas acciones para que todos lleguemos a la meta. Junto a ellos dos, a todos los profesores que nos impartieron docencia en este magnífico Doctorado. A las autoridades de la Universidad de Granada, Universidad de Holguín y la Universidad de Camagüey por darme esta oportunidad única.

A mis compañeros del Departamento de Informática de la Universidad de Camagüey, que han dado un extra para que yo pudiera tener esta oportunidad. De forma especial a Jorge, Eduardo, Julito, Yailé y Olguita.

Gracias también a todas las personas del Departamento de Ciencias de la Computación e Inteligencia Artificial, particularmente a Miguel Prados, Requena, Armando y al resto de los profesores. A los miembros del grupo de Bases de Datos y Sistemas de Información Inteligentes (IDBIS), en especial a Nacho por estar dispuesto a ayudar siempre a todos y a Carlos Molina y Josema, quienes de forma muy generosa me ayudaron a introducirme en los temas de investigación.

A Rocío Pérez y a David Villena por ayudarme con el diseño de la portada.

También me gustaría agradecer a mis amigos cubanos y compañeros en este doctorado: Carlos, Cuevas y Paquito. De forma especial a Darlines por su amistad incondicional, alguien a quien considero muy especial. De igual forma agradecer a mis compañeros becarios que me apoyaron y me dieron su amistad durante este largo período, en especial a Mariola y Javi.

Por último y muy importante, agradecer a José Luis Villena y Elvira, David, Pepi y Juan , al abuelo, Dani y Rocío, y al resto de amigos y familiares. Todos personas con un altruismo y bondad sin límites. Llegue a todos ustedes mis más sinceras e infinitas gracias.

Granada, Julio de 2008.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Desde el inicio del uso masivo de la computación para el procesamiento automatizado de la información, se ha ido incrementando el problema de la acumulación de grandes volúmenes de datos, sin que se traduzcan en información útil y oportuna para la toma de decisiones. Este problema se daba originalmente con tipos de datos estructurados almacenados generalmente en bases de datos relacionales. Posteriormente, sobre todo con el uso masivo de Internet, se repite dicho problema, ahora para otros tipos no estructurados o, en algunos casos, semiestructurados, almacenados en diferentes tipos de sistemas de información.

En este entorno, el principal problema que se debe resolver es cómo representar dicha información, mediante qué tipo de estructuras, de forma que el procesamiento de la misma sea lo más cómodo y eficiente posible. Por otro lado, se han de facilitar técnicas y herramientas para poder procesar la información recogida con el objeto de extraer nuevo conocimiento que pueda resultar útil al usuario. Pero el interés no se centra únicamente en almacenar un tipo concreto de conocimiento, sino en darle una posterior utilidad. Con este objetivo, aparte de las herramientas típicas de consulta con las que cuenta cualquier sistema de almacenamiento y manejo de datos, en ocasiones es necesario el uso de herramientas más específicas para obtener un conocimiento más elaborado y completo a partir de la información almacenada.

Para ello, desde el punto de vista del desarrollo de Sistemas Informáticos

para la recuperación y gestión eficiente de los datos, uno de los tipos de sistemas que han aparecido son los Sistemas de Gestión de Información (en Inglés Information Management Systems (*IMS*)), desde sus versiones iniciales hasta propuestas distribuidas como la que aparece en (Yalagandula y Dahlin, 2004). En (Raghavan y Garcia-Molina, 2001) se realiza una clasificación de un gran número de las propuestas que han aparecido en este tipo de sistemas enfocados a la gestión de información.

El objetivo de los IMS está enfocado a hacer un uso eficiente de toda la información acumulada en las organizaciones para la toma de decisiones. Para ello, explotan todas las fuentes de información de la organización, incluidas las fuentes que son almacenadas en documentos textuales. En el caso particular de los datos de tipo texto, se están convirtiendo en la principal fuente de información en las organizaciones. Dicha información textual es generada por informes de trabajo, los documentos que forman parte de las Intranet, páginas Web, publicaciones, correo electrónico, etc. Aparte de su función de "Memoria de la Organización" , ésta información histórica, es útil para predecir información futura. De aquí la importancia de lograr optimizar los procesos que implican el tratamiento de este tipo de información proveniente de fuentes textuales.

Cada uno de estos sistemas están implementados de manera tal, que puedan manejar eficientemente los tipos de datos para los que están diseñados. Para ello, implementan diferentes modelos lógicos y físicos de datos, lenguajes de consultas y técnicas de procesamiento de consultas. También es factible encontrar diferentes arquitecturas para lograr la integración entre dos o más IMS, con el objetivo de hacer su uso extensivo a otros tipos de datos, reutilizar funcionalidades ya implementadas, etc.

1.1. Planteamiento del problema

La falta de estructura de los datos textuales hace difícil su procesamiento automático para manejarlos de forma masiva. Esto sucede no sólo en repositorios de textos, si no también en atributos de tipo textual en bases de datos relacionales, como las de tipo médico (Mack y Hehenberger, 2002; Prados

et al., 2006), encuestas, bases de datos de correos electrónicos, etc. Existen diversas áreas de investigación tales como el Procesamiento de Lenguaje Natural, la Minería de Textos (Tan, 1999), Recuperación de Información (del Inglés Information Retrieval (*IR*)) (VanRjsbergen, 1979; Salton, 1989), etc., que tratan con este problema e intentan resolverlo desde diferentes puntos de vista. La mayoría de las soluciones vienen de encontrar algún tipo de estructura en el texto y/o transformarlo en lo que es llamado Forma Intermedia (Tan, 1999; Delgado et al., 2002; Justicia et al., 2005).

La Minería de Textos ha emergido como un área de investigación dentro del descubrimiento de conocimiento y el procesamiento de textos, que ha tratado de rellenar el vacío que han presentado los métodos tradicionales de minería (Ciravegna, 2001). Ha sido definida también como la Minería de Datos aplicada a textos, para descubrir nuevo conocimiento no explícito en grandes colecciones de datos (Hearst, 1999).

Las metas que persigue la Minería de Textos son similares a las de la Minería de Datos; por ejemplo, también trata de descubrir agrupamientos, tendencias, asociaciones y desviaciones en en grandes conjuntos de textos. Los procesos de Minería de Textos están compuestos por dos fases principales: Preprocesamiento y Descubrimiento (Tan, 1999). En la etapa de Preprocesamiento los textos son transformados a una Forma Intermedia más estructurada que permita su análisis automático (Hearst, 1999; Tan, 1999). En la etapa de Descubrimiento, esta representación intermedia es analizada para descubrir patrones no triviales e interesantes (Gelbukh et al., 2002).

Dependiendo del tipo de método utilizado en la etapa de Preprocesamiento, la representación del texto construido variará. En la literatura se recogen un gran número de Formas Intermedias. Provenientes del campo de la IR se han trabajado algunas como las palabras, términos(Lent et al., 1997), palabras claves, frecuencia de palabras (Iritano y Ruffolo, 2001), etiquetado del documento (Feldman y Dagan, 1995; Feldman y Hirsh, 1996; Feldman et al., 1997), eventos (Karanikas et al., 2000), grafos semánticos (Gelbukh et al., 2002; Raghavan y Garcia-Molina, 2003), etiquetas XML (Winkler y Spiliopoulou, 2001), etc. Del tipo de representación que se seleccione dependerán los métodos posibles a utilizar en la etapa de Descubrimiento y, por ende, los tipos de patrones descubiertos.

En esta memoria, se aborda el problema de la obtención de una nueva Forma Intermedia de Representación, que se encargue de obtener una representación más estructurada de los atributos textuales en una base de datos Relacional Orientada a Objetos.

Basado en todo lo anteriormente expuesto, el problema de investigación del presente trabajo queda enunciado de la forma siguiente.

Problema a resolver

El tratamiento de atributos textuales en bases de datos relacionales, para realizar operaciones conjuntamente y del mismo modo que con cualquier otro tipo de dato. Estas operaciones pueden incluir desde consultas semánticas hasta resúmenes.

Para darle solución a este problema de investigación, partimos de la formulación de la hipótesis siguiente.

Hipótesis

Se puede obtener una forma de representación del conocimiento que mantenga la semántica de los atributos textuales en una base de datos, a través de técnicas de Minería de Textos que conducen a una nueva Forma Intermedia de Representación. Dicha forma de representación se puede implementar como un TDA[1] *que permita manejar dichos atributos como el resto de los atributos de la base de datos, y también obtener una estructura de conocimiento que mantenga la semántica de dichos atributos.*

Una vez planteada nuestra hipótesis, en la próxima sección se detalla el marco de trabajo en el que se desarrolla la solución al problema planteado.

[1] Tipo de dato abstracto (TDA)

1.2. Marco de trabajo

La idea general de este trabajo es que, una vez que dichos atributos se encuentren representados de una forma más estructurada, puedan ser tratados como el resto de los atributos de la base de datos. La recuperación de la información en los Sistemas de Gestión de Base de Datos (SGBD) está soportada por consultas declarativas con respuestas exactas. Por el contrario, en sistemas de IR las consultas son imprecisas y las respuestas aproximadas. En este sentido, la solución aquí presentada, se enmarca en extensiones de un SGBD (*en nuestro caso con modelo Relacional Orientado a Objetos*) para dotarlo de capacidades de representación y consultas sobre atributos textuales. Para ello, se utilizará una nueva Forma Intermedia propuesta en esta memoria. El Sistema de Gestión de Base de Datos (SGBD) utilizado en nuestro caso es PostgreSQL (PostgreSQL-Global-Devel-Group, 2008b).

Para el caso de las extensiones de almacenamiento, se logró la representación de las estructuras del modelo propuesto mediante un TDA. También se implementaron las extensiones referentes a la extracción de información, mediante la implementación de las operaciones de dicho modelo como métodos del TDA.

En la literatura existen antecedentes de extensión de un SGBD con capacidades de recuperación de textos; aparecen enfoques tales como la extensión del modelo y álgebra relacional, extensiones al lenguaje de consulta de la base de datos, nuevas estructuras de índices (Lynch y Stonebraker, 1988) y tipos de datos (Whitemarsh-IS-Corp., 2000), y estrategias para la ejecución de consultas que optimicen la recuperación de textos.

En el capítulo 2, se amplían los antecedentes que aparecen en este entorno, cuyos enfoques más actuales se aprecian en dos categorías fundamentales:

1. *Modelos relacionales que rompen con la Primera Forma Normal (1FN):* Se rompe la tradicional condición del Modelo Relacional de almacenar una valor atómico en el valor de un atributo para cada tupla, y se permite que se almacenen colecciones. Esto se realiza con el propósito de capturar la estructura jerárquica del documento y almacenarla bajo

estas condiciones; ejemplos son los trabajos (Desai et al., 1987; Zobel et al., 1991; Davis et al., 1995).

2. *Modelos probabilísticos:* Se incorporan elementos de incertidumbre e imprecisión en los SGBD con el objetivo de flexibilizar las consultas, adaptándose de esta forma a la imprecisión y respuestas aproximadas que caracterizan a la IR en textos. Algunos ejemplos son los trabajos (Dalvi y Suciu, 2004, 2005).

Desde el punto de vista de la clasificación anterior, la solución presentada en este trabajo está enmarcada en romper con la 1FN. Se considera una base de datos con Modelo Relacional Orientado a Objetos, donde los atributos textuales, pasan a ser definidos como un TDA en base a la Forma Intermedia propuesta. Para la implementación de dicha solución, se han estudiado las posibilidades de implementación en SQL:99 estándar, lo que derivaría en SGBD como Oracle (Oracle-Corp, 2007), IBM DB2 (IBM-Corp., 2008), etc. Como se comentó anteriormente, también se estudiaron las posibilidades de implementación que brinda PostgreSQL, gestor de bases de datos dentro del Software Libre, y donde finalmente se realizó la implementación.

Con respecto a enfoques similares que aparecen en la literatura, estos se basan, en sentido general, en IMS que trabajan con colecciones de documentos completos que se estructuran generalmente en formato XML, y no con textos cortos como es el caso que nos ocupa, que estructuramos mediante la nueva Forma Intermedia de representación. Se pueden encontrar diferentes tipos de IMS (Goldman y Widom, 2000; Fegaras y Elmasri, 2001) que utilizan diferentes arquitecturas (Raghavan y Garcia-Molina, 2001) con este fin.

Entre estos trabajos, el enfoque más similar al aquí presentado, es el de los SGBD Semiestructurados (Carey et al., 2000; Lahiri et al., 2000). Este tipo de IMS realiza la transformación de la representación original de un atributo textual a una representación más estructurada, usando el formato XML. Para ello, requieren de especificaciones para realizar el mapeo del atributo textual con su representación XML. Este tipo de sistemas implementa extensiones en su lenguaje de consulta para responder a las consultas tradicionales del ámbito de IR, entre ellas, palabras claves, recuperación basada en la similitud, clasificación automática y agrupamiento.

En nuestro caso, se realiza la transformación del atributo textual (textos cortos) a una forma más estructurada. Se utiliza para ello, técnicas de Minería de Textos basadas en Reglas de Asociación (discutidas en el anexo B). Con ellas se obtienen los itemsets[2] frecuentes que mantienen la semántica que contiene el atributo textual procesado. Dicha representación es almacenada en un nuevo atributo de la base de datos, utilizando un TDA y, dentro de éste, el tipo de datos *Array* que incluye el Modelo Relacional Orientado a Objetos. Dicho TDA se almacena para cada tupla de la base de datos y contendrá las agrupaciones lógicas de términos que le corresponden a cada tupla.

Como ya se comentó, desde el punto de vista semántico, la Forma Intermedia obtenida mantiene la semántica que contiene la frase, al obtener los itemsets frecuentes. Dichos itemsets mantienen la agrupación de los términos relevantes para cada tupla. Además, el sistema de esta forma logra realizar consultas semánticas utilizando un tratamiento de sinonimia; recuperando así, información sobre términos que tienen el mismo sentido semántico aunque tengan sintaxis diferente.

Una vez planteado el problema de investigación y el marco de trabajo, a continuación se enuncian los objetivos específicos de esta memoria.

1.3. Objetivos específicos

Como se comentó anteriormente, el objetivo fundamental en este trabajo, es lograr una nueva Forma Intermedia de representación del conocimiento para los atributos textuales de bases de datos. Para ello, se utilizarán técnicas de Minería de Textos y su implementación será llevada a cabo mediante un TDA. También se realizarán las extensiones necesarias al SGBD que se utiliza para la implementación, con vistas a dar respuesta a consultas semánticas, sobre la representación obtenida para dichos atributos.

[2] A lo largo de este trabajo, se utiliza el término ' itemsets ' para referirse al conjunto de items (elementos) frecuentes obtenidos con el algoritmo Apriori. Cuando se use su versión en singular se utilizará el término ' itemset '. Se mantienen estos términos en el idioma Inglés por ser un término acuñado y muy extendido en el ámbito de Minería de Datos.

Todo lo anteriormente expuesto se concreta en una serie de objetivos específicos o tareas a realizar que son las siguientes:

1. *Realizar un estudio de los antecedentes de los sistemas y técnicas que conciernen al problema planteado y la hipótesis para su solución.* Con ello pretendemos situarnos en el estado actual del problema a resolver y estudiar las técnicas necesarias dentro de la solución planteada.

2. *Definición formal del modelo matemático que responde a la nueva Forma Intermedia de Representación obtenida.* Ésta nos proporcionará las estructuras y operaciones que constituyen el modelo abstracto de la Forma Intermedia propuesta en esta memoria.

3. *Estudiar la factibilidad y realizar la implementación del modelo obtenido a través de un TDA en SQL:99 y PostgreSQL.* Es decir, realizar las transformaciones necesarias para lograr la implementación en PostgreSQL del modelo abstracto aquí propuesto.

4. *Plantear la metodología y la arquitectura del sistema para, partiendo de un atributo textual, obtener su representación como un TDA en la base de datos.* Una vez que se ha obtenido y estudiado la factibilidad de implementación de la Forma Intermedia, será necesario formular una metodología y arquitectura del sistema, que contenga de forma detallada todos los pasos a seguir para la obtención de las estructuras del modelo.

5. *Desarrollo y validación de una aplicación software que automatice el proceso de limpieza de datos y la generación de la Forma Intermedia obtenida.* Dicha aplicación se encargará de realizar las tareas necesarias dentro del módulo de Preprocesamiento y obtención de Forma Intermedia, de la arquitectura del sistema propuesto.

6. *Desarrollo y validación de un cliente de consulta que permita la realización de consultas semánticas sobre atributos textuales en base de datos.* Para que estos resultados sean realmente eficaces, será necesario contar con una herramienta software que lleve a la práctica los resultados

teóricos previamente obtenidos. Es por ello que este cliente de consulta permitirá al usuario especificar sus consultas sobre las estructuras obtenidas con la nueva Forma Intermedia.

7. *Comparativa del modelo propuesto con operadores tradicionales de recuperación de textos en bases de datos y validación de los resultados obtenidos.* Por último, para corroborar la calidad del modelo propuesto y las ventajas que introduce sobre la realización de consultas sobre atributos textuales en bases de datos, será necesario realizar comparativas donde se evalúen las respuesta a consultas sobre este tipo de atributos, de la forma tradicional, y utilizando el modelo propuesto. También será necesario realizar la validación de los resultados obtenidos en dicha comparativa, mediante métricas establecidas con este fin en el campo de la Recuperación de Información.

Durante la consecución de los objetivos antes planteados, consideramos que se han llegado a realizar las aportaciones que se relacionan a continuación. Un resumen del cumplimiento de ellas, aparece en el capítulo 8 de esta memoria.

1.4. Aportaciones

Desde el punto de vista teórico y de diseño este trabajo realiza las siguientes aportaciones:

1. Se ha obtenido una nueva Forma Intermedia de Representación para atributos de tipo texto corto en bases de datos. Dicha Forma Intermedia se ha formalizado matemáticamente mediante un modelo de datos abstracto, con sus operaciones asociadas.

2. Se ha realizado el modelado de las estructuras que soportan el modelo propuesto en OQL (en Inglés Object Query Language), SQL:99 y PostgreSQL.

3. Se ha planteado una metodología de desarrollo para la consecución de la Base de Datos Relacional Orientada a Objetos que contiene la

nueva representación de los atributos textuales, así como de la Base de Conocimiento asociada al procesamiento de dichos atributos.

4. Se ha obtenido una arquitectura del sistema que permite integrar las herramientas y procesos planteados en la metodología obtenida.

Desde el punto de vista tecnológico se han desarrollado las siguientes aportaciones:

1. Se ha implementado una herramienta integrada que responde a los pasos de la metodología presentada, para la limpieza y transformación de un atributo textual a la nueva Forma Intermedia de Representación obtenida.

2. Se han realizados las extensiones necesarias en el SGBD PostgreSQL para implementar las funciones de consultas, que responden a la operaciones del modelo obtenido.

3. Se ha implementado un cliente de consulta para la realización de consultas semánticas sobre los atributos textuales, combinándolos con el resto de los atributos de la base de datos.

1.5. Contenidos de la memoria

La memoria está organizada de acuerdo a los objetivos antes mencionados. En primer lugar, tras esta introducción, se profundiza en los antecedentes del problema que nos ocupa y la solución que se la ha dado. Con este objetivo, a continuación se pasa al **Segundo Capítulo: Antecedentes,** donde se estudian los antecedentes del problema y de la solución aquí presentada. Dentro de los antecedentes del problema, se discutirá sobre el entorno que caracteriza el problema abordado y los tipos de soluciones que aparecen en la literatura. Para ello, se discuten los diferentes tipos y arquitecturas de IMS, enfocados a IR en textos. Dentro de los antecedentes de la solución se discutirá el concepto y procesos que forman la Minería de Datos y la

Minería de Textos. También se abordan el proceso de limpieza de datos y las diferentes variantes de formas intermedias que existen, para dotar de estructura a atributos textuales.

En el **Tercer Capítulo: Propuesta teórica de la forma intermedia de representación**, nos centramos en la definición de las estructuras en las que se transforman los atributos textuales y sus operaciones. Se introducen los conceptos de conjunto-AP y estructura-AP como estructuras básicas que componen el modelo de representación que se ha obtenido. También se introducen un conjunto de operaciones y medidas que permitirán realizar operaciones de búsqueda sobre las estructuras de conocimiento asociadas a los atributos textuales procesados. Al final del capítulo se desarrolla un ejemplo práctico, donde se explicitan todas las operaciones que se introducen a lo largo del capítulo.

El **Cuarto Capítulo: Implementación del modelo**, está dedicado al modelado conceptual de las estructuras. Se propone una metodología que permita pasar del modelo matemático obtenido anteriormente, a su implementación concreta en un gestor de base de datos. También se desarrolla la implementación del modelo propuesto en las interfaces que implementa ODL, y su implementación mediante un TDA en SQL:99 y PostgreSQL. Además, se define la estrategia para la implementación del modelo en un gestor Relacional Orientado a Objetos.

En el **Quinto Capítulo: Del atributo textual a la estructura-AP**, basados en las estructuras matemáticas y sus operaciones que se establecen en el capítulo 3, se muestra cómo se implementaron y almacenaron dichas estructuras. Para ello, se utilizarán los aspectos discutidos en el capítulo 4 sobre la implementación de un TDA. Se describe el sistema propuesto y se discute su arquitectura. También se definen los pasos de la metodología obtenida para obtener la estructura-AP partiendo de un atributo textual. Además, se describe cómo se implementó la obtención de la estructura-AP global de conocimiento, utilizando una base de datos médica. Finalmente, se muestra cómo se puede obtener la estructura-AP inducida de cada tupla de la base de datos.

En el **Sexto Capítulo: Consultando el sistema: El cliente de consul-**

ta, a través de varios ejemplos se dan las ideas básicas del tipo de consultas que el usuario puede realizar en un sistema que soporte el modelo propuesto. También se aborda la implementación del cliente de consulta. Para ello, se muestra su arquitectura y las funcionalidades que incorpora. Finalmente, se muestran las características de las que se han dotado a las funciones que trabajan sobre las estructuras definidas en PostgreSQL, para permitirle mayor flexibilidad al usuario a la hora de expresar sus consultas.

El **Séptimo Capítulo: Evaluación del modelo**, está dedicado al uso de medidas provenientes del campo de la Recuperación de Información, para medir la capacidad del sistema de recuperar las tuplas relevantes y determinar la precisión de la información obtenida. También se realiza un análisis comparativo de los resultados obtenidos en consultas similares ejecutadas en PostgreSQL, utilizando las operaciones del modelo propuesto, y operadores como el de igualdad y *Like*. Dichos análisis se realizan para los modelos de relevancia booleana y difusa que se definen. La experimentación se realiza utilizando los conjuntos experimentales definidos en el capítulo 5.

Por último, en el **Octavo Capítulo: Conclusiones y trabajos futuros**, se encuentran las conclusiones obtenidas con la realización de este trabajo y las líneas de investigación a seguir en el futuro.

Con el objetivo de facilitar la mejor comprensión de este trabajo se han incluido dos apéndices donde se recogen detalles de las tecnologías de bases de datos y Minería de Datos utilizadas en la tesis. En el **Apéndice A: El modelo Relacional Orientado a Objetos: Implementación en PostgreSQL**, se discuten las características y elementos que distinguen al Modelo de Datos Relacional Orientado a Objetos del modelo Relacional puro. También se discuten elementos de SQL:99 y PostgreSQL como gestor de bases de datos dentro del Software Libre, bajo este entorno Relacional Orientado a Objetos. En el **Apéndice B: Extracción de Reglas de Asociación: Implementación del algoritmo Apriori**, se discutirán los elementos fundamentales de las Reglas de Asociación, del algoritmo Apriori que se utilizó para la obtención de la Forma Intermedia y los itemset frecuentes. También se dan las características y parámetros de configuración fundamentales de la herramienta integrada que se implementó, para los procesos de limpieza de datos y obtención de las estructuras del modelo propuesto.

Capítulo 2

Antecedentes

El objetivo de este capítulo es dar una idea general sobre la investigación relacionada con el problema que nos ocupa. Dicho problema, como se comentó anteriormente, es la representación del conocimiento para la realización de consultas semánticas sobre atributos textuales en bases de datos relacionales. Para ello, utilizaremos técnicas de Minería de Texto que conducen a una forma de representación intermedia y su implementación utilizando un TDA. Dividiremos los contenidos abordados a lo largo del capítulo en antecedentes del problema y antecedentes de la solución.

En los apéndices A y B de esta memoria se incluye con más detalle, algunos conceptos y las herramientas utilizadas en la solución al problema planteado. En ellos se discutirá sobre el modelo de base de datos Relacional Orientado a Objetos, y la implementación que de este modelo realiza el SGBD PostgreSQL donde se realizó la implementación del modelo propuesto. También sobre Reglas de Asociación, algoritmo Apriori, itemset frecuentes, etc. Debido a esto, en el presente capítulo nos centramos más en los antecedentes específicos del problema.

Dentro de los antecedentes del problema se discutirá sobre el entorno que caracteriza el problema abordado y los tipos de soluciones que aparecen en la literatura. Para ello se describen los diferentes tipos de IMS, enfocados generalmente a recuperación de información en textos. También se dan las diferentes arquitecturas que han surgido con la idea de la integración de

dichos sistemas, y se profundiza en las variantes aparecidas en el entorno que nos ocupa de atributos textuales en bases de datos.

Dentro de los antecedentes de la solución, se discuten el concepto y los procesos de la Minería de Datos y la Minería de Texto. También se abordan el proceso de limpieza de datos y las diferentes variantes de formas intermedias que existen, para dotar de estructura a atributos de tipo texto.

En la próxima sección a partir del enunciado del problema de investigación, se discuten sus antecedentes según la bibliografía consultada.

2.1. Antecedentes del problema

Antes de discutir los antecedentes del problema de investigación, retomamos su enunciado que fue dado anteriormente:

Problema de investigación: *El tratamiento de atributos textuales en bases de datos relacionales, para realizar operaciones conjuntamente y del mismo modo que con cualquier otro tipo de dato. Estas operaciones pueden incluir desde consultas semánticas hasta resúmenes.*

Existen varios acercamientos en la literatura integrando IMS con datos estructurados, semiestructurados y no estructurados (Raghavan y Garcia-Molina, 2001). En todos ellos, la arquitectura del sistema primitivo es modificada en algunos aspectos para soportar la gestión de otros tipos de datos. Tres tipos de IMS son considerados con este propósito[1] (Raghavan y Garcia-Molina, 2001):

1. *Sistemas de Recuperación de Textos sobre SGBD Relacional u Orientado a Objetos:* Se ocupan de la gestión y la recuperación basada en consultas, de colecciones de documentos de textos no estructurados (Salton, 1989, 1991; Witten et al., 1999). El sistema de recuperación de

[1]Los autores para esta clasificación ignoran los IMS no textuales (audio, imágenes y video), así como sistemas que son diseñados para tipos de datos específicos, como por ejemplo, los Sistemas de Información Geográfica.

textos explota las potencialidades del SGBD subyacente sobre el que se implementa, entre ellas, su modelo de datos, lenguaje de consulta, gestión de recuperación, gestión de seguridad, etc.

2. *Sistemas de Recuperación de Textos en modelos Semiestructurados:* Este tipo de IMS es similar al anterior y se encarga de la gestión de datos estructurados, por ejemplo, los datos que conforman un esquema bien definido (Molina et al., 2000; Ullman y Windom, 1997). Para ello, se centran en estructurar los documentos aprovechando las facilidades del SGBD Relacional u Orientado a Objetos que utilizan, fundamentalmente en formato XML (*Extensible Markup Language*).

3. *Sistemas de Gestión de Base de Datos Semiestructurados (XML):* Son SGBD Relacional u Orientado a Objetos, que están diseñados para gestionar eficientemente los datos que sólo conforman parcialmente un esquema, o cuyo esquema puede evolucionar rápidamente (Abiteboul, 1997; Abiteboul et al., 1999; Whitemarsh-IS-Corp., 2000). Este tipo de IMS está enfocado a la recuperación de textos, explotando las potencialidades que implementan estos gestores de estructurar documentos en formato XML (*es el caso contrario de la clasificación anterior*).

Cada uno de estos tres tipos de sistemas están implementados de manera tal que puedan manejar eficientemente los tipos de datos para los que están diseñados. Para ello, implementan diferentes modelos lógicos y físicos de datos, lenguajes de consultas y técnicas de procesamiento de consultas. En (Raghavan y Garcia-Molina, 2001) se puede encontrar una tabla resumen de los modelos y lenguajes empleados por estos sistemas.

A pesar de que esta clasificación es factible de encontrar en la literatura en los trabajos antes mencionados, en la práctica también ha habido un gran interés por combinar, integrar, y lograr la interoperabilidad de estos tres tipos de IMS. Según Raghavan en (Raghavan y Garcia-Molina, 2001), han existido dos motivaciones fundamentales para la mayoría de los trabajos que han surgido en este sentido:

1. *Muchas aplicaciones necesitan procesar tipos de datos que pertenecen a varias categorías:* Por ejemplo una aplicación de procesamiento de

ordenes que puede necesitar gestionar la información del inventario que se encuentra en una base de datos relacional, así como, procesar ordenes de compras que pueden ser recibidas en formato semiestructurado como documentos XML.

2. *Existen muchos avances en reutilizar las facilidades proporcionadas por un IMS para implementar otro:* Por ejemplo un sistema de recuperación de textos que sea construido sobre un SGBD Relacional u Orientado a Objetos. En este caso se aprovechan las posibilidades de gestión avanzada de concurrencia y posibilidades de recuperación que implementa el SGBD, sin necesidad de implementarlas desde cero.

No obstante, los esfuerzos para realizar la integración de este tipo de sistemas suelen ser desafíos importantes. Dependiendo del escenario donde se vaya a realizar la integración, el tipo de sistema a utilizar, los tipos de datos involucrados, etc., así será la solución que se debe adoptar. Algunas de las tendencias principales a la hora de realizar la integración entre diferentes IMS suelen ser (Raghavan y Garcia-Molina, 2001):

1. *Arquitectura por capas:* Los sistemas con esta arquitectura son implementados como un tipo de IMS que opera en una capa superior, como una aplicación que utiliza otro tipo de IMS que funciona en una capa inferior. La principal ventaja de esta arquitectura es que el IMS que opera en la capa superior puede hacer uso de las facilidades que implemente el IMS subyacente. Por ejemplo, el control de concurrencia, recuperación, estructura de índices, etc. sin un esfuerzo significativo y con poco costo de tiempo. Sin embargo, el esfuerzo aparece al tener que acoplar los tipos de datos y operadores utilizados por el IMS de la capa superior de la jerarquía, con los tipos y operadores soportados por el IMS subyacente, así como, maximizar el rendimiento de las consultas.

2. *Integración a través de una capa intermedia:* Esta arquitectura encapsula la lógica de la integración de dos o más tipos de IMS, en una sola capa intermedia. Esta capa garantiza la interfaz unificada para el acceso transparente al sistema integrado, usando sus propios tipos de

datos y lenguajes de consulta. El principal desafío de esta arquitectura, es diseñar un mecanismo eficiente para transformar las consultas especificadas en la interfaz unificada, en consultas con las capacidades soportadas en cada uno de los IMS que componen el sistema integrado. La principal ventaja de esta arquitectura, a diferencia de las otras dos, es que las modificaciones sobre los IMS que se integran son mínimas o innecesarias.

3. *Agregar extensiones a un tipo de sistema:* Esta arquitectura mejora las capacidades de un determinado IMS, agregándole un módulo de extensión que da soporte a nuevos tipos de datos, operadores, o lenguajes de consultas, usualmente disponibles sólo en otro tipo de IMS. Cuando la interfaz de la extensión está disponible en el IMS original (como es el caso de la mayoría de los SGBD Relacionales comerciales (IBM-Corp, 2008; Oracle-Corp, 2008)), el módulo de extensión puede ser implementado utilizando dichas interfaces. De lo contrario, el IMS original es modificado para que de forma nativa soporte las nuevas características.

A pesar de que existen estas tres arquitecturas posibles para lograr la integración entre diferentes tipo de IMS, en algunas ocasiones se encuentran trabajos que mezclan éstas, o realizan algunas variantes de las mismas. Por ejemplo en (Goldman y Widom, 2000) por razones de eficiencia, bajo la arquitectura de integración a través de una capa intermedia, uno de los IMS que se integran contiene físicamente la capa de integración. En (DeFazio et al., 1995) se propone la extensión a un SGBD para facilitar la implementación eficiente de índices invertidos (la estructura más elemental para recuperación de palabras, construye la lista de términos y los documentos donde aparece). En esta solución la capa de extensión y el sistema de recuperación de textos quedan soportados sobre el SGBD mejorado con dichas extensiones.

También estos tres tipos de arquitecturas de integración aparecen relacionadas en la literatura con la clasificación que se discutió anteriormente de tipos de IMS (*Sistemas de Recuperación de Textos sobre SGBD Relacional u Orientado a Objetos*, *SGBD Relacionales u Orientado a Objetos* y *SGBD*

Semiestructurados). En el siguiente apartado se discute esta clasificación y se enmarca la arquitectura a utilizar en la solución de nuestro problema.

2.1.1. Clasificación de IMS según la arquitectura de integración que implementan

A continuación se realiza la clasificación de un grupo de trabajos que han aparecido referentes a la integración de IMS. En la tabla 2.1 se muestra un resumen con algunas actualizaciones de una tabla similar que aparece en (Raghavan y Garcia-Molina, 2001). Como se puede observar, aparecen diferentes trabajos de la literatura clasificados según el tipo de IMS en que puede ser clasificado, y el tipo de arquitectura que implementa.

	Recuperación de Textos en modelo Relacional/OO	**Recuperación de Textos en modelos Semi-estructurados**	**SGBD Relacionales/OO Semiestructurados**
Arquitectura por capas	(DeFazio et al., 1995) (Grossman y Driscoll, 1992) (Grossman et al., 1997)	(Hayashi et al., 2000)	(Fegaras y Elmasri, 2001) (Schmidt et al., 2000) (D.Lee, 2002)
Integración a través de una capa intermedia	(Fagin, 1998) (Fagin et al., 2001)	(Adar, 1998)	(Manolescu et al., 2001) (Christophides et al., 2000)
Agregar extensiones a un tipo de sistema	(Vasanthkumar et al., 1996) (IBM-Corp, 2008) (Oracle-Corp, 2008)	(Goldman y Widom, 2000) (Fuhr y Grossjohann, 2001)	(Carey et al., 2000) (Lahiri et al., 2000) (Oracle-Corp-XML, 2008)

Tabla 2.1: Ejemplos de clasificación de la literatura sobre la integración de diferentes tipos de IMS.

La solución propuesta al problema planteado en este trabajo, se enmarca en la tercera fila de la tabla 2.1, es decir, se trata de una extensión agregada a un tipo de sistema, que en nuestro caso es un SGBD Relacional Orientado a

Objetos. Por esta razón nos centramos en comentar algunas de las soluciones propuestas en la literatura para el caso de agregar extensiones a un tipo de sistema.

El primer tipo de sistema a considerar se basa en la IR sobre colecciones completas de documentos. En él se trata de hacer extensiones al IMS con el objetivo de aprovechar las potencialidades del SGBD que utiliza.

Bajo este entorno, la mayoría de las soluciones dadas para combinar datos estructurados y no estructurados han sido estudiadas considerando documentos de texto completo en una base de datos Relacional u Orientado a Objetos (Hiemstra et al., 2003; Grossman et al., 1997; Mack et al., 2004; Whitemarsh-IS-Corp., 2000), que como se explicó no es el caso que nos ocupa. En nuestro caso se trabaja con textos cortos en bases de datos con un número reducido de palabras que conforman un vocabulario potencialmente controlado. Algunos ejemplos de esta situación son los atributos *intervención propuesta* y *diagnóstico* de una base de datos médica, o el atributo *evaluación cualitativa* en una base de datos de estudiantes universitarios.

Dentro de este tipo de sistemas que tratan de integrar IR con SGBD, tanto la comunidad de las bases de datos como la de IR han realizado aportaciones, pero con diferentes metas y adoptando diferentes arquitecturas. La comunidad de base de datos ha propiciado la extensión de su arquitectura con la meta de proveer eficientemente características de IR dentro de los propios SGBD (última fila de la tabla 2.1). Aquí se incluyen también los SGBD Orientados a Objetos (O2-Tech-Inc, 2007) y Semiestructurados con extensiones para el tratamiento preferentemente del formato XML (Oracle-Corp-XML, 2008; IBM-Corp, 2008).

Por su parte, la comunidad de IR ha mostrado preferencia por la *arquitectura por capas*, con la finalidad de explotar las características de los SGBD (control de concurrencia, seguridad, robustez, etc.) para construir sistemas de IR más escalables y robustos(celda correspondiente a la fila uno y columna uno de la tabla 2.1).

El segundo tipo de IMS bajo arquitectura de *agregar extensiones* son los que se encargan de hacer Recuperación de Textos en modelos Semiestructurados (soportados por *SGBD Relacionales u Orientado a Objetos*). En (Raghavan

y Garcia-Molina, 2001) se dice que son IMS que se centran en la explotación del estándar XML para la recuperación de textos semiestructurados, aprovechando las características de las bases de datos que dan soporte a XML.

Esta aproximación trata el contenido del fichero XML como una colección de documentos de textos estructurados lógicamente. Para lograrlo, realiza la extensión de los modelos e índices de IR para codificar la estructura y la semántica del documento XML. Esto hace posible aplicar técnicas bien conocidas en IR y dar soporte a búsquedas de palabras claves, recuperación basada en la similitud, clasificación automática y agrupamiento (clustering).

Como se puede observar, la clasificación anterior tampoco es el ámbito que nos ocupa pues se basa en IR, explotando las posibilidades de estructuración, normalmente a través del formato XML, que aporta el SGBD subyacente que utiliza. Nuestro problema, como se ha discutido, ni es del ámbito de IR, ni la solución planteada se basa en la estructuración de documentos mediante formato XML.

Finalmente, la tercera clasificación de IMS bajo esta arquitectura, son los SGBD semiestructurados. Este es el caso contrario al anterior; aquí el IMS es propiamente el SGBD, que se aprovecha de las facilidades que han incorporado estos sistemas para publicar datos relacionales con formato XML. Una vez en este formato, el propio SGBD se encarga de realizar la recuperación de información textual. Para ello, los datos relacionales son transformados bajo la perspectiva del formato XML. También se realizan extensiones a los motores de consultas relacionales u orientado a objetos, para permitir consultas sobre los datos en XML.

Según (Raghavan y Garcia-Molina, 2001), existen dos partes en el diseño de un sistema que convierte relaciones en documentos XML. La primera es la necesidad de un lenguaje que permita especificar la conversión y el mapeo entre las relaciones y los documentos XML. La segunda es una estrategia de implementación eficiente para llevar a cabo dicha conversión.

El sistema SilkRoute descrito en (Fernandez et al., 2000) fue uno de los primeros prototipos de investigación que permitió la generación automática de documentos XML desde tablas relacionales. Este sistema utilizó un lenguaje llamado RXL (*Relational-XML query language*), basado en una combinación

de los lenguaje de consulta SQL y XML, para especificar el mapeo de tablas relacionales con una Definición de Tipo de Documento (en Inglés Document Type Definition (*DTD*)) arbitrario. Usando dicho lenguaje, es posible definir vistas XML de los datos relacionales. Para la eficiencia de las consultas sobre las vistas XML, el sistema materializa sólo la porción de XML necesaria para responder la consulta.

En sentido general la clasificación anterior se refiere a IMS que trabajan con documentos completos y no a textos cortos como es el caso que nos ocupa. Este último tipo de IMS con la arquitectura de extensión, es el que más se acerca a nuestra solución, aún cuando difieren en varios aspectos. En este caso se trata de un SGBD que está haciendo la transformación de la representación original de un atributo textual a una representación más estructurada, usando el formato XML.

En el caso de la solución aportada en el presente trabajo, no se puede enmarcar completamente en ninguna celda de la clasificación de la tabla 2.1. Como se comentó anteriormente, la solución implementada claramente se corresponde con la tercera fila de dicha tabla, que se refiere a *agregar extensiones a un tipo de sistema,* pero no se enmarca completamente en ninguno de los tres tipos de sistemas en los que se hace dicha clasificación. De las tres soluciones descritas anteriormente para la arquitectura de extensión, se puede afirmar que la nuestra, es una variante de la última clasificación descrita, la referente a un SGBD semiestructurado.

En este tipo de sistema es el propio SGBD el que realiza la transformación del texto a una forma más estructurada, e implementa las extensiones necesarias para realizar las consultas sobre esta nueva representación. Tomando como referencia la explicación dada de este último tipo de sistema, en la figura 2.1 se muestra la diferencia de ambas propuestas.

Como se puede observar, los SGBD semiestructurados realizan la transformación del texto original a XML. Para ello parten de la tupla original en modelo relacional, que puede estar conformada por atributos de distintos tipos, entre ellos, los de tipo texto. Los SGBD semiestructurados, como se explicó anteriormente, realizan la transformación del texto original a formato XML utilizando una combinación de SQL con XML. En este punto requie-

ren de especificaciones para realizar el mapeo del atributo textual con su representación XML.

Posteriormente se almacena la nueva representación del atributo textual, normalmente como un nuevo atributo de la base de datos (generalmente aprovechando las facilidades del Modelo Relacional Orientado a Objetos). Finalmente este tipo de sistemas implementa extensiones en su lenguaje de consulta para responder a las consultas tradicionales del ámbito de IR, entre ellas, palabras claves, recuperación basada en la similitud, clasificación automática y agrupamiento.

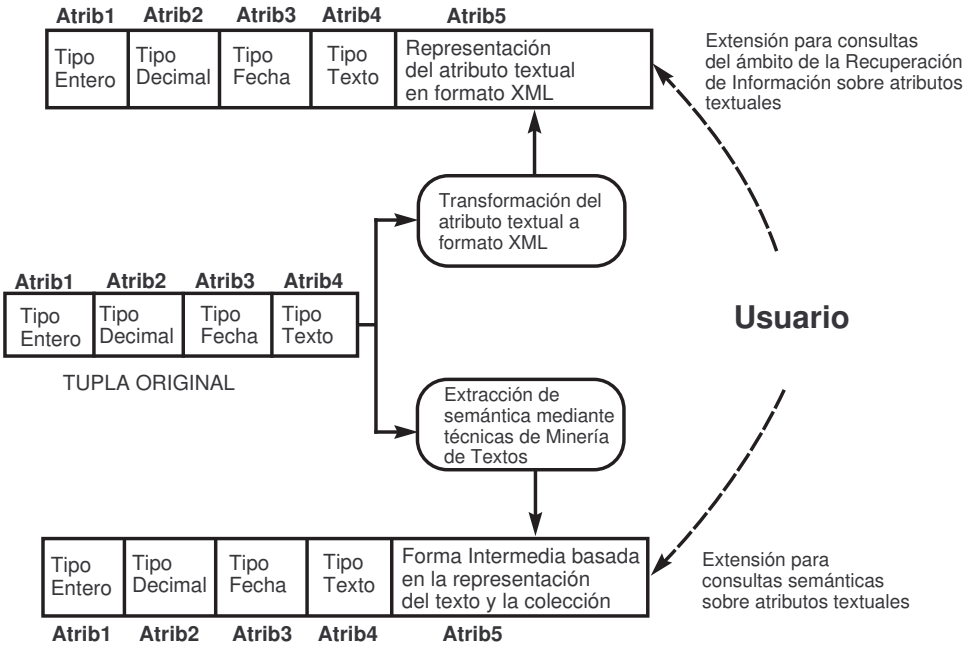


Figura 2.1: Diferencia entre una solución semiestructurada usando XML y la solución propuesta basada en Minería de Texto.

En nuestro caso, se parte también de una tupla original con la misma estructura y se realiza la transformación del atributo textual a una forma más

estructurada. Como se comentó anteriormente, en nuestro caso se considera el contenido del atributo textual, como textos cortos con un vocabulario potencialmente controlado.

Para obtener la representación del texto, se usan técnicas de Minería de Texto basadas en Reglas de Asociación. Con ellas se obtienen los itemsets frecuentes que captan gran parte de la semántica que contiene el atributo textual procesado. Dicha representación es almacenada es un nuevo atributo de la base de datos, utilizando un TDA y dentro de éste el tipo de datos *Array* que incluye el Modelo Relacional Orientado a Objetos. Dicho TDA se almacena para cada tupla de la base de datos y contendrá las agrupaciones lógicas de términos que le corresponden a cada tupla.

Una vez que se tiene el texto en esta nueva representación, nuestro modelo implementa extensiones al SGBD Relacional Orientado a Objetos que le da soporte. En este caso se implementan las funciones específicas que permiten la realización de consultas semánticas sobre el atributo textual.

Teniendo en cuenta todo lo anterior, podemos decir que el entorno de trabajo que nos ocupa viene siendo como una cuarta categoría dentro de esta arquitectura de *agregar extensiones.* Sería entonces la *arquitectura de extensión aplicada al tipo de sistemas que están soportados sobre un SGBD con modelo de datos Relacional Orientado a Objetos y que realiza consultas semánticas sobre textos cortos en bases de datos.*

Con todos estos precedentes, en la próxima sección se enunciará la solución dada al problema de investigación y se discutirán los antecedentes de las técnicas utilizadas en dicha solución.

2.2. Antecedentes de la solución

Antes de discutir los antecedentes de la solución dada al problema de investigación, recordamos en qué consiste la solución propuesta:

Solución al problema de investigación: *Obtener una forma de representación del conocimiento que mantenga la semántica de los atributos textuales*

en una base de datos, a través de técnicas de Minería de Texto que conducen a una nueva Forma Intermedia de Representación. Dicha forma de representación se puede implementar como un TDA que permita manejar dichos atributos como el resto de los atributos de la base de datos, y también obtener una estructura de conocimiento que mantenga la semántica de dichos atributos.

Consideramos necesario resaltar dos aspectos importantes del planteamiento de la solución al problema de investigación:

1. En la solución aquí presentada se obtiene la representación del conocimiento de atributos textuales en bases de datos, con vistas a la realización de consultas semánticas.

2. La solución dada al problema es general y por ende permite su extensión a otros ámbitos donde se tiene el mismo problema con el manejo de atributos textuales. Algunos, como se mencionó anteriormente, pueden ser procesos de sumarización, Datawarehousing y OLAP, etc.

En la actualidad se cuenta con múltiples modelos para la representación de información, con distintas características en función de la semántica que se le quiera dar a los datos. Existen modelos pensados para el almacenamiento de características, otros más adecuados para la representación de acciones, e incluso combinaciones de varios de estos modelos.

Desde hace unos años, se han estudiado y desarrollado una serie de técnicas, metodologías y herramientas (Klösgen, 1992; Anand y Kahn, 1993; Klösgen, 1995; Feldman et al., 1998), conocidas como procesos de *Extracción de Conocimiento en Bases de Datos* (*KDD*) (Frawley et al., 1992; Fayyad et al., 1996; Vila et al., 2000) y *Minería de Datos* (Thuraisingham, 1999) (en Inglés Knowledge Discovery in Databases y Data Mining respectivamente). Dichas técnicas abordan el problema de analizar grandes cantidades de información en busca de conocimiento hasta entonces desconocido y de posible interés.

En la actualidad, la presencia de tipos de datos no estructurados o semiestructurados ha tomado cada vez más relevancia en las organizaciones. En el

caso particular de los datos de tipo texto, se están convirtiendo en su principal fuente de información, a partir de la información textual que se acumula año tras año. Dicha información textual es generada por informes de trabajo, los documentos que forman parte de su Intranet, páginas Web, publicaciones, correo electrónico, etc. La mayoría de las decisiones de empresas, organizaciones e instituciones se basan en información de experiencias pasadas, extraídas de fuentes muy diversas, entre las que desacatan las fuentes textuales. Ésta información histórica es útil para predecir información futura.

Por el hecho anteriormente comentado, se hizo necesario, la revisión del proceso que tradicionalmente seguía el KDD para la extracción de conocimiento, enfocándolo a las características particulares de los textos. La adaptación de los procesos tradicionales de KDD al caso de los textos hizo que surgiera una basta área de investigación conocida como *Descubrimiento de Conocimiento en Textos* (en Inglés Knowledge Discovery in Texts (*KDT*)) (Kodratoff, 1999), particularizando el proceso de Minería de Datos a *Minería de Texto* (en Inglés Text Mining) (Tan, 1999).

De forma casi paralela a las herramientas y técnicas que han surgido para soportar el proceso KDT, en el entorno de las bases de datos se han ido construyendo herramientas que facilitan las consultas sobre el conjunto de documentos. Como se comentó anteriormente, todo con el fin de lograr la recuperación automática o semiautomática de la información que contienen los textos, no sólo de su información explícita, si no de los patrones que estos y su relación pueden encerrar (Baeza y Ribeiro, 1999). Bajo este contexto surgieron los IMS mencionados anteriormente. Dichos IMS, como se vio anteriormente, suelen ser agrupados según el tipo de datos para el que están optimizados. En todos los casos son sistemas que emplean diferentes modelos físicos y lógicos de datos, lenguajes de consultas, y técnicas de procesamiento de consultas apropiadas para el tipo de datos que están manejando (Raghavan y Garcia-Molina, 2001).

Como ya se mencionó, la solución abordada en este trabajo se basa en la utilización de técnicas de Minería de Texto, para la representación de la información semántica que aparece en textos cortos en bases de datos. A continuación se discuten algunos elementos sobre los antecedentes de algunas de las técnicas a utilizar con este fin. Para ello se comienza introduciendo las

definiciones de estos procesos, y posteriormente se discute la relación entre ellos (al no ser el centro del presente trabajo, se darán sólo los elementos fundamentales).

2.2.1. KDD y Minería de Datos

Según Fayyad el proceso KDD *es el proceso no trivial de identificar patrones válidos, novedosos y potencialmente útiles, y en última instancia, comprensibles a partir de los datos* (Fayyad et al., 1996). KDD es un proceso iterativo, de tal manera que se puedan realizar cambios y repetir cada paso para conseguir mejores resultados (Paralic, 2001).

Como sinónimos de KDD los investigadores en esta área suelen utilizar términos como Data Archeology, Dependency Function Analysis, Information Recollector, Pattern Data Analysis o Knowledge Fishing.

Con respecto a la relación entre KDD y Minería de Datos, la idea más extendida en la literatura es la que sitúa a la Minería de Datos como la fase más importante del proceso KDD, tal como se ve en la figura 2.2.

Como se observa en dicha figura, las técnicas de Minería de Datos son consideradas como una etapa dentro del proceso completo de KDD (Paralic y Bednar, 2002). Precisamente es la etapa que se encarga de descubrir o derivar nueva información de los datos, o encontrar patrones a través de los conjuntos de datos dados o bien separando las señales de ruido (o ambos procesos a la vez). El proceso KDD concluye con la extracción de conocimiento a partir de la interpretación y evaluación de los patrones extraídos por la Minería de Datos.

Por otro lado, existen autores como Agrawal y Hearst, para los que Minería de Datos es equiparable con KDD (Agrawal y Hohn, 1996; Hearst, 1999). En (Hernández, 2002) se dice que la causa de esta equiparación es que la Minería de Datos es la fase de generación de hipótesis, la más vistosa y la que produce los resultados sobre los que trabajar y tomar decisiones. Debido a esto, no es extraño que pase a identificar todo el proceso global de KDD. Además como se plantea en (Delgado et al., 2003) no siempre es necesario

realizar todas las fases del proceso KDD por lo que se hace más difícil de distinguir la fase de Minería de Datos.

Según Agrawal la Minería de Datos se define como *el proceso de descubrimiento eficiente de patrones, desconocidos a priori, en grandes bases de datos* (Agrawal y Hohn, 1996).

De forma análoga a como ocurre con KDD, la Minería de Datos aparece referenciada en la bibliografía con diferentes términos que se suelen utilizar como sinónimos. Entre ellos se encuentran: Data Dredging, Data Harvesting, Data Archeology, Knowledge Extraction, Data/Pattern Analysis e Information Harvesting. Por ejemplo Hearst en (Hearst, 1999) menciona el concepto de Information Archeology (Terveen et al., 1993) como sinónimo de Minería de Datos.

Por su parte, las técnicas de Minería de Datos intentan obtener patrones o modelos a partir de los datos recopilados. Decidir si los modelos obtenidos son útiles o no, suele requerir una valoración subjetiva por parte del usuario. En esta área convergen técnicas fuertemente ligadas al aprendizaje automático, tales como redes neuronales, técnicas de agrupamiento, algoritmos genéticos, entre otras (Thuraisingham, 1999). Dentro del proceso se emplean algoritmos de aprendizajes clásicos, métodos estadísticos o técnicas avanzadas de base de datos (Cooley et al., 1999; Dandretta, 2002).

En (Justicia, 2004; Justicia et al., 2005; Escobar, 2007) se puede profundizar en la relación entre estos procesos KDD y Minería de Datos, los modelos y técnicas que en ellos se aplican , así como en las etapas y procesos que los conforman.

Como se comentó anteriormente, la solución presentada se basa en estos procesos de minería, pero enfocados al caso concreto de atributos textuales. En el siguiente apartado se dan las definiciones y se discute sobre la relación que existe entre KDT y Minería de Texto como tipos de procesos que se encargan de trabajar específicamente sobre este tipo de atributos.

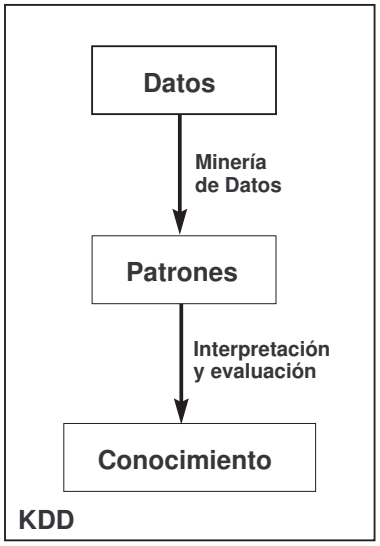


Figura 2.2: Relación entre KDD y Minería de Datos.

2.2.2. KDT y Minería de Texto

Kodratoff, en (Kodratoff, 1999), define KDT como *la ciencia que descubre conocimiento en textos*. Donde *conocimiento* es tomado con el mismo significado que es visto en KDD: el conocimiento extraído tiene que estar basado en los datos fuentes del mundo real, y modificar el comportamiento de un ser humano o agente mecánico.

Basado en el concepto anterior nosotros definimos el KDT como la ciencia que descubre conocimiento en textos propiamente y en las relaciones que entre ellos se establecen.

Dado que la información textual adolece de una estructura previamente definida, o en algunos casos posee una estructura muy compleja, en el proceso KDT algo primordial será encontrar una representación intermedia del texto. El proceso KDT implica áreas tan diversas como la recuperación de información, la extracción de información, tecnología de bases de datos y aprendizaje en bases de datos.

La fase inicial de KDT es el *Preprocesamiento* que se encarga de darle al

texto una Forma Intermedia de representación que permita realizar computaciones sobre él. Una vez que el texto se encuentra en la Forma Intermedia de Representación elegida, se realizan las técnicas de *Minería de Texto*. Por último la fase de *Visualización* es la que se encarga de mostrar al usuario, de la forma más intuitiva posible, los resultados obtenidos. Algunos acercamientos generales sobre KDT y Minería de Texto se pueden encontrar en (Feldman et al., 1998; Hearst, 1999; Kodratoff, 1999; Delgado et al., 2002; Justicia, 2004; Justicia et al., 2005; Escobar, 2007)

La Minería de Texto es usualmente considerada como un dominio de la Minería de Datos, pero en el ámbito de textos. De la misma forma en que la Minería de Datos es un fase del proceso global de KDD, la Minería de Texto puede ser vista como una fase del proceso KDT. Sin embargo, semejante a como se comentó anteriormente para el caso de la Minería de Datos, existen varias interpretaciones en las que no se distingue el proceso de Minería de Texto del proceso KDT global (Feldman y Dagan, 1995; Hearst, 1999; Kodratoff, 2001). En estos trabajos ambos términos se utilizan con un mismo significado.

De forma general, la meta principal de la Minería de Texto según (Croft, 1995; Baeza y Ribeiro, 1999) es el descubrimiento, reconocimiento o la derivación de información nueva de grandes colecciones de textos. En la literatura se recogen una serie de definiciones de Minería de Texto como pudieran ser:

1. *Proceso de extraer patrones interesantes a partir de grandes colecciones de textos para descubrir conocimiento* (Tan, 1999; Delgado et al., 2003).
2. *Descubrimiento de reglas de asociación importantes dentro del corpus del texto* (Wong et al., 2000).
3. *Instancia del descubrimiento óptimo de patrones* (Fukuda y Ñanri, 2000).
4. *Descubrimiento de información útil y previamente desconocida a partir de textos sin estructurar* (Xu et al., 2002).

Finalmente la definición de Minería de Texto con la que nos identificamos más claramente es con la que aparece en (Justicia, 2004) que plantea que

la *Minería de Texto es el proceso que descubre información útil que no está presente explícitamente en ninguno de los documentos objeto de análisis y que surge cuando se estudian adecuadamente y se relacionan dichos documentos.*

Como ya se dijo, el problema principal que surge cuando se aplican técnicas de Minería de Datos sobre atributos textuales, es su falta de estructura. Es por esto que para poder extraer conocimiento de dichos textos, se necesita proveerlos de cierta estructura. Algunos autores como (Rajman y Besançon, 1997) considera que el problema de la falta de estructura en los textos, es más bien que sí poseen una estructura implícita, lo que muy compleja de manejar. Cada documento de texto es una colección ordenada de textos y signos de puntuación, con un significado adjunto, cuya posición en el texto es controlada por complejas restricciones sintácticas y semánticas (Delgado et al., 2002).

La complejidad de la estructura implícita de los textos se puede apreciar viendo alguno de los modelos clásicos que tratan de representarla, tanto parcial como totalmente. Algunos ejemplos son los grafos conceptuales introducidos en (Schank, 1973) y los scripts (Schank, 1980). Esta complejidad también es admitida en (Salton y Buckley, 1988).

En este contexto es que la fase de *Preprocesamiento* de la Minería de Texto aporta la solución a este problema. Para ello, como se mencionó antes, se obtiene una Forma de Representación Intermedia del texto (Tan, 1999) con cierta estructura que dependerá de la forma de representación elegida. Algunos autores llaman a esta fase dentro de la Minería de Texto como Text Refining (Tan, 1999). En (Justicia, 2004) se puede encontrar la relación estrecha que existe entre el *Preprocesamiento,* la *Forma Intermedia de Representación* elegida, y el tipo de *Descubrimiento* al que ésta última puede conducir. También se puede ver un resumen de las diferentes técnicas de *Preprocesamiento* de documentos.

La limpieza de datos es otra de las tareas importantes que se acomete dentro del proceso de *Preprocesamiento*. En cualquier proceso de minería, lograr la eliminación de datos erróneos, inconsistentes, etc., reviste crucial importancia.

Dado su importancia en la solución propuesta en este trabajo, en el apartado

siguiente se profundiza en los aspectos referentes a la limpieza de datos y a la obtención de la forma intermedia de representación, como pasos dentro de la fase de Preprocesamiento previa al proceso de Minería de Texto.

2.2.3. Proceso de limpieza de datos

Teniendo en cuenta que el valor de una base de datos está en función directa a la confiabilidad de sus datos, este proceso de limpieza de datos reviste vital importancia hoy día en las aplicaciones que se desarrollan, especialmente en el ámbito de IR. Por lo general, una herramienta de limpieza de datos incluye programas que son capaces de corregir un número específico de tipos de errores, como completar números telefónicos o encontrar registros duplicados. La utilización de una herramienta de limpieza de datos puede ahorrar un tiempo significativo al administrador de la base de datos y puede ser menos costoso que arreglarlo de forma manual.

Müller, en (Müller, 2003) define que *el proceso de limpieza de datos comprende la identificación y eliminación de errores existentes en conjuntos de datos, para mejorar la calidad global de estos.*

En (Galhardas et al., 2000) sus autores definen el *proceso de limpieza de datos como el proceso que se encarga de resolver los problemas de la ausencia de un identificador universal a través de todas las bases de datos que se integran (conocido como el problema de la identidad de objetos); la existencia de errores de teclado en los datos y el problema de la inconsistencia de los datos provenientes de múltiples fuentes.*

La mayoría de los trabajos en la literatura aparecen ligados a la limpieza de datos en grandes bases de datos, para dar soporte a ámbitos tales como IR y al proceso ETL (*extracción, transformación y carga*) en Datawarehousing (Lee et al., 1999; Rahm y Do, 2000; Galhardas et al., 2001a; Ananthakrishna et al., 2002).

Desde el punto de vista de la Minería de Datos, aparecen en la literatura trabajos como (Bruni y Sassano, 2001; Galhardas et al., 2001b; Iliopoulos et al., 2003; Apweiler et al., 2004). En (Lee et al., 1999) sus autores propone

una metodología genérica de limpieza de datos para minería y Datawarehousing y en (Rahm y Do, 2000) se comentan algunos problemas de los que se ocupa la limpieza de datos. Un ejemplo importante relacionado con minería aparece en (Müller, 2003) donde a través de un ejemplo práctico se muestra un método semántico de cómo realizar la limpieza de datos para mejorar los resultados obtenidos en la base de datos del *Genoma Humano*. En (Müller, 2003) se define que la mayoría de estos y otros trabajos se centran en la transformación de datos, ejecución de restricciones de integridad y eliminación de datos duplicados. Además dice que la mayoría de los trabajos se dedican a identificar errores, pero dejan a los expertos en las áreas temáticas en que son identificados a que planteen el método correcto la solución.

En las técnicas de minería que se basan en la frecuencia de los términos (como es el caso de nuestra solución), este proceso reviste crucial importancia. En estos casos como la solución se basa en el conteo de la frecuencia de aparición de los términos, que los datos sean lo más homogéneos posible es algo primordial. Por eso en la solución obtenida, haciendo uso de estas ideas, se mejora la calidad de la información de entrada para que los resultados obtenidos en consecuencia, sean superiores.

Como se comentó anteriormente, el otro tema importante a la hora de brindar una solución en Minería de Texto particularmente, es la Forma Intermedia de Representación que se elija para darle estructuras a los atributos textuales. En el siguiente apartado se recogen algunas de las formas intermedias más importantes que aparecen en la literatura y nos decantamos por la usada en nuestra solución.

2.2.4. Formas intermedias de representación en Minería de Texto

Como ya se mencionó, la estructura obtenida con el Preprocesamiento proporciona la base para la aplicación de las técnicas de minería, es decir, permiten obtener las diferentes Formas Intermedias de Representación de los textos procesados. Los tipos de Formas Intermedias en los que se puede representar una colección de documentos pueden ir desde la simplicidad de la

palabra hasta la complejidad del documento completo.

El término de *Forma Intermedia* fue acuñado por Tan en (Tan, 1999). En (Delgado et al., 2002) lo plantean como *el proceso de hacer explícita la estructura implícita del texto, mediante su conversión a una estructura de representación más simple.*

La estructura implícita de los datos textuales es tan rica que se puede hacer explícita de diferentes formas (Delgado et al., 2002). Para la realización de este proceso se recogen un gran número de Formas Intermedias en la literatura. Provenientes del campo de la Recuperación de Información se han trabajado algunas como las palabras, términos(Lent et al., 1997), palabras claves, frecuencia de palabras (Iritano y Ruffolo, 2001), etiquetado del documento (Feldman y Dagan, 1995; Feldman y Hirsh, 1996; Feldman et al., 1997), eventos (Karanikas et al., 2000), grafos semánticos (Gelbukh et al., 2002; Raghavan y Garcia-Molina, 2003), etiquetas XML (Winkler y Spiliopoulou, 2001), etc.

La extracción de las características del texto que se procesa, varía según la Forma Intermedia que se utilice. La elección de la Forma Intermedia correcta según Feldman en (Feldman y Dagan, 1995), dependerá de la forma en que el usuario contextualice el dominio que es descrito por los datos. En la práctica, significa que la Forma Intermedia debe estar relacionada con el tipo de patrones que nos interesan descubrir en los datos. Formas como los grafos semánticos aportan mucha más riqueza semántica al resultado obtenido por el proceso de minería. Con otras como las palabras claves, sólo se podrán obtener resultados referentes a la presencia o no de ciertas palabras en el documento procesado.

Cuando se dispone de una colección de textos, la heterogeneidad puede ser uno de los factores característicos de la misma. De este modo, no sería lo mismo encontrar una Forma Intermedia que represente a una Memoria Científica que a un Abstract de un Artículo. Es por esto que no debería ser imprescindible utilizar el mismo tipo de Forma Intermedia para representar a todos los textos de la colección, sino que se podría buscar una combinación de Formas Intermedias para representar dichos textos atendiendo a algún criterio determinado. De este modo, se podrían combinar no sólo palabras y conceptos a

la hora de localizar patrones dentro del texto, como indica Paralic (Paralic y Bednar, 2002), sino cualquier otro tipo de combinación.

Partiendo de la definición de Tan (Tan, 1999), las Formas Intermedias se puede clasificar, en general, como:

1. **Estructurada:** donde los datos se representan de forma relacional.
2. **Semiestructurada:** representación de un grafo conceptual.
    - *Basada en conceptos:* donde cada entidad representa un objeto o concepto de interés de un dominio específico. Deriva patrones y relaciones a través de objetos de conceptos. Se le pueden aplicar operaciones de Minería de Datos como el modelado predictivo y el descubrimiento asociativo.
    - *Basada en documentos:* cada entidad representa un documento. Deduce patrones y relaciones de interés en un dominio específico. La Forma Intermedia basada en documentos se puede transformar en una Forma Intermedia basada en conceptos, extrayendo información relevante de acuerdo a los objetos de interés de un dominio específico.

Para el caso de las Formas Intermedias de tipo *estructurado*, cuando se transforma el documento en una versión estructurada de sí mismo se está perdiendo gran cantidad de información. Esto se debe a que si la técnica elegida para proporcionarle esa estructura no tiene en cuenta la semántica y simplemente se queda con los términos relevantes y alguna relación entre ellos, se estará perdiendo información semántica irrecuperable.

En nuestro caso, la nueva Forma Intermedia que se presenta en este trabajo, está basada en una *forma semiestructurada basada en documentos (siempre considerando estos documentos como textos cortos).* Para cada tupla de la base de datos, *cada valor del texto corto* que se procesa es tomado como un documento independiente y se le aplican técnicas de *Preprocesamiento* enfocadas al proceso de minería que obtiene *Reglas de Asociación.* Siguiendo este proceso se encuentran los itemsets frecuentes de la colección de textos

cortos procesados. A partir de dichos itemsets frecuentes, se construyen las estructuras reticulares que contienen la semántica de los textos procesados (los detalles de este proceso se describen en el capítulo 5 del presente trabajo).

Como se discutió en los antecedentes de la solución, esta nueva forma de representación obtenida, será la base para realizar las consultas semánticas sobre el atributo textual que se procese.

2.3. Conclusiones

En el presente capítulo se han discutido los antecedentes del problema de investigación y la solución que se le ha dado. Para ello se han visto dentro de los antecedentes del problema los diferentes tipos de IMS que aparecen en la literatura, enfocados a la Recuperación de Información en textos. También se han dado las diferentes arquitecturas que han surgido con la idea de la integración de dichos sistemas. Aquí se ha remarcando que la solución obtenida posee una arquitectura de agregación de extensiones a un SGBD, para la realización de consultas semánticas en atributos textuales en bases de datos.

De forma ilustrativa, se han dado las diferencias que posee la solución presentada en este trabajo, con la forma en que los SGBD Semiestructurados resuelven la extracción de información de atributos textuales en una base de datos. Como se resumió, nuestra solución se basa en la aplicación de técnicas de Minería de Texto para dotar de estructura al atributo textual. Por su parte, los SGBD Semiestructurados resuelven esta problemática realizando la conversión de dicho atributo a formato XML.

Por otra parte, se han mencionado las extensiones que implementa nuestra solución para la representación y realización de consultas sobre las estructuras que lo componen.

Dentro de los antecedentes de la solución al problema de investigación se han discutido los conceptos de KDD y Minería de Datos, así como la relación que existe entre ellos. De forma análoga se han dado los conceptos de KDT y Minería de Texto y su relación, la importancia del proceso de Preprocesamiento en la Minería de Texto, así como el concepto e importancia de la tarea

de limpieza de datos. También se han discutido las Formas Intermedias de Representación más importantes que recoge la literatura y se ha enunciado la Forma Intermedia sobre la que se basará la nueva forma obtenida en el presente trabajo.

En el próximo capítulo se realiza la definición matemática de las estructuras y operaciones que conforman la nueva Forma Intermedia de Representación obtenida. Dicha forma se encargará de capturar la semántica que encierran los atributos de tipo texto corto en bases de datos, y define diferentes operaciones que permitirán la realización de consultas semánticas sobre ellos.

Capítulo 3

Propuesta teórica de la forma intermedia de representación

En el presente capítulo, partiendo del hecho de que la falta de estructura de los atributos textuales hace difícil su procesamiento automático para manejarlos de forma masiva, se da una solución a esta problemática. Para ello, la solución propuesta se basa en la transformación del atributo textual en una estructura intermedia que permita su representación de forma más estructurada . Dicha estructura está basada en el concepto de itemset frecuente y sus propiedades (ver apéndice B para más detalles), llamada conjunto-AP. De forma general, dentro de este capítulo se describen las estructuras que componen el modelo de la Forma Intermedia obtenida, así como, las operaciones definidas para cada estructura.

En la sección 1, se comienza definiendo las estructuras en las que se transforman los atributos textuales y sus operaciones. Se introducen los conceptos de conjunto-AP y estructura-AP como estructuras básicas que componen el modelo de representación que se ha propuesto. Durante la presentación de nuestro modelo se irán ilustrando las definiciones con ejemplos sencillos para lograr una mejor comprensión de los conceptos que se introducen. En la sección 2, se introducen un conjunto de operaciones y medidas que permitirán realizar búsquedas sobre las estructuras de conocimiento asociadas a los atributos textuales, y que además nos permitirán en su momento:

1. Sugerirle al usuario conjuntos de términos para refinar su búsqueda.
2. Decirle la cantidad de términos que aparecen representados de todos los que está buscando.
3. Si su frase aparece completamente incluida en la estructura de conocimiento, etc.

Finalmente en la sección 3, se desarrolla un ejemplo más práctico, donde se explicitan todas las operaciones que se introducen en las secciones previas. De esta forma se pretende hacer más comprensible todo el modelo, a partir de un ejemplo más intuitivo.

3.1. Definición formal de las estructuras matemáticas

En esta sección, se comienzan a formalizar las ideas presentadas anteriormente, definiendo las estructuras matemáticas que serán la base para la representación formal de los datos. Se establecerán las definiciones y propiedades de los conjuntos que tienen la propiedad '*APriori* '(conjuntos-AP) (Agrawal y Srikant, 1994). A continuación se da la definición formal y las propiedades de las estructuras subyacentes en los textos, que capturan la semántica que los mismos encierran y que como se verá, está compuesta por un conjunto de conjuntos-AP. Dichas estructuras se denominan estructuras-AP (Martín-Bautista et al., 2006).

3.1.1. Definición y propiedades de los conjuntos-AP

Definición 1. Conjunto-AP

Sean $X = \{x_1...x_n\}$ *un conjunto referencial de items y* $\mathcal{R} \subseteq \mathcal{P}(X)$ *un conjunto de itemsets frecuentes, siendo* $\mathcal{P}(X)$ *las partes de* X*. Diremos que* $\mathcal{R}$ *es un conjunto-AP si y sólo si:*

1. $\forall Z \in \mathcal{R} \Rightarrow \mathcal{P}(Z) \subseteq \mathcal{R}$

2. $\exists Y \in \mathcal{R}$ *tal que* :

   a) $card(Y) = max_{Z \in \mathcal{R}}(card(Z))$ *y no exista* $Y' \in \mathcal{R}$ *tal que* $card(Y') = card(Y)$

   b) $\forall Z \in \mathcal{R}; Z \subseteq Y$

El conjunto Y de máxima cardinalidad caracteriza al conjunto-AP y este será llamado *conjunto generador de* $\mathcal{R}$. Se denotará $\mathcal{R} = g(Y)$, y significa que $g(Y)$ será el conjunto-AP con conjunto generador Y. Obsérvese que $g(Y)$ es el conjunto de las partes de Y.

Se denotará *grado de* $g(Y)$ al cardinal de Y, o sea, a la cantidad de elementos del conjunto generador Y. Obviamente, los conjuntos-AP de grado 1 son los elementos de X, y además consideraremos el conjunto vacío $\emptyset$ como el conjunto-AP de grado cero.

Ejemplo 1

Sea $X = \{1, 2, 3, ..., 10\}$ y
$\mathcal{R} = \{\{1\}, \{3\}, \{5\}, \{1, 3\}, \{1, 5\}, \{3, 5\}, \{1, 3, 5\}\}$, entonces el conjunto generador es $Y = \{1, 3, 5\}$

Como se observa en el ejemplo anterior y teniendo en cuenta la definición 1, el conjunto generador $Y = \{1, 3, 5\}$ se corresponde con el conjunto-AP de mayor cardinalidad y que incluye a su vez, todas las combinaciones presentes en $\mathcal{R}$.

La figura 3.1 muestra el retículo correspondiente al ejemplo 1. Como se observa, el conjunto generador $Y = \{1, 3, 5\}$ es la raíz del árbol que forma el retículo. En las hojas del árbol están los elementos que componen el conjunto generador, y en el nivel intermedio del árbol se encuentran las combinaciones de los elementos de Y con cardinalidad 2.

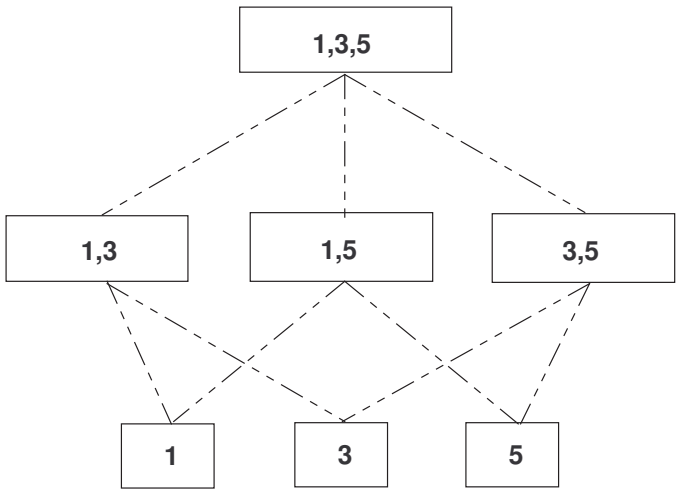


Figura 3.1: Retículo del conjunto-AP.

A continuación se dan algunas operaciones que pueden ser introducidas sobre esta estructura conjunto-AP que ha sido definida. Dichas operaciones serán utilizadas en la definición de operaciones posteriores, así como e incluso en la obtención de la estructura global de conocimiento, que encierra la semántica de los datos procesados. Se comienza por la operación que verifica si un conjunto-AP está incluido en otro.

Definición 2. Inclusión de conjuntos-AP

Sea $\mathcal{R} = g(R)$ *y* $\mathcal{S} = g(S)$ *dos conjuntos-AP con el mismo conjunto referencial de items:*

$$\mathcal{R} \subseteq \mathcal{S} \Leftrightarrow R \subseteq S$$

Como se observa en la definición, los conjuntos-AP $\mathcal{R}$ y $\mathcal{S}$ estarán incluidos unos en otros si alguno de los dos conjuntos generadores R o S está contenido uno en otro. Esto tiene sentido desde el punto de vista de la definición 1 dado que si, por ejemplo, el conjunto generador R está contenido dentro del conjunto generador S, entonces todas las combinaciones posibles de R quedarán contenidas dentro de S. A partir de esto se puede afirmar que el

conjunto-AP $\mathcal{R}$ está completamente contenido dentro del conjunto-AP $\mathcal{S}$, como plantea la definición de inclusión de conjunto-AP.

A continuación se introduce una operación importante en el contexto en el que se plantea este modelo, que es el *subconjunto-AP inducido* por un conjunto determinado. Esta operación se encargará de obtener el conjunto-AP particular que se genera, al intersecar el retículo global del conjunto-AP con un conjunto dado.

Definición 3. Subconjunto-AP inducido

Sea $\mathcal{R} = g(R)$ *y* $Y \subseteq X$ *diremos que* $\mathcal{S}$ *es el* subconjunto-AP inducido por Y *si y sólo si:*

$$\mathcal{S} = g(R \bigcap Y)$$

Como se observa en la definición, el subconjunto-AP inducido $\mathcal{S}$ será obtenido calculando su conjunto generador. Ésto será haciendo la intersección de los conjuntos de R(generadores del conjunto-AP $\mathcal{R}$) y el conjunto dado Y. Dado que Y es un subconjunto del conjunto referencial de items X, se garantiza que la intersección entre R y Y no será vacía ($R \bigcap Y \neq \emptyset$).

De forma análoga, en la siguiente definición se obtiene el superconjunto-AP inducido por un conjunto determinado. Esta operación nos permitirá por ejemplo, construir un nuevo conjunto-AP, a partir de la unión de un conjunto-AP existente con un conjunto dado.

Definición 4. Superconjunto-AP inducido

Sea $\mathcal{R} = g(R)$ *y* $Y \subseteq X$ *diremos que* $\mathcal{V}$ *es el* superconjunto-AP inducido por Y *si y sólo si:*

$$\mathcal{V} = g(R \bigcup Y)$$

Tal como ocurría para la definición del subconjunto-AP inducido, en la definición anterior se calcula el superconjunto-AP inducido $\mathcal{V}$ a partir de su conjunto generador. Para este caso se obtiene a partir de la unión de los conjuntos R (generador del conjunto-AP $\mathcal{R}$) y el conjunto dado Y.

A continuación, tomando como base la estructura conjunto-AP y sus operaciones, se pasa a definir la estructura de conocimiento que encerrará la semántica de los datos textuales.

3.1.2. Definición y propiedades de la estructura-AP

Una vez que ya se han establecido los conceptos de conjunto-AP, se usarán estos para definir las estructuras de información que aparecen cuando son obtenidos los itemsets frecuentes. Debe considerarse que dichas estructuras son obtenidas de forma constructiva, por la generación inicialmente de itemsets con cardinalidad igual a 1; seguidamente, estos son combinados para obtener los de cardinalidad igual a 2, y así sucesivamente hasta obtener los itemsets de máxima cardinalidad, fijando para ello un soporte mínimo. Por consiguiente, la estructura final será un conjunto de conjuntos-AP, la cual será definida formalmente como sigue.

Definición 5. Estructura-AP

Sea $X = \{x_1 ... x_n\}$ *un conjunto referencial de items y* $S = \{A, B, ...\} \subseteq \mathcal{P}(X)$ *un conjunto de itemsets frecuentes, tal que:*

$$\forall A, B \in S \,;\, A \not\subseteq B \,,\, B \not\subseteq A$$

Llamaremos estructura-AP del generador S, $\mathcal{T} = g(A, B, ...)$, *al conjunto de conjuntos-AP cuyos conjuntos generadores son* $A, B, ...$

De la definición anterior queda claro entonces, el hecho que se mencionó anteriormente: que la estructura-AP no es más que una colección de conjuntos-AP. Tal como se definió, los conjuntos generadores de la estructura-AP $A, B, ...$ no pueden estar contenidos unos en otros, utilizando para esta interpretación la definición de conjunto-AP incluido que se dio anteriormente. Entonces, la estructura-AP quedará constituida por todos los conjuntos generadores que se obtengan de las combinaciones de X presentes, dentro de todas las posibles ($\mathcal{P}(X)$).

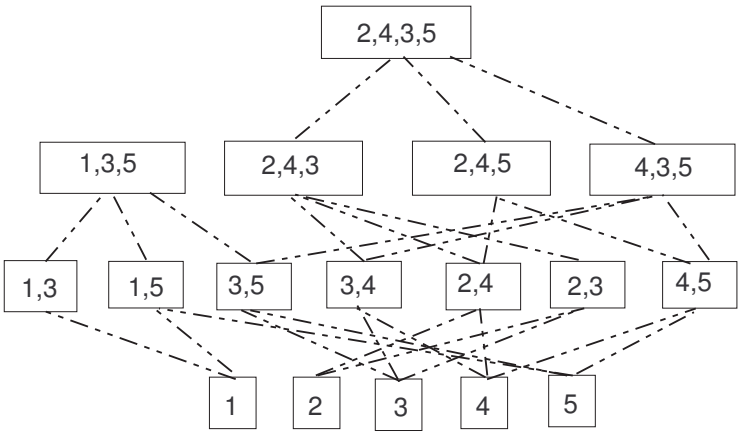


Figura 3.2: Retículo de la estructura-AP.

Debemos destacar también, que cualquier estructura-AP es un retículo de subconjuntos, cuyos extremos superiores son sus conjuntos generadores. La figura 3.2 muestra un ejemplo del retículo subyacente en $g(\{1,3,5\},\{2,4,3,5\})$. Como se observa en la figura, los conjuntos $\{1,3,5\}$ y $\{2,4,3,5\}$ son los conjuntos generadores. Es de destacar que aunque no se encuentran uno completamente incluido en el otro, comparten elementos comunes y se encuentran en la raíz del árbol que forma el retículo. Como también se aprecia, en los conjuntos-AP de cardinalidad dos, ambos conjuntos generadores ya comparten un conjunto-AP común, el $\{3,5\}$, lo mismo que en las hojas del árbol donde ya tienen en común los conjuntos-AP de cardinalidad uno ($\{3\}$ y $\{5\}$).

Ahora se darán algunas definiciones y propiedades de esta nueva estructura que permitirán hacer diferentes operaciones sobre la misma, entre ellas la inclusión de estructuras-AP, la unión y la intersección.

Definición 6. Inclusión de estructuras-AP

Sean $\mathcal{T}_1$, $\mathcal{T}_2$, dos estructuras-AP con el mismo conjunto referencial de items:

$$\mathcal{T}_1 \subseteq \mathcal{T}_2 \Leftrightarrow \forall \mathcal{R} \text{ conjunto-AP de } \mathcal{T}_1 \,,$$

$$\exists \mathcal{S} \textit{ conjunto-AP de } \mathcal{T}_2 \textit{ tal que } \mathcal{R} \subseteq \mathcal{S}$$

Antes que nada, destacar que la inclusión de un conjunto-AP en otro ($\mathcal{R} \subseteq \mathcal{S}$) es la que se dio en la definición 2. De esta definición se puede interpretar que para que una estructura-AP $\mathcal{T}_1$ esté contenida en otra estructura-AP $\mathcal{T}_2$, todos los conjuntos generadores de $\mathcal{T}_1$ tienen que aparecer incluidos en alguno de los conjuntos generadores de $\mathcal{T}_2$. El siguiente ejemplo hace más claro este punto.

Ejemplo 2

Sea $X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ y las estructuras-AP

$\mathcal{T}_1 = g(\{1, 2, 3\}, \{2, 4\}, \{6\})$,

$\mathcal{T}_2 = g(\{1, 2\}, \{6\})$,

$\mathcal{T}_3 = g(\{1, 2, 3\}, \{4\}, \{6\}, \{7\})$, tenemos que:

$$\mathcal{T}_2 \subseteq \mathcal{T}_1 \; ; \; \mathcal{T}_2 \subseteq \mathcal{T}_3 \text{ pero } \mathcal{T}_1 \text{ no está incluida en } \mathcal{T}_3$$

En el ejemplo se puede decir que $\mathcal{T}_2 \subseteq \mathcal{T}_1$, ya que sus conjuntos generadores $\{1, 2\}$ y $\{6\}$ están completamente incluidos en dos de los conjuntos generadores de $\mathcal{T}_1$, el $\{1, 2, 3\}$ y $\{6\}$ respectivamente. De forma análoga ocurre entre $\mathcal{T}_2$ y $\mathcal{T}_3$. Para el caso de $\mathcal{T}_1$ y $\mathcal{T}_3$ no se puede decir que $\mathcal{T}_1$ esté incluida en $\mathcal{T}_3$, dado que el conjunto generador $\{2, 4\}$ de $\mathcal{T}_1$ no está incluido en ninguno de los conjuntos generadores de $\mathcal{T}_3$.

A continuación se introduce una de las operaciones más importantes que se pueden definir sobre esta estructura-AP, la operación subestructura-AP inducida, que no será más que la estructura-AP resultante de intersecar una estructura-AP cualquiera con un conjunto dado. Esta operación reviste especial importancia porque será la que permita encontrar la representación

del tipo de dato abstracto, que corresponde a una tupla dada de la base de datos.

Definición 7. Subestructura-AP inducida

Sea la estructura-AP $\mathcal{T} = g(A_1, A_2, ..., A_n)$ *con conjunto referencial de items* X *y* $Y \subseteq X$. *Definiremos la* estructura-AP de $\mathcal{T}$ inducida por Y *como:*

$$\mathcal{T}' = \mathcal{T} \bigwedge Y = g(B_1, B_2, ..., B_m)$$

donde

$$\forall B_i \in \{B_1, ..., B_m\} \Rightarrow \exists A_j \in \{A_1, A_2, ..., A_n\}$$
$$\textit{tal que } B_i = A_j \bigcap Y$$
$$\forall A_j \in \{A_1, ..., A_n\} \Rightarrow \exists B_i \in \{B_1, B_2, ..., B_m\}$$
$$\textit{tal que } A_j \bigcap Y \subseteq B_i$$

Según esta definición, queda claro que $\mathcal{T}'$ es la estructura-AP generada por las intersecciones de Y con los conjuntos generadores de $\mathcal{T}$. Para ello, se interseca cada elemento de Y con todos los elementos $(A_1, A_2, ..., A_n)$ de $\mathcal{T}$. A los elementos B_i de $\mathcal{T}'$ irán sólo las intersecciones que no estén completamente incluidas en otro elemento B_i, o sea, que las intersecciones resultantes entre los elementos de $\mathcal{T}$ y Y que ya se encuentren incluidas en otro conjunto B_i serán eliminadas. De esta forma, se garantiza que la estructura-AP obtenida sólo esté formada por conjuntos generadores maximales, tal como plantea el concepto de estructura-AP introducido anteriormente. El siguiente ejemplo clarifica estas ideas.

Ejemplo 3

Sea $X = \{1, 2, ..., 9\}$,

$\mathcal{T} = g(\{1, 2, 3\}, \{2, 3, 4\}, \{4, 5\}, \{6\})$,

$Y = \{2, 3, 4, 5\}$ entonces se tiene

$\mathcal{T} \bigwedge Y = g(\{2, 3, 4\}, \{4, 5\})$.

Del ejemplo anterior se debe destacar que el conjunto $\{2,3\}$ no está incluido en $\mathcal{T} \bigwedge Y$, mientras que este conjunto sí pertenece a la intersección de Y y el conjunto generador de $\mathcal{T}$. La razón de la no inclusión es, como se explicó anteriormente, que el conjunto $\{2,3\}$ ya está incluido en uno de los conjuntos de salida de $\mathcal{T} \bigwedge Y$, el $\{2,3,4\}$.

De forma análoga a la definición anterior, se define ahora la operación superestructura-AP inducida, la que permitirá obtener la estructura-AP generada por la unión de un conjunto dado con una estructura-AP determinada.

Definición 8. Superestructura-AP inducida

Sea la estructura-AP $\mathcal{T} = g(A_1, A_2, ..., A_n)$ *con conjunto referencial de items* X *y* $Y \subseteq X$*, se define la* superestructura-AP de $\mathcal{T}$ inducida por Y*:*

$$\mathcal{T}' = \mathcal{T} \bigvee Y = g(B_1, B_2, ..., B_m)$$

donde

$$\forall B_i \in \{B_1, ..., B_m\} \Rightarrow \exists A_j \in \{A_1, A_2, ..., A_n\}$$
$$\textit{tal que } B_i = A_j \bigcup Y$$
$$\forall A_j \in \{A_1, ..., A_n\} \Rightarrow \exists B_i \in \{B_1, B_2, ..., B_m\}$$
$$\textit{tal que } A_j \bigcup Y \subseteq B_i$$

Para este caso la notación es análoga, sólo que en lugar de hacer intersección se hace la unión entre la estructura-AP $\mathcal{T}$ y el conjunto Y. El siguiente ejemplo muestra la unión de una estructura-AP con un conjunto para obtener la superestructura-AP generada por dicha unión.

Ejemplo 4

Sea $X = \{1, 2, ..., 9\}$,

$\mathcal{T} = g(\{1,2,3\}, \{2,3,4\}, \{4,5\}, \{6\})$,

$Y = \{2,3,4,5\}$ entonces se tiene

$\mathcal{T} \bigvee Y = g(\{1,2,3,4,5\}, \{2,3,4,5,6\})$.

Como se observa en el ejemplo anterior, la superestructura-AP obtenida por la operación $\mathcal{T} \bigvee Y$ está formada por los conjuntos $\{1, 2, 3, 4, 5\}$ y $\{2, 3, 4, 5, 6\}$ que aunque tienen elementos comunes, ninguno de ellos está completamente incluido en el otro. Por otro lado, conjuntos que se forman de la operación $\mathcal{T} \bigvee Y$ como el $\{2, 3, 4, 5\}$ no están en la superestructura-AP de salida, pues ya está completamente incluido en un conjunto de mayor cardinalidad como es el $\{2, 3, 4, 5, 6\}$.

La propiedad siguiente es deducida directamente de las dos definiciones anteriores:

Propiedad 1.

Sea $\mathcal{T} = g(A_1, A_2, ..., A_n)$ una estructura-AP, y Y_1 y Y_2 subconjuntos con el mismo conjunto referencial de items, entonces

$$(\mathcal{T} \bigwedge Y_1) \bigwedge Y_2) = \mathcal{T} \bigwedge (Y_1 \bigcap Y_2)$$

$$(\mathcal{T} \bigvee Y_1) \bigvee Y_2) = \mathcal{T} \bigvee (Y_1 \bigcup Y_2)$$

Esta propiedad se refiere a la posibilidad de asociatividad que tienen estas operaciones cuando se trabaja con más de un conjunto. En ella se plantea que se producirá el mismo resultado si se realizan las operaciones $\bigwedge$ y $\bigvee$, operando primero con la estructura-AP $\mathcal{T}$ y un conjunto Y_1, aplicándole a su resultado el mismo operador con otro conjunto Y_2, que realizando la unión o intersección de Y_1 y Y_2 y después la operación $\bigvee$ o $\bigwedge$ respectivamente.

Las operaciones $\bigwedge$ y $\bigvee$ están definidas entre conjuntos y estructuras-AP. Más adelante se discutirá sobre su utilización para consultar la base de datos.

Definición 9. Unión de estructuras-AP

Sean $\mathcal{T}_1 = g(A_1, A_2, ..., A_n)$ *y* $\mathcal{T}_2 = g(B_1, B_2, ..., B_m)$ *dos estructuras-AP, se define:*

$$\mathcal{S} = \mathcal{T}_1 \bigcup \mathcal{T}_2 = g(C_1, C_2, ..., C_h)$$

verificando

- $\forall C_i \in \{C_1, ..., C_h\}$; $(\exists A_j / C_i = A_j)$ *or* $(\exists B_l / C_i = B_l)$
- $\forall A_j \in \{A_1, ..., A_n\}$; $\exists C_i / A_j \subseteq C_i$
- $\forall B_l \in \{B_1, ..., B_m\}$; $\exists C_i / B_l \subseteq C_i$

De la definición anterior se tiene que $\mathcal{T}_1 \bigcup \mathcal{T}_2$ es el resultado de la unión de los conjuntos generadores de las estructuras-AP $\mathcal{T}_1$ y $\mathcal{T}_2$. Para ello se realiza la unión de cada conjunto generador con los conjuntos generadores de la otra estructura-AP y se eliminan las redundancias, que serían, como se explicó anteriormente, los conjuntos que ya se encuentren contenidos en otros de mayor cardinalidad.

De forma análoga se puede definir la intersección entre estructuras-AP de la forma siguiente:

Definición 10. Intersección de estructuras-AP

Sean $\mathcal{T}_1 = g(A_1, A_2, ..., A_n)$ *y* $\mathcal{T}_2 = g(B_1, B_2, ..., B_m)$ *dos estructuras-AP, se define:*

$$\mathcal{S} = \mathcal{T}_1 \bigcap \mathcal{T}_2 = g(C_1, C_2, ..., C_l)$$

verificando

$\forall C_i \in \{C_1, C_2, ..., C_l\}\ \exists A_p \in \{A_1, A_2, ..., A_n\}\ ,\ \{B_q \in B_1, B_2, ..., B_m\};$
$C_i = A_p \bigcap B_q$

De hecho $\mathcal{T}_1 \bigcap \mathcal{T}_2$ *es la estructura-AP cuyos conjuntos generadores son el resultado de hacer la intersección entre cada conjunto generador de* $\mathcal{T}_1$ *y cada conjunto generador de* $\mathcal{T}_2$*, y evitando redundancias.*

Las siguientes propiedades conectan la operación de restricción con ambas, la unión y la intersección. Se dan estas propiedades con la idea de poder realizar la intersección de un conjunto con la estructura-AP resultante de la unión o intersección de dos estructuras-AP dadas.

Propiedad 2.

Sean $\mathcal{T}_1 = g(A_1, A_2, ..., A_n)$ y $\mathcal{T}_2 = g(B_1, B_2, ..., B_m)$ dos estructuras-AP con el mismo conjunto referencial de items X y $Y \subseteq X$, es demostrable que:

$$Y \bigwedge (\mathcal{T}_1 \bigcap \mathcal{T}_2) = (Y \bigwedge \mathcal{T}_1) \bigcap (Y \bigwedge \mathcal{T}_2)$$

Esta propiedad plantea que sería equivalente para realizar la intersección ($\bigwedge$) entre un conjunto Y y la estructura-AP resultante de la intersección ($\bigcap$) de dos estructuras-AP $\mathcal{T}_1$ y $\mathcal{T}_2$, el hacer la intersección del conjunto con cada una de las dos estructuras-AP por separado, y luego realizar la intersección de las dos estructuras-AP resultantes; que realizar primero la intersección de las dos estructuras-AP y la resultante, intersecarla con el conjunto Y. La demostración de que este resultado sería el mismo aparece a continuación.

Demostración:

En lo siguiente se denotará por $sp(\mathcal{T})$ al conjunto de conjuntos generadores de la estructura-AP $\mathcal{T}$, y se considera

$$\mathcal{U} = Y \bigwedge (\mathcal{T}_1 \bigcap \mathcal{T}_2)$$

y

$$\mathcal{V} = (Y \bigwedge \mathcal{T}_1) \bigcap (Y \bigwedge \mathcal{T}_2)$$

Se propone la demostración considerando dos tipos de inclusiones posibles: $\mathcal{U} \subseteq \mathcal{V}$ y $\mathcal{V} \subseteq \mathcal{U}$, y se demostrará que ambas son verdaderas.

A) $\mathcal{U} \subseteq \mathcal{V}$

Sea $B \in sp(\mathcal{U})$ acorde a las definiciones 7,10 $\exists A \in sp(\mathcal{T}_1 \bigcap \mathcal{T}_2)$ tal que $B = A \bigcap Y$ y $\exists A^1 \in \mathcal{T}_1$ $\exists A^2 \in \mathcal{T}_2$ tal que $A = A^1 \bigcap A^2$

Se puede considerar: $B = Y \bigcap A^1 \bigcap A^2 = (Y \bigcap A^1) \bigcap (Y \bigcap A^2)$ y por la definición 7 se tiene:

$$\exists B^1 \in sp((Y \bigwedge \mathcal{T}_1) \ / \ Y \bigcap A^1 \subseteq B^1$$

$$\exists B^2 \in sp((Y \bigwedge \mathcal{T}_2) \ / \ Y \bigcap A^2 \subseteq B^2$$

Finalmente, se tiene: $B \subseteq B^1 \bigcap B^2$ y por la definición10:

$$\exists C \in sp(Y \bigwedge \mathcal{T}_1) \bigcap (Y \bigwedge \mathcal{T}_2) \ / \ B \subseteq C$$

lo cual demuestra la inclusión **A)**.

B) $\mathcal{V} \subseteq \mathcal{U}$

Sea $B \in sp(\mathcal{V})$ acorde a las definiciones 7,10

$$\exists B_1 \in sp(Y \bigwedge \mathcal{T}_1) \ , \ \exists B_2 \in sp(Y \bigwedge \mathcal{T}_2) \ / \ B = B_1 \bigcap B_2$$

y

$$\exists A_1 \in sp(\mathcal{T}_1) \ / \ B_1 = A_1 \bigcap Y \text{ y } \exists A_2 \in sp(\mathcal{T}_2) \ / \ B_2 = A_2 \bigcap Y$$

Adicionalmente por la definición 10

$$\exists C \in sp(\mathcal{T}_1 \bigcap \mathcal{T}_2) \ / \ A_1 \bigcap A_2 \subseteq C$$

y por la definición 7

$$\exists D \in sp(Y \bigwedge \mathcal{T}_1 \bigcap \mathcal{T}_2) \ / \ Y \bigcap C \subseteq D$$

y finalmente se tiene:

$$B = A_1 \bigcap A_2 \bigcap Y \subseteq D$$

lo cual demuestra la inclusión **B)**.

Propiedad 3.

Sean $\mathcal{T}_1 = g(A_1, A_2, ..., A_n)$ y $\mathcal{T}_2 = g(B_1, B_2, ..., B_m)$ dos estructuras-AP con el mismo conjunto referencial de items X y $Y \subseteq X$. Es demostrable que:

$$Y \bigwedge (\mathcal{T}_1 \bigcup \mathcal{T}_2) = (Y \bigwedge \mathcal{T}_1) \bigcup (Y \bigwedge \mathcal{T}_2)$$

Esta propiedad plantea que sería equivalente realizar la intersección ($\bigwedge$) entre un conjunto Y y la estructura-AP resultante de la unión ($\bigcup$) de dos estructuras-AP $\mathcal{T}_1$ y $\mathcal{T}_2$, que hacer la intersección del conjunto con cada una de las dos estructuras-AP por separado, y luego realizar la unión de las dos estructuras-AP resultantes. La demostración de que este resultado sería el mismo aparece a continuación:

Demostración:

Como en la demostración anterior, se considera

$$\mathcal{U} = Y \bigwedge (\mathcal{T}_1 \bigcup \mathcal{T}_2)$$

y

$$\mathcal{V} = (Y \bigwedge \mathcal{T}_1) \bigcup (Y \bigwedge \mathcal{T}_2)$$

y se demuestran ambas inclusiones.

A) $\mathcal{U} \subseteq \mathcal{V}$

Sea $B \in sp(\mathcal{U})$ acorde a las definiciones 7,9 $\exists A \in sp(\mathcal{T}_1 \bigcup \mathcal{T}_2)$ tal que $B = A \bigcap Y$ y $\exists A^1 \in \mathcal{T}_1$ o $\exists A^2 \in \mathcal{T}_2$ tal que $A \subseteq A^1$ o $A \subseteq A^2$.

Entonces se asume que $A \subseteq A^1$, entonces por la definición 7 $\exists C \in sp(Y \bigwedge \mathcal{T}_1)$ tal que $B = Y \bigcap A_1 \subseteq C$ y por la definición 9 $\exists D \in (Y \bigwedge \mathcal{T}_1) \bigcup (Y \bigwedge \mathcal{T}_2)$ tal que $C \subseteq D$ y por consiguiente $B \subseteq D$ lo cual demuestra la inclusión **A)**.

B) $\mathcal{V} \subseteq \mathcal{U}$

Sea $B \in sp(\mathcal{V})$ por la definición 9 $\exists B_1 \in sp(Y \bigwedge \mathcal{T}_1)$ tal que $B \subseteq B_1$ o $\exists B_2 \in sp(Y \bigwedge \mathcal{T}_2)$ tal que $B \subseteq B_2$.

Entonces se asume que $B \subseteq B_1$, entonces $\exists A \in \mathcal{T}_1$ tal que $B \subseteq Y \bigcap A$ y $\exists C \in sp(\mathcal{T}_1 \bigcup \mathcal{T}_2)$ tal que $A \subseteq C$ y $\exists D \in sp(Y \bigwedge(\mathcal{T}_1 \bigcup \mathcal{T}_2))$ tal que $Y \bigcap C \subseteq D$.

Finalmente se tiene:

$$B \subseteq Y \bigcap A \subseteq Y \bigcap C \subseteq D$$

lo cual demuestra la inclusión **B).**

Una vez que se ha definido la estructura-AP con sus operaciones y propiedades, en la siguiente sección se definen algunas operaciones necesarias para poder determinar si un conjunto determinado aparece o no dentro de una estructura-AP determinada.

3.2. Consultando la base de datos: intersección de conjuntos con estructuras-AP

En esta sección, serán establecidas las definiciones necesarias para consultar la base de datos. Teniendo en cuenta que la estructura-AP es obtenida a partir de los términos relevantes del atributo textual que se esté procesando, se puede decir entonces que contendrá la mayoría de los términos relevantes que aparecen en dicho atributo. De aquí que se pueda afirmar entonces que la estructura-AP es el dominio activo del atributo del que fue obtenida. En capítulos posteriores se discute en detalle esta idea, así como la obtención de la estructura-AP particular correspondiente al atributo textual para cada tupla de la base de datos.

La idea es que el usuario expresará sus requerimientos como conjuntos de términos, para ser consultados sobre los atributos textuales en la base de datos. Dado que dichos atributos estarán representados por sus estructuras-AP particulares, algunos tipos de acoplamientos tienen que ser dados para satisfacer las consultas sobre dichas estructuras. Para hacerlo, dos formas aparecen directamente, una cuando se desean encontrar todos los términos

de la consulta en cada tupla, y otra cuando se desea encontrar al menos uno de los términos que se están buscando.

Definición 11. Acoplamiento fuerte

Sea la estructura-AP $\mathcal{T} = g(A_1, A_2, ..., A_n)$ *con conjunto referencial de items* X *y* $Y \subseteq X$. *Se define el* acoplamiento fuerte entre Y y $\mathcal{T}$ *como la operación lógica:*

$$Y \bigodot \mathcal{T} = \begin{cases} \textit{verdadero si} & \exists A_i \in \{A_1, A_2, ..., A_n\} \\ & / Y \subseteq A_i \\ \textit{falso} & \textit{en otro caso} \end{cases}$$

En la definición anterior, se define el tipo de acoplamiento fuerte, que será el que tendrá el conjunto Y con la estructura-AP $\mathcal{T}$ cuando dicho conjunto aparezca completamente incluido en alguno de los conjuntos generadores de $\mathcal{T}$. En caso contrario se dice que Y no tiene un acoplamiento fuerte con la estructura-AP.

De forma análoga, se puede definir la operación que verifica si el conjunto Y, está parcialmente incluido en $\mathcal{T}$.

Definición 12. Acoplamiento débil

Sea la estructura-AP $\mathcal{T} = g(A_1, A_2, ..., A_n)$ *con conjunto referencial de items* X *y* $Y \subseteq X$. *Se define el* acoplamiento débil entre Y y $\mathcal{T}$ *como la operación lógica:*

$$Y \bigoplus \mathcal{T} = \begin{cases} \textit{verdadero si} & \exists A_i \in \{A_1, A_2, ..., A_n\} \\ & / Y \bigcap A_i \neq \emptyset \\ \textit{falso} & \textit{en otro caso} \end{cases}$$

Como se observa, en esta definición se hace uso de la operación intersección entre conjuntos ($\bigcap$) para determinar si la intersección entre Y y los conjuntos generadores de la estructura-AP $\mathcal{T}$ no es vacía. Esto significa que dicho conjunto se acopla, al menos parcialmente, con alguno de los A_i generadores de $\mathcal{T}$.

Estas definiciones se pueden complementar dando alguna medida o índice que cuantifique estos acoplamientos. La idea es considerar que el acoplamiento de

un conjunto largo de términos tendrá un índice mayor que uno con un menor número de términos. Adicionalmente, si algún conjunto de términos se acopla con más de un conjunto generador, éste tendrá un índice mayor que el de otro que sólo se acopla con un sólo conjunto. También, dependiendo del contexto, resultará más importante retornar el máximo índice de acoplamiento entre un conjunto y todos los conjuntos de la estructura-AP, que el grado de acoplamiento con todos los conjuntos que componen dicha estructura.

Con esta idea, en el siguiente apartado se dan las definiciones de las dos variantes de índice de acoplamiento fuerte y débil, para cuando ocurre uno u otro tipo de acoplamiento entre la frase buscada y la estructura-AP consultada.

3.2.1. Cálculo de la bondad de acoplamiento: Índice de Acoplamiento fuerte y débil

Independientemente del tipo de acoplamiento que ocurra entre la frase buscada y la estructura-AP consultada, se pueden definir dos variantes para determinar la bondad con que ocurre el acoplamiento entre ambas estructuras. Esto vendría dado por las siguientes formas de calcular el índice de acoplamiento fuerte o débil:

1. *Cálculo de índices por el promedio:* El índice de acoplamiento (*fuerte o débil*) es calculado teniendo en cuenta todos los conjuntos que componen la estructura-AP. Para ello, después que se calcula la suma del grado de acoplamiento con cada conjunto de la estructura-AP, se divide dicha suma por la cantidad de conjuntos que componen la estructura-AP. De esta forma, se puede afirmar que se calcula el acoplamiento del conjunto de términos buscados con la estructura-AP completa.

2. *Cálculo de índices por el máximo:* El índice de acoplamiento (*fuerte o débil*) es calculado sólo teniendo en cuenta el conjunto de la estructura-AP con el que mejor se acopla el conjunto de términos buscados (*que será el conjunto donde la cardinalidad de la intersección sea mayor*). Para ello, se calcula la intersección del conjunto de términos buscados

con cada conjunto de la estructura-AP, y se determina el máximo de dichos acoplamientos que será el valor retornado para este índice. En este caso, el valor retornado no será referente a la estructura-AP completa, si no, al mejor conjunto de ésta con que se acopla el conjunto de términos buscados.

A continuación se definen el índice de acoplamiento fuerte y débil, teniendo en cuenta estas dos variantes para su cálculo.

Definición 13. Índice de acoplamiento fuerte (débil)

Sea la estructura-AP $\mathcal{T} = g(A_1, A_2, ..., A_n)$ *con conjunto referencial de items* X *y* $Y \subseteq X$, *se define el* índice de acoplamiento fuerte (débil) entre Y y $\mathcal{T}$ *como sigue:*

$\forall A_i \in \{A_1, A_2, ..., A_n\}$ *se denota* $m_i(Y) = card(Y \bigcap A_i)/card(A_i)$, $S = \{i \in \{1, ..., n\} | Y \subseteq A_i\}$, $W = \{i \in \{1, ..., n\} | Y \bigcap A_i \neq \emptyset\}$.

Entonces se define el índice de acoplamiento fuerte y débil entre Y y $\mathcal{T}$ *como sigue:*

3.2.1.1. Cálculo de índices por el promedio

$$\textbf{Índice fuerte} = S(Y|\mathcal{T}) = \sum_{i \in S} m_i(Y)/n$$

$$\textbf{Índice débil} = W(Y|\mathcal{T}) = \sum_{i \in W} m_i(Y)/n$$

Como se puede observar, en este caso se definen ambos índices, teniendo en cuenta todos los conjuntos de la estructura-AP. Para cada caso, se tiene en cuenta que el acoplamiento entre el conjunto de términos buscados y cada conjunto de la estructura-AP haya sido fuerte o débil, según corresponda.

A continuación se definen estas mismas expresiones, pero en función de calcularlas por el máximo acoplamiento entre los términos buscados y la estructura-AP.

3.2.1.2. Cálculo de índices por el máximo

$$\textbf{Índice fuerte} = S(Y|\mathcal{T}) = max(m_i(Y)) \ ; \ i \in S$$

$$\textbf{Índice débil} = W(Y|\mathcal{T}) = max(m_i(Y)) \ ; \ i \in W$$

En este caso se definen ambos índices teniendo en cuenta sólo el conjunto de la estructura-AP con el que la cardinalidad de la intersección sea mayor. También se tiene en cuenta para cada caso, que el acoplamiento entre el conjunto de términos buscados y cada conjunto de la estructura-AP haya sido fuerte o débil, según corresponda.

De estas dos variantes para calcular los índices de acoplamiento fuerte y débil se desprende que:

$$\forall Y \text{ y } \mathcal{T} \ , \ S(Y|\mathcal{T}) \in [0,1] \ , \ W(Y|\mathcal{T}) \in [0,1] \text{ y } W(Y|\mathcal{T}) \geq S(Y|\mathcal{T})$$

Lo que significa que para todo conjunto Y y estructura-AP $\mathcal{T}$, ambos índices de acoplamiento tomarán valor entre 0 y 1; de igual forma, siempre el índice de acoplamiento débil será mayor o igual que el índice de acoplamiento fuerte.

En el capítulo referente a la evaluación del sistema, se verá el uso de estas dos variantes para calcular los índices de acoplamiento. Ellas serán utilizadas para determinar la relevancia que da el sistema a cada tupla analizada.

A continuación, la próxima sección se dedica a desarrollar un ejemplo concreto con el que se resuelven la mayoría de los conceptos introducidos hasta este punto a lo largo del capítulo.

3.3. Ejemplo práctico

En la presente sección se pretende hacer una recapitulación de todos los conceptos introducidos a lo largo de este capítulo. Para ello se desarrollará un ejemplo más ilustrativo con el que se retoman la mayoría de las operaciones sobre las estructuras básicas introducidas en el modelo propuesto.

En el ejemplo se consideran los atributos textuales de una base de datos de recetas de postres. Como se observa en la figura 3.3, se han extraído atributos textuales referentes al atributo *nombre de postre,* para algunos registros de la base de datos y se muestran en su estado original en dichos registros. Los valores de los atributos son textos cortos que en su mayoría dan una idea de los ingredientes fundamentales de la receta en sí.

Nombre de Postre	**Autor**	$\cdots$	**Dificultad**	**Tiempo de preparación**
Fresas con nata y caramelo	María		Alta	18
Tarta de fresas	Guillermo		Media	25
Tarta de almendras	Elizabet		Media	25
Tarta de chocolate	Idelino		Media	28
Fresas con nata	Nérida		Baja	15
Nueces con nata	Claudia		Baja	15
Tarta de nata y chocolate	Carlos		Media	30
Tarta de nata	Teresa		Baja	15
Tarta de fresas, nueces y nata	Clarisbel		Alta	35
$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$

Figura 3.3: Ejemplos de registros para atributos textuales en una Base Datos de recetas de postres.

Se toman estos datos como punto de partida, y se sigue el procedimiento que posteriormente se discute en el capítulo 5 del presente trabajo. En dicho procedimiento, se obtiene el diccionario de palabras con los términos más representativos de todo el vocabulario que se utiliza en los registros del atributo *nombre de receta,* eliminando las palabras de parada. Una vez que se tienen los términos del diccionario, utilizando estos, se pueden transformar los registros en una base de datos transaccional, y con ella, utilizar el algoritmo Apriori para generar los itemsets. Algunos de estos itemsets se muestran en

la figura 3.4, donde el soporte es definido como la probabilidad de encontrar el itemset en la base de datos transaccional.

Item1	**Item2**	**Item3**	**Item4**	$\cdots$	**Soporte**
fresas	nata	nueces	tarta		1.412
chocolate	nata	tarta			1.489
fresas	nueces	tarta			1.503
chocolate	tarta				1.633
nata	tarta				1.702
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$

Figura 3.4: Ejemplos de itemsets y su soporte.

Estos itemsets forman el conjunto de conjuntos-AP en la forma de una estructura de inclusión reticular. Una vez que se tienen los conjuntos-AP, el conjunto de todos ellos forma el retículo de la estructura-AP. En la figura 3.5 se muestra un segmento del retículo de la estructura-AP que se deriva de nuestro ejemplo.

Una vez definido el retículo, a continuación se desarrollan los ejemplos de las definiciones introducidas a lo largo de este capítulo. Para clarificar las ideas que se han transmitido con la introducción de cada estructura y su operación, partiremos de la definición de las estructuras básicas de nuestro ejemplo.

El referencial X es el conjunto de todos los items distintos que aparecen en los itemsets obtenidos. A partir de X se define $\mathcal{P}(X)$ como el conjunto de todas sus combinaciones posibles. En el caso del ejemplo sería $X = \{chocolate, fresas, nata, nueces, tarta\}$.

Se define $\mathcal{R}$ como un subconjunto de todas las combinaciones posibles de $\mathcal{P}(X)$, esto es, $\mathcal{R} \subseteq \mathcal{P}(X)$. Para nuestro ejemplo sería:

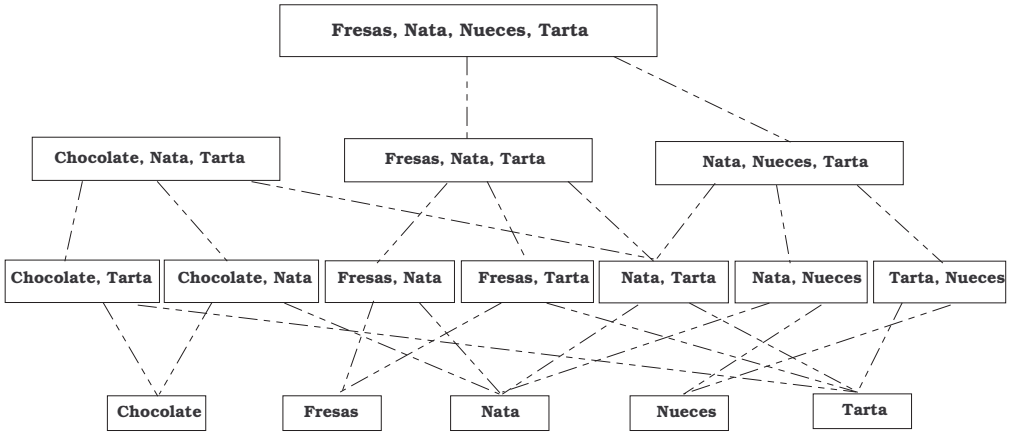


Figura 3.5: Retículo de la estructura-AP del ejemplo de recetas de postres.

$$\mathcal{R} = \{\{chocolate\}, \{nata\}, \{tarta\}, \{chocolate, nata\}, \{chocolate, tarta\}, \{nata, tarta\}, \{chocolate, nata, tarta\}\}$$

El conjunto generador de $\mathcal{R}$ es $Y = \{chocolate, nata, tarta\}$.

Una vez que ya se han definido las estructuras básicas del ejemplo, se comenzará ahora a resolver las definiciones que introducen las estructuras y sus operaciones, comenzando con las referentes al concepto de conjunto-AP.

Ejemplo 5 Conjunto-AP

En el ejemplo, el conjunto $\mathcal{R}$ se puede decir que es un conjunto-AP. El mismo cumple los requisitos de la definición, al estar presentes todas las combinaciones posibles de su conjunto generador $Y = \{chocolate, nata, tarta\}$ y garantiza que existe este conjunto maximal, sin que otro, dentro de $\mathcal{R}$, lo iguale en cardinalidad. El retículo que forma este conjunto-AP se puede observar en la figura 3.2 como todos los conjuntos debajo del conjunto-AP generador $Y = \{chocolate, nata, tarta\}$.

Ejemplo 6 Inclusión de conjuntos-AP

Como se explicó anteriormente cuando se introdujo esta definición, la inclusión de conjuntos-AP se verifica teniendo en cuenta si los conjuntos generadores de los conjuntos-AP en cuestión, están contenidos unos dentro de

otros. Para mostrar la aplicación de esta definición se construyen dos nuevos conjuntos-AP S_1 y S_2.

$S_1 = \{\{chocolate\}, \{nata\}, \{tarta\}, \{chocolate, nata\}, \{chocolate, tarta\},$

$\{nata, tarta\}, \{chocolate, nata, tarta\}\}$, y

$S_2 = \{\{fresas\}, \{nata\}, \{fresas, nata\}$

Se trata ahora de obtener los conjuntos generadores de los conjuntos-AP S_1 y S_2, que no serán más que los conjuntos de cardinalidad máxima dentro del conjunto-AP, donde todas sus combinaciones posibles estén presentes. Para el caso del conjunto-AP S_1, el conjunto generador será $Y_1 = \{chocolate,$ $nata, tarta\}$ y para el caso del conjunto-AP S_2 el conjunto generador será $Y_2 = \{fresas, nata\}$. Para el caso de $\mathcal{R}$, conjunto-AP del ejemplo, su conjunto generador es $Y = \{fresas, nata, tarta\}$, como ya se dijo anteriormente.

Entonces se verifican la inclusión de S_1 y S_2 en $\mathcal{R}$:

1. $S_1 \subseteq \mathcal{R}$**:** como $Y_1 \not\subseteq Y$, entonces podemos decir que S_1 no está incluido en $\mathcal{R}$, ya que su conjunto generador no está incluido con el conjunto generador de $\mathcal{R}$.

2. $S_2 \subseteq \mathcal{R}$**:** como $Y_2 \subseteq Y$, entonces podemos decir que S_2 sí está incluido en $\mathcal{R}$, ya que su conjunto generador está incluido con el conjunto generador de $\mathcal{R}$.

Ejemplo 7 Subconjunto-AP inducido

Para el caso de esta definición, como se refiere al subconjunto-AP inducido sobre un conjunto-AP por un conjunto Y dado, se comienza por definir el conjunto Y. Después se ejemplifica la obtención del subconjunto-AP, que induce sobre el conjunto-AP $\mathcal{R}$ de nuestro ejemplo.

Sea el conjunto $Y = \{chocolate, nata, tarta\}$, para obtener el subconjunto-AP inducido por Y sobre $\mathcal{R}$, sería, como plantea esta definición, obtener $\mathcal{S} = g(R \bigcap Y)$. Para ello se calcula el subconjunto-AP inducido $\mathcal{S}$, por el conjunto

Y, como el conjunto-AP que se obtiene de intersecar el conjunto Y con el conjunto generador R de $\mathcal{R}$.

Entonces, sería obtener la intersección entre $Y = \{chocolate, nata, tarta\}$ y $R = \{fresas, nata, tarta\}$, esto es:

$\mathcal{S} = g(R \bigcap Y) = g(\{nata, tarta\})$

La interpretación del subconjunto-AP inducido S que se obtiene, es que sería el conjunto-AP que tiene como conjunto generador el conjunto $\{nata, tarta\}$. De aquí que todo el retículo derivado de este conjunto, será el subconjunto-AP inducido por Y sobre el conjunto-AP original de nuestro ejemplo $\mathcal{R}$.

Ejemplo 8 Superconjunto-AP inducido

Para el caso de esta definición, al ser su estructura e interpretación similar a la de la definición anterior, basados en esas mismas estructuras que se introdujeron para explicar el ejemplo 3, se desarrolla el ejemplo de ésta definición. Según plantea la definición 4, sería obtener el superconjunto-AP $\mathcal{V}$ inducido por el conjunto Y sobre $\mathcal{R}$, de la siguiente forma:

$\mathcal{V} = g(R \bigcup Y) = g(\{chocolate, fresas, nata, tarta\})$

En este caso $\mathcal{V}$ significa el superconjunto-AP inducido por el conjunto Y, sobre el conjunto-AP $\mathcal{R}$. Éste ha sido obtenido calculando su conjunto generador como la unión entre Y y el conjunto generador de $\mathcal{R}$. En este caso se obtiene un super conjunto generador, del conjunto generador $R = \{fresas, nata, tarta\}$, pues con la unión se le adiciona el elemento $\{chocolate\}$ que no se encontraba en el generador original. De aquí que $\mathcal{V}$ es entonces todo el retículo derivado del conjunto generador $\{chocolate, fresas, nata, tarta\}$.

A continuación se ejemplifica el grupo de definiciones que comprenden la estructura-AP con sus operaciones. Para ello se comienza con la definición de estructura-AP. Para definir la estructura-AP resultante de nuestro ejemplo, se retoman en este punto, las estructuras iniciales que se definieron al inicio de este ejemplo. Se hace uso del retículo de la figura 3.2, ahora completamente. De aquí se obtienen las siguientes estructuras para la continuación de este ejemplo:

Sea el conjunto referencial de items $X = \{chocolate, fresas,$
$nata, nueces, tarta\}$ y $\mathcal{P}(X)$ el conjunto de todas las combinaciones posibles de X (*que por su extensión ponemos, estrictamente, sólo las que aparecen representadas en el retículo de la figura* 3.2).

$\mathcal{P}(X) = \{\{fresas, nata, nueces, tarta\}, \{chocolate, nata, tarta\},$
$\{fresas, nata, tarta\}, \{nata, nueces, tarta\}, \{fresas, nata\},$
$\{fresas, tarta\}, \{chocolate, tarta\}, \{chocolate, nata\}, \{nata, nueces\},$
$\{tarta, nueces\}, \{chocolate\}, \{fresas\}, \{nata\}, \{nueces\}, \{tarta\}\}$

Ejemplo 9 Estructura-AP

Como plantea la definición 5, la estructura-AP estará compuesta por el conjunto de conjuntos-AP generadores que se tengan en $\mathcal{P}(X)$. Para introducir la definición se ha denotado al conjunto $S = \{A, B, ...\}$, como un subconjunto de $\mathcal{P}(X)$ donde los componentes $\{A, B, ...\}$ de S no pueden estar contenidos unos en otros, y cada elemento representa los conjuntos generadores de los conjuntos-AP que componen la estructura-AP. Esto es:

$\mathcal{T} = g(A, B, ...)$, donde $\mathcal{T}$ es la estructura-AP compuesta por el conjunto de conjuntos-AP cuyos generadores son $A, B, ...$

En nuestro caso, tenemos un conjunto maximal que contiene la inmensa mayoría de las combinaciones posibles de X, que es el conjunto $\{fresas, nata,$ $nueces, tarta\}$. Éste contiene a todos los elementos de $\mathcal{P}(X)$ excepto al conjunto $\{chocolate, nata, tarta\}$ con algunos de los nodos que se derivan de él. Debido a esto, se obtienen estos dos conjuntos como generadores de los dos conjuntos-AP que compondrán la estructura-AP, que quedará definida, como se dijo, en función de sus conjuntos generadores:

$\mathcal{T} = g(\{fresas, nata, nueces, tarta\}, \{chocolate, nata, tarta\})$

Ejemplo 10 Inclusión de estructuras-AP

Para analizar la inclusión de estructuras-AP, como aparece en su definición, se tiene que verificar la inclusión o no de todos los conjuntos generadores de una estructura-AP $\mathcal{T}_1$ en los de otra $\mathcal{T}_2$. Esto sería verificar que $\forall \mathcal{R}$ conjunto-AP de $\mathcal{T}_1$, $\exists \mathcal{S}$ conjunto-AP de $\mathcal{T}_2$ tal que $\mathcal{R} \subseteq \mathcal{S}$. Para hacer esta verificación,

serán definidos dos estructuras-AP de ejemplo $\mathcal{T}_1$ y $\mathcal{T}_2$ (*no necesariamente con el mismo referencial del ejemplo*) y se verificará su inclusión o no, en la estructura-AP $\mathcal{T}$ obtenida de nuestro ejemplo:

$$\mathcal{T} = g(\{fresas, nata, nueces, tarta\}, \{chocolate, nata, tarta\})$$

$$\mathcal{T}_1 = g(\{fresas, tarta\}, \{chocolate\})$$

$$\mathcal{T}_2 = g(\{fresas, tarta, vainilla\}, \{chocolate\})$$

Analizamos ahora las siguientes inclusiones:

1. $\mathcal{T}_1 \subseteq \mathcal{T}$: como se observa, los dos conjuntos generadores de $\mathcal{T}_1$ se encuentran completamente incluidos en los conjuntos generadores de $\mathcal{T}$. Por eso podemos decir que $\mathcal{T}_1$ está contenido en $\mathcal{T}$.

2. $\mathcal{T}_2 \subseteq \mathcal{T}$: como se observa, de los dos conjuntos generadores de $\mathcal{T}_2$ sólo el conjunto $\{chocolate\}$ está completamente incluido en uno de los generadores de $\mathcal{T}$, mientras que el conjunto $\{fresas, tarta, vainilla\}$ no está completamente incluido en ninguno de los dos conjuntos generadores de $\mathcal{T}$. Por eso podemos decir que $\mathcal{T}_2$ no está contenido en $\mathcal{T}$.

Se debe destacar, que en el caso de la definición anterior, cuando se habla de inclusión de conjuntos generadores, se está haciendo alusión a la misma definición de inclusión de conjuntos-AP discutida anteriormente.

Ejemplo 11 Estructura-AP inducida

Para determinar la estructura-AP inducida por un conjunto Y, se tiene que resolver la intersección de dicho conjunto con la estructura-AP de nuestro ejemplo $\mathcal{T}$. Para ello, se define el conjunto Y como:
$Y = \{\{chocolate, nata\}, \{vainilla\}\}$ y

$$\mathcal{T} = g(\{fresas, nata, nueces, tarta\}, \{chocolate, nata, tarta\})$$

por tanto para obtener la estructura-AP inducida por Y sobre $\mathcal{T}$ sería obtener : $\mathcal{T}' = \mathcal{T} \bigwedge Y = g(B_1, B_2, ..., B_m)$, lo que implica, como se explicó anteriormente, hacer la intersección entre una estructura-AP y un conjunto

(operación $\bigwedge$). Esto no es más que obtener una nueva estructura-AP $\mathcal{T}'$, que tendrá como conjuntos generadores, los conjuntos resultantes de la intersección de Y con cada conjunto generador de $\mathcal{T}$. En nuestro ejemplo quedaría:

$\mathcal{T} \bigwedge Y = g(\{chocolate, nata\})$ siendo $\{chocolate, nata\}$ el resultado de la intersección de Y con $\mathcal{T}$ que no está incluido en ningún otro subconjunto de la intersección. Por el contrario, el conjunto $\{nata\}$ que sale como resultado de la intersección del primer elemento de Y con el primero de $\mathcal{T}$, ya está contenido en el conjunto de salida $\{chocolate, nata\}$ y por eso se descarta.

Ejemplo 12 Superestructura-AP inducida

Para esta definición, que es muy similar a la anterior, se tomarán las mismas estructuras usadas en la explicación anterior, sólo que en lugar de hacer la intersección, se hace la unión entre la estructura-AP $\mathcal{T}$ y el conjunto Y. La operación quedaría:

$\mathcal{T} \bigvee Y = g(\{chocolate, fresas, nata, nueces, tarta\},$
$\{fresas, nata, nueces, tarta, vainilla\},$

$\{chocolate, nata, tarta, vainilla\})$ que es el resultado de unir cada conjunto de Y con cada conjunto de $\mathcal{T}$, eliminando loa conjuntos redundantes.

Ejemplo 13 Unión e intersección de estructuras-AP

Al ser estas dos operaciones muy similares, se crearán las estructuras necesarias para resolverlas y se explica cada caso. Para la definición de estas dos operaciones necesitamos dos estructuras-AP $\mathcal{T}$ y $\mathcal{T}_1$. De ellas, $\mathcal{T}$ será la de nuestro ejemplo y se define $\mathcal{T}_1$.

$$\mathcal{T} = g(\{fresas, nata, nueces, tarta\}, \{chocolate, nata, tarta\})$$

$$\mathcal{T} = g(\{nueces, tarta\}, \{chocolate\})$$

La unión e intersección de $\mathcal{T}$ y $\mathcal{T}_1$ serían respectivamente:

1. $\mathcal{S} = \mathcal{T} \bigcup \mathcal{T}_1 = g(\{chocolate, fresas, nata, nueces, tarta\})$; en este caso, se han unido todos los elementos de los conjuntos generadores de $\mathcal{T}$ con los de $\mathcal{T}_1$ eliminando las redundancias.

2. $\mathcal{S}_1 = \mathcal{T} \bigcap \mathcal{T}_1 = g(\{nueces, tarta\}, \{chocolate\})$; en este caso, se han intersecado todos los elementos de los conjuntos generadores de $\mathcal{T}$ con los de $\mathcal{T}_1$ eliminando las redundancias.

Ejemplo 14 Acoplamientos fuerte y débil

Para determinar los acoplamientos fuerte y débil de un determinado conjunto Y con una estructura-AP $\mathcal{T}$, se tiene que resolver de qué forma ocurre la intersección de dicho conjunto con la estructura-AP. Si todos los elementos de Y se acoplan completamente con alguno de los conjuntos generadores de $\mathcal{T}$, será un acoplamiento fuerte, si esto no ocurre, y los elementos de Y se acoplan parcialmente con los conjuntos generadores de $\mathcal{T}$, será un acoplamiento débil. Para ejemplificar estas dos operaciones se toman como base las estructuras siguientes: el conjunto $Y = \{chocolate, vainilla\}$ y teniendo la estructura-AP de nuestro ejemplo:

$$\mathcal{T} = g(\{fresas, nata, nueces, tarta\}, \{chocolate, nata, tarta\})$$

Se define el acoplamiento fuerte y débil entre Y y $\mathcal{T}$ respectivamente como:

1. $Y \bigodot \mathcal{T} = falso$, ya que no existe acoplamiento fuerte entre Y y $\mathcal{T}$, porque no todos los elementos de Y se acoplan completamente con algún generador de $\mathcal{T}$. Concretamente, el conjunto $\{chocolate, vainilla\}$ no aparece definido en ninguno de los dos conjuntos generadores de $\mathcal{T}$.

2. $Y \bigoplus \mathcal{T}$ =verdadero, ya que existe acoplamiento débil entre Y y $\mathcal{T}$, porque el conjunto $\{chocolate, vainilla\}$ está parcialmente contenido en el conjunto generador $\{chocolate, nata, tarta\}$ de $\mathcal{T}$.

Ejemplo 15 Índices de acoplamiento fuerte y débil

Para determinar los índices de acoplamiento fuerte y débil de un determinado conjunto Y con una estructura-AP $\mathcal{T}$, se tiene que resolver con qué grado ocurre la intersección de dicho conjunto con la estructura-AP. Para calcular

estos índices se tienen en cuenta la cardinalidad del conjunto resultante del acoplamiento y la cardinalidad del conjunto con que se acopla.

Como se discutió anteriormente, este grado de acoplamiento se puede obtener de dos formas: calculando el promedio de la intersección con todos los conjuntos de la estructura-AP, o determinando el máximo de todos los acoplamientos calculados. Para ejemplificar estas dos operaciones, por ambas vías de cálculo, se toman como base las estructuras: el conjunto $Y = \{chocolate, tarta\}$ y la estructura-AP de nuestro ejemplo:

$$\mathcal{T} = g(\{fresas, nata, nueces, tarta\}, \{chocolate, nata, tarta\})$$

- **Cálculo de índices por el promedio**

En este caso, se define el índice de acoplamiento fuerte y débil entre Y y $\mathcal{T}$ respectivamente como:

1. $S(Y|\mathcal{T}) = \sum_{i \in S} m_i(Y)/n$
2. $W(Y|\mathcal{T}) = \sum_{i \in W} m_i(Y)/n$

Donde el coeficiente $m_i(Y)$ es el que define el grado con que se acopla el conjunto Y a cada conjunto generador de la estructura-AP. Para el caso del índice de acoplamiento fuerte, en la sumatoria sólo se tiene en cuenta cuando ocurre un acoplamiento fuerte entre Y y el conjunto A_i de la estructura-AP. Para el caso del índice de acoplamiento débil, en la sumatoria se tiene en cuenta cuando ocurre un acoplamiento débil entre Y y el conjunto $A_i$de la estructura-AP. El coeficiente $m_i(Y)$ se calcula dividiendo la cardinalidad de la intersección de Y con A_i entre la cardinalidad de A_i. El coeficiente n es la cantidad total de conjuntos generadores que componen la estructura-AP.

En el caso del ejemplo, estos índices tomarán los siguientes valores:

1. $S(Y|\mathcal{T}) = ((2/3)/2) = 0.66/2 = 0.33$
2. $W(Y|\mathcal{T}) = (((1/4) + (2/3))/2) = (0.25 + 0.66)/2 = 0.45$

Como se observa, para el cálculo del índice de acoplamiento fuerte, sólo se tiene en cuenta la intersección del conjunto $Y = \{chocolate, tarta\}$ con el conjunto generador de la estructura-AP $\{chocolate, nata, tarta\}$ que es el que lo contiene completamente. Para este caso el coeficiente $m_i(Y)$ toma el valor (2/3) donde 2 significa la cardinalidad de la intersección resultante de estos dos conjuntos y 3 la cardinalidad del conjunto $\{chocolate, nata, tarta\}$ con que se acopla completamente el conjunto Y. Ese resultado se divide entre 2, que es la cantidad de conjuntos generadores que tiene la estructura-AP. Realizando estos cálculos se obtiene un índice de acoplamiento fuerte de 0.33.

Para el cálculo del índice de acoplamiento débil, se tiene en cuenta la intersección del conjunto $Y = \{chocolate, tarta\}$ con los dos conjuntos generadores de la estructura-AP, pues ambos lo contienen al menos parcialmente. Para estos casos, el coeficiente $m_i(Y)$ toma dos valores, uno para cada conjunto de la estructura-AP. El valor (1/4) donde 1 significa la cardinalidad de la intersección resultante entre Y y el conjunto generador $\{fresas, nata, nueces, tarta\}$ y 4 la cardinalidad de dicho conjunto. El otro valor $m_i(Y)$ es (2/3) donde 2 significa la cardinalidad de la intersección resultante entre Y y el conjunto generador $\{chocolate, nata, tarta\}$ y 3 la cardinalidad de dicho conjunto. La sumatoria de ambos valores de $m_i(Y)$ se divide entre 2, que es la cantidad de conjuntos generadores que tiene la estructura-AP. Realizando estos cálculos se obtiene un índice de acoplamiento débil de 0.45.

Se debe destacar, que en ambos casos, el resultado cumple con lo planteado en las definiciones de índice de acoplamiento fuerte y débil que dichos valores siempre están entre $[0, 1]$. Además se obtiene que el índice de acoplamiento débil es mayor que el índice de acoplamiento fuerte para el mismo conjunto (0,45 > 0,33), algo también correcto acorde con dichas definiciones.

- **Cálculo de índices por el máximo**

Por esta otra variante, los cálculos de los valores de $m_i(Y)$ para cada tipo de acoplamiento son idénticos, sólo varían las expresiones para el cálculo del índice, que en este caso se definen de la forma siguiente:

1. $S(Y|\mathcal{T}) = max\,(m_i(Y))\,;\; i \in S$

2. $W(Y|\mathcal{T}) = max\,(m_i(Y))\,;\, i \in W$

En el caso del ejemplo, estos índices tomarán los siguientes valores:

1. $S(Y|\mathcal{T}) = max(2/3)$ =0.66
2. $W(Y|\mathcal{T}) = max(1/4, 2/3)) = max(0.25, 0.66) = 0.66$

Como se puede observar, por esta variante de cálculo de los índices de acoplamiento fuerte y débil, su valor es el mismo en ambos casos 0.66. Esto ocurre por la forma en que son calculados, dado que el único acoplamiento fuerte que ocurre, tiene un índice superior que el otro acoplamiento que ocurre cuando se calcula el índice débil (1/4).

Comparando ambas vías de cálculo, se puede apreciar que cuando estos índices son calculados por la variante del máximo, el valor final de ambos será mayor que cuando se obtiene por la variante de promediar el $m_i(Y)$ calculado para cada conjunto de la estructura-AP. En el caso que la estructura-AP tenga un sólo conjunto, el cálculo por ambas vías, sí dará el mismo resultado.

Cabe insistir en que cada vía será utilizada en dependencia de si se desea obtener el acoplamiento de un conjunto con la estructura-AP completa (*cálculo mediante el promedio*), o sólo determinar el índice de acoplamiento con el conjunto de la estructura-AP que mejor se acopla el conjunto buscado (*cálculo mediante el máximo*).

3.4. Conclusiones

Al iniciar el presente capítulo, se tenía como objetivo fundamental la formalización del modelo matemático que representa la nueva forma de representación intermedia obtenida, para estructurar un atributo textual. Para ello, se ha introducido el concepto de conjunto-AP como la estructura base, para representar los términos más frecuentes que aparecen en el atributo textual que se está procesando. Además, se definieron las operaciones para estos conjuntos-AP que permiten realizar acciones como verificar si un

conjunto-AP está incluido en otro y obtener los sub y super conjuntos-AP inducidos por un conjunto determinado. A continuación, se han definido el concepto de estructura-AP, así como sus operaciones. Se ha discutido que la estructura-AP no es más que el conjunto de conjuntos-AP generadores del retículo que forma dicha estructura-AP.

Entre las operaciones definidas para la estructura-AP se discutieron la inclusión de estructura-AP, así como, la sub y super estructura-AP inducida por un conjunto de términos determinado. Además se definieron las operaciones enfocadas a la realización de consultas sobre los datos almacenados. Dichas operaciones permiten la búsqueda de un conjunto de términos dentro de la estructura que contiene el conocimiento subyacente en los atributos textuales del problema que se aborde. Además, permiten hacer refinamiento de las búsquedas utilizando las definiciones de índices para acoplamiento fuerte y débil que fueron dadas. En dichas definiciones se dieron dos formas de cálculo según sea el interés del usuario, de encontrar el grado de acoplamiento con la estructura-AP completa, o con el conjunto de ella que mejor se acople el conjunto buscado.

En el próximo capítulo, se continúa con la formalización conceptual del modelo propuesto, esta vez, realizando su definición desde diferentes enfoques conceptuales que permitan llevar a cabo su implementación. Para ello se abordará la definición conceptual del modelo en lenguajes de definición de objetos y lenguaje de consulta estructurado estándar. Además, se ejemplificará su implementación en un gestor de bases de datos específico.

Capítulo 4

Implementación del modelo

En el presente capítulo se aborda el modelado conceptual de las estructuras propuestas, con vistas a llevar a cabo sus posibles implementaciones. Para ello, se propone una metodología que permite pasar del modelo matemático obtenido anteriormente, a su implementación concreta en un gestor de base de datos.

Se comienza en la sección 1 introduciendo algunos elementos del modelado de datos, así como haciendo una breve introducción a las herramientas a utilizar en la metodología de modelado que se propone. En la sección 2 se realiza un análisis comparativo de las posibilidades que brindan las herramientas seleccionadas. En la sección 3 se define la metodología a utilizar, a partir de la misma. En las secciones 4 y 5 se desarrolla la implementación del modelo propuesto en ODL y SQL:99 respectivamente. En la sección 6 se dan las ideas propuestas para la implementación del modelo en un SGBD con modelo de datos Relacional Orientado a Objetos (R.O.O). Se concluyen los pasos de la metodología propuesta en la sección 7 con la implementación del modelo en PostgreSQL. Finalmente, en la sección 8 se realiza un resumen y discusión de posibilidades que brindan las herramientas utilizadas para el modelado de datos.

4.1. Introducción al modelado de datos

A continuación se hace una introducción a las estructuras y operaciones que tendremos en cuenta para el modelado desde el punto de vista computacional. Además, se dan algunos elementos del surgimiento de las herramientas que utilizaremos en la metodología que se propone.

Un modelo de datos es aquel que describe de una forma abstracta cómo se representan los datos. Básicamente consiste en una descripción de algo conocido como contenedor de datos (algo en donde se guarda la información), así como de los métodos para almacenar y recuperar información de esos contenedores. Todo modelo de datos, está compuesto por tres elementos fundamentales:

1. **Atributos:** Son los descriptores del estado de una clase del modelo, y toman su valor sobre determinados dominios de datos. Normalmente son tipos primitivos de datos (cadena, numéricos, etc) o tipos más estructurados (tipo registro, vectores, etc).

2. **Relaciones:** Las relaciones conectan un objeto de una clase con uno o más objetos de otra clase.

3. **Métodos:** Implementan el comportamiento de la clase, dotándola de las rutinas necesarias para operar sobre sus propios atributos e intercambiar mensajes con el resto de las clases del modelo.

Estos tres elementos están interrelacionados entre sí, y en su conjunto conforman la implementación que realiza hoy día cualquier sistema con persistencia y manejo de datos. Desde los primeros sistemas orientados a dispositivos y ficheros, hasta las bases de datos puramente orientadas a objetos, cada uno con sus ventajas y desventajas, han logrado el modelado de estos tres componentes. El objetivo final siempre ha sido el mismo: lograr que las representaciones abstractas de los datos se conviertan en implementaciones físicas, con capacidades añadidas para su manejo. Precisamente, lograr este objetivo sobre el modelo matemático que se ha propuesto en el capítulo anterior, será la meta del presente capítulo. Para ello, realizaremos primero el modelado

en un lenguaje orientado a objetos como ODL, posteriormente en un lenguaje de consultas estructurado como SQL:99 y también en un gestor de bases de datos específico como lo es PostgreSQL. A continuación se enmarca el surgimiento histórico de estas herramientas que utilizaremos en el modelado.

4.1.1. Marco histórico

ODL

En la actualidad, el Paradigma de Programación Orientado a Objetos se ha impuesto como modelo para el análisis, diseño e implementación de sistemas, dado su robustez, escalabilidad y posibilidades de modelado semántica. Por otro lado CORBA(Common Object Request Broker Architecture) ha permitido el desarrollo de aplicaciones orientadas a objetos en entornos distribuidos. A menudo, estas aplicaciones necesitan almacenar de manera persistente, la información que manejan mediante Sistemas Gestores de Bases de Datos Orientados a Objetos (SGBDOO) (Cattell y Barry., 2000; de˜Miguel y Piattini., 1999). Los fabricantes de SGBDOO han realizado un esfuerzo realmente importante, especialmente en estos últimos años, para que se conviertan en populares.

Los aportes más importantes en este campo, sin duda, constituyeron los provenientes del Object Database Management Group, fundado en el verano de 1991, y que posteriormente se llamó Object Data Management Group (ODMG) (Cattell y Barry., 2000). Dicho grupo desarrolló un modelo de datos común basado en el modelo OMG, con su lenguaje asociado de consulta llamado Object Query Language (OQL). El estándar ODMG-93 de bases de datos orientadas a objetos (Cattell y Barry., 2000) tiene como objetivo garantizar la portabilidad entre sistemas, para lo cual define tres interfaces:

1. **ODL**: Como se discutió anteriormente, este lenguaje define el modelo de datos, manteniendo la compatibilidad con IDL (Interface Definition Language) (Vila, 2002) para la definición de objetos complejos, las relaciones entre ellos y sus métodos asociados.

2. **OQL:** Este lenguaje permite hacer consultas sobre los objetos anteriores, envío de mensajes a objetos y realizar reuniones y otras operaciones de tipo asociativo. Su sintaxis es muy similar a SQL.

3. **Conexión a través de C++ y SmallTalk:** Define las interfaces para programar una aplicación en estos lenguajes sobre una base de datos definida en ODL.

Precisamente utilizando ODL como ya se dijo, se comienza el proceso de pasar desde el modelo matemático propuesto hasta su implementación.. Posteriormente se realizará la implementación SQL:99, y por último se realizará la implementación en el gestor de bases de datos PostgreSQL.

SQL:99

La historia de SQL empieza en 1974 con la definición, por parte de Donald Chamberlin (Astrahan et al., 1976) y de otras personas que trabajaban en los laboratorios de investigación de IBM, de un lenguaje para la especificación de las características de las bases de datos que adoptaban el modelo relacional. Este lenguaje se llamaba SEQUEL (Structured English Query Language) y se implementó en un prototipo llamado SEQUEL-XRM entre 1974 y 1975. Las experimentaciones con ese prototipo condujeron, entre 1976 y 1977, a una revisión del lenguaje (SEQUEL/2), que a partir de ese momento cambió de nombre por motivos legales, convirtiéndose en SQL. En los años siguientes, éste ha sufrido diversas revisiones que han conducido primero a la versión SQL/89, SQL/92 y, posteriormente, a la actual SQL:99 (*también conocida como SQL3*) (Gulutzan y Pelzer, 1999; Melton y Eisenberg, 2000).

PostgreSQL

Dicho gestor tiene como ancestro a Ingres, desarrollado en la Universidad de California en Berkeley entre 1977-1985. El código de Ingres fue mejorado después por *Relational Technologies/Ingres Corporation*[1] la cual produjo uno

[1] Ingres Corporation fue posteriormente comprada por Computer Associates.

de los primeros sistemas comerciales exitosos de servidores de bases de datos relacionales. También en Berkeley, Michael Stonebraker lideró el equipo que desarrolló un servidor de base de datos llamado Postgres entre 1986-1994. Posteriormente la compañía Illustra[2] tomó el código de Postgres desarrollándolo en un producto comercial. Entre 1994-1995 se le añadieron a Postgres capacidades de SQL y el proyecto resultante se nombró Postgres95. Finalmente en 1996 se le cambió el nombre de Postgres95 a PostgreSQL en honor al nombre dado en Berkeley y a sus capacidades SQL.

A continuación se realiza un análisis comparativo de las posibilidades que brindan para el modelado de datos estas tres herramientas.

4.2. Análisis comparativo del modelado conceptual: ODL, SQL:99 y PostgreSQL

A continuación en la tabla 4.1 se resumen, de manera general, las potencialidades que tienen las herramientas utilizadas para el modelado conceptual. Para ello se tienen en cuenta las posibilidades que ofrecen para implementar los tres componentes de un modelo de datos. En las siguientes secciones de este capítulo se discutirán más en detalle las particularidades de cada una de ellas.

ODL

Como se observa, para el caso del modelado en orientación a objetos, el lenguaje ODL permite la definición de interfaces, como estructura para modelar una clase de objetos del mundo real. Dichas interfaces permiten la definición de los atributos de la clase de varios tipos: los tradicionalmente conocidos como primitivos o atómicos y los estructurados. Entre los atómicos están los

[2]Illustra más tarde fue comprado por Informix e integrado dentro de Informix's Universal Server.

	Interfaz ODL	Tipos y tablas SQL:99	Tipos, tablas y funciones PostgreSQL
Atributos	Permite la definición de atributos de tipos primitivos, estructurados y colecciones de ambos tipos de datos. No están permitidos atributos de tipos clase	Permite la definición de atributos de tipos primitivos y tipos estructurados. Además los tipos fila (Row), tipo de dato definido por el usuario (TDA) y colecciones de tipo vector	Permite la definición de atributos de tipos primitivos y tipos estructurados. Además los tipos TDA (con restricciones) y tipo vector
Relaciones	Permite la definición de relaciones bidireccionales entre objetos	Permite la definición de todos los tipos de relaciones presentes en el modelo relacional. Implementa el concepto de herencia de la Programación Orientada a Objetos a nivel de tipos y tablas	Permite la definición de todos los tipos de relaciones presentes en el modelo relacional. Implementa el concepto de herencia de la Programación Orientada a Objetos a nivel de tablas
Métodos	Permite la signatura de métodos que serán escritos en un lenguaje de implementación orientado a objetos	Permite las operaciones tradicionales de inserción, actualización y eliminación. Además la definición de procedimientos y funciones definidas por el usuario, disparadores y funciones que implementan el comportamiento de los TDA	Permite las operaciones tradicionales de inserción, actualización y eliminación. Además la definición de funciones definidas por el usuario y disparadores. No permite la definición de métodos asociados a los TDA ni procedimientos

Tabla 4.1: Análisis de implementación de un modelo de datos con las herramientas utilizadas.

de tipo booleano, cadena, enteros, etc. y el estructurado definido por *Struc* cuyos elementos son registros con n campos. ODL también permite la implementación de diferentes tipos de colecciones tanto de los tipos atómicos como del tipo estructurado, entre ellas las de tipo conjunto, lista y vector.

Para el caso de las relaciones en ODL, son propiedades que sirven para reflejar las conexiones entre objetos; estas conexiones son entre objetos de la misma u otra clase y pueden describirse desde un objeto a otro o desde un objeto a un conjunto de objetos. En muchas ocasiones, una pareja de relaciones establecidas entre dos clases son relaciones inversas entre sí, o sea, que se da la relación de una clase a otra y viceversa (Vila, 2002).

Los métodos en ODL permiten sólo su signatura pues tienen que ser escritos en un lenguaje orientado a objetos externo. En ODL, un método es una función asociada a una clase que puede tener además argumentos adicionales y que devuelve un determinado valor. Las declaraciones de los métodos aparecen junto con los atributos y las relaciones en la declaración de las interfaces. Como es normal en lenguajes objeto-orientados, cada método se asocia con una clase y los métodos se invocan sobre los objetos de la misma. La sintaxis de la declaración de métodos es similar a la de las declaraciones de funciones en *C* con dos importantes adiciones (Vila, 2002):

1. Permiten la declaración de parámetros de tipo *in*, *out* o *inout* según sea de entrada, salida o de entrada/salida respectivamente. Además, toda función tiene un valor de retorno que supone una forma alternativa de conseguir un resultado.

2. Permiten manejar excepciones que son respuestas especiales. Una excepción indica habitualmente una condición anormal que a su vez se tratará mediante otro método. En ODL, una declaración de función puede ser seguida por la cláusula *Raises*, seguida por una lista parentizada de una o más excepciones que la función puede tratar.

SQL:99

Para el caso de SQL:99, se debe destacar que su enfoque no es puramente orientado a objetos, si no que es un lenguaje estructurado que ha incorporado

extensiones para el modelado orientado a objetos. De aquí que permite la definición de un modelo de datos a partir de las estructuras básicas, tablas y tipos de datos que caracterizan al Modelo Relacional tradicional.

De manera muy similar a ODL, permite la definición de atributos de tipos primitivos, y tipos estructurados. Además permite la creación de tipos de datos fila (*Row*), comúnmente utilizados para definir varias tablas haciendo uso de un conjunto de atributos comunes, ya definidos previamente en este tipo de dato fila. Para el caso de las columnas se utiliza el concepto de *tablas con tipo,* donde los atributos que sean definidos dentro del tipo se convierten en columnas de la tabla. Para ello, se añade una columna más a la tabla, para definir el valor *REF* de la fila (el identificador OID), o sea, definir unívocamente el identificador que le corresponde dentro de las tuplas del tipo creado. Hay que destacar del concepto anterior, que el tipo *REF* no tiene la misma semántica que una restricción de integridad referencial, ya que la integridad referencial implica una dependencia de inclusión y las referencias no.

SQL:99 también permite la definición de TDA que se puede componer de uno o varios atributos, y además encapsula los métodos que operan sobre dichos atributos. También, para la definición de atributos, este lenguaje permite la definición de colecciones a través de vectores (*Array*). Esto posibilita romper la restricción tradicional del Modelo Relacional de que el valor de un atributo tiene que ser atómico (Codd, 1970). Con este tipo de dato *Array*, un atributo para una tupla determinada puede contener grupos repetitivos.

Para establecer las relaciones entre entidades en SQL:99 se utiliza la definición de restricciones de clave extranjera como lo plantea el modelo relacional. Además implementa el concepto de herencia de la Programación Orientada a Objetos a dos niveles (Melton, 2003):

1. **Herencia de tipos:** a través del uso del operador *under* en la sentencia *Create Type,* para definir herencia simple y múltiple entre tipos.

2. **Herencia de tablas:** a través del uso de los operadores *of* y *under* en la sentencia *Create Table,* para definir herencia entre tablas.

Para la definición de métodos en SQL:99 se tienen las operaciones tradicionales de un sublenguaje de manipulación de datos, la inserción, actualización y eliminación. Además se permite la definición de procedimientos y funciones definidas por el usuario, disparadores (procedimientos especiales que se ejecutan cuando ocurren determinadas operaciones que modifican la tabla en la que fueron definidos), y funciones que implementan el comportamiento de los TDA. Estas funciones ligadas a un TDA se definen en el tipo, pero sólo su signatura, pues se define de forma separada la especificación del cuerpo de la función. La diferencia fundamental entre un método y una función en SQL:99 es que la función de propósito general puede ser implementada en cualquier esquema de la base de datos, mientras que los métodos tienen que ser obligatoriamente implementados en el esquema que se encuentra en el TDA que las define (Melton, 2003).

PostgreSQL

Como se comentó anteriormente, PostgreSQL realiza su propia implementación del SQL:99 estándar. De aquí que la mayoría de las posibilidades que han sido discutidas para este lenguaje, ocurren de manera muy similar en este gestor de base de datos. La definición de atributos en PostgreSQL permite tipos primitivos y tipos estructurados. De manera similar al estándar permite el tipo vector.

Lo mismo sucede con los TDA, pero con una limitación muy importante, pues deja sólo definir operaciones asociadas sobre los atributos que conforman el tipo, referentes a su forma de escritura y lectura, y alguna función para brindar estadística sobre los contenidos del tipo de dato. Como se discutirá posteriormente, ésta ha sido la principal limitación que ha presentado PostgreSQL en la implementación del modelo propuesto.

Para la definición de relaciones, cumple todo lo conocido de integridad referencial del modelo relacional, y además permite la herencia entre tablas con la cláusula *INHERITS* de la sentencia *Create Table*. En este punto, a diferencia del SQL:99 estándar, se puede señalar que no permite la herencia entre tipos. En el caso de los métodos, permite las operaciones relacionales

tradicionales. Además, también permite la definición de funciones definidas por el usuario y disparadores muy poderosos, como métodos especiales asociados a tablas. Se debe destacar que los disparadores en PostgreSQL brindan una mayor flexibilidad y potencialidades en su definición y funcionamiento que gestores de bases de datos tan conocidos como SQL Server. Un ejemplo ilustrativo es que permiten especificar si el contenido de la función que ellos definen, se ejecuta para cada sentencia que afecta la base de datos, o para cada fila que es afectada por una sentencia. Esta posibilidad no está presente en SQL Server.

Finalmente, sobre la definición de métodos en PostgreSQL se debe señalar que no permite procedimientos, debido a que todos los métodos los implementa en funciones. Dichas funciones pueden ser definidas en lenguaje SQL o C. Además permite escribir funciones en otros lenguajes, que son generalmente llamados Lenguajes Procedurales (PL). Existen en la actualidad cuatro lenguajes procedurales disponibles en una distribución estándar de PostgreSQL: PL/pgSQL, PL/Tcl, PL/Perl y PL/Python. Otros lenguajes como Java pueden ser añadidos por el usuario. Esta robustez en cuanto a la cantidad de lenguajes procedurales que soporta PostgreSQL para la definición de funciones, hace que sea una de las mejores ofertas del mercado de gestores de bases de datos en cuanto a esta característica de soporte multilenguaje, tanto en el mundo del software libre como propietario.

Una vez que ya se han discutido las principales cuestiones de las herramientas que se utilizarán para el modelado conceptual de nuestro modelo, en la siguiente sección proponemos la metodología seguida para su implementación.

4.3. Metodología propuesta para la implementación del modelo

A continuación se discute la metodología que se ha utilizado para la implementación del modelo obtenido. El objetivo primordial es partir del modelo matemático e ir pasando por diferentes fases en su modelado que permitan llevar a cabo su implementación en un SGBD específico. El proceso seguido es el que describe la figura 4.1.

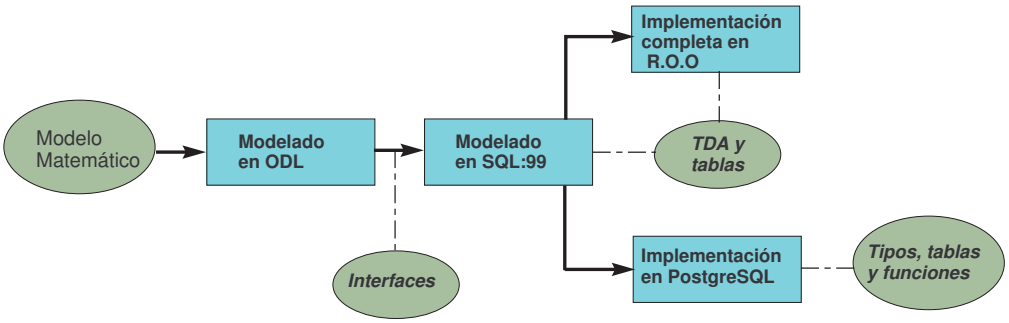


Figura 4.1: Metodología propuesta para la implementación del modelo.

Como se observa en la figura 4.1, se parte del modelo matemático que se ha definido en el capítulo anterior y se comienza con el modelado en ODL (Cattell y Barry., 2000) de las estructuras y operaciones definidas en él. Como resultado de este modelado se producen las interfaces que representan los conjuntos-AP y la estructura-AP. Dichas interfaces definen las clases del modelo, con sus atributos, relaciones y métodos. A continuación se realiza el modelado en SQL:99 como representación intermedia entre el modelado conceptual y la implementación en un Sistema de Gestión de Bases de Datos (SGBD). De este proceso salen las estructuras que implementa el SQL:99 estándar (Gulutzan y Pelzer, 1999) para modelar las interfaces antes obtenidas: el Tipo de Dato Abstracto (TDA), conocidos también como Tipo de Dato Definido por el Usuario (Melton, 2003) y las tablas propiamente dichas.

A partir de este punto, se está en condiciones de llevar a cabo su implementación en un SGBD que utilice el modelo de datos R.O.O y en PostgreSQL. Para el primero de estos dos casos, la idea sería utilizar un SGBD que implemente todas o la mayoría de las posibilidades que implementa el modelo R.O.O que es soportado por SQL:99. El mejor ejemplo sin dudas de este tipo de SGBD es Oracle 10G (Abbey, 2004; PRICE, 2004). Para ello se podría utilizar la definición de TDA tal como lo expresa el estándar SQL:99 y hacer uso de las tablas para el almacenamiento físico de los datos. Posteriormente en la sección 4.6 se propondrá una estrategia para lograr esta implementación.

Finalmente, para el caso de PostgreSQL, se realiza la implementación de estas estructuras derivadas de SQL:99. PostgreSQL (Stonebraker y Rowe, 1986) hace su propia implementación del estándar SQL:99. Utilizando los tipos, tablas y funciones que posee, queda implementado finalmente el modelo inicial, con la posibilidad de realizar consultas flexibles sobre los atributos textuales.

Hasta aquí se han discutido los elementos preliminares de cada herramienta para la definición de un modelo de datos, así como la metodología a seguir para la implementación del mismo. En la siguiente sección se discutirá más en detalle su implementación en lenguaje ODL.

4.4. Modelado en ODL

Teniendo en cuenta los aspectos discutidos sobre ODL en la tabla 4.1, se pasa ahora a la definición de las dos estructuras que componen nuestro modelo con sus operaciones, los conjuntos-AP y la estructura-AP. Para ello debido a que, como se discutió en el capítulo anterior, la estructura-AP es una colección de conjuntos-AP, se definirá la base de la jerarquía de interfaces que sería la interfaz *conjunto-AP*. La definición en ODL de dicha interfaz es la que se muestra en la figura 4.2.

Como se muestra en la figura 4.2, se construye la interfaz *conjunto-AP* para la que se definen dos atributos y tres métodos, sin que tenga ninguna relación con otra interfaz. En la línea (3) se define el identificador del conjunto-AP de tipo entero, mientras que en la línea (4) se define el atributo *conjunto-AP* como un conjunto de tipo cadena. Nótese que se utiliza este tipo de datos conjunto para almacenar en un vector los términos que pueden conformar el conjunto-AP, como se explicó en el capítulo anterior.

En las líneas (7), (8) y (9) se definen los métodos para esta estructura. En la línea (7) se define el método *inclusion_ conjunto-AP* que recibe como parámetro de entrada (*in*) un conjunto (*conjunto_ Y*) y retorna un valor booleano. Dicho valor se referirá a si está o no incluido el *conjunto_ Y* en el atributo *conjunto_ AP* de la instancia sobre la que se invoque dicho método.

1. ***interface*** **conjunto-AP {**
2. // Atributos
3. ***attribute*** *integer* **ID_conjunto-AP**;
4. ***attribute*** *Set <string>* **conjunto-AP**;
5. // Relaciones
6. // Métodos
7. *boolean* **inclusion_conjunto-AP** (*in Set <string>* conjunto_Y);
8. **subconjuntoAP_inducido** (*in Set <string>* conjunto_Y, *out Set <string>* subconjunto-AP_inducido);
9. **superconjuntoAP_inducido** (*in Set <string>* conjunto_Y, *out Set <string>* superconjunto-AP_inducido);**};**

Figura 4.2: Definición de la interfaz *conjunto-AP* en ODL.

Vale destacar que cada una de las operaciones a implementar por esta interfaz, se corresponde con las que fueron definidas en el modelo propuesto en el capítulo 3.

Los métodos *subconjuntoAP_ inducido* y *superconjuntoAP_ inducido* son implementados como procedimientos que reciben el parámetro de entrada *conjunto_ Y* y devuelven como parámetro de salida (*out*) los conjuntos resultantes de realizar sus operaciones. De forma análoga al método de la línea (7), estos conjuntos de salida de estos métodos, se referirán al conjunto obtenido de realizar dichas operaciones sobre el atributo *conjunto_ AP* de la instancia para la que se invoquen dichos métodos.

Ahora, como se comentó anteriormente, se hace uso de la interfaz *conjunto-AP* para, definir la interfaz *estructura-AP.* Como se ha dicho antes, las relaciones son las que permiten hacer la herencia entre interfaces o definir atributos en una interfaz a partir de otro tipo de interfaz que exista en el modelo.

La definición en ODL de la interfaz *estructura-AP* es la que se se muestra en la figura 4.3. Como se observa en dicha figura, en las líneas (2) y (3) se

definen los dos atributos de esta interfaz. El atributo *ID_ estructura-AP* es definido de tipo entero y se refiere al identificador de la clase, mientras que *atributo_fuente* de tipo cadena, se refiere al nombre del atributo textual que se ha procesado para obtener dicha estructura-AP.

```
1.  interface estructura-AP {
2.  // Atributos
3.  attribute integer ID_estructura-AP;
4.  attribute string atributo_fuente;
5.  // Relaciones
6.  relationship Set <conjunto-AP> estructura-AP;
7.  // Métodos
8.  boolean inclusion_estructura-AP (in Set <conjunto-AP> otra_estructura-AP );
9.  subestructura_inducida (in Set <string> conjunto_Y, out estructura-AP );
10. superestructura_inducida (in Set <string> conjunto_Y, out estructura-AP );
11. union_estructura-AP (in Set <conjunto-AP> otra_estructura-AP, out estructura-AP );
12. interseccion_estructura-AP (in Set <conjunto-AP> otra_estructura-AP, out estructura-AP
    );
13. boolean acoplamiento_fuerte ( in Set <string> conjunto_Y);
14. boolean acoplamiento_debil ( in Set <string> conjunto_Y);
15. float indice_acoplamiento_fuerte ( in Set <string> conjunto_Y);
16. float  indice_acoplamiento_debil ( in Set <string> conjunto_Y); };
```

Figura 4.3: Definición de la interfaz estructura-AP en ODL.

En la línea (6) se observa la relación que se ha creado utilizando la cláusula *relationship.* Dicha relación tiene por nombre *estructura-AP* y, como se ve, es un conjunto de conjuntos-AP, que es exactamente lo que significa la estructura-AP del modelo propuesto. Mediante esta relación *estructura-AP*

será que se puede hacer referencia entonces desde la interfaz *estructura-AP* a los atributos y métodos definidos en la interfaz *conjunto-AP*.

En las líneas de la (8) a la (16) se definen los métodos correspondientes a las operaciones del modelo propuesto para una estructura-AP. Como se observa por ejemplo en la línea (8), el método *inclusion_ conjunto-AP* recibe como parámetro de entrada el parámetro *otra_ estructura-AP* (definido como una colección de la interfaz *conjunto-AP*), que no es más que una estructura-AP que se pasa, para verificar su inclusión o no, en el atributo *estructura-AP* de la instancia de la interfaz sobre la que se haga la llamada del método antes mencionado.

Los métodos de las líneas (11) y (12) también reciben como parámetro una estructura-AP y devuelven otra en su parámetro de salida (*out*). Dicho parámetro de salida resulta de hacer sus operaciones entre la estructura-AP de entrada, y la que contiene la instancia de la clase desde la que se invoca el método.

Los métodos de las líneas (13) y (14) toman como parámetro de entrada un conjunto de términos (*conjunto_ Y*) y dicen si tiene o no acoplamiento fuerte o débil, respectivamente, con la estructura-AP que contiene la instancia de la clase desde la que se invocan estos métodos.

Finalmente, los métodos de las líneas (15) y (16) devolverán un decimal correspondiente al índice de acoplamiento fuerte o débil respectivamente, con que se acopla el *conjunto_ Y* con la estructura-AP que contiene la instancia de la clase desde la que se invocan estos métodos.

Hasta este punto, se ha realizado ya la definición de las estructuras del modelo propuesto y sus operaciones en ODL, bajo el enfoque de orientación a objeto que implementa este lenguaje. En la siguiente sección, se discutirá el paso siguiente de la metodología propuesta, referente ahora al modelado del sistema en lenguaje SQL:99 estándar. Para ello, tomaremos como punto de partida el conocimiento obtenido en el paso precedente.

4.5. Modelado en SQL:99 estándar

Es importante tener en cuenta que, mientras que OQL no tiene una notación específica para el concepto de relación (*del modelo relacional*), que podrían ser modeladas como un un *set* o *bag* de estructuras; en SQL:99 lo mismo que en SQL la relación es un concepto central. Por ello, los objetos en SQL:99 se pueden definir de dos maneras:

1. Objetos fila que son esencialmente tuplas.
2. Tipos abstractos de datos (TDA), objetos generales que sólo pueden ser usados como componentes de una tupla, es decir, como dominios donde puede valorarse un atributo de una tupla.

A continuación se muestran los tipos de implementaciones que realiza SQL:99 para los TDA, quedando explicadas de esta forma, las dos posibilidades de definición de objetos antes mencionadas.

4.5.1. Tipos de implementaciones de TDA en SQL:99

Como se comentó anteriormente, para la implementación de las estructuras del modelo en SQL:99 se utilizarán fundamentalmente los TDA con sus métodos asociados, y el tipo vector para representar la colección de términos del atributo textual. Aunque ya en la explicación que se ofreció de la tabla 4.1 se dieron algunos detalles de cómo SQL:99 implementa este tipo de dato, a continuación se realiza un análisis más detallado de las tres variantes de implementación posibles (Melton, 2003). También se dan algunos ejemplos.

1. El TDA como un tipo distinto (*distinct type (Melton y Simon, 2002)*): Es un tipo de dato que puede construir el usuario a partir de un sólo tipo de dato base (numérico, cadena, etc). Este tipo de dato tiene la restricción de que no puede ser mezclado en ninguna operación con otros atributos de su mismo tipo base, ni con otros datos de tipo distinto (que sea de su mismo tipo base). Dicho tipo se puede utilizar para ser aplicado a una variable o a un atributo de una tabla.

2. El TDA como valor de objeto: Su uso concreto es para modelar entidades con relaciones y comportamiento. Esta nueva forma como valor de objeto, permite la implementación del concepto de identificador de objeto (normalmente representado por OID) de la Programación Orientada a Objetos. El SQL lo implementa, como comentamos anteriormente, utilizando el concepto de *tablas con tipo*, donde entre sus atributos se encuentra un tipo fila (*Row*) que ha sido construido a partir de un TDA estructurado mediante la sentencia *Create Row Type*. En la nueva tabla creada a partir de este TDA se añade una columna más a la tabla, para definir el valor *REF* de la fila (el identificador OID) que hace que las instancias de este tipo sean identificadas unívocamente.

3. El TDA como valor de atributo: Se utiliza para definir un tipo estructurado, compuesto por varios atributos con su tipo, que pueden ser de un tipo base u otro tipo estructurado definido por el usuario. Su función principal es modelar atributos de las entidades. Un ejemplo clásico en la literatura es el del tipo *Direccion* que se define como un tipo estructurado compuesto por los atributos *nombre_calle*, *numero_apartamento*, *ciudad*, *estado*, *pais* y *codigo_postal*. Además de sus atributos, este tipo de dato abstracto, como se ha comentado anteriormente, permite la definición de sus métodos. Comúnmente, éstos son utilizados en comparaciones de instancias del tipo creado, para la conversión de valores a dicho tipo (*casting*) y para la implementación de su comportamiento específico como tipo de dato. La figura 4.4 muestra la sintaxis genérica de la definición de un TDA. La sintaxis completa y explicada en detalle se puede encontrar en (Melton, 2003).

1. **CREATE TYPE** *<nombre-tipo>*
2. *lista de atributos y sus tipos*
3. *declaraciones opcionales de los operadores =,>,<, etc.. asociados al tipo*
4. *declaraciones de funciones (métodos) asociados al tipo*

Figura 4.4: Sintaxis general de la instrucción *Create Type* de SQL:99.

En esta sintaxis general de la figura 4.4, de las líneas (1) y (2) se debe recordar que es en este punto donde la creación de tipos permite la herencia, al

poder utilizar la cláusula UNDER para crear un subtipo de un tipo previamente definido. Además, la lista de atributos también permite definirlos de un tipo creado por el usuario previamente. La línea (3) indica que es posible establecer operadores de comparación específicos para el tipo de que definimos. Estos operadores se implementarán posteriormente como funciones. Aquí realmente lo que se hace es dar su signatura. La forma genérica de este tipo de declaraciones es:

OPERADOR *<nombre de la función que implementa el operador >*

donde **OPERADOR** se sustituirá en su caso por *EQUALS*, *LESS THAN*, etc. Se debe tener en cuenta que para poder incluir comparaciones asociadas a estos tipos de datos en una sentencia SQL, cuando estos aparezcan como valores de un atributo, es necesario tener definidos estos operadores previamente. Esto se debe a que los conceptos de igualdad y orden son altamente dependientes del tipo de dato, si éste es complejo y definido por el usuario. La línea (4) indica que se pueden declarar métodos adicionales. SQL:99 proporciona ciertas funciones previamente construidas que no necesitan ser declaradas. Estas funciones son:

1. **Constructor:** Una función que devuelve un nuevo objeto del tipo. Todos los atributos del objeto creado son inicialmente nulos. Si T es el nombre del tipo, $T()$ es la función constructor.

2. **Observadoras:** Son funciones para cada uno de los atributos del objeto; si X es una variable de un determinado objeto, entonces $A(X)$ es el valor del atributo A para dicha variable. También se suele usar la notación de punto ($X.A$) para nombrar a dicho valor.

3. **Cambiadoras (mutator):** Estas funciones sirven para asignar un valor a cada atributo de un objeto. Se utilizan normalmente en el lado izquierdo de una sentencia de asignación. Hay que destacar que si se desea un verdadero encapsulamiento es necesario evitar que estas funciones sean de uso público. En SQL:99 se utiliza para ello un mecanismo de gestión de privilegios(Melton, 2003).

Dado que de estas tres variantes discutidas para la implementación de un TDA en SQL:99, se utilizará esta última (*el TDA como valor de atributo*), se presentan a continuación algunas características de los métodos (funciones que encapsulan el acceso a los atributos del TDA e implementan su comportamiento) en SQL:99.

4.5.2. La definición de métodos en un TDA como valor de atributo en SQL:99

En SQL:99, un método es un tipo especial de función, que es invocado usando una variación a la sintaxis de invocación de una función ordinaria. Los métodos y las funciones ordinarias presentan muchas similitudes, pero existen diferencias importantes entre ellas, tanto en su declaración como en su uso (Melton, 2003). Algunas de las más importantes son:

1. Cada método está fuertemente ligado al TDA que lo define, mientras que las funciones son libres de definirse en cualquier esquema de la base de datos.

2. Cada método tiene que ser declarado en el mismo esquema de la base de datos donde fue definido el TDA al que corresponde.

3. Cada método asociado a un TDA tiene que ser definido en la declaración del TDA. Una vez que es declarado, tiene que ser implementado.

4. La lista de parámetros de métodos de instancia y constructores (no métodos estáticos), siempre incluye un parámetro implícito adicional (parámetro que no tiene que ser definido explícitamente). Dicho parámetro, usualmente llamado *parámetro SELF*, es efectivamente el primer parámetro en la lista y su tipo de datos es siempre el tipo de dato TDA al que está asociado. Dentro de la implementación del método, se usa para acceder con la notación de punto a los atributos del TDA (*SELF.nombre_atributo*).

5. En la invocación de métodos de instancia y constructores, también es añadido un parámetro adicional (*el argumento SELF*), que no es

incluido en la lista de parámetros que se le pasan al método en la invocación. El valor de este argumento es dado con la notación de punto.

6. Los métodos sólo permiten parámetros de entrada *IN*; no permiten los tipos de parámetros *OUT* o *INOUT* como las funciones. Esto es para reforzar su función de encapsular sus datos.

7. La resolución de los métodos ocurre parcialmente en tiempo de compilación, mientras que la de las funciones queda completamente resuelta. Esto se debe a la posibilidad del uso de *métodos sobrecargados.* De aquí que se resuelven en tiempo de compilación un conjunto de métodos con el mismo nombre, entre las clases y subclases (recordemos que SQL:99 permite tanto la herencia entre tablas como entre tipos), y según el valor de los parámetros o desde donde sea invocado, será la resolución final del método que se ejecuta.

La forma de una declaración de funciones en SQL:99 es la siguiente (Vila, 2002):

FUNCTION <*nombre*> (<*argumentos*>) **RETURNS** <*tipo*>

Como cualquier lenguaje, en SQL:99 la declaración de una función especifica su nombre, lista de argumentos entre paréntesis (aunque no lleve ninguno es obligatorio poner paréntesis vacíos) y tipo que retorna la función en su invocación. Cada argumento consta de un nombre de variable y de un tipo de variable. Los argumentos están separados por comas. Las funciones son de dos tipos: internas y externas. Las funciones externas se escriben en el lenguaje anfitrión y sólo aparecen en la declaración las signaturas de las mismas. Las funciones internas se escriben en un SQL extendido que permite la declaración de variables, operaciones de asignación, construir expresiones, etc. Para iniciar y terminar el cuerpo de una función se utiliza *BEGIN* y *END*.

Teniendo en cuenta todos los elementos que se han dado en las dos apartados anteriores, en en el siguiente pasamos directamente a realizar la implementación de las estructuras que componen nuestro modelo en SQL:99.

4.5.3. Definición de conjunto-AP y estructura-AP en SQL:99

Como se discutió en el capítulo anterior, la colección de conjuntos-AP que se obtiene del procesamiento del atributo textual será la que conforma la estructura-AP, que será el Dominio Activo sobre el que se valorará la representación que se obtiene para cada tupla de dicho atributo. Es por ello que se comienza por la representación en SQL:99 de la estructura conjunto-AP para posteriormente, definir la estructura-AP. En la figura 4.5 se da dicha definición. Como se observa en dicha figura, se crea el TDA *conjunto-AP* con los atributos *id_ conjunto-AP* y *conjunto-AP*; el primero como identificador del conjunto-AP y el segundo que tendrá un vector de tipo cadena, donde se almacena los conjuntos-AP.

Este TDA *conjunto-AP* se crea como *INSTANTIABLE* y *NOT FINAL.* Se utiliza la cláusula *INSTANTIABLE* para especificar que se pueden crear instancias de dicho tipo, o sea, se habilita la posibilidad de que pueda ser referenciado como tipo de dato para un valor de cualquier otro atributo. Lo que no inhabilita la posibilidad que se pueda crear un subtipo de este TDA *conjunto-AP,* según lo que se especifique en la cláusula *FINAL.* Por el contrario, si se quisiera que no se puedan crear instancias de él directamente, y que sólo existiera como una superclase a partir de la cual se creen subclases, sería necesario definirlo como *NOT INSTANTIABLE.*

La cláusula *NOT FINAL*, como se explicó anteriormente, indica que está permitida la herencia entre tipos, o sea, que se puede crear un nuevo tipo de datos tomando como punto de partida TDA *conjunto-AP* con la cláusula *UNDER.*

1. **CREATE TYPE** *conjunto-AP* **(**
2. *id_ conjunto-AP* **INTEGER,**
3. *conjunto-AP* **CHARACTER VARYING** (40) **ARRAY** [2000],
4. **INSTANTIABLE**
5. **NOT FINAL**
6. **METHOD** *inclusion_ conjunto-AP* (: *conj_ Y* **CHARACTER ARRAY** [10]) **RETURNS BOOLEAN** ;
7. **METHOD** *subconjunto-AP_ inducido* (: conj_Y **CHARACTER ARRAY** [10]) **RETURNS conjunto-AP** ;
8. **METHOD** *superconjunto-AP_ inducido* (: conj_Y **CHARACTER ARRAY** [10]) **RETURNS conjunto-AP** ;
9. **);**

Figura 4.5: Definición en SQL:99 de la estructura *conjunto-AP*.

En las líneas de la (6) a la (8) de la figura 4.5, también se pueden ver los tres métodos declarados para el TDA *conjunto-AP*. Como se observa, la declaración del método comienza con la instrucción *METHOD*, a continuación el nombre del método seguido de su lista de parámetros y finalmente la cláusula *RETURNS* que permite especificar el tipo del valor de retorno de la función que implementa el método. También se puede ver que en SQL:99 los parámetros van precedidos de los dos puntos (:).

Como el significado de los métodos y sus parámetros de entrada y salida son muy similares a la definición discutida en la figura 4.2, por eso no se comentarán en detalle.

A continuación, en la figura 4.6 se muestra cómo se realiza la implementación del método *inclusion_ conjunto-AP* en SQL:99. Dicho método fue definido por el TDA *conjunto-AP* que aparece en la figura 4.5.

Como se observa en la figura 4.6, el método se declara de tipo *INSTANCE*, lo que quiere decir que es un método que opera sólo sobre una instancia

```
1. CREATE INSTANCE METHOD inclusion_ conjunto-AP (
       : conj_ Y CHARACTER ARRAY [10 ])
2. FOR conjunto-AP
3. BEGIN
4.    IF (: conj_ Y  IN  SELF.conjunto-AP)
5.       THEN RETURN TRUE;
6.       ELSE RETURN FALSE;
7.    ENDIF;
8. END;
```

Figura 4.6: Implementación del método *inclusion_ conjunto-AP* en SQL:99.

del TDA para el que se define (Melton, 2003). Normalmente un método de tipo *INSTANCE* necesita conocer el estado de los atributos de la instancia para obtener su valor de retorno, de aquí que sólo se puedan invocar sobre instancias del TDA y no sobre el tipo TDA propiamente. En SQL:99 el tipo de método *INSTANCE* es el por defecto; por ende la definición anterior se pudo encabezar con **CREATE METHOD** *inclusion_ conjunto-AP*, teniendo el mismo significado.

Como se comentó anteriormente, en SQL:99 las funciones permiten parámetros, que se pueden definir de entrada (*IN*), de salida (*OUT*) o de entrada salida (*INOUT*). En el caso de los métodos sólo están permitidos los parámetros de entrada; su valor de retorno va en el tipo que devuelve la función que lo implementa. El tipo de parámetro por defecto es *IN*; por eso en la implementación del método *inclusion_ conjunto-AP* se ha omitido el tipo de parámetro. Esto quiere decir que el parámetro *conj_ Y* es de entrada. La lista de parámetros y su tipo irán separadas por coma, y en caso que el método no tenga parámetros de entrada, es obligatorio indicar paréntesis vacío a continuación del nombre del método.

En la línea (2) de la mencionada figura, aparece la cláusula *FOR* que es la que indica para qué tipo, de los que existan en el esquema de la base de

datos, se está definiendo el método; en el caso del ejemplo se dice que es para el tipo *conjunto-AP*.

Desde la línea (3) a la (8) aparece entre *Begin-End* la definición del cuerpo del método. En la línea (4) se verifica si el conjunto de entrada *:conj_ Y* está en el atributo *conjunto-AP* que tiene el TDA sobre el que se está definiendo en método. Obsérvese el uso del parámetro *SELF* para hacer referencia al tipo *conjunto-AP* que es sobre el que se esta definiendo el método. Con la notación *SELF.conjunto-AP* se está accediendo de forma encapsulada al atributo del tipo. Como se planteó anteriormente, el parámetro *SELF* se pasa implícitamente en la definición de un método, y es el primero en la lista de parámetros. En esa misma línea (4) se utiliza el operador *IN* para verificar si el *conj_ Y* está o no incluido en el *conjunto-AP* de la instancia que invoque este método. Finalmente en las líneas (5) y (6) se utiliza la instrucción *RETURN* para indicar que se retorne verdadero o falso respectivamente, según sea la evaluación de la expresión de la línea (4).

A continuación, en la figura 4.7 de igual forma, se muestra la definición en SQL:99 del TDA que permitirá luego crear la tabla que contendrá las estructuras-AP. Como se indicó anteriormente, usando el TDA definido para el conjunto-AP se define el TDA para las estructuras-AP.

Como se observa en la figura 4.7, se crea el TDA *estructura-AP* con sus atributos *id_ estructura-AP* que se refiere al identificador de la estructura-AP y *estructura-AP* como un vector del tipo *conjunto-AP* que fue el TDA definido anteriormente. De igual forma al caso anterior, se define el tipo como *INSTANTIABLE* y *NOT FINAL*.

En las líneas de la (6) a la (14) se definen los métodos del TDA. Los métodos de las líneas (6), (9) y (10) reciben el parámetro : *otra_ estructura-AP* que como su nombre lo indica, es una estructura-AP definida como un vector de tipo *conjunto-AP*. Las operaciones de estos métodos se realizarán entonces entre esta estructura-AP que se pasa como parámetro y la estructura-AP de la instancia que invoque dichos métodos. El resto de los métodos, reciben como parámetro *:conj_ Y* que será el conjunto necesario par realizar el resto de las operaciones que define el modelo para esta estructura, siempre sobre la estructura-AP de la instancia que invoque dichos métodos.

1. **CREATE TYPE** *estructura-AP* **(**
2. *id_estructura-AP* **INTEGER,**
3. *estructura-AP* **conjunto-AP ARRAY** [2000],
4. **INSTANTIABLE**
5. **NOT FINAL**
6. **METHOD** *inclusión_estructura-AP* (: *otra_estructura-AP* **conjunto-AP ARRAY** [2000]) **RETURNS BOOLEAN** ;
7. **METHOD** *subestructura-AP_inducida* (: conj_Y **CHARACTER ARRAY** [10]) **RETURNS conjunto-AP** ;
8. **METHOD** *superestructura-AP_inducida* (: conj_Y **CHARACTER ARRAY** [10]) **RETURNS conjunto-AP** ;
9. **METHOD** *union_estructura-AP* (: *otra_estructura-AP* **conjunto-AP ARRAY** [2000]) **RETURNS conjunto-AP** ;
10. **METHOD** *interseccion_estructura-AP* (: *otra_estructura-AP* **conjunto-AP ARRAY** [2000]) **RETURNS conjunto-AP** ;
11. **METHOD** *acoplamiento_fuerte* (: conj_Y **CHARACTER ARRAY** [10]) **RETURNS BOOLEAN** ;
12. **METHOD** *acoplamiento_debil* (: conj_Y **CHARACTER ARRAY** [10]) **RETURNS BOOLEAN** ;
13. **METHOD** *indice_acoplamiento_fuerte* (: conj_Y **CHARACTER ARRAY** [10]) **RETURNS NUMERIC** ;
14. **METHOD** *indice_acoplamiento_debil* (: conj_Y **CHARACTER ARRAY** [10]) **RETURNS NUMERIC** ; **);**

Figura 4.7: Definición en SQL:99 de la *estructura-AP*.

Una vez que ya se tienen definidos los tipos para la creación de la tabla que físicamente almacenará los datos correspondientes a los conjuntos-AP y la estructura-AP, en la figura 4.8 aparece la definición de dicha tabla.

1. **CREATE TABLE** *estructuras-AP* (
2. *fecha_creacion* **DATE**
   CONSTRAINT *fecha_creacion_no_nulo* **NOT NULL**,
3. *valor_estructura-AP* **estructura-AP**,
4. *atributo_fuente* **CHARACTER VARYING (20)**,
5. **CONSTRAINT** *estructuras-AP_llave_primaria*
   PRIMARY KEY (*fecha_creación*));

Figura 4.8: Definición de la tabla *estructuras-AP* en SQL:99.

Como se observa en la figura 4.8, se ha creado la tabla *estructuras-AP* tomando como base el TDA *estructura-AP* para definir el atributo *valor_estructura-AP* en la línea (3). Además se añaden los atributos *fecha_creacion* de tipo fecha que puede indicar la fecha en la que se creó la estructura-AP, y el *atributo_fuente* de tipo cadena, que puede indicar el nombre del atributo textual sobre el que se ha obtenido dicha estructura-AP.

Como se ve en la línea (2), sobre el atributo *fecha_creacion* se construye la restricción *fecha_creacion_no_nulo* la que indica que dicho atributo no puede tomar valores nulos. Además también en la línea (5) se crea la restricción *estructuras-AP_llave_primaria* sobre el mismo atributo, para indicar que será la llave primaria de la tabla.

De esta definición que se ha realizado en la figura 4.8 se debe destacar la definición realizada en la línea (3). Aquí se muestra el uso que se le ha dado al TDA, para definirlo como tipo de dato para un atributo de una tabla. Este uso del TDA, como se dijo anteriormente, es el que se ha usado en estas definiciones en SQL:99.

Una vez concluida la implementación del modelo en SQL:99 y siguiendo la metodología propuesta en la figura 4.1, a continuación se discute el próximo paso a seguir, que sería la implementación en un sistema con modelo de datos

Relacional Orientado a Objetos (R.O.O) de las estructuras que componen nuestro modelo. Concretamente, en la siguiente sección se discuten todo un conjunto de elementos a tener en cuenta, para la implementación del modelo en un SGBD específico que implemente este modelo R.O.O.

4.6. Estrategia para la implementación en un sistema R.O.O

Acorde a las ideas presentadas hasta el momento en este trabajo, a continuación se introduce la estrategia de implementación seguida para, haciendo uso de los conjuntos-AP y estructura-AP, dotar a una base de datos de la posibilidad de realizar consultas flexibles y más poderosas sobre sus atributos de tipo texto corto. Retomando las ideas presentadas en (Marín et al., 2006), se discuten en detalle algunas ideas previas a tener en cuenta para la implementación del modelo obtenido en un modelo de datos Relacional Orientado a Objetos. Dicho modelo está soportado por el lenguaje SQL:99; por tanto todas las cuestiones discutidas sobre ese lenguaje podrían ser utilizadas.

La idea básica que se desarrollará en esta sección es lograr lo que se expresa en la tabla 4.2, que no es más que lograr la implementación de los valores de atributos de tipo texto corto, como conjunto de términos que tomarán valores sobre su dominio activo que es la estructura-AP.

Elemento	**Representación extendida**
Valores de atributos	Conjunto de términos
Domino activo del atributo	Estructura-AP

Tabla 4.2: Representación extendida de atributos textuales.

Para conseguir esta idea, se ha desarrollado una estrategia para lograr el almacenamiento de los conjuntos-AP y la estructura-AP, de la forma más óptima posible. A continuación se discuten algunos elementos a tomar en cuenta en este proceso.

4.6.1. Representación de los Metadatos

Con el propósito de dar soporte al objetivo deseado de mejorar la recuperación de información sobre atributos de tipo texto corto, se necesitará almacenar la estructura-AP para cada atributo de tipo texto corto presente en una relación. Por consiguiente la base de datos en cuestión necesitará algunos metadatos asociados a cada tabla, como muestra la figura 4.9.

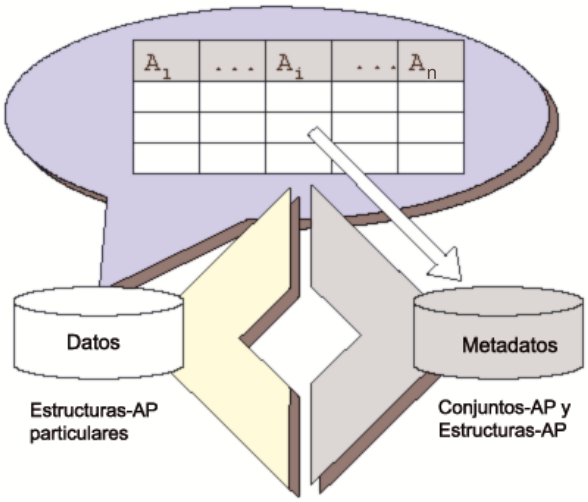


Figura 4.9: Soporte para relaciones con atributos textuales.

Como se observa en la figura, se trata de almacenar las estructuras-AP particulares que corresponde a cada tupla de atributo textual A_i, y obtener los metadatos correspondientes a dicho atributo. Estos metadatos serán los referentes a los conjuntos-AP, que son obtenidos durante el procesamiento de A_i, además de la estructura-AP que describe el dominio activo, sobre el que se valorará dicho atributo A_i.

En la figura 4.10 se representa la relación entre los datos y los metadatos. Como se puede observar, cada atributo de tipo texto corto en una relación necesitará:

1. La capacidad de representar sus distintos valores como un conjunto de términos y su estructura-AP particular. Esto es, la estructura-AP

inducida por el valor del atributo para cada tupla, sobre la estructura-AP que forma su dominio activo.

2. Almacenar la estructura-AP que representa su dominio activo.

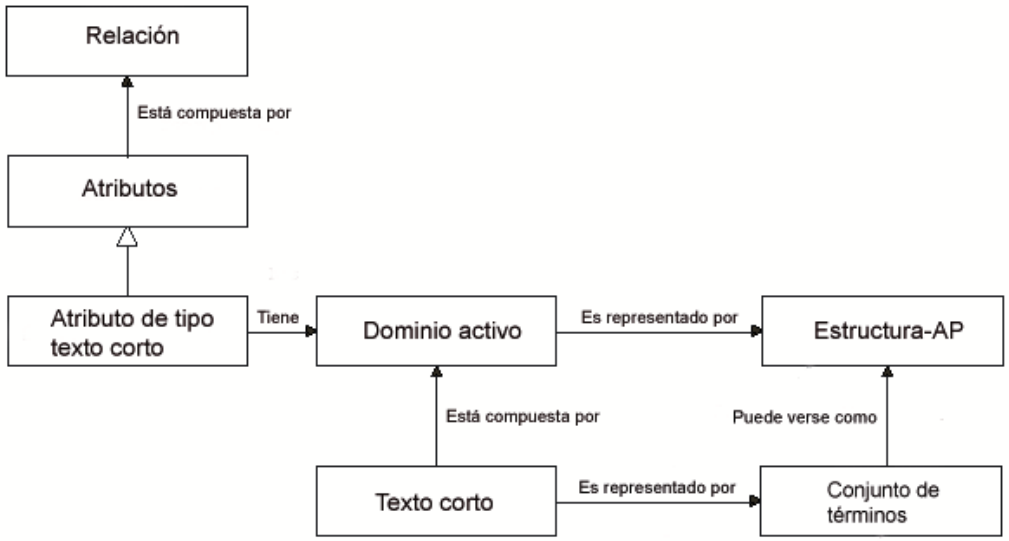


Figura 4.10: Relación entre los datos y los metadatos.

Para llevar a cabo el primero de estos dos requerimientos, se puede simplemente implementar una extensión de alguno de los tipos cadena tradicionales, como pudieran ser los vectores soportados por SQL:99. De esta manera se lograría el almacenamiento de los textos cortos como un conjunto de términos (con la capacidad de interactuar con la estructura-AP). Para mejorar la eficiencia, se puede añadir una columna adicional a la relación, en la que se escriba para cada tupla, el conjunto de términos que componen el atributo de tipo texto corto.

Para soportar el almacenamiento de la estructura-AP que representa el dominio activo, se puede adicionar una nueva relación en la base de datos. Esta nueva relación será un metadato en la cual cada tupla tiene que ser capaz de indicar:

1. El nombre del atributo textual.
2. El nombre de la relación original (relación de la que proviene el atributo textual procesado).
3. La estructura-AP que representa el dominio activo del atributo textual.

En la siguiente subsección se realiza una discusión más detallada de, cómo se puede realizar la representación física en un gestor de bases de datos, con modelo de datos R.O.O de los TDA correspondientes a estas dos estructuras, conjunto-AP y estructura-AP.

4.6.2. Alternativas de almacenamiento de los TDA conjunto-AP y estructura-AP

Como se ha comentado anteriormente, el cálculo de los conjuntos-AP que se obtienen de los textos cortos, se realiza por medio del algoritmo APriori (Agrawal y Srikant, 1994), junto con algunos de los mecanismos bien conocidos para eliminar las palabras de parada en textos (Salton y McGill, 1983), tales como artículos, pronombres, preposiciones, etc. Las poderosas capacidades de modelado de los SGBD objeto-relacionales facilita la implementación y uso de los conjuntos-AP y estructuras-AP en bases de datos convencionales. Ambas estructuras, son grafos orientados, que pueden ser almacenadas con dos niveles diferentes de detalle:

1. Representando sólo el conjunto generador del conjunto-AP y la colección completa de conjuntos generadores para el caso de la estructura-AP.
2. Representando los grafos completos, es decir, almacenar cada estructura desde su conjunto generador, hasta los elementos de longitud 1 (nodos hojas).

En la tabla 4.3 se muestran las ventajas y desventajas de cada una de las dos posibles alternativas.

Alternativa	Ventajas y posibilidades	Desventajas
Almacenar sólo el conjunto generador	1. Fácil de implementar y no requiere de grandes necesidades de información en la base de datos 2. Posibilidad de los sistemas R.O.O para implementar conjuntos y sus operaciones 3. La estructura-AP se puede modelar como un tipo de dato definido por el usuario, como una colección de conjuntos-AP con sus operaciones	1. Posible computaciones adicionales, ante tipos de consultas que requieran expandir el conjunto generador 2. Imposibilidad de responder a preguntas flexibles, que requiera almacenar los soportes
Almacenar el conjunto generador expandido hasta las hojas	1. Permite representar más información relacionada con cada nivel del árbol, tales como los soportes, para flexibilizar las consultas 2. Posibilidad de implementar la estructura expandida, representando en cada nivel sólo la variación con respecto al nivel anterior planteado en (Berzal et al., 2001) 3. Posibilidad de uso del modelo de lista adyacente (Celko, 2005) para representar la estructura-AP	1. Más complejo de implementar y necesita mucho más espacio en disco para almacenar la base de datos 2. Dada la complejidad de la estructura y el tipo de consultas que puede responder, el tiempo de respuesta es mayor

Tabla 4.3: Alternativas de representación de conjuntos-AP y estructuras-AP.

Como se observa en la tabla 4.3, ambas alternativas son válidas con sus ventajas y desventajas correspondientes. En ambos casos, las extensiones introducidas en el lenguaje SQL:99 pueden ayudar a su implementación. Para explicar en detalle las ventajas desde el punto de vista de almacenamiento de una alternativa sobre la otra, se hará basado en la estructura-AP que aparece en la figura 3.5 del capítulo anterior. La representación de dicha estructura-AP según el modelo matemático propuesto era la siguiente:

$\mathcal{T} = g(\{fresas, nata, nueces, tarta\}, \{chocolate, nata, tarta\})$

Como se observa, la estructura-AP $\mathcal{T}$ está compuesta por dos conjuntos generadores, el conjunto $\{fresas, nata, nueces, tarta\}$ y el conjunto $\{chocolate, nata, tarta\}$. Si se realizara el almacenamiento físico de $\mathcal{T}$ por cada una de las dos variantes discutidas en la tabla 4.3, el costo de almacenamiento quedaría de la siguiente forma:

1. **Variante de almacenar sólo el conjunto generador:** Con sólo almacenar los dos conjuntos generadores ya quedaría representada la estructura-AP. Físicamente se podrían almacenar utilizando una tupla para cada conjunto maximal con un identificador que permitiera la reunión de la estructura completa, o utilizando una sola tupla con un atributo de tipo *array* si se utiliza un gestor que soporte vectores multidimensionales. Adoptando la primera de estas dos variantes físicas de almacenamiento, por ser la más general, con dos tuplas en una tabla de la base de datos quedaría almacenada la estructura-AP $\mathcal{T}$. La estructura resultante es sencilla, pero para dar respuesta a consultas que requieran de los niveles intermedios del retículo, habría que calcularlos haciendo las posibles combinaciones de los términos que componen los conjuntos generadores. Esto presupone un costo de procesamiento adicional para dar la respuesta.

2. **Variante de almacenar el conjunto generador expandido hasta las hojas:** Se hace necesario almacenar tanto los conjuntos generadores, como todas sus combinaciones posibles. En el caso del segmento de la estructura-AP que se muestra en el ejemplo de la figura 3.5, sería necesario almacenar 16 tuplas en la base de datos, una para cada uno

de los 16 nodos del retículo (grafo). De esta forma el espacio de almacenamiento sería mayor, y la complejidad de la estructura también sería mayor. Para el caso de la respuesta ante consultas que necesitaran nodos intermedios sería más rápida, pues ya se tienen almacenados directamente.

Además de estas dos variantes de almacenamiento, también se puede pensar en un híbrido entre las dos formas, haciendo uso de la idea planteada en (Berzal et al., 2001). De esta forma se almacenaría el retículo en todos sus niveles, desde el conjunto generador hasta los nodos hojas, pero sólo se almacenaría la variación con respecto al nivel anterior. Con esta idea se lograrían las ventajas de ambas posibilidades.

Para el caso de la implementación que se realiza en este trabajo, se decidió utilizar una combinación de los dos métodos discutidos en la tabla 4.3. Para la representación de la mayor cantidad de conocimiento posible, en el valor de una tupla para la tabla se representarán los conjuntos generadores correspondientes. Esta es la variante menos costosa desde el punto de vista de almacenamiento, y permite realizar fácilmente la implementación de las estructuras utilizando vectores y funciones definidas por el usuario (*dado que los TDA no están completamente soportados por el SGBD utilizado*). Además, en el diccionario se almacenarán las estructuras-AP extendidas hasta las hojas. Esto permitirá dar flexibilidad y rapidez en las consultas. En el siguiente capítulo, cuando se discuta la arquitectura del sistema propuesto se verán los detalles de esta alternativa de almacenamiento implementada.

A continuación, en la siguiente sección, se discute la implementación del modelo en el mejor exponente dentro del Software Libre de un SGBD, el PostgreSQL. Dicho gestor, es de hecho, el único de los SGBD de libre disposición que implementa el modelo de datos R.O.O. Este elemento fue decisivo en su elección como herramienta para la implementación del modelo propuesto.

4.7. Implementación del modelo en PostgreSQL

Existen diferencias en la implementación que hacen los diferentes SGBD de los procedimientos, funciones y métodos que define el SQL:99 estándar(Melton, 1998). Cada uno de dichos sistemas hace sus propias interpretaciones del estándar y en algunos casos, gestores como PostgreSQL permiten sólo la implementación de métodos asociados a la lectura y escritura del TDA para el que fue definido, y algunas funciones estadísticas sobre los datos almacenados. El resto de las funcionalidades del TDA tienen que ser implementadas como funciones definidas por el usuario, en el esquema en que está definido dicho TDA.

Debido a esta limitación importante que presenta PostgreSQL para la implementación de TDA, se decidió realizar la implementación en este SGBD, utilizando los vectores multidimensionales y las funciones definidas por el usuario que implementa. Dichas funciones como se comentó anteriormente pueden ser escritas en varios Lenguajes Procedurales siendo PL/pgSQL el lenguaje nativo de PostgreSQL.

Teniendo en cuenta todo lo discutido sobre SQL:99 y la propuesta realizada para hacer la implementación en un SGBD que soporte modelo de datos R.O.O, a continuación se muestra cómo queda implementado el modelo en PostgreSQL. Este gestor como se comentó anteriormente, soporta dicho modelo de datos y realiza su propia implementación del lenguaje SQL:99 estándar. En la figura 4.11 se puede ver cómo quedaría la definición de la tabla que contendrá las estructuras-AP.

Como se observa en la figura 4.11 se crea la tabla *estructura-AP*. En la línea (2) se crea el atributo llave *fecha* al que se especifica que no puede tomar valores nulos y en la línea (7) se crea la restricción *estructura-AP_llave_primaria* para indicar que dicho atributo es la llave de la tabla. Se debe destacar que todos los identificadores que aparecen encerrados entre doble comilla (") es porque contienen el signo menos (-) que no está permitido para los identificadores no delimitados en PostgreSQL. También resaltar que el atributo fecha se declara del tipo *TIMESTAMP*, que es un tipo fecha que incluye la hora hasta milisegundos, por eso puede funcionar como atributo llave.

1. **CREATE TABLE** public."*estructura-AP*" **(**
2. *fecha* **TIMESTAMP NOT NULL**,
3. "*conj-AP1*" **TEXT[ ]**,
4. "*conj-AP2*" **TEXT[ ]**,
5. "*conj-AP3*" **TEXT[ ]**,
6. *atributo_fuente* **VARCHAR(20)**,
7. **CONSTRAINT** "estructura-AP_llave_primaria" **PRIMARY KEY** (*fecha*)
8. **) ;**

Figura 4.11: Implementación de la tabla *estructura-AP* en PostgreSQL.

En la línea (6) se declara el *atributo_fuente* de tipo cadena, y tiene el mismo significado que el visto en las definiciones en ODL y SQL:99, el nombre del atributo textual a partir del que se obtuvo la estructura-AP.

En las líneas de la (3) a la (5) se definen los atributos que contienen los valores de la estructura-AP. Como se observa, han sido definidos como vectores de tipo *TEXT*, que es uno de los tipos cadena que implementa PostgreSQL. Dicho gestor no permite la creación de vector de vectores con longitud variable, que permitiera como en los casos anteriores, almacenar la matriz de los conjuntos-AP (un vector de vectores de tipo conjunto-AP). Debido a esta limitación, fue necesario realizar el almacenamiento de los conjuntos-AP que forman la estructura-AP, teniendo en cuenta las ideas siguientes:

1. Almacenar en una sola tupla de la base de datos la estructura-AP: para ello se añade un atributo para cada una de las longitudes de los conjuntos-AP que se obtienen. De esta forma, se evita el problema de no poder insertar varias estructuras-AP si no tienen la misma cardinalidad.

2. Antes de insertar la estructura-AP en la base de datos, se verifica que la cantidad de atributos que tiene la tabla para almacenar los conjuntos-AP, sea mayor o igual que la longitud máxima de los conjuntos-AP que

se generan. Si es menor, se agregan la cantidad de atributos de tipo vector que hagan falta.

3. PostgreSQL permite completar con *Null* las posiciones de los vectores que queden vacías.

Con estas ideas, por ejemplo, la estructura-AP representada en la figura 4.11 permitiría almacenar estructuras-AP de hasta nivel 3, o sea, que la longitud del conjunto generador sería de tres elementos. En el atributo *conj-AP1*, se almacenarían los nodos hojas del retículo que no estén incluidos en ningún otro conjunto-AP de mayor longitud, y de forma análoga para los de longitud 2.

Un ejemplo de una función implementada en PostgreSQL para responder a las operaciones del modelo, puede verse en el apéndice A.

Hasta este punto se han discutido todas las etapas de la metodología propuesta en la figura 4.1 para la implementación del modelo obtenido. A continuación, se realiza una discusión final sobre las herramientas utilizadas en dicha implementación, comparando algunas de sus características, así como sus ventajas y desventajas fundamentales.

4.8. Discusión final sobre el modelado

En esta sección, a modo de resumen, se discuten las principales cuestiones de las implementaciones realizadas a lo largo de este capítulo. Para ello, partimos de una tabla resumen similar a la tabla 4.1, pero esta vez, enfocada a las características concretas utilizadas de cada herramienta. Para ello se muestran ejemplos de algunas de las instrucciones utilizadas para la definición de los atributos, relaciones y métodos, en cada caso.

Como se observa, cada herramienta brinda las posibilidades de definir los elementos que componen un modelo de datos de una forma u otra. El caso de ODL, como se comentó anteriormente, tiene la restricción que sólo se

INTERFAZ ODL	
Definición de Atributos	**attribute** ***integer*** *ID_conjunto-AP*; **attribute** ***Set*** *<**string**> conjunto-AP*;
Definición de Relaciones	**relationship** ***Set*** *<**conjunto-AP**> estructura-AP*;
Definición de Métodos	**boolean** *acoplamiento_debil* (***in Set*** *<**string**> conjunto_Y*); **float** *indice_acoplamiento_fuerte* (***in Set*** *<**string**> conjunto_Y*);
TIPOS Y TABLAS SQL:99	
Definición de Atributos	**Create Type** *idEstructura-AP* **AS Integer Final;** **Create Row Type** *TBaseEstruc-AP* (*nombAtribTextual* **Char (20)**, *conjuntos-AP* **Varchar (15) Array** [2000]);
Definición de Relaciones	**Create Table** *Estructura-AP* **Of Type** *TEstructura-AP*; *estructura-AP* **conjunto-AP Array** [2000] // (creación de vectores de conjuntos-AP)
Definición de Métodos	**Method** *inclusion_conjunto-AP* (: *conj_Y* **Character Array** [10]) **Returns Boolean** ; **Method** *interseccion_estructura-AP* (: *otra_estructura-AP* **conjunto-AP Array** [2000]) **Returns conjunto-AP** ;
TIPOS, TABLAS Y FUNCIONES POSTGRESQL	
Definición de Atributos	*fecha* **Timestamp Not Null**, "*conj-AP1*" **Text[ ]**,
Definición de Relaciones	Se realizan en el sistema, pero ninguna fue definida en este capítulo
Definición de Métodos	**Create Function** *public.enlace_debil* (*texto_entrada* **Text**, *hacer_limpieza* **Boolean**) **Returns** *Boolean* **As** .....

Tabla 4.4: Resumen de los elementos utilizados de cada herramienta en la implementación del modelo.

permite la signatura de los métodos, pues tienen que ser implementados en un lenguaje externo como pudiera ser C o C++. De aquí que sea un lenguaje de definición de objeto. Las funcionalidades de este modelo de consultar los datos las aporta OQL. En el caso de SQL:99 permite la implementación de procedimientos, funciones y métodos. Estos últimos, están siempre asociados al TDA que los define, siendo un requisito que su implementación esté en el mismo esquema en que se ha definido el TDA. Para el caso de PostgreSQL, se permite sólo la implementación de funciones, y métodos asociados a un TDA, estos últimos con la restricción que se ha comentado anteriormente.

Los tipos fila de SQL:99, junto con el concepto de referencia, proporcionan gran parte de la funcionalidad de los objetos. Comparativamente con OQL, podemos además modificar objetos mediante los operadores asociados en SQL: inserción, modificación y borrado, mientras que en ODL dichas modificaciones han de hacerse utilizando un lenguaje de programación orientado a objetos subyacente.

También en SQL:99, los tipos tupla no contemplan en absoluto el encapsulamiento que es inherente a la orientación a objetos. En este sentido, las clases de ODL tampoco son completamente encapsuladas ya que contemplan la posibilidad de consultar a través de OQL. No obstante consultar no es peligroso para la integridad de una clase; el problema surge cuando se modifican los elementos de la misma y esto, en ODL, siempre ha de hacerse a través de los métodos que le son propios. Para evitar estas posibles disfuncionalidades en SQL:99, se utiliza el concepto de TDA que soporta el encapsulamiento. En este caso, en los TDA quedarán encapsulados los atributos y se definirán los métodos para acceder a ellos y realizar las operaciones que le sean inherentes al TDA. En este sentido PostgreSQL, a pesar que permite la definición de TDA, tampoco garantiza completamente el encapsulamiento de los datos, pues permite sólo la definición de métodos asociado con la lectura y escritura de los valores de los atributos, así como algunas funciones estadísticas sobre los datos almacenados.

En todos los casos, la definición de atributos permite varios tipos de datos primarios y los tipos colecciones. En tal sentido, ODL tiene mayor flexibilidad pues permite los tipos colecciones *Set*, *Bag*, *List* y *Array*. Para el caso de SQL:99, a pesar que en el estándar estaban pensadas estructuras simila-

res (Melton y Simon, 2002), no se han logrado incluir en su versión actual, que se basa sobre el tipo colección *Array*. Se piensa que sí estén incluidas en versiones posteriores estructuras similares a las de ODL (Melton y Simon, 2002). De forma análoga a SQL:99 ocurre con PostgreSQL, que implementa vectores con las limitaciones que se discutieron anteriormente.

Para el caso de las relaciones, en ODL se permite la definición de relaciones en ambas direcciones entre dos interfaces, que es la forma en que maneja la asociación entre clases, pues por ejemplo no deja definir el atributo de una clase, en función del tipo de otra interfaz definida previamente. En el caso de SQL:99 y PostgreSQL, permiten el modelado de las relaciones mediante las restricciones de llave extranjera del Modelo Relacional, y además soportan la herencia entre tablas. Para el caso de la herencia de tipos, está permitida en SQL:99 pero no en PostgreSQL, lo que supone una limitación importante para este SGBD.

4.9. Conclusiones

Al iniciar este capítulo se tenía como objetivo lograr el modelado del sistema en varias herramientas que permitirán llevar a cabo su implementación. Para ello, se ha propuesto una metodología de cómo transformar el modelo abstracto obtenido, en su implementación en el SGBD PostgreSQL.

Se ha realizado el modelado en los lenguajes ODL y SQL:99, donde se han definido los principales componentes de un modelo de datos (*atributos, relaciones y métodos*) para las estructuras propuestas por nuestro modelo. Dichas definiciones han permitido obtener una representación intermedia entre el modelado matemático y la implementación que se ha realizado posteriormente en PostgreSQL. Además, ha permitido realizar la propuesta formal de cómo sería la implementación de nuestro modelo en un SGBD con modelo de datos R.O.O. También se han discutido cuestiones referentes a las posibilidades y restricciones de cada herramienta utilizada. Finalmente se ha planteado la estrategia seguida para almacenar los conjuntos-AP y estructuras-AP.

Una vez que ya se ha mostrado que es factible realizar la implementación de las estructuras y operaciones del modelo propuesto, a continuación en

el próximo capítulo, se describe todo el proceso de cómo se ha obtenido la estructura-AP partiendo de un atributo textual. Se describirá en detalle la implementación completa con que se obtienen los conjuntos-AP, con ellos la estructura-AP global y las estructuras-AP particulares, que corresponde al atributo textual para cada tupla de la base de datos. Además se realizarán algunos análisis de calidad de la metodología propuesta con este fin.

Capítulo 5

Del atributo textual a la estructura-AP

En el presente capítulo, basándonos en las estructuras matemáticas y sus operaciones establecidas en el capítulo 3, se muestra cómo se implementaron y almacenaron dichas estructuras. Para ello, se utilizarán los aspectos discutidos en el capítulo anterior sobre la implementación de un TDA. De esta forma, los atributos textuales pueden ser representados de forma estructurada para facilitar su manejo. Con esta idea, se ha desarrollado la arquitectura del sistema que se propone. También se realizan los procesos de Minería de Textos para la transformación del atributo textual hasta obtener su forma intermedia de representación.

En la sección 1 del presente capítulo se describe el sistema propuesto, se discute su arquitectura y cómo se realiza el proceso para obtener la estructura-AP partiendo de un atributo textual. En la sección 2 se describe de forma detallada todo el proceso de limpieza de datos realizado como una fase de la Minería de Textos. En la sección 3 se describe cómo se implementó el proceso de obtención de la estructura-AP global de conocimiento, utilizando una base de datos médica. Finalmente, en la sección 4 se muestra cómo se puede obtener la estructura-AP inducida de cada tupla de la base de datos, para dicho atributo textual.

5.1. Descripción del sistema

El sistema desarrollado trata con textos cortos en ciertos dominios. Estos pueden estar no sólo en repositorios de textos si no también en bases de datos (Relacionales, Relacional Orientado a Objetos, etc.). En este último caso, éstos pueden ser atributos textuales compuestos por algunas palabras o varias frases. Las operaciones sobre estos atributos son las clásicas en las bases de datos para atributos textuales, tales como, preguntar sobre el contenido del atributo, o buscar si el atributo contiene determinadas palabras. Sin embargo, la falta de un dominio definido y una estructura en estos atributos les impide que sean tratados conjuntamente como el resto de los atributos de la base de datos. La representación y operadores tradicionales que provee el modelo de bases de datos relacionales, hacen que procesos como Datawarehousing, Minería de Datos o, simplemente una consulta semántica, sean prácticamente imposibles de resolver, o al menos con la calidad que se requiere, sobre este tipo de atributos textuales.

Para resolver esta problemática, se parte de la hipótesis de que existe una semántica subyacente en determinados atributos textuales de la base de datos (Martín-Bautista et al., 2008). A pesar de que el dominio de éstos no es estructurado y es muy complicado de manejar, puesto que está compuesto por elementos del lenguaje natural, la semántica e incluso el vocabulario en los atributos, es en alguna medida restringido. De esta forma, se pueden encontrar conjuntos de términos relacionados semánticamente que aparecen repetidamente en el atributo analizado. Para este propósito, como se explicó anteriormente, se usa una forma intermedia basada en itemsets frecuentes obtenidos vía el algoritmo Apriori con la propiedad Apriori. Dicha forma intermedia es llamada conjunto-AP. Como ya se ha comentado también, con todos los conjuntos-AP obtenidos en la base de datos, es formada una estructura global intermedia nombrada estructura-AP. Esta estructura refleja la semántica del atributo textual en la base de datos.

5.1.1. Arquitectura del sistema

El sistema está compuesto básicamente por una base de datos inicial, un modulo de manejo de textos cortos, la base de datos modificada y un cliente de consulta. Si bien el principal objetivo de esta arquitectura es el manejo de atributos textuales conjuntamente como el resto de los atributos de la base de datos, se pueden implementar también herramientas analíticas para la toma de decisiones que incluyan Data Warehouse y OLAP y que se aprovechen de las ventajas del sistema, como se muestra en la figura 5.1.

Como ya se mencionó, la solución propuesta parte de una base de datos inicial en PostgreSQL; en ella se almacena la información sensible a ser consultada por el cliente final. Para soportar la consulta sobre atributos textuales conjuntamente como el resto de los atributos de la base de datos, dicha base de datos inicial es sometida a un proceso de transformación. A los atributos textuales que sean de interés en la toma de decisiones, se les aplica un proceso de transformación para lograr su representación estructurada que pueda facilitar su posterior consulta. Este proceso será realizado por un módulo de preprocesamiento y obtención de la forma intermedia de representación. Dicho módulo toma como entrada todos los atributos textuales seleccionados, y obtendrá una base de datos inicial modificada soportada por el modelo objeto-relacional de bases de datos y además una base de datos con conocimiento subyacente, como se muestra en la figura 5.1.

Dado que la forma de representación intermedia escogida para la representación de los textos cortos, está basada en el concepto de itemset frecuente utilizando el algoritmo Apriori, este proceso se encargará de obtener los conjuntos-AP generadores y las estructuras-AP resultantes del proceso. Como resultado, se obtiene además un diccionario que está compuesto por todos los términos relevantes correspondientes a cada atributo procesado, con su correspondiente frecuencia de aparición y la lista de itemsets frecuentes con su soporte. También se almacenan los conjuntos-AP detallados con su soporte. Ambas estructuras pueden ser utilizadas en el proceso de actualización de la base de datos inicial modificada, teniendo en cuenta la frecuencia de aparición de los términos con respecto a un umbral determinado y/o para realizar consultas sobre la estructura general. Dichas consultas podrían buscar po-

sibles asociaciones entre términos, para sugerir otras posibles respuestas de interés al cliente, como se comentó anteriormente. Tanto la base de datos inicial modificada como el diccionario y la base de datos que contiene los conjuntos-AP detallados, serán almacenados en PostgreSQL.

Por otra parte, la arquitectura cuenta con una herramienta de consulta que permitirá al usuario construirse información muy variada, y de una forma flexible será asistido por la misma. Esta herramienta será el cliente de consulta; aquí el usuario podrá indagar sobre información almacenada en la base de datos. Podrá hacer marcado hincapié en la información almacenada en el diccionario y los conjuntos-AP detallados; también podrá realizar preguntas para cualquiera de los atributos de textos previamente procesados, y además recibirá sugerencias sobre los términos consultados. Además, la arquitectura tiene en cuenta la posibilidad de dar soporte para que herramientas que implementan Data Warehouse y OLAP puedan explotar la forma intermedia de representación obtenida para textos cortos. De esta forma el usuario podrá construir cubos multidimensionales, donde pueden formar parte de éste dimensiones que respondan a atributos textuales. Por consiguiente, se podrán realizar las operaciones clásicas del modelo multidimensional (Roll up, Drill Down, Slice y Dice) que incluyan atributos de este tipo. Dichas posibilidades están hoy en discusión teórica, y se pueden encontrar algunas implementaciones incipientes por los principales proveedores de soluciones de este tipo.

De forma resumida, los principales componentes de esta arquitectura serán:

- **Base de datos inicial:** La base de datos almacena diferentes tipos de datos, incluyendo atributos textuales. Aquellos atributos textuales con relevancia en el proceso de toma de decisiones pueden ser transformados, a fin de realizar consultas simples incluyendo estos atributos. A pesar de que la solución presentada está soportada sobre una base de datos Relacional Orientada a Objeto, cualquier otro tipo de base de datos puede ser considerada.

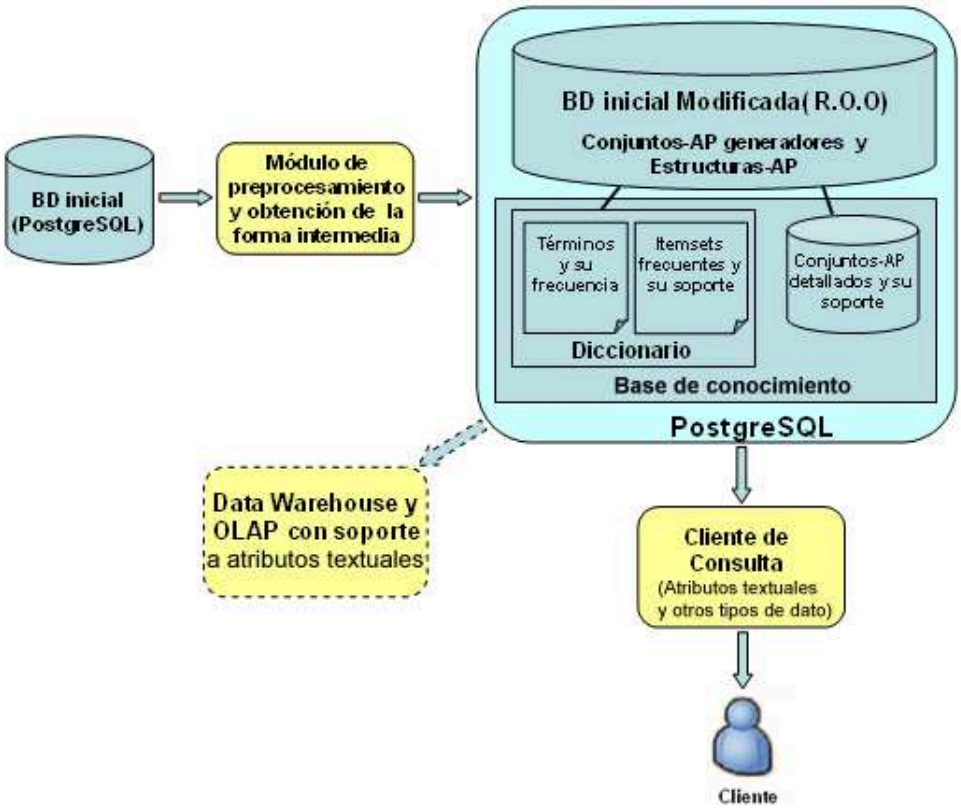


Figura 5.1: Arquitectura del Sistema.

- **Módulo de preprocesamiento y obtención de la forma intermedia:** En este módulo, los atributos textuales de la base de datos son procesados a fin de obtener la forma intermedia. La nueva estructura de conocimiento permitirá consultar todos los atributos de la base de datos y llevar a cabo cualquier proceso de sumarización, Datawarehousing, etc.

- **Base de datos modificada:** Esta base de datos almacena las nuevas estructuras de conocimiento correspondientes a los atributos textuales de la base de datos inicial. Estas estructuras están basadas en los conjuntos de términos obtenidos por el algoritmo Apriori (conjuntos-AP y estructuras-AP). Diferentes datos intermedios son obtenidos en

este. El diccionario está compuesto por todos los términos relevantes de cada atributo textual (una vez eliminadas las palabras de parada, preprocesados los acrónimos, etc.), y por los itemsets frecuentes y su soporte. El diccionario y los conjuntos-AP expandidos con su soporte son almacenados en la base de conocimiento. Esta base de datos puede ser usada para actualizar la base de datos modificada, así como para completar las respuestas del sistema al Cliente de Consulta, sugerir un nuevo término por el que se puede consultar, etc.

- **Cliente de Consulta:** Mediante este módulo, el cliente realiza consultas sobre la base de datos inicial modificada y sobre la base de conocimiento. Puede realizar operaciones específicas para los atributos textuales, tales como los tres tipos de consultas discutidas en el capítulo anterior.

- **Data Warehouse y OLAP con soporte a textos cortos:** El usuario puede construir cubos multidimensionales donde una o más dimensiones pueden estar relacionadas a los atributos textuales de la base de datos. La nueva representación de estos atributos, permite realizar sobre ellos las operaciones usuales del modelo multidimensional (Roll up, Drill Down, Slice y Dice).

Una vez discutida la arquitectura del sistema propuesto para la obtención de la forma intermedia de representación y las consultas de atributos textuales, a continuación se detalla la forma en que se obtiene la estructura-AP a partir de dichos atributos.

5.1.2. Desde un atributo textual hasta la estructura-AP: el mecanismo para obtener el TDA

La base matemática de las estructuras que sustentan la forma intermedia de representación obtenida, así como sus operaciones, fueron definidas en el capítulo 3. En el capítulo anterior se discutió que es factible su implementación como un TDA en un SGBD con modelo R.O.O. En este punto se discutirá

entonces, de manera práctica, cómo se obtuvo dicho TDA utilizando el *módulo de preprocesamiento y obtención de la forma intermedia* de la arquitectura del sistema propuesto.

Con este fin, se comienzan por definir los procesos que componen dicho módulo. Para ello en la figura 5.2 se detallan los procesos que lo forman con sus entradas y salidas. Como se observa, se parte desde un texto corto hasta obtener el TDA, que no será más que la estructura-AP que le corresponde a cada tupla de la base de datos. Para la implementación práctica de dichos procesos, se ha creado la herramienta *Text Mining Tool V1.0 (en el apéndice B se describe su interfaz en detalle),* que será la encargada de realizar todas las operaciones que se describen en la figura 5.2.

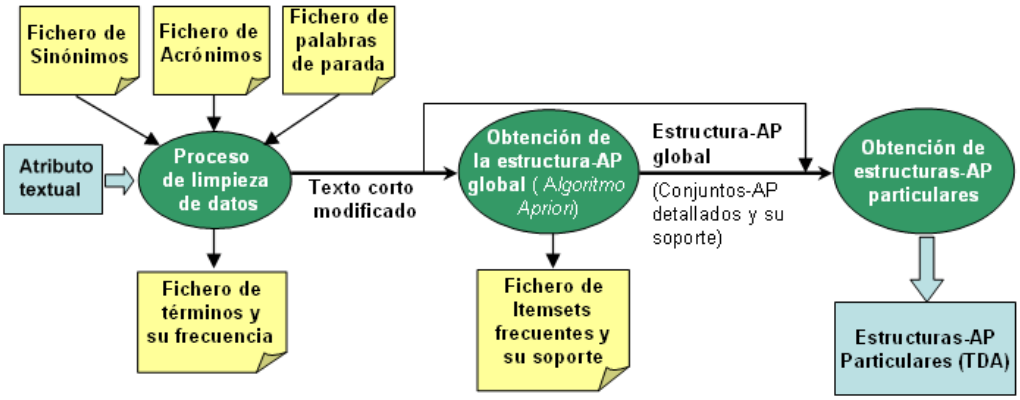


Figura 5.2: Módulo de Preprocesamiento y obtención de Forma Intermedia.

Como se observa en la figura 5.2, el módulo de preprocesamiento y obtención de forma intermedia está compuesto por tres procesos fundamentales:

1. **Proceso de limpieza de datos:** Este proceso se encarga de realizar el preprocesamiento inherente a cualquier proceso de minería de datos, para realizar la limpieza y homogenización de la muestra inicial. En el caso concreto de la implementación realizada, el proceso toma como entrada frases de texto corto provenientes de un atributo textual

en una base de datos, como se vio anteriormente. Además, utiliza los ficheros de sinónimos y acrónimos para realizar la sustitución de estos en las frases iniciales. También utiliza el fichero de palabras de paradas para eliminarlas de la frase original. Este proceso produce a la salida el fichero de todos los términos diferentes que componen el vocabulario del atributo textual procesado, con su frecuencia de aparición en el total de tuplas procesadas. Además, da como principal salida, un nuevo atributo con el texto corto original modificado después de haber realizado la sustitución de sinónimos y acrónimos, y eliminado las palabras de paradas de las frases originales.

2. **Obtención de la estructura-AP global (*Algoritmo Apriori*):** En este proceso, se toma como entrada el texto corto original modificado que obtiene el proceso anterior. Sobre dichos textos se ejecuta el algoritmo Apriori para obtener el fichero de itemsets frecuentes con su soporte, y los conjuntos-AP detallados con su soporte. Con los conjuntos-AP maximales se forma la estructura-AP global que encierra el conocimiento del atributo textual procesado.

3. **Obtención de estructuras-AP inducidas:** Este proceso es el que obtiene las estructuras-AP inducidas para cada tupla de la base de datos, y escribe su representación como un TDA en un nuevo atributo en la tabla desde la que provienen los datos originales. Para obtener el TDA, se realiza la intersección entre la estructura-AP y el texto corto modificado que se obtiene a la salida del proceso de limpieza de datos.

En las siguientes secciones se discutirá en detalle cada uno de estos tres procesos. Antes, de forma general, se describe el mecanismo seguido para obtener el TDA a partir de un atributo textual. Para ello se plantean, por orden, los pasos seguidos con este fin.

Sea R una relación con atributos $\{A_1, \ldots, A_T, \ldots, A_n\}$, donde A_T representa un atributo textual y sin una estructura predecible. Por la heterogeneidad y estructura en sí que puede tener A_T y el conjunto de valores que puede tomar, se hace impredecible obtener una forma de representación intermedia que posibilite manejar su semántica. Dicha forma permitirá realizar consultas

más potentes que las tradicionales verificaciones de si el atributo es igual a un determinado valor, o si sigue algún patrón especificado en una máscara. Para obtener el TDA que encapsule la semántica del mencionado atributo, y posibilite realizar éstas consultas más potentes, se realizan las siguientes tareas utilizando el software *Text Mining Tool V1.0*:

1. Realizar el proceso de limpieza sobre A_T, teniendo en cuenta la sustitución de sinónimos y acrónimos, además de tener en cuenta la posible separación de frases diferentes en una misma tupla con uno o varios caracteres separadores (Ejemplo: signo +, punto y coma, coma).

2. Eliminar las palabras vacías que pose el atributo A_T; aquí se crea el diccionario con los términos relevantes y la frecuencia con que aparecen en dicho atributo. Además da a la salida el texto corto original modificado, después de la limpieza de datos y eliminación de palabras de parada. Esta salida se crea como un nuevo atributo en la tabla cuando se trabaja con bases de datos.

3. Obtener la base de datos transaccional donde los atributos son los diferentes términos del diccionario, y cada tupla corresponde a un registro. El valor de un determinado atributo en una tupla será 1 si el término del atributo aparece en el registro original, en otro caso será 0. A partir de dicha base de datos transaccional, calcular los itemsets frecuentes siguiendo el algoritmo Apriori.

4. A partir de los itemsets frecuentes se calculan los itemsets maximales que son considerados los conjuntos-AP generadores. Con dichos conjuntos-AP se forma la estructura-AP, que será el dominio considerado para el TDA obtenido para cada tupla de la base de datos inicial.

5. Para obtener el valor del TDA para cada tupla en el atributo A_T, se utilizará la salida del proceso de limpieza de datos, que contiene para cada fila, la representación de A_T después de este proceso. Para ello se calcula la intersección entre cada valor de las filas de dicho fichero y la estructura-AP. El resultado final será el que se escriba en otra columna A_{TN} como valor de TDA para A_T.

6. Una vez que se tiene el atributo A_{TN} que contiene el valor del TDA para el texto corto, se está en condiciones de realizar operaciones de consultas más complejas que incluso encierren resultados semánticos. La implementación de estas operaciones de consulta serán las que se discutan en el próximo capítulo.

Como se comentó anteriormente, en las siguientes secciones, se discutirán los tres procesos que forman el módulo de *preprocesamiento y obtención de forma intermedia.* Para hacerlo, se utilizará un ejemplo experimental para clarificar las ideas que se van introduciendo. Este ejemplo experimental es descrito brevemente en el apartado siguiente.

5.1.3. Conjuntos de datos experimental: base de datos médica

En el entorno hospitalario vienen repitiéndose los problemas que fueron abordados al inicio de este trabajo, sobre la acumulación de grande volúmenes de información sin que repercutan en la consecución de información útil y oportuna para la toma de decisiones. El entorno hospitalario en la actualidad, en la mayoría de los casos, se cuenta con diferentes y variadas herramientas que automatizan el quehacer diario del hospital, como pudieran ser sistemas para la gestión de urgencias médicas e intervenciones quirúrgicas o de gestión de recursos humanos. Para el caso de sistemas que extraigan conocimientos de los datos almacenados, que realmente ayuden a mejorar la gestión general de hospital, y por ende, la atención de los pacientes que a él acuden, la situación es claramente deficitaria (Prados y Peña, 2002).

Como plantea el mismo autor Prados en (Prados y Peña, 2004) "el decisor solicita conocimientos, no simples resultados, basados en agrupaciones lógicas previamente establecidas". Como demuestra esta cita, el problema consiste entonces en lograr estructurar el conocimiento en agrupaciones lógicas que permitan analizar la información bajo una determinada óptica con un determinado interés.

Para el caso particular de la información en formato texto, un hospital es un ejemplo clásico de lo que se ha venido discutiendo a lo largo de este trabajo, donde la acumulación de información en este formato crece por día. Las historias clínicas de los pacientes, los reportes médicos sobre el estado de un paciente, la valoración de urgencias médicas, entre otros, son algunos ejemplos. Es por eso que contar con una herramienta que de forma eficaz pueda extraer conocimiento de estas informaciones se hace muy importante.

Como ya se comentó anteriormente, normalmente el hospital cuenta con poderosos sistemas operacionales que automatizan el quehacer diario del hospital, pero sistemas inteligentes para la extracción de información son más escasos o inexistentes. Inclusive no todos los sistemas que pueden extraer conocimientos de las bases de datos hospitalarias, dan soporte de forma eficiente a la posibilidad de realizar consultas semántica sobre dichos tipos de atributos textuales. Esta situación debido a la causa que hemos venido mencionando, de la falta de una estructura en este tipo de datos.

En este entorno, la solución aquí presentada constituye una alternativa nada despreciable a estas problemáticas, pues además de dar soporte a consultas semánticas en atributos textuales, ha sido implementada utilizando software libre, algo también importante, en un entorno tan sensible a los costos.

A continuación se describen los datos que han sido extraídos de este entorno, con la idea de a lo largo de este trabajo, realizar experimentos sobre un caso real.

Los conjuntos de datos están compuestos por atributos textuales de la base de datos médica del Hospital Clínico "San Cecilio" de Granada, España. Se ha trabajado con la información referente a las Intervenciones Quirúrgicas y Urgencias Médicas. Para el caso de las Intervenciones Quirúrgicas se ha utilizado la tabla TIntervenciones con 24481 registros y para el caso de las Urgencias Médicas la tabla TUrgencias con 18940 registros. El contenido de estos atributos es corto (desde uno a cincuenta términos), y ellos pueden incluir una o más frases. Los conjuntos de muestra para la experimentación se definieron de la forma siguiente:

1. **Conjunto1:** Atributo *Diagnostico* de la tabla *TIntervenciones* con 24481 registros.

2. **Conjunto2:** Atributo *IPropuesta* de la tabla *TIntervenciones* con 24481 registros.

3. **Conjunto3:** Atributo *Motivo* de la tabla *TUrgencias* con *18940* registros.

4. **Conjunto4:** Atributo *Juicio* de la tabla *TUrgencias* con *18940* registros.

Una vez definidos los datos para la experimentación, en la siguiente sección se realiza la descripción detallada de todas las tareas que se realizan en el proceso de limpieza de datos.

5.2. Descripción detallada del proceso de limpieza de datos

En prácticamente todos los contextos en los que se quieran aplicar técnicas de minería de datos y textos para la recuperación de información, el proceso de limpieza de los datos juega un papel fundamental en la calidad de los resultados obtenidos. En él se eliminan posibles desviaciones, valores irrelevantes, etc. En nuestro caso particular, que se trabaja con información procedente del ámbito hospitalario, donde la información es introducida al sistema por diferentes personas, utilizando muy diversos patrones de escritura, este proceso se hace de vital importancia.

Después del análisis y manipulación experimental sobre los datos, se llegó a la conclusión de que existen términos escritos de diferentes formas y que tienen un mismo significado, o sea, son sinónimos, lo mismo ocurre con diversos símbolos que son acrónimos y no siempre se escriben de igual forma. Por otra parte, también se detectaron relaciones semánticas a partir de la sintaxis en que es escrita la información en los distintos atributos textuales. Se identificó

que el valor de un atributo de este tipo, en una misma tupla, puede tener más de una frase delimitada por algún carácter separador.

La primera columna de la tabla 5.2 representa un ejemplo de estos textos originales, para el conjunto experimental *Conjunto2*. Como se puede observar, inicialmente están en lenguaje natural, incluyendo signos de puntuación, abreviaturas y hasta símbolos como " + ". Por consiguiente, la primera tarea de limpieza de datos se hace importante para este caso médico.

Dato de tipo Texto (Texto corto)	**Dato limpio (Texto corto modificado)**
CURA DE ABCESO PERIANAL	CURA ABSCESO PERIANAL
HISTERECTOMIA ABDOMINAL	HISTERECTOMIA ABDOMINAL
EMBOLECTOMIA BYPASS AXILO-FEMORAL IZDO	EMBOLECTOMIA BYPASS AXILO-FEMORAL IZQUIERDO
EECC + L.I.O. O.I.(20.O)	EECC + LIO OI(20.O)
ARTROPLASTIA DE RODILLA DCHA	ARTROPLASTIA RODILLA DERECHA

Tabla 5.2: Ejemplos de textos originales y limpios del *Conjunto2*.

A continuación se describe en detalle, cómo se resolvieron los problemas antes descritos, creando ficheros auxiliares para los sinónimos y acrónimos que permitan sustituir términos diferentes por un único término con el mismo significado, elevando de esta forma el soporte con que es obtenido dicho término. Además se describe el algoritmo general de limpieza implementado, y se exponen ejemplos de los resultados obtenidos después de todo este proceso.

5.2.1. Fichero de Sinónimos

Como se comentó anteriormente, se ha obtenido un fichero de sinónimos a partir del estudio de los conjuntos-AP generados, más la realización de consultas tradicionales sobre los datos y el criterio de expertos. El objetivo con este fichero es hacer la sustitución de diferentes términos que tienen un mismo significado por uno común a todos, y de esta forma conseguir aumentar el soporte de los itemsets obtenidos que incluyan a dicho término. La estructura de este fichero y algunos ejemplos se muestran en la tabla 5.3.

ANTEROSEPATAL ANTEROSEPAL ANTEROSEPATAL ANTEROCEPTAL ANTEROSEPT ANTEROSEPTA ANTEROSEPTOAL ANTEROSPTAL **:ANTEROSEPTAL**
DCHA DCHO DCH DECHO DERCH DERCHA DERCHO DERECH DER DRCH DRCHA DERECHA **:DERECHO**
IZQ IZDA IZDO IZQD IZQDO IZQDA IZ IZQIERDO IZQU IZQUI IZQUERDO HIZQ IZQ1UIERDA IZQUIE IZQUIEDO IZQUIER IZQUIERA IZQUIERD IZQUIERDOP IZQUIERTDO IZQUIRDO IZQUUIERDO IZQUIERDA **:IZQUIERDO**

Tabla 5.3: Estructura y ejemplos del fichero de sinónimos utilizado.

Como se observa, cada agrupación tiene la estructura *lista de sinónimos :término común*. El algoritmo implementado, sustituye para cada tupla del atributo al que se le está haciendo el proceso de limpieza, cada término que aparece en *lista de sinónimos* por el *término común*. De esta forma, se consigue aumentar el soporte de los itemsets que contienen al *término común* y desaparecen los itemsets que contenían los términos que están en la *lista de sinónimos*. Más adelante se discutirá sobre la eficiencia del algoritmo implementado, y se mostrarán ejemplos de las agrupaciones que se realizan, con las correspondientes mejoras del soporte.

Con este tratamiento de sinonimia, además de conseguir las mejoras antes mencionadas en el soporte, cabe destacar que se logra homogeneizar el vocabulario del atributo textual procesado. Con esto se logra reducir su complejidad y variabilidad, y por ende se facilita el trabajo de obtención de las estructuras del modelo. Además, desde el punto de vista semántico, será posible realizar consultas buscando por un término determinado, y se pueden retornar tuplas que contengan términos que sean sinónimos del que se está buscando.

5.2.2. Fichero de Acrónimos

El fichero de acrónimos que se utiliza durante el proceso de limpieza, cumple la función principal de obtener los acrónimos sin los puntos intermedios que posee cada uno de ellos. Esto se realiza, dado que hemos considerado el punto como posible separador entre dos frases en la misma tupla. Debido a esto, se tratar de descartar puntos que no significan separadores entre dos frases en una misma tupla, para posteriormente poder utilizar el signo de punto como posible separador. Por ejemplo, se sustituye el acrónimo *O.I* que significa *OJO IZQUIERDO* por *OI*, o *A.A.A* que significa *ANEURISMA AORTA ABDOMINAL* por *AAA*. La estructura de este fichero y algunos ejemplos se muestra en la tabla 5.4.

AVCA ACCIDENTE VASCULAR CEREBRAL AGUDO
BAVC BLOQUEO AURICULO VENTRICULAR COMPLETO
OD OJO DERECHO
OI OJO IZQUIERDO
OMS OTITIS MEDIA SUPURADA
OP OSTEOSINTESIS PLACA
PA PERITONITIS AGUDA

Tabla 5.4: Estructura y ejemplos del fichero de acrónimos utilizado.

Como se observa, cada línea del fichero comienza con el acrónimo sin sus puntos intermedios y a continuación el significado del mismo. Dentro de las

líneas del fichero estos acrónimos aparecen ordenados alfabéticamente para optimizar el algoritmo de sustitución implementado. De forma análoga ocurre para el caso anterior del fichero de sinónimos.

Una vez que se ha discutido la estructura y función de los ficheros de sinónimos y acrónimos utilizados en el proceso de limpieza de datos, a continuación se darán los detalles del algoritmo de limpieza implementado.

5.2.3. Algoritmo de limpieza implementado

El algoritmo de limpieza implementado se encarga de recorrer cada valor del atributo textual sobre el que se realiza el proceso de limpieza, sustituyendo los acrónimos y sinónimos, y eliminando las palabras de parada utilizando los ficheros antes descritos. El resultado es escrito en el atributo que se seleccione como parámetro para recibir el resultado del proceso de limpieza. De forma general, la descripción del algoritmo es la que aparece en la tabla 5.5.

Como se puede observar, el algoritmo toma como entrada las tuplas T del atributo textual sobre el que se va a realizar el proceso de limpieza. Se obtiene como resultado las tuplas T_M, que se escriben en el atributo de la tabla que se le pasa al algoritmo como parámetro, indicando dónde se escribirá el resultado. Básicamente, el algoritmo lo que hace es recorrer cada tupla del atributo textual, y para cada una, recorrer todas las palabras que contiene y buscarlas en el fichero de acrónimos y de sinónimos, si aparece, realiza la sustitución por el término equivalente.

Durante la implementación del algoritmo anterior, se tuvieron en cuenta diferentes aspectos que permiten mejor su eficiencia. Especialmente se prestó atención a la búsqueda de cada palabra de una tupla en los ficheros de sinónimos y acrónimos para realizar la sustitución de cada término por su equivalente. Ambos ficheros se organizaron alfabéticamente, y cuando se realiza la búsqueda del término W_j, se van recorriendo las filas de dichos ficheros, mientras el ordinal del primer carácter de W_j sea menor que el ordinal del primer término de cada fila de ambos. Cuando se deja de cumplir esta condición, significa que el término no aparece en el fichero en cuestión, y se detiene

Entrada: T tuplas del atributo textual.
Salida: T_M tuplas modificadas del atributo textual.

1. **PARA CADA** T_i ($i = 1..n$) **HACER**
2. **HACER** $T_M \leftarrow \emptyset$
3. **PARA CADA** palabra W_j de T_i ($j = 1..m$) **HACER**
4. **SI** W_j (sin puntos) en fichero de acrónimos **ENTONCES**
   Sustituir W_j por su término equivalente
5. **SI** W_j en fichero de sinónimos **ENTONCES**
   Sustituir W_j por su término equivalente
6. **HACER** $T_M \leftarrow T_M + W_j$
7. **HACER** $j \leftarrow j + 1$
8. **FIN PARA**
9. **ESCRIBIR** T_M en atributo de salida
10. **HACER** $i \leftarrow i + 1$
11. **FIN PARA**

Tabla 5.5: Algoritmo de limpieza de datos implementado.

la búsqueda sin necesariamente leer siempre todas las filas, lo que hace que el algoritmo gane bastante en eficiencia.

En el próximo apartado se muestra un ejemplo concreto de la aplicación del algoritmo de limpieza antes descrito.

5.2.3.1. Ejemplo de aplicación del algoritmo de limpieza

En la tabla 5.6 se ilustra con varios ejemplos, los resultados obtenidos con la limpieza de datos sobre el conjunto experimental *Conjunto2*. En la columna *IPropuesta Modificada* se muestra el resultado que se obtiene de aplicar la

limpieza de datos sobre el conjunto experimental antes mencionado. Como se observa, se realiza la sustitución de acrónimos, sinónimos y puntos.

IPROPUESTA	IPROPUESTA MODIFICADA
F.A.V. 1* VEZ	FAV. 1* VEZ
EECC+ LIO O.D.	EECC+ LIO OD
VARICES PIERNA IZQ1UIERDA	VARICES PIERNA IZQUIERDO
TEA carotidea izqda	TEA CAROTIDEA IZQUIERDO
EECC + L.I.O. O.I. 16.DT.	EECC + LIO OI 16DT
EXTIRAPCION	EXTIRPACIÓN
UNILAT.	UNILATERAL.
EECC + L.I.O. O.D. (22.50)	EECC + LIO OD (22.50)

Tabla 5.6: Ejemplo del resultado del proceso de limpieza aplicado sobre el *Conjunto2*.

Como se observa en la tabla 5.6, se han sustituido términos por su sinónimo, como por ejemplo los términos *izqda* por *izquierdo* y *unilat* por *unilateral*. También se han sustituido acrónimos por sus términos equivalentes sin los puntos intermedios, como por ejemplo $F.A.V$ por FAV y $O.D$ por OD. De igual forma en las filas 2, 5 y 8 se observa cómo se ha mantenido el signo + para utilizarlo como posible separador de dos frases en la misma línea. De esta forma se contarán como dos frases separadas a los efectos de computar lo itemsets frecuentes.

El proceso de limpieza de datos realizado trae aparejado la mejora de los resultados obtenidos en el cálculo de los itemsets frecuentes, pues al homogeneizar el vocabulario se obtienen con mayor soporte dichos itemsets. También se incluyen otros que no se consideraban, y desaparecen algunos itemset que significaban lo mismo que otros que ya habían sido generados. Para tener una idea clara de las mejoras que se obtienen, en el siguiente apartado se realiza un análisis de estas cuestiones.

5.2.4. Análisis de los resultados y mejoras obtenidas con la limpieza de los datos

A continuación, se muestran algunos resultados y mejoras que se obtienen, una vez que se realiza el proceso de limpieza de los datos debido a que, como se explicó anteriormente, al realizar las sustituciones de acrónimos, sinónimos y puntos, se logra obtener un vocabulario más homogéneo. Esto implica que posteriormente se obtenga un número menor de conjuntos-AP, y con un mayor soporte.

Como se comentó anteriormente el resultado fundamental que trae aparejado el proceso de limpieza de datos, es que permite obtener un menor número de conjuntos-AP y con un mayor soporte. En la tabla 5.7 se muestran la cantidad de conjuntos-AP obtenidos sobre el conjunto experimental *Conjunto2*, antes y después del proceso de limpieza y se realiza el comentario de los resultados obtenidos para los diferentes casos que se observan.

En la tabla 5.8 se muestran los resultados similares a los mostrados en la tabla 5.7, pero ahora para los cuatro conjuntos experimentales definidos anteriormente. Para los cuatro casos, se adiciona una columna con el nombre del campo y para al mismo atributo después del proceso de limpieza (*modificado*). Los resultados obtenidos son análogos a los comentados en la tabla 5.7.

Como se observa en la tabla 5.8, la tendencia en los 4 conjuntos experimentales es el mismo. Para los conjuntos-AP de longitud 1, en todos los casos desaparecen conjuntos-AP después de aplicar el proceso de limpieza. Esto ocurre por dos motivos fundamentales: primero, que desaparecen conjuntos-AP que eran tenidos en cuenta y ahora se unen al grupo del sinónimo por el que fueron sustituido; y segundo porque al incrementarse el soporte con que son obtenido los términos algunos de ellos aparecen ya incluidos en conjuntos-AP de cardinalidad superior.

Longitud Conjunto-AP	IPropuesta	IPropuesta Modificada	Comentario de la limpieza de datos
1	287	278	En la columna modificada se obtienen 9 nodos hojas(conjuntos-AP de longitud 1) menos, pues el programa sustituye términos que son sinónimos que antes se obtenían como término frecuente independientes (13) y aparecen nuevos términos que al agruparse con otros sube su soporte y se incluyen(4)
2	288	293	En la columna modificada se obtienen más conjuntos-AP, pues al obtenerse con mayor soporte, agrupaciones que en la columna original se eliminaban por no cumplir con el soporte mínimo, ahora se incluyen
3	167	171	Ídem a los conjuntos-AP de longitud 2
4	70	72	Ídem a los conjuntos-AP de longitud 2
5	21	21	
6	3	3	
7	2	2	

Tabla 5.7: Conjuntos-AP obtenidos antes y después del proceso de limpieza de datos sobre el *Conjunto2*.

También se observa en la tabla 5.8 que ocurre de forma análoga en todos los casos con los conjuntos-AP de longitud dos y tres. La tendencia es la misma en los 4 atributos procesados, pero en este caso ocurre lo contrario. Como se ve, en todos los casos se obtienen más conjuntos-AP de estas longitudes después de la limpieza de datos. Este hecho da la medida de cómo el proceso de limpieza de datos mejora la calidad de los conjuntos-AP que se obtienen, pues mientras mayor sea la cardinalidad de los conjuntos-AP más semántica recogen de la frase inicial. Esto traerá aparejado por ende, mayor calidad de información en la respuesta antes consultas sobre los atributos textuales.

Obsérvese además en dicha tabla, que para el caso del atributo *Juicio* en la tabla de *Urgencias* se obtiene un conjunto-AP de cardinalidad cuatro, que antes de la limpieza de datos no existía.

	TINTERVENCIONES				URGENCIAS			
Longitud Conjunto-AP	IPropuesta	IPropuesta Modif	Diagnóstico	Diagnóstico Modif.	Juicio	Juicio Mo-dif.	Motivo	Motivo Mo-dif.
1	287	278	360	336	248	241	211	204
2	288	293	256	296	107	121	133	140
3	167	171	75	89	7	11	9	10
4	70	72	11	13		1		
5	21	21	1	1				
6	3	3						
7	2	2						

Tabla 5.8: Conjuntos-AP obtenidos antes y después del proceso de limpieza para todos los conjuntos experimentales.

En la tabla 5.9 se muestra un listado de conjuntos-AP con su soporte que desaparecen después de la limpieza de los datos. Como se comentó anteriormente desaparecen porque son sustituidos por sus correspondientes sinónimos o acrónimos.

Finalmente se muestra en la tabla 5.10 se muestra un resumen donde se ilustran algunas de las agrupaciones fundamentales que surgen en los conjuntos-AP antes del proceso de limpieza, y el término porque el que son sustituidos con su correspondiente aumento de soporte. Como se observa, esto justifica la disminución que siempre se obtiene en los conjuntos-AP de longitud uno pues siempre varios términos son sustituidos por su sinónimo. También se agrupan acrónimos que se escribían con y sin sus puntos intermedios.

Se debe destacar, que el proceso de limpieza implementado tiene en cuenta el caso particular para cuando una tupla posee más de una frase asociada al mismo atributo textual. Para ello, el proceso permite definir como parámetro

Conjunto-AP	Soporte
Abceso	0.118237
Dcha	0.13454558
Dcho	0.14473845
Derecha	0.99074364
Izda	0.28336138
Izdo	0.19366418
Izq	0.15289277
Izquierda	0.69311315
l.i.o	3.6653504
o.d	1.0478255
o.i	1.1742163
r.t.u	0.24055102
t.e.c	0.31393975

Tabla 5.9: Ejemplos de conjuntos-AP que desaparecen después del proceso de limpieza de datos en el *Conjunto2*.

de entrada el conjunto de caracteres que pueden funcionar como separadores de frases en una misma tupla. Teniendo en cuenta estos términos, en el proceso de limpieza no son eliminados y para el caso particular del punto, se realiza un tratamiento especial para cuando está al final de un acrónimo. Si tiene la posibilidad de separar dos frases, no se elimina. De esta forma, este proceso sienta las bases para que en la obtención de los itemsets, las tuplas que contengan estos separadores, sean tratadas como más de una tupla realmente, según la cantidad de frases diferentes que puedan tener.

El algoritmo de limpieza implementado, se aplicó sobre el conjunto de datos experimental antes descrito, y se obtuvieron las estadísticas que se muestran en la siguiente sección.

5.2.5. Estadísticas obtenidas del proceso de limpieza de datos

Con el objetivo de medir algunos parámetros y conocer en que cuantía se realizaban los cambios introducidos por el programa de limpieza, se le adicionó

Antes de la limpieza de datos	Después de la limpieza de datos
abceso 0.118237 absceso 0.358788	absceso 0.48517942
dcha 0.13454558 dcho 0.14473845 derecha 0.99074364 derecho 0.65845805	derecho 1.9855702
izda 0.28336138 izdo 0.19366418 izq 0.15289277 izquierda 0.69311315 izquierdo 0.49537328	izquierdo 1.9522555
l.i.o 3.6653504 lio 1.4208826	lio 5.092354
o.d 1.0478255 od 0.6360346	od 1.5289277
o.i 1.1742163 oi 0.30782405	oi 1.4820417
r.t.u 0.24055102 rtu 0.9336649	rtu 1.1742163
t.e.c 0.31393975 tec 0.86842954	tec 1.1823702

Tabla 5.10: Agrupaciones de conjuntos-AP que se forman antes y después del proceso de limpieza en el *Conjunto2*.

la posibilidad de generar un fichero de salida que contiene la información de la cantidad de filas procesadas, y cada una de las cantidades de los tipos de cambios que se realizan. En la tabla 5.11 se muestra la estructura y un ejemplo de dicho fichero.

Como se observa en dicha tabla, el fichero brinda la información de la cantidad de tuplas procesadas, la cantidad de puntos sustituidos (*los de los acrónimos más otros que se eliminan para tratar de los restantes utilizarlos como separadores*), la cantidad de acrónimos y sinónimos sustituidos y finalmente la cantidad total de cambios que se hicieron.

Para el caso particular de este ejemplo, se observan más cambios de puntos (*1883*) que de acrónimos (*1465*), pues en muchos casos, los acrónimos aparecen con los puntos intermedios y estos son eliminados (al menos, los intermedios). Como se ve, la cantidad de sinónimos sustituidos es de 1485, una cifra importante que nos da la medida de cómo se ha homogeneizado el vocabulario del atributo textual. Finalmente, se muestra el número total de cambios realizados que sumando los tres tipos es de 4806. Para el caso de estas estadísticas, no se tiene en cuenta la eliminación de las palabras de parada.

1. **Estadísticas obtenidas del proceso de limpieza de los datos**
   ———————————————————————————

2. Cantidad de **tuplas procesadas**: 24481
3. La cantidad de **puntos sustituidos** es: 1883
4. La cantidad de **acrónimos sustituidos** es: 1465
5. La cantidad de **sinónimos sustituidos** es: 1458
6. **LA CANTIDAD TOTAL DE CAMBIOS REALIZADOS ES: 4806**

Tabla 5.11: Estructura y ejemplo del fichero de estadísticas obtenido del proceso de limpieza de datos sobre el *Conjunto2*.

En la figura 5.3 se puede observar un ejemplo de las métricas obtenidas sobre el conjunto experimental *Conjunto2*. Para ello, se muestran los diferentes parámetros del fichero de estadísticas obtenido, para diferentes cantidades de tuplas en dicha tabla.

Como se ve en dicha figura, la cantidad total de puntos sustituidos es siempre mayor, que la cantidad de sinónimos y acrónimos sustituidos. Este resultado es lógico si se tiene en cuenta que, normalmente, por cada acrónimo que se sustituye, en muchas ocasiones se sustituye más de un punto, además de los que son eliminados cuando no tiene posibilidad de ser un separador entre dos frases.

Para el caso de los sinónimos y acrónimos, la cantidad sustituida es similar, aunque ligeramente, se sustituyen más sinónimos que acrónimos, sobre todo a partir de las 9792 tuplas en adelante. Este resultado es también correcto ya que, como se dijo anteriormente, existen muchas formas posibles en que se puede escribir un mismo término, y cada variación es sustituida por el término correspondiente.

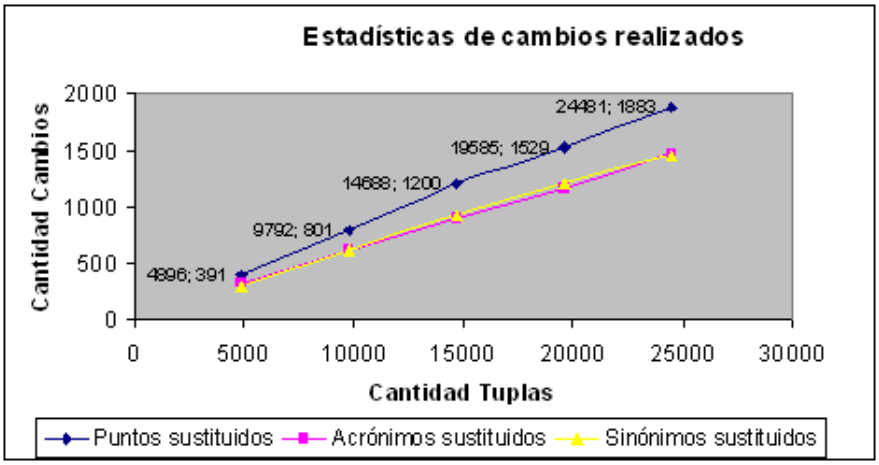


Figura 5.3: Estadísticas obtenidas en el proceso de limpieza sobre el conjunto2.

Hasta este punto, se han discutido los detalles del proceso de limpieza de datos, y las mejoras que aporta a la obtención de nuestra forma intermedia de representación. En la siguiente sección se discutirá la obtención de dicha forma intermedia utilizando el algoritmo Apriori.

5.3. Obtención de la estructura-AP global de conocimiento

Como se mencionó en la sección 5.1, nuestra hipótesis básica es que el conocimiento incluido en un atributo de tipo texto libre, de cualquier base de

datos, puede ser obtenido por medio de un simple proceso de Minería. Se asume que la estructura de itemsets frecuentes [1] de los datos del Diccionario, involucran la mayor parte de la semántica del atributo considerado. También consideramos que, una vez que el soporte es fijado, los itemsets frecuentes son los conjuntos de términos más frecuentes que aparecen en los datos limpios. Por consiguiente, ellos pueden ser vistos como la representación de las sentencias más frecuentes incluidas en los datos. Además, los itemsets frecuentes tienen la propiedad "Apriori"; así, ellos forman una estructura-AP cuyos conjuntos generadores son aquellos itemsets frecuentes no incluidos en ningún otro de cardinalidad superior, o sea, los itemsets maximales. Por consiguiente:

Se puede obtener la estructura semántica global del atributo textual, obteniendo la estructura-AP formada por los itemsets frecuentes del diccionario de datos. Los itemsets frecuentes maximales son los conjuntos generadores de dicha estructura-AP global.

Para este propósito, una vez fijado el soporte mínimo, el algoritmo Apriori es ejecutado sobre la base de datos transaccional con los datos textuales. Los itemsets frecuentes son obtenidos por este algoritmo y la estructura-AP global es representada considerando los conjuntos-AP maximales. A continuación se describen en detalle todas las tareas realizadas en este proceso.

5.3.1. Implementación del algoritmo Apriori

Con la implementación del algoritmo Apriori, como se comentó anteriormente, se obtienen las dos estructuras de conocimiento fundamentales que encierran la semántica de los datos textuales, estas estructuras son:

- **Conjuntos-AP:** Conjuntos de itemsets del procesamiento de todas las tuplas del atributo A_T, obtenidos con un soporte igual o superior, al que se especifica como soporte mínimo en el algoritmo Apriori. Es de destacar que los conjuntos-AP obtenidos poseen diferente cardinalidad y el algoritmo comienza incrementalmente por los de longitud 1. Así se obtienen todas las combinaciones de éstos que cumplen con el soporte

mínimo establecido, hasta llegar a los de máxima cardinalidad posible que cumplan con dicho soporte. Bajo este principio, la estructura-AP quedaría definida de la forma siguiente.

- **Estructura-AP:** Conjunto de itemsets de máxima cardinalidad. Dichos itemsets son obtenidos de forma incremental, realizando todas las combinaciones de los itemsets de longitud uno. La condición de parada será que la cardinalidad del itemset formado agrupando los itemsets de longitud uno, deje de cumplir con el soporte fijado al algoritmo Apriori.

Para la obtención de estas estructuras, como se comentó anteriormente, se utilizó la herramienta *Text Mining Tool V1.0.* En dicha herramienta es posible definir todos los parámetros de entrada para obtener las estructuras comentadas anteriormente.

La herramienta *Text Mining Tool V1.0*, como se comentó en el proceso anterior, se encarga de remover las palabras de parada del valor original de A_T para cada tupla, además crea el diccionario con los términos relevantes y su frecuencia que posee A_T. También crea la base de datos transaccional con la estructura que se comentó anteriormente. Finalmente, obtiene un nuevo atributo, para cada tupla de la tabla sobre la que se trabaja, el valor del atributo A_T limpio de las palabras de parada y con las actualizaciones que realiza el proceso de limpieza de datos. Este nuevo atributo será el que se utilice para realizar la intersección de A_T con la estructura-AP y obtener el TDA que representará el valor de dicho atributo A_T.

Para realizar todo este proceso, la herramienta implementada permite definir los parámetros de entrada necesarios para realizar todo este procesamiento. También da la posibilidad de definir los documentos de entrada para el análisis, como ficheros de texto o atributos de una base de datos. Además permite definir el fichero que contiene las palabras de parada y el nombre del fichero de salida, entre otros.

Para tener una idea de la flexibilidad que brinda la herramienta implementada, de forma resumida, los cuatro parámetros de entrada más importantes que se definen en su fichero de configuración (*TextMiningTool.cfg*) para realizar estos procesos son:

1. **source:** indica la fuente de los datos a procesar. Si es *cero* es de ficheros texto, y si es *uno*, indica que provienen de un atributo de una base de datos.

2. **txt.stoplist:** indica el nombre del fichero que contiene las palabras de parada.

3. **minsup:** soporte mínimo que deben cumplir los itemsets generados para ser tenidos en cuenta a la salida.

4. **output.file.dictionary:** indica el nombre del fichero que contendrá el diccionario con los términos relevantes y su frecuencia, obtenidos del atributo o fichero de entrada.

Lo mismo que estos parámetros, en el fichero de configuración *TextMiningTool.cfg* también se pueden definir el nombre de los ficheros de sinónimos y acrónimos, los parámetros para realizar la conexión con la base de datos en caso que la fuente sea desde una base de datos, etc.

La implementación del algoritmo Apriori, a partir del diccionario y la matriz transaccional creada en el paso anterior, obtiene los itemsets que cumplan con el soporte que se le especifica como parámetro. La herramienta implementada brinda la posibilidad de crear un fichero para cada una de las cardinalidades de los itemsets generados, o almacenar en un sólo fichero todos los itemsets obtenidos. La estructura de estos ficheros garantiza posteriormente, la optimización del algoritmo de intersección para la obtención del TDA. En dichos ficheros los términos aparecen ordenados alfabéticamente dentro de cada tupla, y por ende, el algoritmo de intersección puede realizar la búsqueda de una palabra dentro de una tupla, mientras el cardinal del primer carácter de cada palabra que se busca sea menor o igual al cardinal del primer carácter de la palabra que se lee del fichero de itemsets.

De forma resumida, los parámetros de entrada más importantes que aparecen en el fichero de configuración *TextMiningTool.cfg* referente a estos procesos son:

1. **output.storeSeparateIsets:** indica si se almacenarán en ficheros separados o en un sólo fichero, los conjuntos-AP generados.

2. **output.itemset:** nombre del fichero de itemsets que se genera, en caso que el parámetro *output.storeSeparateIsets* sea verdadero, el fichero que se genera lleva el nombre definido en *output.itemset*. Al final del nombre del fichero se le adiciona un número, que indica la cardinalidad de los itemsets generados en dicho fichero. Por ejemplo: *tintervenciones.tf.isets2.*

3. **itemsets.count:** indica la máxima cantidad de ficheros de itemsets que es obtenida, en caso que cada cardinalidad se almacene en un fichero separado. Se corresponde entonces con la longitud máxima con que se obtienen los itemsets generados.

5.3.2. Ejemplos de los conjuntos-AP y la estructura-AP global obtenida

Como se comentó anteriormente, la herramienta implementada brinda la posibilidad de obtener los ficheros de itemsets separados, un fichero para cada cardinalidad de las que se generan. Haciendo uso de esa posibilidad, se realizó la implementación del algoritmo que obtiene la estructura-AP global, generando los conjuntos-AP maximales que la componen, para cada cardinalidad.

A continuación se muestran algunos ejemplos de los ficheros de itemsets obtenidos y se discute su estructura. En la tabla 5.12 se puede observar un segmento del fichero de itemsets de longitud dos, obtenido sobre el conjunto experimental *Conjunto2*.

Como se puede observar, la estructura de cada línea del fichero consiste en los términos ordenados alfabéticamente, un delimitador (*#delim#*) y a continuación el soporte con que es obtenido ese itemset. De manera global, también cada fila se ordena alfabéticamente. Es importante destacar, que tanto los ficheros de itemsets, como los de conjuntos-AP maximales que componen la estructura-AP tienen este mismo ordenamiento alfabético. Por tanto, como son precisamente estos últimos los que se utilizan para realizar la intersección; ambas organizaciones alfabéticas, la de las palabras dentro una fila, y

axilar cuadrantectomia #delim# 0.22467342
axilar diseccion #delim# 0.6535948
axilar mastectomia #delim# 0.36764735
axilar vaciamiento #delim# 0.1225492
benigna extirpacion #delim# 0.2042486
benigna lesion #delim# 0.2042486
benigna piel #delim# 0.2042486
bio da #delim# 0.2450984
bio ev #delim# 0.1021243
bio eventual #delim# 0.1633987
bio expl #delim# 0.1021243
bio ht #delim# 0.30637267

Tabla 5.12: Ejemplo de un fichero de itemsets de longitud dos del Conjunto2.

las de las líneas dentro del fichero, permiten que el algoritmo implementado para la intersección tenga esto en cuenta y logre mayor eficiencia.

En la tabla 5.13 se muestra el algoritmo seguido para obtener los conjuntos-AP generadores de la estructura-AP. Para ello se toman como entrada los ficheros de itemsets de todas las cardinalidades generados, y para cada uno de ellos, se eliminan del fichero de longitud n, los itemsets que se encuentren incluidos en el fichero de longitud $n+1$. Para el caso de las longitudes de itemsets mayores que uno, tienen que aparecer completamente incluidos, todos los términos de cada fila del fichero de longitud n, en alguna fila del fichero de longitud $n+1$.

Como se observa en dicha tabla, se hace la descripción general del algoritmo implementado para obtener los conjuntos-AP maximales. Como se ha dicho, éstos no serán más que la estructura-AP generada como dominio activo para el atributo textual que se procesa. El algoritmo trabaja sobre tres ficheros a la vez: los dos ficheros de entrada que contienen los itemsets de cardinalidad n y $n+1$ y el fichero de salida que contiene los conjuntos-AP maximales de cardinalidad n.

De forma general, el algoritmo lo que hace es leer todas las filas del fichero de itemsets de cardinalidad n, y verificar si todos sus términos se encuentran

Entrada: Máxima cantidad de ficheros de itemsets (*cantidad_ itemsets*) y ficheros de itemsets (se leen en A y B).

Salida: Ficheros maximales de conjuntos-AP (*uno por casa valor final de S*).

```
1.  HACER cantidad ←1
2.  HACER encontrados ←0
3.  MIENTRAS cantidad <cantidad_itemsets HACER
4.      HACER A ← ABRIR Fichero_de_itemsets de longitud igual a cantidad
5.      HACER B ← ABRIR Fichero_de_itemsets de longitud igual a cantidad+1
6.      HACER S ← ABRIR Fichero_de_Conj-AP_Maximal de longitud igual a cantidad
7.      MIENTRAS NO SEA FIN DEL FICHERO A HACER
8.          HACER encontrar ←falso, i←0, j←0
9.          HACER a ← LEER LINEA de A
10.         HACER b ← LEER LINEA de B
11.         MIENTRAS ordinal a[0] <= ordinal b[0] Y NO SEA FIN DEL FICHERO B Y
                        encontrar =falso HACER
12.             HACER a_j ← LEER PALABRA de a[i]
13.             HACER b_j ← LEER PALABRA de b[j]
14.             MIENTRAS ordinal a_j [0] <= ordinal b_j [0] Y a_j <>∅ Yb_j <>∅ HACER
15.                 SI a_j = b_j ENTONCES
16.                     HACER encontrados ← encontrados+1
17.                     HACER i←i+ longitud de a_j +1
18.                     HACER a_j ← LEER PALABRA de a[i]
19.                 FIN SI
20.                 SINO
21.                     HACER j←j+ longitud de b_j +1
22.                    HACER b_j ← LEER PALABRA de b[j]
23.                 FIN SINO
24.             FIN MIENTRAS
25.             SI encontrados= cantidad ENTONCES encontrar← verdadero
26.             SINO
27.                 HACER b ←LEER LINEA de B
28.                 HACER i ←0
29.                 HACER a_j ← LEER PALABRA de a[i]
30.             FIN SINO
31.         FIN MIENTRAS
32.         SI encontrar = falso ENTONCES ESCRIBIR a en S
33.     FIN MIENTRAS
34.     HACER cantidad← cantidad +1
35.     CERRAR A, B, S
36. FIN MIENTRAS
```

Tabla 5.13: Algoritmo para obtener los conjuntos-AP maximales.

completamente incluidos en alguna línea del fichero de itemsets de cardinalidad $n+1$. Si no se encuentra, escribe dicha línea en el fichero de conjuntos-AP maximales de salida. Si por el contrario se encuentra la línea del fichero de itemset de longitud n en el de longitud $n+1$, no se escribe a la salida para esa cardinalidad, pues quiere decir que ya todos sus términos están incluidos en un fichero de cardinalidad superior. De esta forma, el algoritmo implementado garantiza que se obtienen, para cada cardinalidad, sólo los conjuntos-AP maximales.

El algoritmo también tiene en cuenta que el fichero de itemset de máxima cardinalidad no se procesa, pues no existe la posibilidad de que este incluido en otro de cardinalidad mayor; por ende el contenido de este fichero el sistema lo pasa directamente a la salida como conjuntos-AP maximales. Esta última condición se cumple con la verificación que se realiza en la línea (3).

Como se comentó anteriormente, para optimizar el algoritmo que obtiene los conjuntos-AP maximales se hace uso del ordenamiento alfabético dentro de la fila. En la línea (14) por ejemplo, se verifica la condición ($ordinal\, a_j[0] <= ordinal\, b_j[0]$) que chequea que la palabra que se busca dentro de una fila del fichero de itemsets de longitud n, comience con una letra cuyo ordinal es menor o igual que el ordinal de la primera letra de la palabra con que se va a comparar del fichero de itemsets de longitud $n+1$. Si se cumple esa condición y las palabras comparadas son iguales, se incrementa el contador de palabras encontradas y se leen las próximas palabras de las líneas actuales de cada fichero, si no, se lee la próxima palabra del fichero de longitud $n+1$. De forma análoga ocurre en la línea (11) cuando se leen las filas de cada fichero, donde se realiza la búsqueda de las palabras sólo si la primera palabra de la línea del fichero de longitud n, comienza con un carácter cuyo ordinal es menor o igual que la primera palabra de la línea del fichero de longitud $n+1$ ($ordinal\, a[0] <= ordinal\, b[0]$).

Es de destacar que todo este proceso descrito anteriormente es fuertemente dependiente tanto del soporte mínimo (*minsup*), como de la máxima cardinalidad de los itemsets frecuentes generados (*maximalItemsetLength*). De hecho, estos dos parámetros están relacionados ya que un elevado *minsup* produce maximales de baja cardinalidad.

A continuación en la tabla 5.14 se muestra un ejemplo donde se toma como entrada, algunas líneas de dos ficheros de itemsets de longitud dos y tres, y se muestra el fichero de conjuntos-AP maximales de longitud dos que se generaría.

Como se observa en la tercera fila de la tabla 5.14, los conjuntos-AP maximales de longitud dos que se obtienen, serán sólo los itemsets de longitud dos que no se encuentren completamente incluidos en los de longitud tres. Dado que la estructura-AP no será más que la suma de todos estos conjuntos-AP maximales que se obtienen, el proceso de intersección realizado para obtener el TDA asociado a cada tupla, contra esta estructura, será más eficiente. Esto se logra precisamente porque sólo se almacenan los conjuntos-AP maximales, y no todas sus posibles combinaciones que cumplan con el soporte especificado, por eso al tener menos filas dichos ficheros, se obtiene en menor tiempo el resultado de la intersección.

En el siguiente apartado se muestran brevemente algunos ejemplos de retículos obtenidos sobre los datos experimentales. Para ello se utilizan los datos experimentales de los conjuntos 2 y 3.

5.3.3. Ejemplos de retículos de conjuntos-AP y estructuras-AP obtenidos.

En las figuras 5.4 y 5.5 se muestran dos ejemplos de retículos de conjuntos-AP de cardinalidad 2 y 3 respectivamente. Dichos retículos han sido obtenidos utilizando el *Conjunto2* de los datos experimentales.

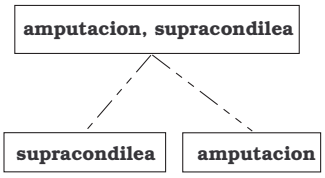


Figura 5.4: Ejemplo conjunto-AP cardinalidad 2 sobre el *Conjunto2*.

Itemsets de longitud dos:
ab histerectomia #delim# 0.1021243
ab profilaxis #delim# 0.1225492
abdominal anexectomia #delim# 0.1021243
abdominal da #delim# 0.2450984
abdominal histerectomia #delim# 0.7557195
abdominal profilaxis #delim# 0.63317
abdominal simple #delim# 0.1429738
abdominal subtotal #delim# 0.22467342
abdominal total #delim# 0.4697711
Itemsets de longitud tres:
abdominal anexectomia histerectomía #delim# 0.1021243
abdominal da histerectomia #delim# 0.2450984
abdominal da profilaxis #delim# 0.2450984
abdominal da total #delim# 0.2042486
abdominal histerectomia profilaxis #delim# 0.63317
abdominal histerectomia simple #delim# 0.1429738
abdominal histerectomia subtotal #delim# 0.22467342
abdominal histerectomia total #delim# 0.4697711
abdominal profilaxis simple #delim# 0.1429738
abdominal profilaxis subtotal #delim# 0.18382367
Conjuntos-AP maximales de longitud dos:
{ab, histerectomía} →0.1021243
{ab, profilaxis} → 0.1225492

Tabla 5.14: Ejemplo de obtención de conjuntos-AP maximales de longitud dos sobre el *Conjunto2*.

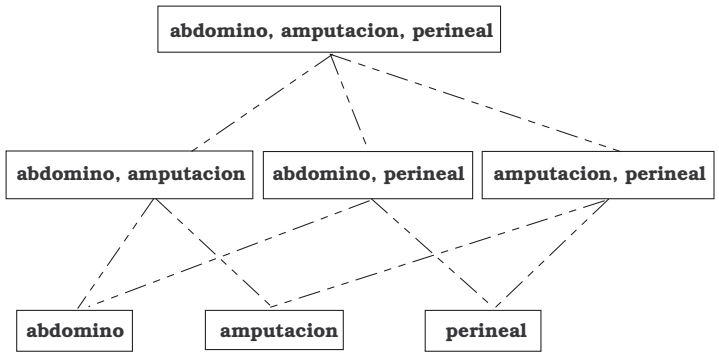


Figura 5.5: Ejemplo de conjunto-AP de cardinalidad 3.

Con los conjuntos-AP que aparecen representados en las figuras 5.4 y 5.5 se puede conformar una estructura-AP como la que aparece en la figura 5.6.

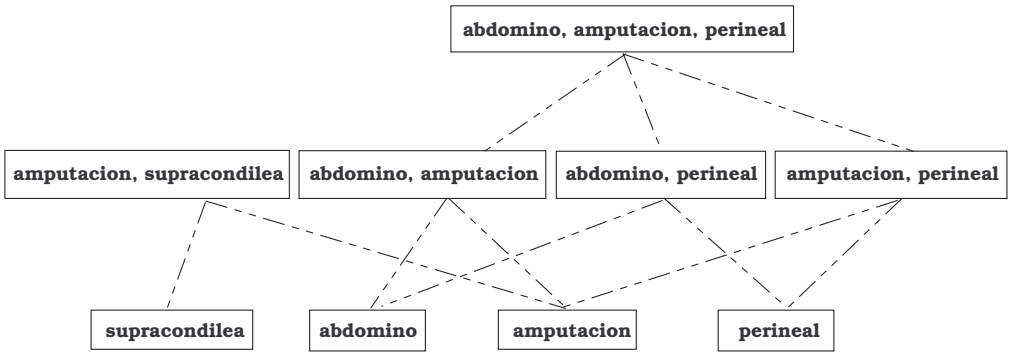


Figura 5.6: Ejemplo de estructura-AP sobre el *Conjunto2*.

De forma análoga, en las figuras 5.7 y 5.8 se muestran dos ejemplos de retículos de conjuntos-AP de cardinalidad 3 y 4 respectivamente. En estos casos, han sido obtenidos utilizando el *Conjunto3* de los datos experimentales.

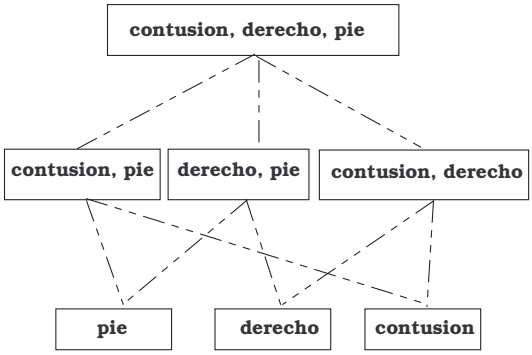


Figura 5.7: Ejemplo conjunto-AP de cardinalidad 3 sobre el *Conjunto3*.

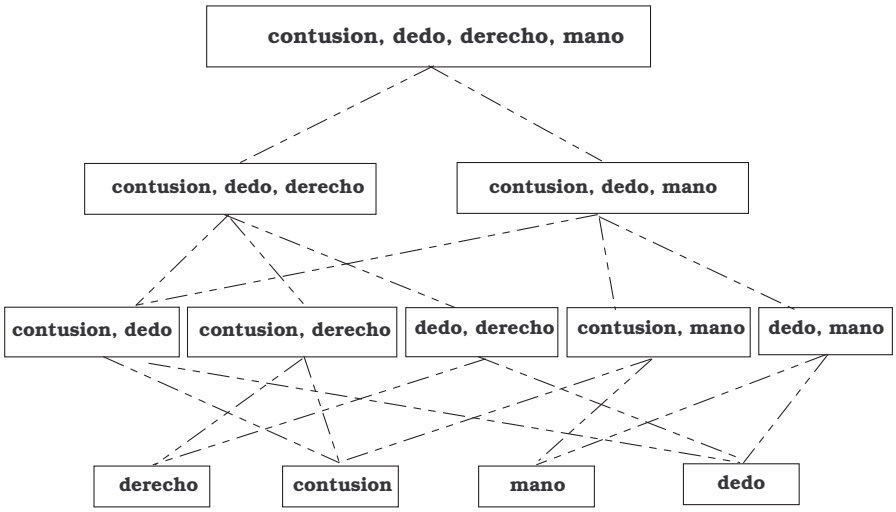


Figura 5.8: Ejemplo de segmento de conjunto-AP de cardinalidad 4 sobre el *Conjunto3*.

También con los conjuntos-AP que aparecen representados en las figuras 5.7 y 5.8 se puede conformar una estructura-AP como la que aparece en la figura 5.9.

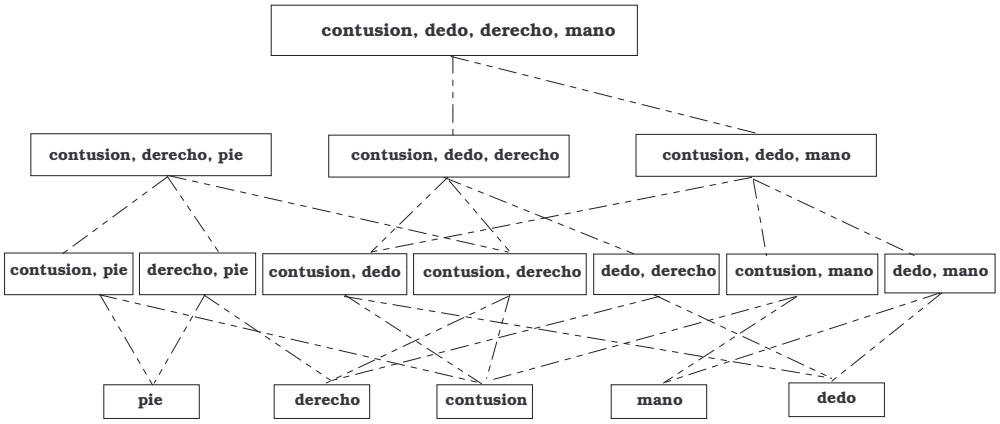


Figura 5.9: Ejemplo estructura-AP sobre *Conjunto3*.

Hasta este punto ya se tiene la estructura-AP y los conjuntos que la forman, así como el atributo textual limpio. En la siguiente sección se discute la forma en que se obtiene la estructura-AP inducida, para cada tupla del atributo textual que se procesa.

5.4. Obtención de la estructura-AP inducida para cada tupla de la base de datos

Antes de ver detalles de implementación de cómo se obtuvo la estructura-AP inducida para cada tupla de la base de datos, se retoman algunos aspectos teóricos que fueron introducidos en el capítulo 3. Dichos aspectos serán la base para la obtención de las estructuras-AP inducidas para cada tupla (TDA asociado).

Sea R una relación con atributos $\{A_1, A_2, A_T, ..., A_n\}$, donde A_T es un atributo textual sin una estructura predecible. En este punto del proceso, se asume que el atributo A_T está limpio. En nuestro ejemplo se encuentran en la segunda columna de la tabla 5.2.

Como establecimos anteriormente, la estructura-AP global obtenida, cubre la semántica del atributo A_T y así, éste nos proveerá con el dominio para el

TDA, el cual reemplazará el mencionado atributo. Siendo $\mathcal{T} = g(A, B, ...)$ dicha estructura-AP.

Considerando ahora la tupla $x \in R$, para obtener la instancia del TDA para dicha tupla, es suficiente con obtener la subestructura-AP de $\mathcal{T}$ inducida por el valor de A_T para x, $x[A_T]$. Si denotamos por A_{TN} este nuevo atributo, tenemos:

$$\forall x \in R \; x[A_{TN}] = \mathcal{T} \bigwedge x[A_T]$$

Está claro que cada valor de este nuevo atributo A_{TN} es una subestructura-AP de la global. En este sentido, como se comentó anteriormente, se considera la estructura-AP global como el "*valor de dominio*" de este atributo. Ambas, la estructura-AP global y las subestructuras-AP inducidas son del mismo tipo TDA. El nuevo atributo resultante, es almacenado en una nueva columna de la base de datos modificada.

De forma general, a este proceso de obtención de las subestructuras-AP lo denominamos, intersección, refiriéndonos, como ya se explicó, a la intersección que se realiza entre los valores de los conjuntos-AP que describen el valor del atributo textual y la estructura-AP global de conocimiento. Respecto a nuestro caso real, realizando dicha intersección, se han computado las subestructuras-AP correspondientes a cada tupla en la base de datos, diseñando un algoritmo con fines específicos (*ad hoc*). La segunda columna en la tabla 5.16 muestra los conjuntos generadores de las estructuras-AP inducidas por los valores de la primera columna.

Es posible que se pierdan algunos términos en el valor de alguna tupla dado que, para calcular la subestructura-AP inducida, se restringe el conjunto de términos que forman el atributo después de la limpieza, contra la estructura-AP. Dado la forma en que se obtiene dicha estructura-AP, ésta puede perder los términos del lenguaje inicial del atributo, que no cumplan con el soporte mínimo, por eso, estos términos no aparecerían en el TDA. Un ejemplo de este caso aparece en la segunda fila de la tabla 5.16. En dicha fila desaparece el término *bajo* pues no se encuentra en la estructura-AP obtenida al no cumplir con el soporte mínimo fijado.

Este hecho de que la forma de representación obtenida "discrimina" términos que no cumplen con un soporte mínimo, se reconoce que constituye una limitación con respecto a la forma tradicional en que se procesan los textos cortos en bases de datos.

A continuación se discute más en detalle cómo ocurre el proceso de intersección entre el valor de una tupla y la estructura-AP global que encierra el conocimiento del atributo que se procesa. Para ello, de forma análoga a los procesos descritos anteriormente, se darán detalles de su implementación.

5.4.1. Descripción del proceso de intersección entre el valor de una tupla y la estructura-AP

La implementación de la intersección, al igual que el resto de los procesos hasta aquí descritos, se realizó en la herramienta *Text Mining Tool V1.0*, en cuyo fichero de configuración *TextMiningTool.cfg* también se definen todos los parámetros de entrada que requiere dicho proceso. Para obtener el valor del TDA del atributo A_T, para cada tupla, se utilizará la salida del proceso de limpieza. Como se explicó anteriormente, dicha salida es una nueva columna en la base de datos que contiene, para cada tupla, la representación de A_T removiendo las palabras de parada y realizando el proceso de limpieza. Para cada valor de sus tuplas se realiza el proceso de intersección con la estructura-AP y el resultado final será el que se escriba en otra columna A_{TN} como valor del TDA para A_T .

Entre los parámetros más importantes que se especifican en el fichero *TextMiningTool.cfg* para realizar el proceso de intersección se tienen:

1. **maxItemsets.name:** indica el nombre con que son generados los conjuntos-AP maximales que forman la estructura-AP global. Estos ficheros serán los utilizados en el proceso de intersección.

2. **itemsets.count:** como se definió anteriormente, indica la máxima cardinalidad que poseen los conjuntos-AP que se procesan.

3. **intersection.filename:** indica el nombre del fichero en el que se escribe el resultado de la intersección, en caso que se seleccione la salida a fichero y no a base de datos.

4. **intersection.attribute:** indica el nombre del atributo donde se escribirá el valor de la intersección de cada fila, en caso que se seleccione la salida a base de datos. Este será el nombre que recibe la columna donde se escribe el TDA.

Como se deduce de los parámetros anteriores, la herramienta *Text Mining Tool V1.0* brinda la posibilidad de especificar en su fichero de configuración, si la intersección calculada va a un fichero o a la base de datos. Para el primer caso deja especificar el nombre del fichero de salida en el parámetro *intersection.filename*, y en caso que la salida sea a base de datos, en el parámetro *intersection.attribute* permite especificar el nombre del atributo donde se escribe el TDA.

Como se explicó anteriormente, para obtener el valor del TDA para cada fila en el atributo A_T, se utilizará la salida del proceso de limpieza de datos, que contiene para cada tupla, la representación de A_T después de este proceso. Para ello se calcula la intersección entre cada valor de las filas de dicho fichero y la estructura-AP. El resultado final será el que se escriba en otra columna A_{TN} como valor de TDA para A_T. La implementación de este proceso de intersección de forma detallada realiza, en este orden, los siguientes pasos:

1. Fijar la máxima cantidad de los ficheros de conjuntos-AP que componen la estructura-AP, leyendo el parámetro *itemsets.count.*

2. Para cada tupla del atributo A_T después de la limpieza de datos y sin las palabras de parada, calcular la intersección del valor leído de la tupla con el fichero de conjuntos-AP de máxima cardinalidad posible.

3. Escribe en la tupla correspondiente el valor de la intersección calculada en el atributo A_{TN}.

De la descripción del algoritmo anterior cave resaltar, que su implementación tiene en cuenta estrictamente la operación de intersección ($\bigwedge$) definida

Entrada: Estructura-AP global $\mathcal{T}\{t_1, t_2, \ldots, t_m\}$ y tuplas después de la limpieza del atributo textual que se procesa $A_T\{A_{T1}, A_{T2}, \ldots, A_{Tn}\}$.
Salida: TDA asociado al atributo textual $I\{I_1, I_2, \ldots, I_n\}$.

1. **HACER** $I \leftarrow \emptyset$, $j \leftarrow 1$
2. **PARA CADA** tupla A_{Tj} ($j = 1 \ldots n$) **HACER**
3. **HACER** $h \leftarrow 1$, $i \leftarrow 1$ $encontrar = falso$
4. **HACER** $K \leftarrow \emptyset$
5. **MIENTRAS** $i <= m$ **Y** $encontrar$ = falso **HACER**
6. **HACER** $K \leftarrow A_{Tj} \bigcap t_i$
7. **MIENTRAS** $h <= m$**Y** $encontrar$ = falso
8. **SI** $K \subseteq I_h$**ENTONCES HACER** $encontrar = verdadero$
9. **SINO**
10. **SI** $I_h \subseteq K$ **ENTONCES** $I_J \leftarrow (I_j - I_h) \cup K$
11. **SINO** $I_J \leftarrow I_J \cup K$
12. **HACER** $h \leftarrow h + 1$
13. **FIN MIENTRAS**
14. **HACER** $i \leftarrow i + 1$
15. **FIN MIENTRAS**
16. **HACER** $j \leftarrow j + 1$
17. **FIN PARA**

Tabla 5.15: Algoritmo de intersección implementado.

en el capítulo 3, entre un conjunto y una estructura-AP. En esa operación se garantiza que el resultado de la intersección, elimina las redundancias posibles entre intersecciones parciales del conjunto con la estructura-AP y se da a la salida, sólo las intersecciones que no están contenidas completamente en otras. Para optimizar este proceso, el algoritmo implementado comienza buscando la intersección de cada tupla contra el fichero de conjuntos-AP de máxima cardinalidad. De igual forma va calculando la intersección con cada uno de los ficheros de las diferentes cardinalidades que componen la estructura-AP de forma decreciente. De esta manera siempre la salida que se da al TDA, garantiza que es la que se acopla con la frase de mayor cardinalidad en la estructura-AP.

La implementación realizada de este algoritmo de intersección, ha sido validada con diferentes cantidades de tuplas y sobre varios atributos en las tablas de *Intervenciones Quirúrgicas* y *Urgencias*. A continuación se muestran algunos ejemplos que ilustran algunas intersecciones particulares que se obtienen para reforzar estos aspectos que se han descrito anteriormente.

5.4.2. Ejemplos y discusión sobre intersecciones obtenidas

En la tabla 5.16 se muestran algunas tuplas de uno de los ejemplos que ilustran los resultados obtenidos. Dichas tuplas son también del mismo experimento que se muestra en la tabla 5.2, lo que en estas muestras se incluyen dos casos particulares que se pueden dar en la obtención del TDA.

En la tabla 5.16 la columna *IPropuesta Modificada*, se refiere al atributo *Intervención Propuesta*, después de haber realizado sobre él todo el proceso de limpieza de datos que se describió anteriormente. La columna TDA, contiene el valor del resultado de la intersección de las tuplas que se muestran en la columna *IPropuesta Modificada* con la estructura-AP global. Como se observa, prácticamente todos los términos estaban completamente incluidos en la

IPropuesta Modificada	**TDA**
amputacion	{amputacion}
bajo exploracion narcosis	{exploracion, narcosis}
izquierdo pelvis plastia	{izquierdo, pelvis, plastia}
reimplantacion vesicoureteral	{reimplantacion, vesicoureteral}
corneal herida sutura	{corneal, herida, sutura}
abducion yeso	{yeso},{abducion}
incruenta reduccion yeso	{incruenta, reduccion},{reduccion, yeso}

Tabla 5.16: Ejemplo de intersección obtenida sobre el *Conjunto2*.

estructura-AP global. Para los casos en que ocurre esto, coinciden los valores de ambas columnas, y en otros casos como en las filas 6 y 7, se obtienen combinaciones de los términos con los que se realiza la intersección con la estructura-AP, según aparezcan estos en los conjuntos-AP que la conforman.

Para el caso del valor *abducion yeso*, como se observa en la tabla 5.16, se obtiene un valor de TDA que contiene ambos términos por separado (*{yeso},{abducion}*). Esto significa, que los términos *abducion yeso* no se encuentran unidos como un sólo conjunto en ninguno de los conjuntos-AP obtenidos con cardinalidad mayor que uno. Lo que quiere decir que se acoplan sólo con sus conjuntos-AP de cardinalidad uno, y es por eso que se obtienen como términos por separado.

Para el caso del valor *incruenta reduccion yeso*, se obtienen dos conjuntos-AP resultantes, cada uno con dos términos, ellos son *{incruenta, reduccion},{reduccion, yeso}*. De forma análoga a como ocurre en el caso anterior, para el caso de este tipo de salida, significa que el término *incruenta reduccion yeso* no aparece completamente contenido en ningún conjunto-AP de cardinalidad igual o superior a su longitud tres. Es por esto que la intersección ocurrió con los conjuntos-AP de cardinalidad dos y en la forma en que ellos se encuentran almacenados. Como se observa se repite el término *incruenta*, algo que se corresponde con la definición que se ha dado en nuestro modelo de la intersección entre un conjunto y la estructura-AP.

El otro caso particular que puede ocurrir en la intersección es el que se explicó anteriormente, que algún término que está contenido en el atributo

A_T después de la limpieza, no este incluido en el TDA. Esto se debe, como ya se explicó, a que dicho término no se encuentra en la estructura-AP global obtenida debido a que no cumple con el soporte especificado al algoritmo Apriori.

A continuación en las figuras 5.17, 5.18, 5.19 se muestran ejemplos de intersecciones obtenidas para el resto de los conjuntos experimentales. Como se observa, los patrones descritos para el caso del *conjunto2* se repite en estos ejemplos.

Diagnostico Modificado	**TDA**
catarata dilatar od	{catarata, dilatar}
depresion	{depresion}
ambos oidos perforacion	{ambos},{oidos, perforacion}
dorso quiste sebaceo	{quiste,sebaceo}
derecho inferior miembro varices	{derecho, inferior, miembro, varices}
cuello quiste	{cuello},{quiste}
cronica insuficiencia renal	{cronica,insuficiencia,renal}

Tabla 5.17: Ejemplo de intersección obtenida sobre el *Conjunto1*.

Motivo Modificado	**TDA**
vomitos cefalea	{vomitos},{cefalea}
asmatica crisis	{asmatica, crisis}
contusion	{contusion}
fiebre vomitos	{fiebre},{vomitos}
molestia ocular	{molestia, ocular}
dedos mano traumatismo	{dedos, mano, traumatismo}
insuficiencia respiratoria	{insuficiencia},{respiratoria}

Tabla 5.18: Ejemplo de intersección obtenida sobre el *Conjunto3*.

Juicio Modificado	TDA
INGRESO ABORTO COMPLETO	{esginces, tobillo}
abdominal dolor inespecifico	{abdominal, dolor, inespecifico}
inflamado quiste sebaceo	{quiste, sebaceo}
aguda gastritis	{aguda, gastritis}
neumonia	{neumonia}
colico derecho nefritico	{colico, derecho, nefritico}
obstruccion urinarias vias	{obstruccion, urinarias, vias}

Tabla 5.19: Ejemplo de intersección obtenida sobre el *Conjunto4*.

Para tener una idea de en qué magnitud se pierden términos durante el proceso de intersección y su fuerte dependencia con respecto al soporte fijado al algoritmo Apriori, en el siguiente apartado se realizan experimentos que clarifican este punto.

5.4.3. Ejemplos y discusión de las estadísticas obtenidas en el proceso de intersección

Para tener una idea de la eficiencia con la que se obtiene la intersección de las tuplas con la estructura-AP global, la herramienta implementada, genera un fichero con las estadísticas principales de este proceso. Dicho fichero contiene:

1. **Cantidad de palabras que se pierden:** contiene el número de palabras que están en el atributo A_T después del proceso de limpieza, y no se encuentra en su TDA asociado.

2. **Cantidad de tuplas que incluyen palabras perdidas:** contiene el número de tuplas que contienen al menos un términos perdido.

3. **Porcentaje de palabras que se pierden**: contiene el porcentaje de palabras que se pierden con respecto al total de palabras que se procesan en todo el atributo A_T después del proceso de limpieza.

4. **Porcentaje de tuplas que incluyen palabras perdidas:** contiene el porcentaje de tuplas que al menos pierden un término con respecto al total de tuplas procesadas.

Con el propósito de medir cuanta información es reducida por reemplazar el conjunto de términos por su intersección con la estructura-AP, se calcularon estos cuatro parámetros usando diferentes valores de *minsup*, y tomando diferentes cantidades de tuplas de la base de datos. El gráfico de la figura 5.10 muestra los resultados obtenidos. Dicho gráfico es obtenido también sobre el conjunto experimental *Conjunto2*. En él se muestra lo expuesto anteriormente sobre la dependencia del soporte con la cantidad de palabras que se pierden.

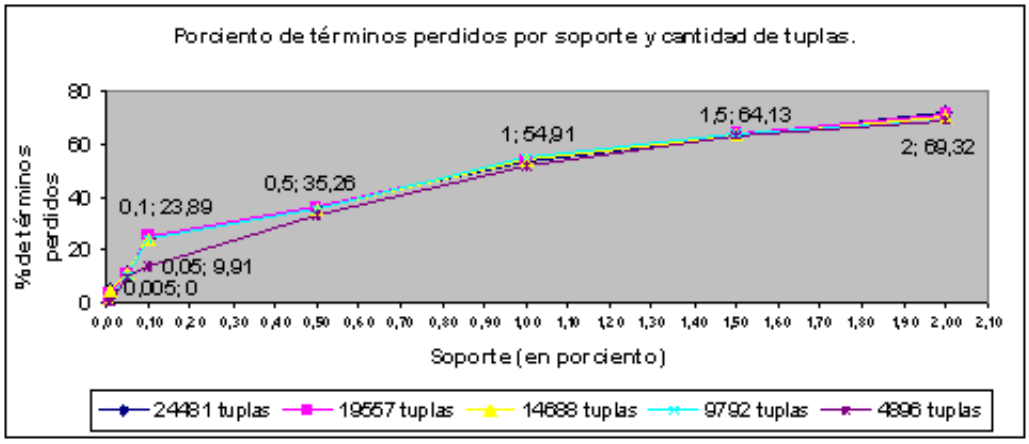


Figura 5.10: Resultados de los experimentos con términos perdidos sobre el *Conjunto2*.

En en el caso concreto de dicha figura, se trabajó con el porcentaje de palabras que se pierden. Se observa claramente cómo a medida que aumenta el soporte, también aumenta el porcentaje de palabras que se pierden. Además se ve, que el comportamiento de este fenómeno para las diferentes cantidades de tuplas utilizadas es prácticamente el mismo. Este resultado, a nuestro juicio, es el más interesante ya que demuestra que esta medida es casi independiente

del número de tuplas consideradas. Este resultado refuerza la hipótesis que la estructura-AP global verdaderamente captura la semántica del atributo, ya que como se observa, para valores bajo del soporte el porcentaje de palabras que se pierden con respecto al total es bajo.

Por otra parte, el resultado anterior también sugiere que es posible obtener la semántica subyacente en un atributo textual sin utilizar la base de datos completa. Esto hace que se pueda afirmar que el proceso de minería es escalable.

A continuación en las figuras 5.11, 5.12 se muestran ejemplos de estadísticas sobre porcentaje de términos perdidos, obtenidas para el resto de los conjuntos experimentales. Como se observa, los patrones descritos para el caso del *Conjunto2* se repite en estos ejemplos.

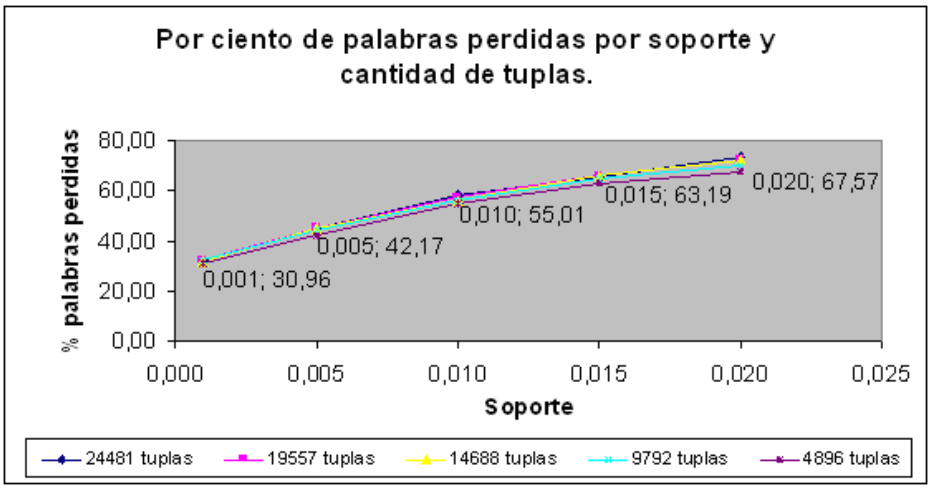


Figura 5.11: Resultados de los experimentos con términos perdidos sobre el *Conjunto1*.

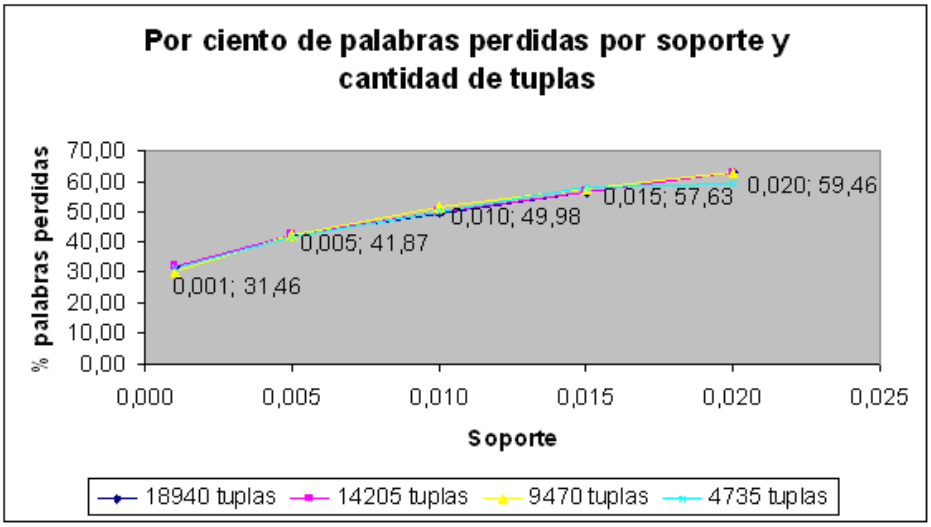


Figura 5.12: Resultados de los experimentos con términos perdidos sobre el *Conjunto3*.

En la figura 5.13 se muestra un gráfico similar al anterior, obtenido sobre el *conjunto2* de los datos experimentales. Esta vez se muestra el porcentaje de tuplas que poseen términos perdidos.

Como se observa en la figura 5.13, la distribución de las tuplas que pierden términos, para diferentes cantidades de soporte, tiene el mismo comportamiento que el caso anterior. A medida que el soporte aumenta, el porcentaje de tuplas que incluyen términos perdidos también aumenta. De forma análoga también, para las diferentes cantidades de tuplas que se procesan el comportamiento del por ciento de tuplas con palabras perdidas es muy similar.

Hasta este punto, a lo largo del capítulo, se han discutido todos los detalles del proceso de obtener el TDA asociado a un atributo textual. A continuación, y

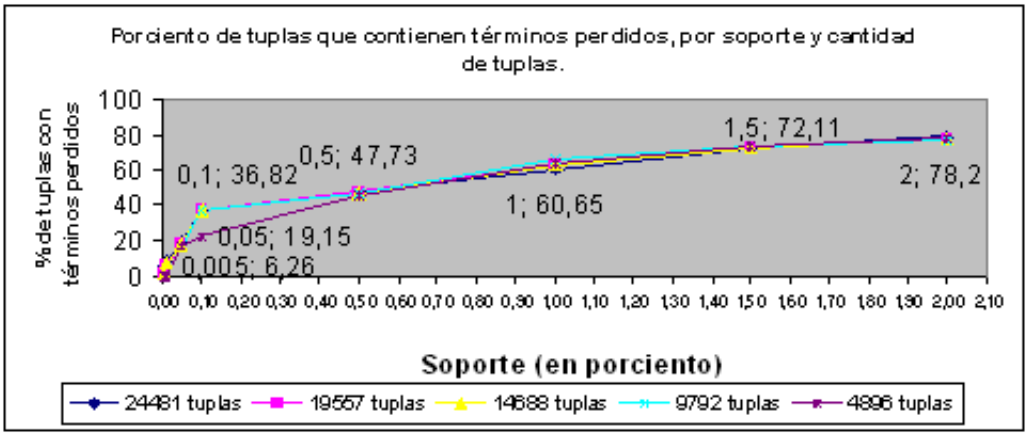


Figura 5.13: Resultados de los experimentos con tuplas con términos perdidos sobre el *Conjunto2*.

a modo de resumen, en la siguiente sección se desarrolla un ejemplo práctico sobre un atributo textual de la base de datos. El objetivo fundamental del ejemplo será ver cómo ocurre la transformación de los datos del atributo textual, hasta obtener su estructura-AP inducida representada por un TDA.

A continuación en las figuras 5.14, 5.15 se muestran ejemplos de estadísticas sobre por ciento de términos perdidos, obtenidas para el resto de los conjuntos experimentales. Como se observa, los patrones descritos para el caso del *conjunto2* se repite en estos ejemplos.

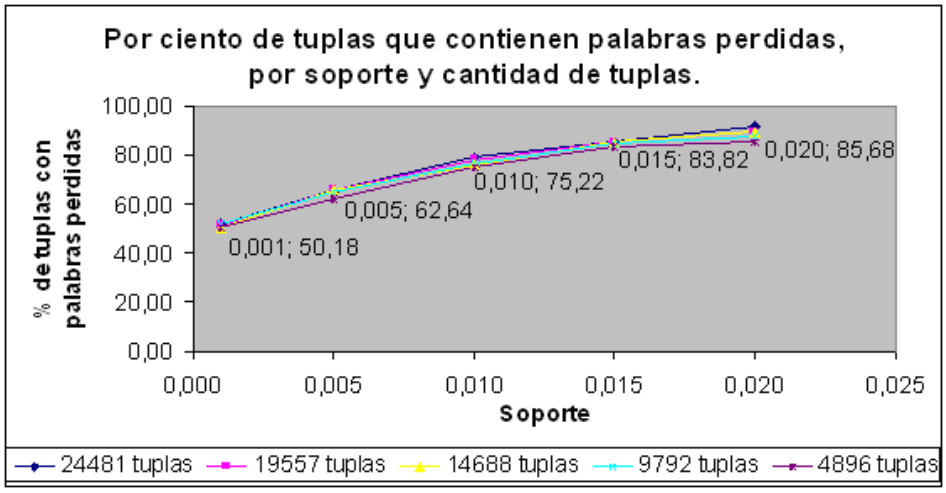


Figura 5.14: Resultados de los experimentos con tuplas con términos perdidos sobre el *Conjunto1*.

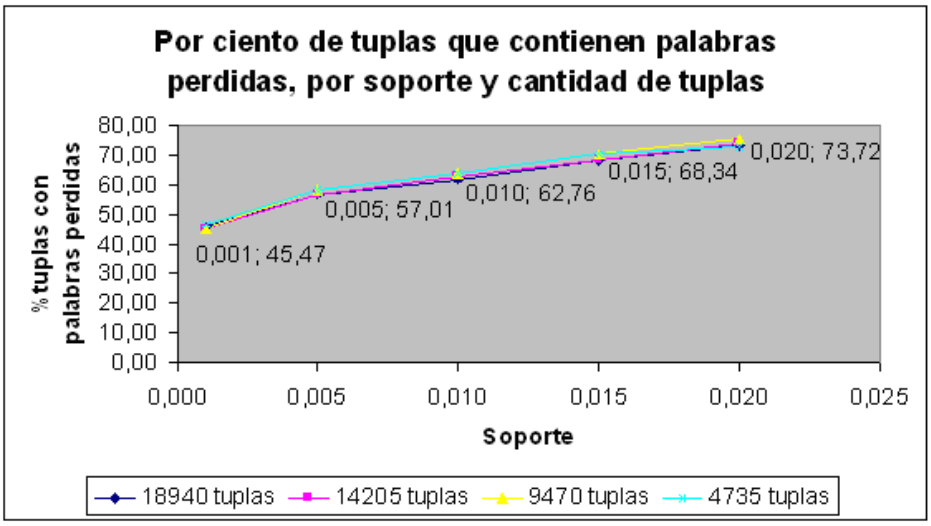


Figura 5.15: Resultados de los experimentos con tuplas con términos perdidos sobre el *Conjunto3*.

5.5. Conclusiones

La metodología presentada en este capítulo nos da el procedimiento a seguir para manejar un atributo textual, como y conjuntamente al resto de los atributos en bases de datos (Relacionales, Relacional Orientado a Objetos, etc.). Se ha explicado y ejemplificado la importancia del preprocesamiento de los datos en el proceso de Minería de Textos. También se ha visto cómo utilizando el algoritmo Apriori se generan los itemsets frecuentes, con los que se construye la estructura global de conocimiento que encierra la semántica del atributo textual. Utilizando dicha estructura es obtenido finalmente el TDA que le corresponde a cada tupla de la base de datos para dicho atributo.

Con la aplicación de la metodología propuesta a una base de datos médica, se ha demostrado la posibilidad real de su implementación. Los resultados estadísticos obtenidos durante el proceso, refuerzan la hipótesis inicial de que la estructura-AP realmente encierra la semántica presente en el atributo que se procesa. De aquí que dicha estructura-AP pueda ser definida como el dominio activo sobre el que se valora cada tupla del atributo textual que se procesa.

Hasta este capítulo se ha demostrado cómo se obtuvo la representación física de la forma intermedia de representación propuesta. En el capítulo siguiente, se discutirán los posibles tipos de consultas que se podrían hacer sobre las estructuras obtenidas. También se muestra la arquitectura y funcionalidades del Cliente de Consulta creado para la realización de dichas consultas. Además se ejemplifican las características de las que se han dotado a las extensiones implementadas en PostgreSQL, que hacen el sistema más general y adaptable a las necesidades de información del usuario.

Capítulo 6

Consultando el sistema: El cliente de consulta

El objetivo fundamental del presente capítulo será demostrar la utilidad del modelo abstracto obtenido para la implementación de consultas semánticas sobre atributos textuales en bases de datos. Para ello, los contenidos aquí discutidos son eminentemente prácticos, y se centran en describir los tipos de consultas que son posibles realizar por parte del usuario y a describir la herramienta implementada.

En la sección 1 se dan las ideas básicas del tipo de consultas que el usuario pudiera realizar en un sistema que soporte el modelo propuesto. A continuación, a través de ejemplos, se muestran cómo los resultados obtenidos en dichas consultas utilizando los métodos que le fueron definidos al TDA, a partir de las operaciones definidas en el modelo abstracto.

Los elementos de la implementación del Cliente de Consulta son dados en la sección 2. Aquí se define su arquitectura y el lenguaje, tecnologías y SGBD utilizado en su implementación. En la sección 3 se describe la interfaz y funcionalidades que implementa dicha herramienta. Finalmente, se muestran las potencialidades de las que se han dotado a las funciones que trabajan sobre las estructuras definidas en PostgreSQL. Dichas potencialidades permitirán al usuario realizar sus consultas al sistema de una forma flexible y obtener los mejores resultados.

6.1. Consultando el sistema

Una vez que ya se discutió en el capítulo 4 la representación del modelo matemático propuesto en el capítulo 3, en esta sección se dan algunos elementos de los tipos de consultas que se pueden realizar sobre los datos y estructuras almacenadas en una base de datos. Además, se muestran ejemplos concretos de consultas implementadas en PostgreSQL.

Del análisis de las estructuras obtenidas para representar nuestro modelo, y de las características de los atributos de tipo textual, aparecen tres tipos de consultas que, inicialmente, pueden ser solicitadas sobre la base de datos:

1. **Consultas sobre la base de datos completa:** El usuario puede preguntar por toda la información incluida en la base de datos, incluyendo la información que se encuentra almacenada en la estructura-AP global obtenida.

2. **Consultas sobre la estructura-AP**: El usuario puede dar una lista inicial de términos en su consulta, sin tener conocimiento alguno sobre el vocabulario de la estructura-AP. En este caso puede preguntar inicialmente sobre el dominio activo (la estructura-AP), buscando los términos que se acoplan con su lista inicial. Los acoplamientos tenidos en cuenta en la consulta pueden ser fuertes o débiles, basados en la coincidencia entre los términos usados en la consulta por el usuario y los términos almacenados en la estructura-AP. Si los términos introducidos están incluidos completamente en el vocabulario de la estructura-AP, se llevará a cabo un acoplamiento fuerte, de lo contrario, se llevará a cabo un acoplamiento débil. Además, el usuario puede ser ayudado sugiriéndole nuevos términos relacionados con su búsqueda, que se encuentran almacenados en la estructura-AP.

3. **Consultas sobre el TDA particular de cada tupla:** El usuario puede dar una lista inicial de términos preguntando directamente sobre el TDA particular de cada tupla. En este caso, los procedimientos de acoplamiento son invocados directamente sobre dichas estructuras.

> Para este tipo de consulta, debe ser fijado un soporte por el usuario, con el propósito de evitar un grupo numeroso de respuestas.

De lo anterior, se puede deducir, que existirán dos grupos de consultas posibles a realizar por el usuario sobre la base de datos; en el primero, el usuario consulta sobre la base de datos completa, buscando términos haciendo uso de la estructura-AP global. Para ello, verifica la forma en que los términos que busca están acoplados con dicha estructura-AP. En este caso el usuario no conoce nada sobre los datos almacenados; y en este mismo grupo, las también relacionadas sólo con la estructura-AP global. El segundo grupo de consultas corresponde a las que asumen que el usuario tiene algún conocimiento previo sobre los términos almacenados en la base de datos y realiza la búsqueda directamente sobre el TDA particular de cada tupla.

Por otra parte, también es posible crear un grupo de consultas por los diseñadores de la base de datos que, aprovechando algunos de los métodos definidos por las estructuras del modelo, permitan brindarle más información al usuario sobre los datos que se almacenan en la base de datos.

Antes de discutir los ejemplos de estos tres grupos de consultas, es necesario recalcar las ideas discutidas en la sección 4.7, donde se explicó la limitación que presenta PostgreSQL para la implementación de un TDA. Como se comentó anteriormente, PostgreSQL no permite escribir las funciones que implementan los métodos de las estructuras como métodos del TDA. Es por esto que dichos métodos son implementados como funciones independientes en el mismo esquema donde es definido el TDA. Debido a esta limitación, se verá en las consultas de los ejemplos que las funciones que representan los métodos del TDA son invocadas sin la notación *tabla.método*, pues dichas funciones no están asociadas a las tablas que representan el TDA.

En la explicación de todos los ejemplos de consultas que se discutirán, se utilizará *estructuras-AP* como nombre de la tabla que contiene las estructuras-AP de la base de datos. La estructura de dicha tabla es la planteada al final de la sección 4.6.1, cuando se discutió la estrategia para almacenar los metadatos. Concretamente, se asume que la tabla contiene un atributo que se llama *nombre_relacion* refiriéndose a la relación original desde la que se obtuvo la estructura-AP, y otro que se llama *nombre_atributo* refiriéndose al nombre

del atributo de tipo texto corto desde el que se obtuvo dicha estructura-AP. Como valores de estos dos atributos, se utilizará *Postres* como nombre de la relación y *nombre_postres* como nombre del atributo. A continuación, se muestran algunos ejemplos de estos tres tipos de consultas posibles implementadas en PostgreSQL.

6.1.1. Ejemplos del uso de métodos en consultas sobre la estructura-AP

En este grupo de consultas, el usuario introducirá un conjunto de términos para ver si aparecen o no en la estructura-AP. Puede verificar si el grupo de términos que introduce aparecen completamente representados como un conjunto de los que forman la estructura-AP, o si aparecen parcialmente incluidos y con qué índice ocurren estos acoplamientos. A continuación se muestran varios ejemplos de este tipo de consultas, con la respuesta que retorna PostgreSQL tras la ejecución de cada una de ellas. Para la implementación de las consultas se tomó como base el segmento de la estructura-AP que aparece en el retículo de la figura 3.5 del capítulo anterior. Para una mejor comprensión de los ejemplos a continuación aparece la estructura-AP que genera dicho retículo.

$\mathcal{T} = g(\{fresas, nata, nueces, tarta\}, \{chocolate, nata, tarta\})$

Ejemplo 16

Determinar si un conjunto de términos está completamente incluido en la estructura-AP.

Método a utilizar: *acoplamiento_fuerte*.

Consulta:

SELECT *acoplamiento_fuerte* (' nata,nueces ,tarta ') **FROM** *estructuras-AP*
WHERE *nombre_relacion* = ' Postres' AND *nombre_atributo* = ' nombre_postres';

Query OK (0,08 sec) **Return Value:** True

Como se observa en el ejemplo anterior, el usuario introducirá una lista de términos y se invocará la función *acoplamiento_fuerte* sobre la tabla que contiene las estructuras-AP. Particularmente, se ejecuta sobre la estructura-AP que ha sido obtenida de la relación *Postres* y sobre el atributo *nombre_postres*. En el caso de este ejemplo, la consulta demora 0,08 segundos en ejecutarse y al retornar *True*, significa que todos los términos introducidos por el usuario aparecen en un mismo conjunto generador de la estructura-AP. Como se puede observar, todos los términos introducidos en la consulta aparecen en el primero de los dos conjuntos generadores que forman la estructura-AP del ejemplo.

Ejemplo 17

Determinar si un conjunto de términos está parcialmente incluido en la estructura-AP.

Método a utilizar: *acoplamiento_ debil*.

Consulta:

SELECT *acoplamiento_ debil* (' vainilla, almendra ') **FROM** *estructuras-AP*
WHERE *nombre_ relacion* = ' Postres' AND *nombre_ atributo* = ' nombre_postres';

Query OK (0,05 sec) **Return Value:** False

Como se observa en el ejemplo anterior, el usuario introducirá una lista de términos y se invocará la función *acoplamiento_ debil* sobre la tabla que contiene la estructura-AP directamente. En el caso de este ejemplo, la consulta demora 0,05 segundos en ejecutarse y al retornar *False*, significa que ninguno de los términos introducidos aparece en ninguno de los conjuntos generadores que componen la estructura-AP. Se debe recalcar que para el caso de esta función, con que uno de los términos introducidos aparezca en al menos uno de los conjuntos generadores de la estructura-AP, la función retornaría verdadero. Como se observa, en efecto los términos *vainilla* y *almendra* no aparecen en la estructura-AP.

Ejemplo 18

Determinar el grado de acoplamiento total de un conjunto dado con la estructura-AP.

Método a utilizar: *indice_ acoplamiento_ fuerte.*

Consulta:

SELECT *indice_ acoplamiento_ fuerte* (' nata, nueces ') **FROM** *estructuras-AP*
WHERE *nombre_ relacion* = ' Postres' AND *nombre_ atributo* = ' nombre_postres';

Query OK (0,10 sec) **Return Value:** (2/4/2) = 0.25

Para el caso del ejemplo anterior, el usuario introduce un conjunto de términos para la búsqueda y el resultado (tal como plantea la definición de índice acoplamiento fuerte discutida en el capítulo anterior), es la sumatoria del cálculo de la cantidad de términos que se encuentran incluidos en un conjunto generador entre la longitud de dicho generador. En el caso del ejemplo la consulta se obtiene en 0,10 segundos y la expresión del resultado ($2/4/2 = 0{,}25$) significa que, los dos elementos introducidos en la consulta se acoplan con un conjunto generador de cardinalidad 4, y se divide el resultado entre los 2 conjuntos generadores que forman la estructura-AP; de ahí que el índice de acoplamiento fuerte sea de 0.25. Como se plantea en la definición de índice de acoplamiento fuerte, este valor siempre está entre $[0, 1]$, lo que es correcto en el resultado obtenido.

Ejemplo 19

Determinar el grado de acoplamiento parcial de un conjunto dado con la estructura-AP.

Método a utilizar: *indice_ acoplamiento_ debil.*

Consulta:

SELECT *indice_ acoplamiento_ debil* (' nata, nueces ') **FROM** *estructuras-AP*
WHERE *nombre_ relacion* = ' Postres' AND *nombre_ atributo* = ' nombre_postres';

Query OK (0,09 sec) **Return Value:** (0.8/2=0.4)

En el ejemplo anterior, el usuario introduce un conjunto de términos para la búsqueda y el resultado es el número de coincidencias (*incluidas las coincidencias parciales*) que hay de todos los elementos del conjunto introducido,

con todos los conjuntos generadores de la estructura-AP. En este caso la consulta se obtiene en 0,09 segundos y retorna que el conjunto introducido se acopla con un valor de 0.8 con todos los conjuntos generadores, este resultado es dividido entre los 2 conjuntos generadores que componen la estructura-AP y da un índice de acoplamiento débil de 0.4. Como se observa, el resultado es correcto ya que la definición de índice de acoplamiento débil plantea que este valor siempre está entre $[0, 1]$. Además, se obtiene que el índice de acoplamiento débil es mayor que el índice de acoplamiento fuerte para el mismo conjunto ($0{,}4 > 0{,}25$), algo también correcto acorde con dichas definiciones.

Una vez que se han discutido algunos ejemplos de consultas sobre la estructura-AP, a continuación se introducen algunos ejemplos de tipos de consultas que pueden ser formuladas sobre la estructura-AP particular que es obtenida como representación del atributo de tipo texto corto. En este grupo de consultas, se pueden usar los métodos de estas estructuras combinados con otros atributos en la lista de salida de la sentencia *Select,* o además formando parte de una expresión en la cláusula *Where* de dicha sentencia.

6.1.2. Ejemplos del uso de métodos en consultas sobre el TDA inducido de cada tupla

Para la implementación de este grupo de consultas, se tomó como punto de partida el segmento de la tabla de recetas de postres que aparece en la figura 3.3 del capítulo anterior. Para obtener el TDA particular de cada tupla se partió de dicha tabla y se tomó como estructura-AP la que genera el retículo de la figura 3.5 de dicho capítulo. El proceso seguido fue el que se describirá en el próximo capítulo para obtener la estructura-AP particular para cada tupla de la base de datos. En la figura 6.1 aparece el TDA obtenido para cada tupla de la base de datos.

A continuación se introducen algunos ejemplos que utilizan el atributo TDA para brindar información al usuario. En ellas se asume *Postres* como nombre de la tabla en la que se guardan las recetas de postre, y que sólo aparecen en la tabla las tuplas que están en la figura 6.1.

Ejemplo 20

Nombre Postre	TDA	Autor	...	Dificultad	Tiempo de preparación
Fresas con nata y caramelo	{fresas, nata}	María		Alta	35
Tarta de fresas	{tarta, fresas}	Guillermo		Media	25
Tarta de almendras	{tarta}	Elizabet		Media	25
Tarta de chocolate	{tarta, chocolate}	Idelino		Media	28
Fresas con nata	{fresas, nata}	Nérida		Baja	15
Nueces con nata	{nueces, nata}	Claudia		Baja	15
Tarta de nata y chocolate	{tarta, nata, chocolate}	Carlos		Media	30
Tarta de nata	{tarta, nata}	Teresa		Baja	15
Tarta de fresas, nueces y nata	{tarta, fresas, nueces, nata}	Clarisbel		Alta	35
.	.	.	.		.

Figura 6.1: Ejemplos de estructuras-AP particulares en el ejemplo de recetas de postres.

Determinar el autor, el tiempo de preparación y el índice de acoplamiento fuerte del TDA con la estructura-AP.

Método a utilizar: *indice_acoplamiento_fuerte.*

Consulta:

SELECT *Autor*, "*Tiempo de preparacion*" , *indice_acoplamiento_fuerte* (*TDA*) **AS** "*Índice acoplamiento fuerte*" **FROM** *Postres* **WHERE** "*Tiempo de preparacion*" <25;

3 rows fetched (0,06 sec)

Autor	Tiempo de preparacion	Índice acoplamiento fuerte
Nérida	15	(0,5/2 = 0,25)
Claudia	15	(0,5/2 = 0,25)
Teresa	15	(1,16/2 = 0,58)

Antes de comentar el resultado de la consulta del ejemplo anterior, se debe destacar el uso de la función *indice_ acoplamiento_fuerte* en la lista de salida de la sentencia *Select*. En este contexto, PostgreSQL permite el uso de funciones y, en el caso del ejemplo, a dicha función se le pasa como parámetro el atributo TDA; por tanto, la función se calculará sobre dicho atributo para cada tupla de la tabla *Postres*. Con la cláusula *AS* se ha renombrado el nombre del atributo de salida como *Índice acoplamiento fuerte.* Este uso de las funciones que implementan los métodos permiten al usuario combinar otros atributos de la base de datos, con el resultado de operaciones sobre el TDA particular de cada tupla.

Las tres tuplas retornadas por la consulta del ejemplo anterior, son las que cumplen la condición que el atributo *Tiempo de preparacion* sea menor que 25. En ellas se obtienen el autor de la receta, el tiempo que demora prepararla y el índice de acoplamiento fuerte que tiene el valor del TDA para cada tupla con la estructura-AP del ejemplo. La consulta se resuelve en 0.06 segundos. Para el caso de la tercera fila que es la que mayor índice de acoplamiento fuerte posee, se debe a que su TDA está formado por el conjunto $\{tarta, nata\}$ que tiene acoplamiento fuerte con los dos conjuntos generadores que forman la estructura-AP.

Ejemplo 21

Contar la cantidad de tuplas cuyo TDA tiene acoplamiento fuerte con la estructura-AP.

Método a utilizar: *acoplamiento_fuerte.*

Consulta:

SELECT Count(*) AS "*Cantidad*" **FROM** *Postres*
WHERE (*acoplamiento_fuerte* (*TDA*) = TRUE);

1 rows fetched (0,11 sec)

Cantidad 9

Como se observa en el ejemplo anterior, la función que implementa el método *acoplamiento_fuerte* es utilizada en la cláusula *Where* de la sentencia *Select.*

En este contexto PostgreSQL también permite el uso de funciones y, en el caso del ejemplo, a dicha función se le pasa como parámetro el atributo TDA; por tanto la función se calculará sobre dicho atributo para cada tupla de la tabla *Postres*. Con la cláusula *AS* se ha renombrado el resultado de la función *Count* a *Cantidad*. Este uso de las funciones que implementan los métodos permiten al usuario tener una idea global, del grado de acoplamiento total que existe entre el TDA particular de cada tupla y la estructura-AP.

La consulta retorna que las 9 tuplas del ejemplo tienen acoplamiento total con la estructura-AP, lo que indica una correspondencia total entre el valor del TDA particular de cada tupla con la estructura-AP.

A continuación, se introducen un grupo de consultas más generales, que pudieran ser implementadas por el diseñador de la base de datos. Dichas consultas, utilizando algunos de los métodos que fueron definidos anteriormente por las estructuras del modelo, pudieran sugerir términos de búsqueda al usuario, retornar los conjuntos generadores de una estructura-AP, etc.

6.1.3. Ejemplos de consultas posibles por el diseñador

Como se mencionó anteriormente, este grupo de consultas pueden ser implementadas por el diseñador de la base de datos. Las funciones que se invocan en ellas, no son métodos que han aparecido como operaciones del modelo propuesto, aunque algunas de ellas sí están basadas en dichas operaciones. En la definición de los ejemplos se muestran los métodos del modelo que pueden ser utilizados para responder la consulta en caso de que sea posible.

Ejemplo 22

Determinar los conjuntos generadores de la estructura-AP.

Consulta:

SELECT *conjuntos_generadores()* **AS** "*Generadores*" **FROM** *estructuras-AP*
WHERE *nombre_relacion* = ' Postres' AND *nombre_atributo* = ' nombre_postres';

2 rows fetched (0,06 sec)

Generadores
$\{fresas, nata, nueces, tarta\}$
$\{chocolate, nata, tarta\}$

Esta consulta podría ser muy útil para tener una idea clara del contenido de la estructura-AP ya que, al obtener sus conjuntos generadores, se están dando a conocer todos los términos que ella almacena. Con esta función *conjuntos_ generadores()* cualquier usuario final que no conozca nada del contenido de la estructura-AP, se puede familiarizar con su contenido inmediatamente. Para el caso concreto de la consulta realizada, se devuelven los dos conjuntos generadores que forman la estructura-AP del ejemplo.

Ejemplo 23

Determinar el conjunto maximal de la estructura-AP con que se acopla un conjunto dado.

Posibles métodos a utilizar: *indice_ acoplamiento_fuerte* e *indice_ acoplamiento_ debil.*

Consulta:

SELECT *acoplamiento (' chocolate, vainilla')* **AS** Acoplamiento **FROM** *estructuras-AP*
WHERE *nombre_ relacion* = ' Postres' AND *nombre_ atributo* = ' nombre_postres';

1 rows fetched (0,11 sec)

Acoplamiento
Índice de acoplamiento débil: (0,33/2 = 0,16)
Conjunto máximo con que se acopla:$\{chocolate, nata, tarta\}$

La consulta del ejemplo anterior podría ser muy útil para tener una idea sobre el tipo de acoplamiento que tienen un conjunto de términos con la estructura-AP. Además se le puede sugerir al usuario el conjunto maximal con que se acoplaron los términos que él introdujo en su consulta, de manera

que él pueda ver la mayor cantidad de términos posibles que tienen relación con los que introduce en su búsqueda.

Para el caso del ejemplo, el conjunto introducido $\{chocolate, vainilla\}$ tiene acoplamiento débil con la estructura-AP, ya que el término *vainilla* no aparece en dicha estructura y el conjunto máximo con que se acopla es el conjunto generador que contiene el término *chocolate*.

En este caso como se puede observar, se ha utilizado el método *acoplamiento*, que se encarga de determinar si existe un acoplamiento fuerte o débil del término introducido con la estructura-AP global, y retorna el índice de acoplamiento calculado, con el conjunto máximo con que se acopla. En el caso particular de este ejemplo se ha utilizado la variante del cálculo del índice de acoplamiento débil por el promedio.

Ejemplo 24

Sugerir un conjunto de términos para refinar una búsqueda dado un conjunto de términos iniciales.

Posibles métodos a utilizar: *indice_ acoplamiento_ fuerte* e *indice_ acoplamiento_ debil.*

Consulta:

SELECT *Sugerir (' nueces')* **AS** *Sugerencia* **FROM** *estructuras-AP*
WHERE *nombre_ relacion* = ' Postres' AND *nombre_ atributo* = ' nombre_postres';

1 rows fetched (0,11 sec)

Sugerencia
Índice de acoplamiento fuerte: (0,25/2 = 0,125) Sugerencia:$\{fresas, nata, nueces, tarta\}$

La consulta del ejemplo anterior podría ser muy útil para sugerirle al usuario términos que él no conoce que se encuentran almacenados y que guardan relación con su búsqueda inicial. A diferencia del método *acoplamiento* que

sólo devuelve el conjunto maximal con que se acoplan los términos introducidos, este método sugeriría todos los conjuntos con el que los términos introducidos tuviesen algún grado de acoplamiento.

Para el caso del ejemplo, el conjunto introducido $\{nueces\}$ tiene acoplamiento fuerte con la estructura-AP, ya que aparece en un conjunto generador completamente contenido. Precisamente es ese generador el que sugiere la función *Sugerir* en su salida.

Una vez discutidos los tipos de consultas que se le pueden realizar al modelo propuesto. En la siguiente sección, se discutirán los detalles del Cliente de Consulta implementado, con vistas a que el usuario pueda expresar dichos tipos de consultas. Para ello se verá su arquitectura, interfaz y funcionalidades que implementa.

6.2. Implementación del Cliente de Consulta

En esta sección se dan los elementos fundamentales de la implementación del cliente de consulta. Para ello se comienza discutiendo sobre su arquitectura y posteriormente se discutirá sobre la elección realizada del SGBD a utilizar en la implementación del modelo obtenido. La herramienta ha sido implementada en lenguaje Java, tomando como entorno de desarrollo integrado (del Inglés IDE) el Eclipse v3.3 (The-Eclipse-Foundation, 2008) y como gestor de Base de Datos PostgreSQL v8.2.1 (PostgreSQL-Global-Devel-Group, 2008a).

6.2.1. Arquitectura del Cliente de Consulta

La arquitectura del Cliente de Consulta ha sido implementada tomando como base la arquitectura propuesta por Microsoft para el desarrollo de aplicaciones Cliente/Servidor en varias capas (Microsoft-Corp., 2006). Concretamente, esta arquitectura propone un grupo de capas lógicas en las que se puede dividir la aplicación cliente, para un mejor funcionamiento y comunicación entre todos los componentes de un software. En la figura 6.2 se muestra la arquitectura del Cliente de Consulta, diseñada bajo esta perspectiva.

Como se puede observar en la figura, dicha arquitectura se ha descompuesto en las siguientes capas lógicas:

1. **Interfaces:** Esta es la capa de la aplicación que contiene la interfaz de usuario que se le muestra para que realice sus consultas a la base de datos. La interfaz de usuario se implementa utilizando los componentes y controles que permita procesar y dar formato a los datos de los usuarios, así como adquirir y validar los datos entrantes procedentes de éstos. En esta capa se han definido las diferentes vistas (*las que se discuten en detalle en la siguiente sección*) que se le muestran al usuario para que cree y manipule las consultas *Select*.

2. **Controladores y Servicios:** Esta capa es la que se encarga de facilitar la sincronización y organización de las interacciones con el usuario. En ella se encapsulan los servicios que están disponibles para cada clase del modelo. También contiene las clases controladoras que se encargan del intercambio de mensajes entre la interfaz de usuario y las clases del modelo. De este modo, el flujo del proceso y la lógica de administración del estado de las clases no se incluye en el código de los elementos de la interfaz de usuario. Dichos mensajes no modifican el modelo directamente, sino que son atendidas por los controladores que son los encargados de modificar el modelo, ajustándonos de esta forma al patrón MVC (Modelo Vista Controlador).

3. **Modelo:** En esta capa se incluyen las clases que componen el modelo abstracto del Cliente de Consulta. Dicha capa aparece dividida en tres submodelos diferentes: *modelo de la base de datos*, *modelo de consulta* y *modelo de la representación visual*. El primero, como su nombre indica, contiene las clases del modelo que se encuentran en la base de datos (urgencias, intervenciones, provincias, sexos, etc). El *modelo de consulta* contiene las clases necesarias para crear una consulta *Select* (se corresponde con las diferentes cláusulas de la sentencia Select: *Where*, *Group By*, *Order By*, etc). Finalmente, el *modelo visual* contiene las

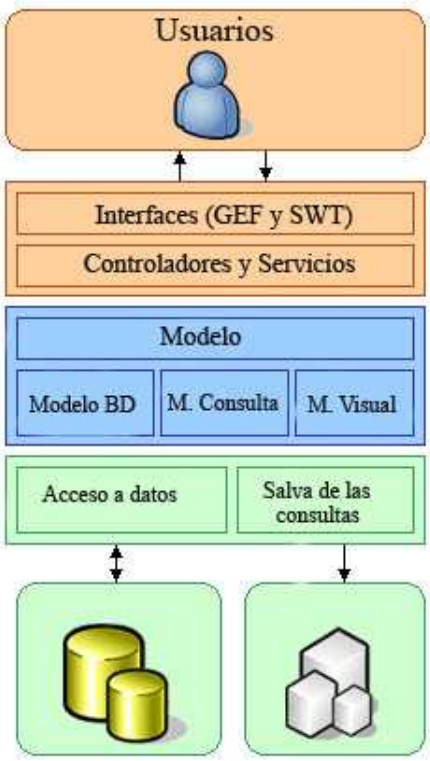


Figura 6.2: Arquitectura del Cliente de Consulta.

clases correspondiente a la creación visual de la consulta *Select* (se corresponde con los componentes visuales para establecer una tabla en el Editor de Consulta, una relación entre dos tablas, etc).

4. **Acceso a datos:** Contiene las clases necesarias para obtener acceso a los datos, mediante una capa independiente de componentes lógicos de acceso a datos. De esta forma, se centraliza la funcionalidad de acceso a datos y se facilita la configuración y el mantenimiento de la misma. Dicha capa se subdivide en el *acceso a datos* propiamente que se encuentra en la base mediante un driver nativo JDBC (Java Database Connectivity) que se encargará de la comunicación con la base de datos. La subdivisión *salva de las consultas* contiene las clases que se encargan de la escritura y lectura hacia y desde el fichero de las consultas que

cree el usuario.

Con la correcta comunicación entre todas estas capas de software, se le garantiza al usuario que puede expresar de una forma muy intuitiva sus consultas sobre la base de datos y recibir sus respuestas. De forma general, todas estas capas lógicas se comunican entre sí y el sistema garantiza la seguridad de los datos. Se implementa también la seguridad a nivel de base de datos requiriendo la autenticación por parte del usuario a la base de datos.

Como se comentó anteriormente, un aspecto importante en la implementación realizada, fue la elección del SGBD a utilizar. En la siguiente sección se dan algunos elementos que justifican la elección realizada en este sentido.

6.2.2. Elección del SGBD de libre disposición para la implementación del Cliente de Consulta

Para la implementación tanto del modelo propuesto como de este Cliente de Consulta, un aspecto importante a tener en cuenta fue la elección del SGBD de libre disposición a utilizar. Entre los gestores de libre disposición se pueden citar: SAPDB, MySQL, db4o, PostgreSQL, entre otros. Todos ellos se encuentran diseñados y optimizados para un ambientes de trabajo determinados, e implementan modelos de datos específicos.

Dado los requerimientos de este trabajo, en que como se comentó se realizan extensiones en el almacenamiento de los datos rompiendo la 1FN, un requerimiento importante es que el SGBD elegido implemente el modelo de datos Relacional Orientado a Objetos, que es el que introduce esta característica. Además, dado que la forma elegida para almacenar las estructuras resultantes del modelo se basa en colección, será necesario el uso de un gestor que implemente el tipo Array. Otro de los requerimientos de la herramienta a utilizar, es que permita la definición de extensiones en la base de datos, para implementar las funciones que responden a las operaciones del modelo propuesto.

De los gestores mencionados anteriormente, el único que cumple con todos estos requisitos es el PostgreSQL. Su modelo de datos es Relacional Orientado

a Objetos que implementa el tipo Array, herencia entre tablas, y permite la definición de extensiones mediante la programación de funciones. Dicho gestor permite darle una mayor robustez a las aplicaciones que lo utilizan, explotando sus potencialidades de programación en el servidor de bases de datos y utilizando su potente lenguaje de consulta PL/pgSQL y/o utilizando algunas características de la orientación a objetos.

El mayor competidor por su popularidad y características de PostgreSQL es MySQL. Es un gestor más sencillo y su funcionamiento es más veloz que en el caso de PostgreSQL. El motivo fundamental por el que fue descartado, es porque no implementa el modelo de datos Relacional Orientado a Objetos, y dentro de dicho modelo, concretamente el tipo Array. Este tipo de dato, como se dijo anteriormente, es un requisito esencial en la implementación realizada.

PostgreSQL, además, cuenta con una amplia documentación y listas de discusión. Está concebido sin límite de usuarios y soportado por multitud de plataformas, entre ellas Windows y Linux, lo que permite de forma más sencilla extender el Cliente de Consulta a otras plataformas de trabajo, también teniendo en cuenta que está implementado en lenguaje Java, que tiene la misma característica de ser multiplataforma.

En la siguiente sección, se darán los elementos fundamentales de la interfaz de usuario y las funcionalidades que implementa el Cliente de Consulta implementado. Para ello se mostrarán las diferentes vistas que componen dicha interfaz, y mediante ellas se explicarán las funcionalidades que implementan.

6.3. Interfaz y funcionalidades del Cliente de Consulta

La interfaz del Cliente de Consulta implementado, ha sido diseñada de forma tal que se le pueda dar las mayores posibilidades al usuario para confeccionar su consulta. Dado que se utiliza el modelo de datos R.O.O, la realización de dicha consulta a la base de datos se expresará a partir de una sentencia *Select* escrita en este caso en lenguaje SQL:99. Debido a esto, se ha diseña-

do la interfaz para dar la mayor asistencia posible al usuario a la hora de confeccionar su sentencia Select. De esta forma, el proceso de creación de la consulta se convierte en una simple interacción visual con las diferentes cláusulas que componen la sentencia *Select*, cláusulas que son agrupadas con la misma lógica funcional que poseen dentro de dicha sentencia.

Como se comentó anteriormente, la implementación del Cliente de Consulta se realizó en lenguaje Java, tomando como IDE de desarrollo el Eclipse. Como principal herramienta empleada en la creación de la aplicación se utilizó el proyecto GEF (del Inglés Grafical Editing Framework) de la comunidad Eclipse, además del mismo Eclipse. Para el desarrollo de la interfaz visual se utilizó la biblioteca SWT (del Inglés Standard Widged Toolkit) que incorpora Eclipse, utilizando tecnología RCP(del Inglés Rich Client Platform). Esto posibilita la creación de futuros *plug in* para perfeccionar la herramienta sin necesidad de más cambios que generar el "*producto*", que sería el ejecutable de la aplicación final.

En la figura 6.3 se nuestra la interfaz principal del Cliente de Consulta, diseñada con las herramientas y bajo los principios antes mencionados. Se observa el ambiente de trabajo que proporciona el Cliente de Consulta. Como cualquier aplicación, posee una barra de título donde aparece el nombre de la aplicación (*Query Maker*) seguido del nombre de la base de datos que se tiene abierta. A continuación, aparece una barra de herramienta que da acceso a través de botones de rápido acceso a las funcionalidades principales del sistema y después, aparecen un grupo de vistas que permiten el trabajo con el sistema. Finalmente, aparece la barra de estado donde aparecerán mensajes contextuales según la acción que esté llevando a cabo el usuario.

Todas las pantallas que brinda el sistema mantienen una misma línea de diseño y se caracterizan por un estilo sencillo e interactivo. Se utilizan componentes visuales que le facilitan al usuario su interacción con el sistema, además que minimizan la ocurrencia de errores.

Las funcionalidades principales que implementa el sistema están soportadas por las seis vistas que componen su interfaz. A continuación se brinda una

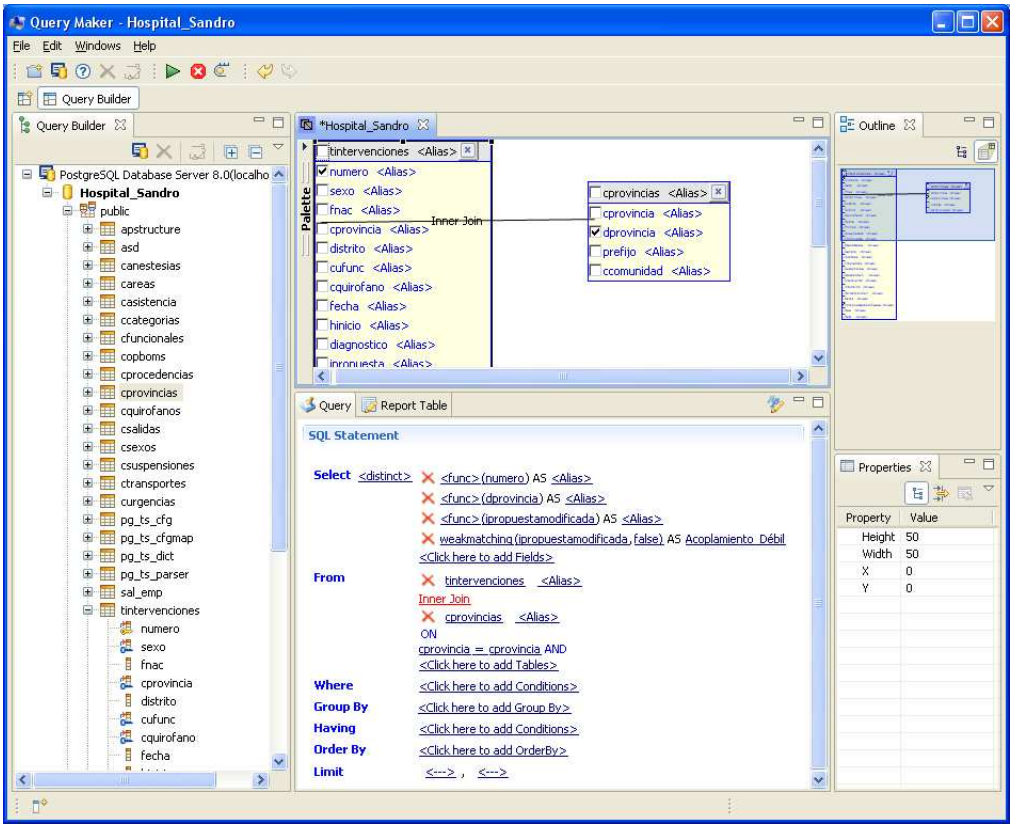


Figura 6.3: Interfaz principal del Cliente de Consulta.

breve descripción de cada una de las vistas, que da una idea más clara de las funcionalidades del sistema.

6.3.1. Vista de la base de datos

Esta es la vista base de todo el trabajo con el sistema. Responde a las funcionalidades referentes a la autenticación del usuario con el servidor de base de datos y a utilizar y mostrar la base de datos con que se conecta el usuario dentro del servidor seleccionado. También se encarga de mostrar todos los esquemas de la base de datos y dentro de un esquema seleccionado, mostrar las tablas y atributos que los componen.

Para más claridad y una mejor organización de la información, esta vista ha

sido implementada como un árbol, donde la conexión es el nivel más externo y los distintos elementos se van adicionando sucesivamente hasta llegar al nivel mas interno. En la figura 6.4 se muestra un ejemplo de una *vista de la base de datos*, obtenida de la base de datos médica sobre la que se implementan los ejemplos de este capítulo.

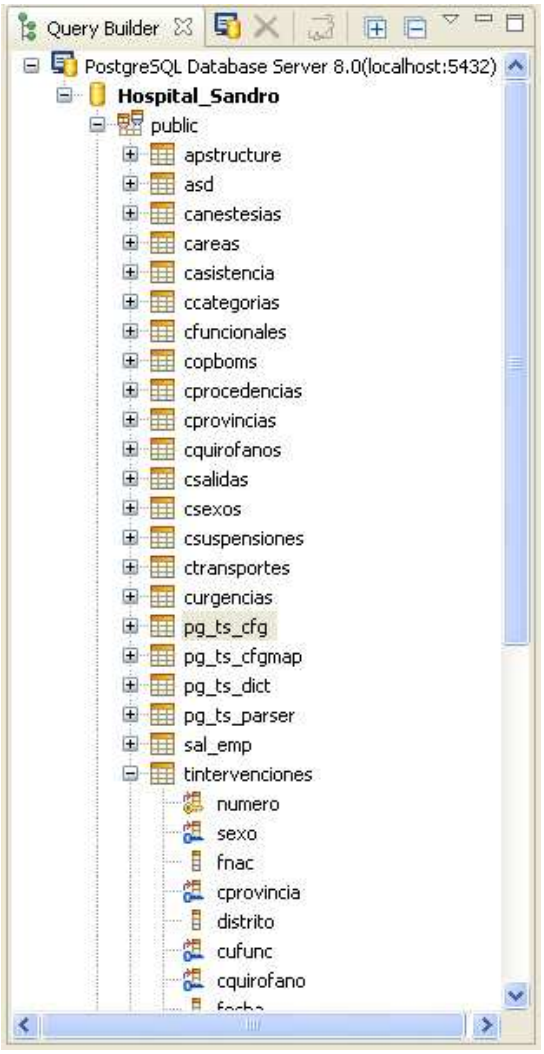


Figura 6.4: *Vista de la base de datos* del Cliente de Consulta.

Como se observa en la figura, el nivel más externo del árbol se corresponde con el nombre del servidor de base de datos. En este caso, se ha realizado la conexión con un servidor PostgreSQL local. A continuación, en el segundo nivel del árbol aparece la base de datos con la que se ha establecido la conexión dentro del servidor PostgreSQL. En el tercer nivel del árbol, aparecen los esquemas que se tienen definidos dentro de la base de datos; en el caso del ejemplo aparece el esquema *public* de PostgreSQL. En el cuarto nivel, aparecen las tablas que están dentro del esquema seleccionado. Finalmente, en el quito nivel, como se muestra para el caso de la tabla *TIntervenciones*, están los atributos de dichas tablas.

Con el objetivo de asistir al usuario a la hora de crear de manera visual las relaciones entre las tablas en la vista *editor gráfico de consultas*, en esta vista se le señalan al usuario con un icono específico la llave primaria de cada tabla (*llave de color amarillo*). Los atributos que son llave extranjera en dicha tabla se señalan también con otro icono específico (llave de color azul). De esta forma, el usuario tiene información adicional sobre la relación que tiene una tabla en cuestión con el resto de las tablas de la base de datos. Es de destacar que esta funcionalidad no la incorpora ningún cliente de consulta para PostgreSQL.

Esta *vista de la base de datos* trabaja estrechamente relacionada con la *vista del editor gráfico de consulta*, que se discutirá a continuación. La vista de la base de datos permite que el usuario marque una tabla determinada y la arrastre hasta soltarla (*drag and drop*) en el *editor gráfico de consultas*. Esta funcionalidad hace que el usuario de forma visual e intuitiva pueda ir seleccionando las tablas que formarán parte de su consulta a la base de datos.

6.3.2. Vista del editor gráfico de consultas

Además de manejar las tablas que formaran parte de la consulta, en esta vista se puede seleccionar los atributos que estarán en la lista de salida de la sentencia Select y construir relaciones entre las tablas. A medida que se van realizando todas estas funcionalidades, el sistema las va sincronizando con la *vista de sentencia Select* en la que se va construyendo la sintaxis de la sentencia *Select* resultante.

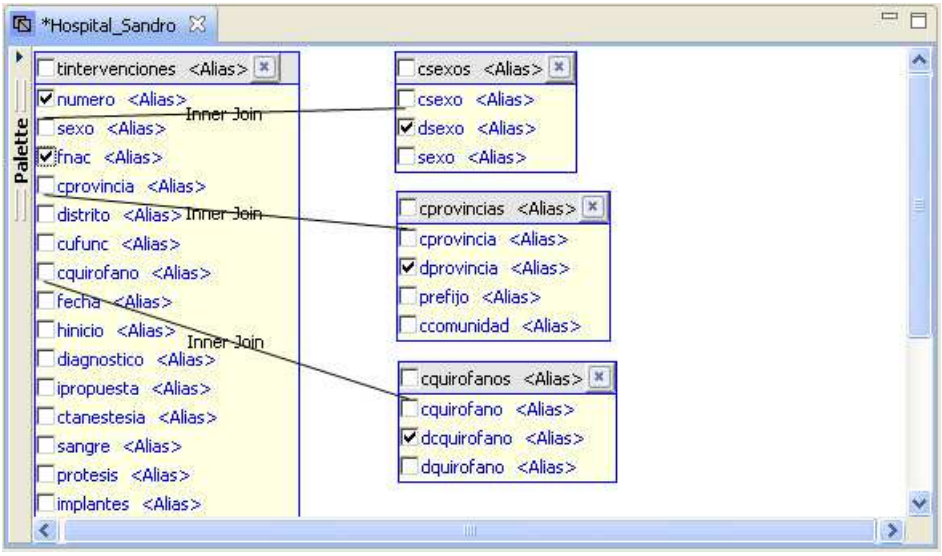


Figura 6.5: Vista del *editor gráfico de consultas* del Cliente de Consulta.

En la figura 6.5 se muestra un ejemplo de esta vista. Como se observa en la figura, esta vista será en la que el usuario puede depositar las tablas que formarán parte de su consulta de forma visual. Aquí tiene la posibilidad de seleccionar los atributos que desea que estén en la lista de salida de la sentencia *Select*. También puede establecer las relaciones entre las tablas. Para ello, esta vista cuenta con una paleta (*Palette*), donde vienen los tres tipos de punteros que son permitidos en dicha vista. Con el puntero *Connection* podrá construir una relación entre dos tablas, seleccionando los dos atributos por los que desea formar la relación. También el puntero *select* permite la selección de objetos de forma individual, mientras que el puntero *marquee* permite la selección de múltiples objetos a la vez.

Como se comentó anteriormente, todos los cambios que se van haciendo o deshaciendo en esta vista, se van reflejando en la *vista de sentencia Select*. Para el caso de las herramientas que aparecen hoy día para la creación de consultas en PostgreSQL, ninguna de estas dos vistas está disponible, si no que cuando el usuario quiere especificar una consulta, lo tiene que hacer construyendo la sentencia Select sin ningún tipo de asistencia, como ocurre con la herramienta gráfica oficial que se distribuye en el paquete de insta-

lación de PostgreSQL, la herramienta PGAdmin III. Esto sin duda alguna, hace que se pueda afirmar que el Cliente de Consulta implementado supera a herramientas similares dentro del mundo del software libre e incluso algunas herramientas dentro del software comercial como el EMS PostgreSQL Manager Pro (EMS-DMSolutions-Inc, 2008).

6.3.3. Vista de sentencia Select

Como su nombre indica, en esta vista se puede ver y modificar la sentencia *Select* que se está construyendo. Esta vista permite una mayor cantidad de opciones que la vista del *editor gráfico de consultas,* pero requiere por parte del usuario un dominio de la sintaxis de la sentencia *Select.*

En la figura 6.6 se muestra un ejemplo de dicha vista. Como se muestra en la figura, básicamente la vista está compuesta por las cláusulas fundamentales que componen la sentencia Select. Para todas ellas, el sistema incluye un texto indicativo que tiene el funcionamiento de un botón, que cuando el usuario lo pulse, en cada caso, le permitirá agregar los diferentes elementos de la sentencia. Una vez que un elemento es adicionado, el sistema incluye un icono (*cruz de color rojo*), para permitirle al usuario eliminar el elemento introducido.

Entre otra funcionalidades, esta vista le permite al usuario las siguientes:

- Definir la lista de atributos que desea visualizar en la consulta.
- Renombrar los atributos en la vistas de resultados a través de la cláusula *AS.*
- Aplicarle a un atributo tanto las funciones agregadas tradicionales del lenguaje SQL (*Max*, *Min*, *Sum*, etc.) como las funciones que implementan las operaciones del modelo propuesto (*acoplamiento_fuerte*, *acoplamiento_ debil*, etc.).
- Especificar condiciones en la cláusula *Where* utilizando varios operadores, entre ellos el *Like* y el *Similar To.*

Figura 6.6: *Vista de sentencia Select* del Cliente de Consulta.

- También permite hacer agrupamientos (*Group By*) , ordenamientos (*Order By*), entre otras.

Una vez que el usuario construye su consulta, la ejecuta y los resultados de la misma se muestran el la *vista de datos*.

6.3.4. Vista de Datos

Es la vista que se utiliza para mostrar los datos que se obtienen al ejecutar una consulta determinada.

Un ejemplo de esta vista se muestra en la figura 6.7. Como se observa en la figura, los datos que muestra el ejemplo son los que obtendría el usuario que ejecuta la consulta que aparece diseñada en la figura 6.6. En la figura 6.7 también se puede observar que esta vista contiene otra etiqueta (*Message*). En dicha etiqueta, le muestra al usuario información estadística del tiempo que tardó la consulta que ejecutó y la cantidad de tuplas que retornó dicha consulta. También se le muestra la sentencia Select elaborada de forma

asistida por el sistema. Un ejemplo de esta información que se le brinda al usuario aparece en la figura 6.8.

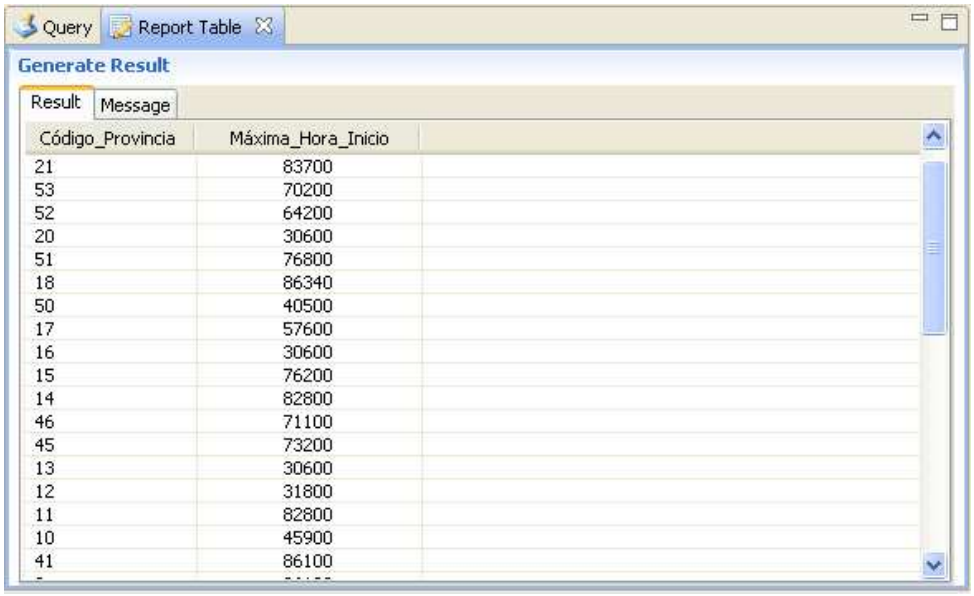


Figura 6.7: *Vista de datos* del Cliente de Consulta.

6.3.5. Vista resumen (outline)

Esta vista está diseñada con la idea de mostrar un resumen de los elementos visuales que aparecen en la *vista editor gráfico de consulta*. Contiene dos botones en el extremo superior derecho. El primero muestra un resumen de las tablas que han sido agregadas. El segundo, muestra un resumen visual de las tablas y relaciones que han sido agregadas en dicha vista y permite desplazarse hacia arriba y hacia abajo, de forma sincronizada con la vista antes mencionada.

Esta funcionalidad permite al usuario acceder de forma más rápida y fácil a una determinada tabla que no se encuentre visible en la porción actual que se esté mostrando del *editor gráfico de consultas*. Un ejemplo de esta vista con este segundo botón activado aparece en la figura 6.9 (a).

Figura 6.8: Mensaje de la *vista de datos* del Cliente de Consulta.

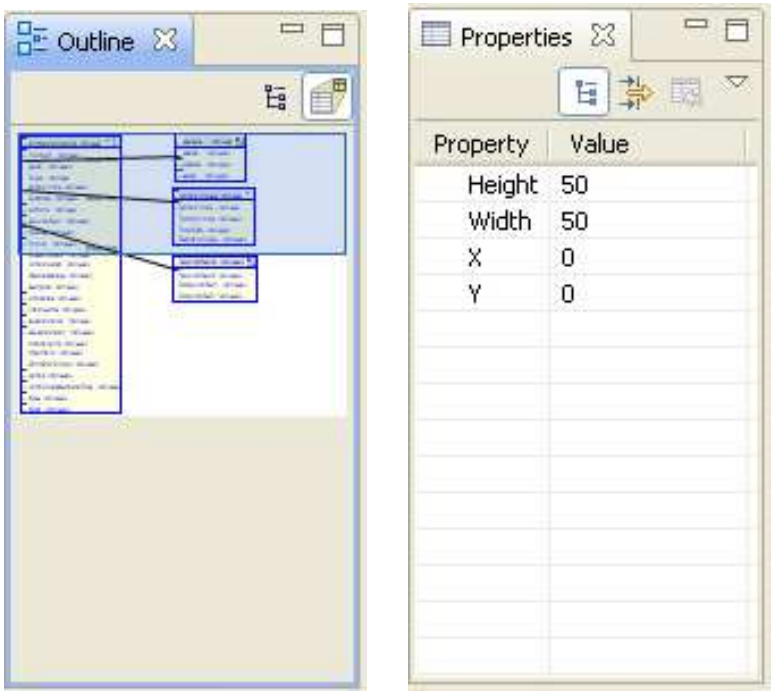
6.3.6. Vista de propiedades

Es la vista que contiene las propiedades fundamentales del objeto que esté seleccionado en ese momento en la vista editor gráfico de consulta. Un ejemplo de esta vista aparece en la figura 6.9 (b).

Una parte importante en el desarrollo de un determinado sistema lo constituye la validación de los datos que son introducidos en el mismo y el análisis de los errores que puedan ser introducidos por el usuario. Tener esto en cuenta y solucionarlo contribuye en gran medida a la calidad del producto terminado. En el próximo apartado se describe brevemente cómo se realizó el tratamiento de errores en el Cliente de Consulta aquí propuesto.

6.3.7. Tratamiento de errores

Como se comentó anteriormente, la aplicación fue implementada en lenguaje Java. Este lenguaje tiene el tratamiento de excepciones como una de sus principales fortalezas. Es tan así esta afirmación, que las clases no compilan de forma correcta hasta tanto se especifique el manejos de excepciones a determinados bloques de códigos, que el propio compilador detecta como posibles a provocar errores. Varias son las formas que implementa dicho lenguaje para



(a) Vista resumen (b) Vista de propiedades

Figura 6.9: Vistas *resumen* y *propiedades* del Cliente de Consulta

el manejos de excepciones; entre ellas se pueden citar: interfaces con métodos que pueden lanzar excepciones, métodos que capturan las excepciones, métodos que propagan las excepciones, entre otros.

Basados en esta posibilidad que da el lenguaje, el tratamiento de errores ha quedado cubierto de forma exhaustiva, quedando el sistema siempre en un estado consistente ante cualquier error producido, e informando al usuario la causa del error producido. Al cometerse un error, la aplicación lo detecta y trata de corregirlo. Aunque el mejor remedio para los errores es la prevención, evidentemente los errores ocurrirán incluso con los usuarios de más habilidad y experiencia. Es por esto que si no se pueden evitar los errores, al menos hay que tratar de minimizar sus consecuencias.

En el caso del Cliente de Consulta se permite la recuperación ante un error producido, y se ha realizado el diseño de la interfaz, de forma tal que el usuario pueda cometer los menos errores posible. Si se observa la figura 6.6

se verá el mejor ejemplo. Aquí cada cláusula de la sentencia Select se ha puesto en un orden lógico y admisible por el SGBD; de esta forma el usuario no podría especificar por ejemplo la cláusula *Where* antes que la cláusula *From*, ni los atributos de la lista de salida separados por otro separador que no sea la coma, pues el sistema las agrega automáticamente, sin darle la posibilidad al usuario a que introduzca dicho separador.

De igual forma, las funciones agregadas (*Max*, *Min*, *Avg*, etc.) ya aparecen escritas y el usuario sólo las tiene que seleccionar; de esta forma se evita la posibilidad de que se equivoque al escribirlas. De forma análoga ocurre con las funciones que realizan extensiones al sistema para implementar las operaciones del modelo propuesto. También ocurre lo mismo con los operadores a la hora de construir expresiones, por ejemplo en la cláusula *Where* y en todos los casos con la propia sintaxis de las cláusulas de la sentencia Select (*From*, *Where*, *Group By*, *Order By*, etc.).

Por todos estos aspectos señalados, podemos decir que el diseño y programación del Cliente de Consulta le garantizan al usuario un entorno de trabajo integrado donde, basado en una interfaz sencilla y amigable, puede expresar sus consultas a la base de datos. También puede trabajar de forma asistida y cómoda y sin la preocupación de que un error provocará el cierre de la aplicación.

En la próxima sección se muestran algunas de las características con que se han dotado las extensiones implementadas en PostgreSQL. Dichas características hacen que el usuario pueda expresar sus consultas de una forma más flexible, sin necesidad de cuidar si expresa sus términos en letras mayúsculas o minúsculas, si deja espacios intermedios, iniciales o finales en la cadena de términos que está buscando, etc. También se implementa la posibilidad de encontrar los términos que introduce el usuario sin importar el orden en que los escriba, etc.

6.4. Características de las funciones de consulta implementadas en PostgreSQL

A continuación se muestran, a través de ejemplos concretos, algunas de las posibilidades antes mencionadas para las funciones que implementan las operaciones del modelo propuesto. Hay que destacar que la mayoría de estas características no están presentes en los operadores tradicionales del trabajo con atributos textuales en bases de datos. De esta forma, como se verá, los resultados obtenidos por el usuario respecto a la forma tradicional en que hace su búsqueda son significativamente superiores.

Para ganar en comprensión en las ideas presentadas en esta sección, se utiliza para todos los ejemplos la misma consulta. En cada caso sólo se variará lo que interesa para explicar las características antes mencionadas. Para la implementación de estos ejemplos se utilizó el conjunto experimental *Conjunto2* que fue definido en el capítulo anterior.

- ***Ejemplos que muestran que las funciones admiten espacios en blanco en cualquier posición, sin alterar el resultado de las mismas.***

Esta característica está implementada con la idea de que el usuario no tenga que cuidarse de no incluir algún espacio en blanco adicional, en la frase que introduce en su búsqueda. Esto le permitirá obtener el mismo resultado, si algún espacio fuera introducido de forma intencional o no. En los siguientes ejemplos se clarifican estas ideas.

1. **Select** *count (*)* **As** *Cantidad* **From** *TIntervenciones* **Where** *acoplamiento_fuerteTDA* (IPropuestaTDA, ' abdomino, amputacion ,perianal ', TRUE) = TRUE ;

Query OK (0,7 sec)

Cantidad: 341

En el caso de la consulta del ejemplo (1) se retorna la cantidad de tuplas que cumplen la condición que el atributo *IPropuestaTDA* (se refiere al TDA asociado al *Comjunto2*) tiene un acoplamiento fuerte con la frase ' *abdomino, amputacion ,perineal* '. Para hacer esta verificación se utiliza el método *acoplamiento_fuerteTDA* que recibe tres parámetros. El primero es el nombre del TDA sobre el que se va a buscar las frase que aparece como segundo parámetro. El tercer y último parámetro es booleano y significa si dentro de la función se realiza o no el proceso de limpieza descrito en capítulos anteriores.

Como se observa, en la frase que se busca se han dejado espacios en blanco antes del término *abdomino* y después de *perianal*. La implementación realizada garantiza que el usuario puede dejar espacios en blanco en cualquier posición, antes o después de cualquier término que se introduce en la búsqueda sin que se afecte el resultado. Otro ejemplo de esta situación aparece en el ejemplo (2).

2. **Select** *count (*)* **As** *Cantidad* **From** *TIntervenciones* **Where** *acoplamiento_fuerteTDA* (IPropuestaTDA, ' abdomino, amputacion , perianal ', TRUE) = TRUE ;

Query OK (0,7 sec)

Cantidad: 341

Como se observa en el ejemplo (2), en la misma consulta del ejemplo (1) se han dejado espacios intermedios en varias otras posiciones de la misma cadena que se busca y el resultado es exactamente el mismo. En ambos ejemplos se localizan 341 personas a las que se les ha propuesto la intervención '*amputacion abdomino perianal*'(que sería el significado correcto de esta frase). Se debe recordar en este punto que por una cuestión de eficiencia, la implementación del sistema, almacena los términos en orden alfabético para optimizar los algoritmos que se implementan.

Si se tratara de obtener este resultado de la forma tradicional en que se procesan los atributos textuales en bases de datos utilizando el operador *Like*, habría que escribir una consulta como la del ejemplo (3).

3. **Select** *count (*)* **As** *Cantidad* **From** *TIntervenciones* **Where** *ipropuesta* Like '%CURA DE ABSCESO PERIANAL%';

Query OK (0,2 sec)

Cantidad: 21

En una consulta como la del ejemplo (3), si el usuario introduce al menos un espacio en blanco en cualquier posición de la cadena de caracteres que está buscando, la consulta retornaría cero tuplas que cumplan esa condición. Esto demuestra que el operador *Like* es sensible a los espacios en blanco en el patrón de búsqueda que se la pasa. En este sentido, la solución aquí presentada representa una mejora significativa, ya que como se demostró anteriormente, el usuario puede colocar espacios al inicio o el final de cualquiera de los términos sin que se afecte el resultado final. Esta característica de las funciones implementadas ayudan al usuario a la hora de escribir sus consultas, ya que si por error de teclado introdujera un espacio de más, no repercute en su búsqueda como sí ocurre en la forma tradicional.

A continuación se muestra otra de las características incorporadas en las funciones que implementan las operaciones del modelo propuesto. En este caso se verá que no importa el orden en que se introducen los términos a la hora de realizar una consulta determinada.

- ***Ejemplos que muestran que las funciones admiten los términos sobre los que se realiza la búsqueda, en cualquier posición sin alterar el resultado de las mismas.***

En el caso de esta funcionalidad, la idea es la misma que la anterior, el usuario puede especificar los términos que desea buscar sin importar el orden en que los escriba. Esto resulta de vital importancia, pues en muchas ocasiones, el usuario no sabe el orden en el que son escritos en la base de datos. De la forma tradicional, si estos valores no son introducidos en el orden exacto en

el que aparecen en la base de datos, la búsqueda no retornará ningún valor. En el caso de nuestra solución, como se verá en los ejemplos que se presentan a continuación, este orden no interesa y el resultado es el mismo.

4. **Select** *count (*)* **As** *Cantidad* **From** *TIntervenciones* **Where** *acoplamiento_fuerteTDA* (IPropuestaTDA, 'perianal, abdomino, amputacion', TRUE) = TRUE ;

Query OK (0,73 sec)

Cantidad: 341

5. **Select** *count (*)* **As** *Cantidad* **From** *TIntervenciones* **Where** *acoplamiento_fuerteTDA* (IPropuestaTDA, 'perianal, amputacion, abdomino', TRUE) = TRUE ;

Query OK (0,75 sec)

Cantidad: 341

Como se puede observar en los ejemplos (4) y (5), los términos de la consulta han sido introducidos en diferentes posiciones y el resultado obtenido es el mismo, con una variación despreciable del tiempo de respuesta. Concretamente, los términos *abdomino* y *amputacion* han sido intercambiados entre una consulta y la otra. Cualquier otro cambio de este tipo, produce el mismo resultado.

Tratando de realizar esta misma operación en una consulta similar utilizando el operador *Like*, sería construir una consulta ídem a la del ejemplo (3). Si sobre esa consulta se realizara una alteración en el orden en el que aparecen expresados sus términos, no se recuperaría ninguna tupla, tal como se muestra en el ejemplo (6).

6. **Select** *count (*)* **As** *Cantidad* **From** *TIntervenciones* **Where** *ipropuesta* Like '%CURA DE PERIANAL ABSCESO%';

Query OK (0,09 sec)

Cantidad: 0

Como se había comentado, en el ejemplo (6) se han modificado el orden de los términos *PERIANAL* y *ABSCESO* y no se obtiene ninguna tupla como resultado de ejecutar la consulta. Esta característica también muestra la mejoría que en este sentido supone la solución aquí presentada, con respecto a la forma tradicional en la que se procesan atributos textuales en bases de datos.

Finalmente, la siguiente característica está enfocada al uso de las mayúsculas y minúsculas indistintamente. Este aspecto también en ocasiones puede producir resultados indeseados por el usuario.

- ***Ejemplos que muestran que las funciones admiten mayúsculas y minúsculas indistintamente sin alterar el resultado de las mismas.***

Normalmente, es preferible en el ámbito de búsqueda de términos en textos, que el uso de las mayúsculas y minúsculas pueda utilizarse de forma insensitiva, o sea, no distinguir entre mayúsculas y minúsculas. Este tipo de búsqueda se centra en la sintaxis en sí de los términos introducidos sin verificar si se encuentran escritos de una u otra forma. Bajo este principio se han diseñado las funciones que implementan las operaciones del modelo propuesto. En los siguientes ejemplos se muestran las diferencias que existen entre dichas funciones y la forma tradicional de hacer búsquedas de textos en bases de datos, utilizando el operador *Like*.

7. **Select** *count (*)* **As** *Cantidad* **From** *TIntervenciones* **Where** *acoplamiento_fuerteTDA* (IPropuestaTDA, ' ABDOMINO,amputacion ,perianal ', TRUE) = TRUE ;

Query OK (0,71 sec)

Cantidad: 341

8. **Select** *count (*)* **As** *Cantidad* **From** *TIntervenciones* **Where** *acoplamiento_fuerteTDA* (IPropuestaTDA, ' Abdomino, AmputaCion , PERIANAL ', TRUE) = TRUE ;

Query OK (0,71 sec)

Cantidad: 341

Como se observa en los ejemplos (7) y (8), los términos de la consulta han sido introducidos combinando indistintamente las mayúsculas y minúsculas. En ambos casos el resultado es el mismo, lo que demuestra que el usuario puede expresar los términos sin importar si utiliza las mayúsculas o minúsculas. Esta opción podría resultar particularmente importante para aportar el mismo resultado independientemente de como estén escritos los valores en la base de datos. En el caso de la solución implementada antes de hacer la comparación, lleva a mayúsculas tanto los términos introducidos para realizar la búsqueda como los términos que aparecen en la base de datos.

Si se tratara de utilizar indistintamente las mayúsculas con el operador *Like*, el resultado no será el esperado, y sólo se encontraría las tuplas que estén escritas exactamente a la forma en que son expresados los términos en la consulta. Un ejemplo de esta situación se muestra en el ejemplo siguiente.

9. **Select** *count (*)* **As** *Cantidad* **From** *TIntervenciones* **Where** *ipropuesta* Like '%cura de PERIANAL ABSCESO % ';

Query OK (0,08 sec)

Cantidad: 0

Como se observa en el ejemplo (9) al escribir los términos *cura* y *de* en minúsculas, esto hace que el operador no pueda localizar esa cadena y por eso no se devuelve ninguna tupla.

Para resolver esta problemática, PostgreSQL implementa una extensión al operador *Like* para dar la posibilidad al usuario de escribir los términos tanto en mayúsculas como en minúscula. Para ello implementa el operador *ILike (del Inglés insensitive Like)*. Utilizando este operador, sí el usuario podrá expresar los términos de su consultas tanto en mayúsculas como en minúsculas. Un ejemplo es el siguiente.

10. **Select** *count (*)* **As** *Cantidad* **From** *TIntervenciones* **Where** *ipropuesta* ILike '%cura de PERIANAL ABSCESO%';

Query OK (0,11 sec)

Cantidad: 21

En el caso del ejemplo (10) se utiliza el operador *ILike* y como se observa sí retorna las 21 tuplas que cumplen con la condición expresada en la cláusula *Where*. Aquí se ha escrito indistintamente los términos de la búsqueda en mayúsculas y minúsculas.

Para el caso de esta funcionalidad, sería necesario que el usuario domine la existencia de esta extensión del operador tradicional *Like*, para equiparar su solución con la que aquí se presenta. En cualquier caso, se demuestra que la solución dada a este problema del uso indistintamente de mayúsculas y minúsculas es ídem a las extensiones que implementa PostgreSQL.

6.5. Conclusiones

Como era el objetivo fundamental de este capítulo, en él se ha mostrado con varios ejemplos las potencialidades del modelo propuesto para la realización de consultas sobre textos cortos en bases de datos.

De forma práctica se ha ejemplificado el uso de las operaciones del modelo propuesto mediante las extensiones implementadas en PostgreSQL. Se ha mostrado el uso de dichas operaciones y se han discutido las mejoras que añade nuestra solución, con respecto a los operadores tradicionales para el trabajo con textos en los SGBD.

También se han mostrado la arquitectura y funcionalidades que posee el Cliente de Consulta implementado. En este sentido, cabe destacar que el diseño de la herramienta permite al usuario realizar su consulta de forma visual y muy intuitiva, guiado las diferentes cláusulas de la sentencia *Select*.

Finalmente se han mostrado las diferentes características que se le han añadido a las funciones implementadas, con el objetivo de que el usuario pueda expresar sus consultas utilizando frases como está adaptado a escribirlas en su entorno de trabajo. Dichas características ayudan a mejorar los resultados obtenidos, ya de por sí superiores, a los que resultan del uso de los operadores tradicionales para el trabajo con texto en SGBD.

El siguiente capítulo estará dedicado a la evaluación del modelo propuesto. Para ello, se definirán medidas provenientes de campo de la Recuperación de Información, para medir la capacidad del sistema de recuperar las tuplas relevantes y determinar la precisión de la información obtenida. También se realiza un análisis comparativo de los resultados obtenidos en consultas similares sobre atributos textuales, utilizando las operaciones del modelo propuesto, y operadores relacionales como el de igualdad y *Like*.

Capítulo 7

Evaluación del modelo

Las bases de datos relacionales se consideran un modelo determinístico con respuestas exactas, donde la respuesta a una consulta es un conjunto de tuplas que cumplen con una determinada condición de búsqueda. Sin embargo, al manejar atributos textuales, el modelo pasa a ser probabilístico porque no hay certeza de que las tuplas recuperadas sean relevantes, ni que se vayan a recuperar todas las que sí lo sean. Para evaluar el sistema en este sentido, no es suficiente con devolver el número de tuplas como respuesta, si no que se necesitan medidas adicionales, que evalúen además la calidad o bondad de esa respuesta en términos de recuperación y relevancia.

Con esta idea, en el presente capítulo se realiza la evaluación del modelo obtenido. Para ello, se utilizan medidas provenientes del campo de IR para medir la capacidad del sistema de recuperar las tuplas relevantes y determinar la precisión de la información obtenida.

Se comienza definiendo el concepto de *relevancia,* y a partir de los diferentes modelos que han aparecido para calcularla, se definen las medidas a utilizar: la *exhaustividad* y la *precisión* (Salton y McGill, 1983). Utilizando dichas medidas, se compara la efectividad del modelo propuesto, con operadores tradicionales de consultas sobre atributos textuales en bases de datos. Dichos operadores son el operador de igualdad y el operador *Like.*

Para realizar la comparación con estos operadores, se realizan diferentes ejemplos que muestran como se calculan las medidas para los modelos de relevancia booleano y difuso. Para ello, se define cómo se calculan las medidas para cada uno de los

operadores y el modelo propuesto, además de para cada modelo de relevancia, para los diferentes tipos de consultas que se pueden presentar. Estas consultas incluyen la búsqueda de un sólo término, de varios términos a la vez, o varios términos sin que todos estén incluidos.

El análisis comparativo del modelo propuesto mediante estas medidas, se realiza mediante ejemplos prácticos implementados sobre una base de datos médica.

7.1. Tratamiento de la relevancia

Varios factores son los que influyen en la calidad de la información recuperada por el sistema. Entre ellos la escritura correcta de los términos que se buscan, el tipo de operador que se utilice para encontrarlos, la calidad y homogeneidad de los datos que se encuentran almacenados en la base de datos, etc. De aquí que sea importante contar con algunas métricas que ayuden a tener una idea de la calidad de la información recuperada por un sistema.

Como se dijo anteriormente, las medidas que se utilizarán para medir la calidad de la solución aquí presentada, provienen del área de IR. En este contexto dichas medidas son aplicadas sobre documentos textuales completos. En nuestro caso, como se ha venido comentando, se aplica nuestra solución sobre atributos textuales en bases de datos. Debido a esto, en la definición y aplicación de estas medidas en nuestra solución, se utilizará el término tupla, en lugar de documento. Entendiendo esto como que cada tupla de la base de datos que contiene el atributo textual es, al fin y al cabo, un conjunto de términos. Hay que tener en cuenta además, que en nuestro modelo, una tupla puede contener varios conjuntos-AP con varios términos en cada uno de ellos.

Para definir la *exhaustividad* y la *precisión*, será necesario definir antes el concepto de *relevancia*, ya que es un concepto central en cualquier medida de IR. *Relevancia* es aquello que responde a lo solicitado al sistema; hay relevancia cuando lo recuperado se ajusta a la petición efectuada al sistema.

El concepto de relevancia se ha estudiado desde distintos puntos de vista (Saracevic, 1997): lógica, filosofía, psicología, semántica, documentación, etc. Estos enfoques se pueden resumir en dos tendencias: la relevancia objetiva y la subjetiva. La primera hace hincapié en los sistemas, normalmente define cómo la materia de la información recuperada coincide con la de la pregunta. La subjetiva, es la que tiene en cuenta al usuario (Wanson, 1986).

Muy ligado al concepto de *relevancia* está el de *pertinencia*; con frecuencia se entremezclan y confunden. Según Korfhage (Korfhage, 1997), relevancia es la medida de cómo una pregunta se ajusta a una tupla, (*esta visión coincide con el enfoque de la relevancia objetiva*) y *pertinencia* es la medida de cómo una tupla se ajusta a una necesidad informativa (*lo que otros autores definen como relevancia subjetiva*).

Además de la relevancia, para el cálculo de la *exhaustividad* y la *precisión*, será necesario conocer la cantidad de *tuplas recuperadas* por el sistema para responder a una determinada consulta. Dicha cantidad no será más que la cantidad total de tuplas que el sistema retorne que cumplen de alguna forma con la consulta que se ha realizado. Según la relevancia que tenga la tupla recuperada, será que se pueda considerar como tupla relevante recuperada. Esto significa que no sólo es recuperada, si no que posee el nivel de relevancia necesario para incluirla en la respuesta que se le brinda al usuario.

Para calcular la relevancia de una tupla, han aparecido entre otras, las siguientes formas:

- *Modelo de relevancia booleano:* Donde lo más habitual es establecer valores binarios: si una tupla es relevante, es decir, sirve como respuesta a la búsqueda que se realiza, se le asigna valor 1 de relevancia. Si por el contrario no sirve como respuesta a la búsqueda se le da valor 0.

- *Modelo de relevancia difuso:* Al contrario del caso booleano donde sólo se tienen dos grados de relevancia, en este modelo se establecen grados intermedios de relevancia de la tupla. Los valores de relevancia están dentro del rango $[0, 1]$. De esta forma, las tuplas de mayor relevancia tendrán valores más cercanos a l y las de menor relevancia, tendrán valores más cercanos a 0. Este modelo incluye el grado de *relevancia subjetiva difusa del usuario*, donde éste le asigna un valor acorde a su interpretación de la relevancia de la tupla analizada. Además, incluye el grado de *relevancia objetiva difusa*, que será un valor calculado automáticamente por el sistema.

A continuación se muestran tres ejemplos que ilustran el cálculo de la *relevancia*, siguiendo las dos variantes enunciadas anteriormente. Con la idea de aplicar estos conceptos, a un ejemplo práctico, se retomará el presentado en la figura 6.1 sobre *recetas de postres*.

Los tres ejemplos se realizan utilizando la forma en que sería calculada utilizando nuestro modelo. Para ello, en cada ejemplo se muestran los términos que se consultan, se selecciona una tupla concreta, y se le calcula la relevancia a dicha tupla, para cada una de dichas variantes.

Ejemplo 25

Seleccionar las tuplas que contienen los términos $\{Tarta, chocolate\}$.

Términos: $\{Tarta, chocolate\}$ ***TDA:*** $\{Tarta, nata, chocolate\}$		
Relevancia objetiva	*Relevancia objetiva difusa*	*Relevancia subjetiva difusa*
1	0.66	0.8

Como se puede observar en el ejemplo 25, se realiza el cálculo de la *relevancia* del conjunto $\{Tarta, chocolate\}$, con una tupla cuyo TDA toma como valor $\{Tarta, nata, chocolate\}$. En este caso, los dos términos que se buscan se encuentran contenidos en el valor del TDA, cuyo conjunto posee otro elemento (*nata*).

La *relevancia objetiva* que sigue el modelo booleano, da valor de relevancia 1 a la tupla, dado que contiene los términos que se están buscando. La *relevancia objetiva difusa* se puede calcular como el grado de acoplamiento de los términos que se buscan con el TDA. En este caso, podría ser el cociente entre la cardinalidad de la intersección entre ambos conjuntos, y la cardinalidad del TDA. Este cociente sería *(2/3)*. De ahí el valor de 0.66 de *relevancia* que obtiene el modelo para este TDA.

Para el caso del cálculo de la *relevancia subjetiva difusa,* el usuario podría dar un valor de 0.8 a esta tupla. Esto es debido a que ambos términos aparecen en la tupla. No se daría valor 1, pues posee un elemento más que los que se buscan. Dado que ese elemento adicional es un ingrediente más de la *Tarta*, se puede justificar ese valor elevado de relevancia.

Ejemplo 26

Seleccionar las tuplas que contienen el término $\{fresas\}$.

Término:$\{fresas\}$ ***TDA:***$\{Tarta, fresas\}$		
Relevancia objetiva	*Relevancia objetiva difusa*	*Relevancia subjetiva difusa*
1	0.5	0.7

En el ejemplo 26, se realiza el cálculo de la *relevancia* del conjunto $\{fresas\}$, con una tupla cuyo TDA toma como valor $\{Tarta, fresas\}$. En este caso, el término que se busca se encuentra contenido en el valor del TDA, cuyo conjunto posee otro elemento (*Tarta*).

La *relevancia objetiva*, da valor de relevancia 1 a la tupla dado que contiene el término que se está buscando. La *relevancia objetiva difusa* calculada ídem al ejemplo anterior y sería el cociente (*1/2*). De ahí el valor de 0.5 de *relevancia* que obtiene este modelo para este TDA.

Para el cálculo de la *relevancia subjetiva difusa* el usuario podría dar un valor de 0.7 a esta tupla. Esto debido a que el término aparece en la tupla. No se daría valor 1, pues posee un elemento más que el que se busca. En este caso el elemento adicional (*Tarta*) posee más peso semántico que el término adicional del ejemplo anterior por eso se le da un valor menor de relevancia.

Ejemplo 27

Seleccionar las tuplas que contienen los términos $\{fresas, nata\}$.

Término:$\{fresas\}$ ***TDA:***$\{Tarta, fresas\}$		
Relevancia objetiva	*Relevancia objetiva difusa*	*Relevancia subjetiva difusa*
1	0.5	0.5

Como se puede observar en el ejemplo 27, se realiza el cálculo de la *relevancia* del conjunto $\{fresas, nata\}$ con una tupla cuyo TDA toma como valor $\{Tarta, fresas\}$. En este caso sólo el término (*fresas*) aparece contenido en el TDA.

La *relevancia objetiva* da valor de relevancia 1 a la tupla, dado que contiene al menos uno de los términos que se buscan. La *relevancia objetiva difusa* calculada de

la forma antes discutida sería el cociente (*1/2*). De ahí el valor de 0.5 de *relevancia* que obtiene el modelo para este TDA.

En este ejemplo, el usuario podría dar un valor de *relevancia subjetiva difusa* de 0.5 a esta tupla. Esto debido a que sólo aparece en ella, uno de los dos términos que se buscan.

En función de estas formas en que puede ser calculada la *relevancia*, se podrán definir las expresiones para el cálculo de la *exhaustividad* y la *precisión*. En el siguiente aparatado se darán los conceptos de ambas medidas y se formularán las expresiones para su cálculo.

7.2. Definición de exhaustividad y precisión según el modelo utilizado

A continuación, se definen las dos medidas antes mencionadas en función de tuplas de base de datos y no en función de documentos, como se hace tradicionalmente. También se darán las variantes en que pueden ser calculadas dependiendo de qué modelo consideremos, el booleano o el difuso.

La **exhaustividad** indica la capacidad de un Sistema de Recuperación de Información para mostrar las tuplas relevantes. Ella mide el volumen de tuplas relevantes recuperadas respecto al total de relevantes disponibles en el sistema. Dicho de otra forma, es el cociente entre las tuplas relevantes recuperadas y el total de tuplas relevantes. Se podría definir también como la medición de la efectividad del sistema para evitar el silencio (*entendiendo silencio como no recuperar ninguna tupla*).

Por su parte la **precisión** mide el nivel de tuplas relevantes recuperadas respecto al total de tuplas recuperadas, relevantes o no. Dicho de otra forma, es el cociente entre las tuplas relevantes recuperadas y el total de tuplas recuperadas. Se podría definir también como la medición de la efectividad del sistema para evitar el ruido (*entendiendo ruido como las tuplas recuperadas que no son relevantes*).

Como se comentó anteriormente, la formulación matemática de ambas medidas queda determinada por el tipo de modelo que se utilice para calcular la relevancia de las tuplas. En este sentido, dado que el grado de relevancia subjetivo definido por el usuario es un concepto difuso en sí mismo, su expresión será la misma que la utilizada en el modelo difuso. Por su parte, el modelo booleano tendrá

sus expresiones basadas en las características de dicho modelo. Ambas formas de calcular la exhaustividad y la precisión aparecen a continuación.

7.2.1. Exhaustividad y precisión en el modelo booleano

Ambas medidas definidas en notación de conjuntos, donde $| A |$ significa la cardinalidad del conjunto A, quedarían de la siguiente forma:

$$Exhaustividad = \frac{| N_{Relv} \bigcap N_{Rec} |}{| N_{Relv} |}$$

$$Precisión = \frac{| N_{Relv} \bigcap N_{Rec} |}{| N_{Rec} |}$$

donde,

N_{Rec}: significa el número de tuplas recuperadas.

N_{Relv}: significa el número de tuplas relevantes dentro del total de tuplas recuperadas.

Como se ve, ambos valores estarán situados en el rango $[0, 1]$. Un valor 0 de *exhaustividad* significa que ninguna tupla relevante es recuperada, y un valor 1 indica que todas las tuplas relevantes son recuperadas. Para el caso de la *precisión* un valor 0 indica que ninguna tupla relevante es recuperada, y un valor 1 indica que todas las tuplas recuperadas son relevantes.

Normalmente, para los sistemas de recuperación de información que ordenan las tuplas de mayor a menor relevancia de acuerdo a la búsqueda realizada, la *exhaustividad* y la *precisión* de cada consulta serán inversamente proporcionales. Para las primeras tuplas la *precisión* será muy alta (muchas tuplas con información relevante) pero poca *exhaustividad* (quedan muchas tuplas relevantes por mostrarse). Según aumente el número de tuplas que se muestran, la *precisión* irá descendiendo y aumentando la *exhaustividad*.

En el caso contrario, una *exhaustividad* muy alta y una *precisión* muy baja nos mostraría la mayoría de las tuplas relevantes (*poco silencio*) pero también bastantes no relevantes (*mucho ruido*).

Este modelo de relevancia objetiva, es el modelo booleano que le asigna *relevancia* 0 a la tupla que no sea relevante y valor 1 a la tupla que sí lo sea. Para este modelo aparecen dos casos a la hora de considerar una tupla como relevante y recuperada:

1. Cuando la frase que se introduce para la búsqueda en la consulta aparece completamente incluida en cada tupla de la base de datos (*para consultas con uno o más términos*).
2. Cuando la frase que se introduce para la búsqueda en la consulta aparece parcialmente incluida en cada tupla de la base de datos (*para consultas con más de un término*).

Como se puede observar, la inclusión o no de los términos que se buscan en una o más tuplas de la base de datos, puede variar en dependencia de si se hace la búsqueda sobre un sólo término, o sobre más de un término. Teniendo en cuenta estos dos tipos de casos posibles, en las tablas 7.1 y 7.2 se hace un resumen que muestra, lo que a efectos de los cálculos que se realizarán de la *exhaustividad* y la *precisión*, se entiende por recuperado y relevante. También en dichas tablas se muestra cómo serán obtenidos dichos valores, para cada uno de los operadores que se utilizarán en las consultas de ejemplo: el operador de igualdad, el operador *Like* y el modelo propuesto en este trabajo.

Como se puede observar en la tabla 7.1, en todos los casos se considera como número de tuplas relevantes, las tuplas que contengan en cualquier posición el término que se está buscando. Para el caso de las tuplas recuperadas, se observa la clara desventaja del operador de igualdad, pues sólo recupera las tuplas que contienen únicamente el término que se busca. En este caso, todas las tuplas que contengan una frase en la que aparece el término que se busca no serán retornadas.

Para el caso de las tuplas recuperadas con el operador *Like*, se puede especificar un patrón de búsqueda que contenga el término que se busca con un signo de % al inicio

	Operador de igualdad	**Operador Like**	**Modelo propuesto**
Número de tuplas relevantes (N_{Relv})	Las tuplas que contengan en cualquier posición el término que se está buscando	Las tuplas que contengan en cualquier posición el término que se está buscando	Las tuplas que contengan en cualquier posición el término que se está buscando
Número de tuplas recuperadas (N_{Rec})	Las tuplas que contengan exactamente sólo el término que se está buscando	Las tuplas que retorne el patrón *%término % (indica que el término puede aparecer en cualquier posición)*	Las tuplas donde la operación acoplamiento_debil, aplicada sobre el TDA de cada tupla, se valore como verdadera con el término que se busca

Tabla 7.1: Criterios para el cálculo de la *exhaustividad* y la *precisión* en consultas con un sólo término y usando *relevancia objetiva*.

y final del término. En este caso, se recuperarán todas las tuplas que contengan el término, sin importar la posición en que aparece. Finalmente, con el modelo propuesto, las tuplas recuperadas serán las que tengan un acoplamiento débil tal como lo define esa operación. Para ello, se verificará la función *acoplamiento_ debil*, aplicada sobre el TDA de cada tupla y la frase que se busca. Las tuplas donde dicha función se valore como verdadera, serán retornadas.

Cabe destacar en este caso en el que se realizan consultas con un sólo término, que se podría considerar la relevancia difusa para admitir sinónimos como respuesta a la consulta. De esta forma, se podría dotar a las consultas de un grado de flexibilidad semántica. Para ello, se podrían definir diferentes grados de sinonimia entre varios términos que pudieran ser interpretados como uno sólo, con distinto grado de similitud. Por ejemplo, si en la consulta se busca "fractura" y el sistema encuentra "rotura" ó "fisura", suponiendo un grado de sinonimia de estas palabras con fractura, en ese grado se basaría el grado de acoplamiento para determinar la relevancia de la tupla.

En la tabla 7.2, se hace el mismo análisis que en la tabla anterior, pero ahora

para cuando se buscan varios términos. En dicha tabla se da la forma de calcular el número de tuplas relevantes y recuperadas, para los dos casos posibles. Primero, para cuando se requiere encontrar la frase que se busca completamente y después, para cuando se requiere encontrar sólo parte de dicha frase.

	Operador de igualdad	**Operador Like**	**Modelo propuesto**
Número de tuplas relevantes (N_{Relv}) para encontrar la frase completa	Las tuplas que contengan completamente, en cualquier posición, la frase buscada	Las tuplas que contengan completamente, en cualquier posición, la frase buscada	Las tuplas que contengan completamente, en cualquier posición, la frase buscada
Número de tuplas relevantes (N_{Relv}) para encontrar sólo parte de la frase	Las tuplas que contengan al menos un término, en cualquier posición, de la frase buscada	Las tuplas que contengan al menos un término, en cualquier posición, de la frase buscada	Las tuplas que contengan al menos un término, en cualquier posición, de la frase buscada
Número de tuplas recuperadas (N_{Rec}) para encontrar la frase completa	Las tuplas que contengan exactamente y en el mismo orden, la frase buscada	Las tuplas que retorne el patrón *%frase %* *(indica que la frase puede aparecer en cualquier posición)*. Sí importa el orden en que aparezcan los términos	Las tuplas donde la operación acoplamiento_fuerte, aplicada sobre el TDA de cada tupla, se valore como verdadera con la frase buscada. No importa el orden
Número de tuplas recuperadas (N_{Rec}) para encontrar sólo parte de la frase	Las tuplas que contengan exactamente y en el mismo orden, la frase buscada	Dependerá de las tuplas que retorne el patrón que se cree. Puede ser uno para cada término o la frase completa. Sí importa el orden en que aparezcan los términos	Las tuplas donde la operación acoplamiento_debil, aplicada sobre el TDA de cada tupla, se valore como verdadera con la frase buscada. No importa el orden

Tabla 7.2: Criterios para el cálculo de la *exhaustividad* y la *precisión* en consultas con varios términos usando *relevancia objetiva*.

Como se puede observar, cuando se requiere encontrar la frase completa, el número de tuplas relevantes se obtiene de igual forma para los tres operadores. Se hace recuperando las tuplas que contengan completamente, en cualquier posición, la frase buscada. Para el caso de este valor, cuando se quiere encontrar sólo parte de la frase, será el de las tuplas que contengan al menos un término, en cualquier posición, de la frase buscada.

Para el caso del número de tuplas recuperadas cuando se requiere encontrar la frase completa, sí varía en dependencia del operador que se utilice. En este caso, el operador de igualdad vuelve a presentar serios problemas, pues sólo recupera las tuplas que contengan exactamente, y en el mismo orden, la frase buscada. Aquí quedarán fuera las frases que contengan más términos que los buscados, o que no aparezcan en el mismo orden. De forma análoga ocurre con el operador *Like*. Para el caso del modelo propuesto, no presenta ninguno de estos problemas pues con la operación *acoplamiento_fuerte* encuentra la frase completa. Esto lo hace aún cuando el TDA tenga más términos que los buscados o los tenga en posiciones diferentes.

Para el caso del número de tuplas recuperadas cuando se requiere encontrar sólo parte de la frase que se busca, también varía en dependencia del operador que se utilice. En este caso, el operador de igualdad vuelve a presentar serios problemas, pues sólo recupera las tuplas que contengan exactamente, y en el mismo orden, la frase buscada. Este operador no tiene posibilidad de sólo expresar parte de la frase pues si no, se pierde el sentido de lo que busca el usuario.

Para el operador *Like,* dependerá de la calidad del patrón que especifique el usuario; en este caso, si pone sólo algunos términos de la frase original, se pueden recuperar tuplas que no son correctas. Por ejemplo, si el usuario busca la frase '*fractura cadera*'y se crea el patrón %*fractura* %, todos los tipos de fracturas serán recuperados y no es correcto. Si se pone más de un término en el patrón que se cree, dependerá ya del orden en el que sean escritos. Esto hace este operador dependiente del orden de los términos introducidos.

Con el modelo propuesto, se contarán como recuperadas todas las tuplas donde la operación *acoplamiento_debil*, aplicada sobre el TDA de cada tupla, se valore como verdadera con la frase buscada. En este caso, no importa el orden en que el usuario introduzca los términos, ni si en la tupla hay más términos que los que se buscan.

En el siguiente apartado, se discuten los parámetros requeridos para el cálculo de la *exhaustividad* y *precisión* utilizando la *relevancia objetiva difusa*. Para ello, de forma análoga a como se hizo con la *relevancia objetiva*, se verá cómo se calculan los parámetros en el caso de los operadores de igualdad y *Like*, y también para el modelo propuesto.

7.2.2. Exhaustividad y precisión en el modelo difuso

Tomando como base las definiciones del modelo booleano, en (Buell y Kraft, 1981) se definieron dichas expresiones en un ambiente difuso. Este modelo se basa en los grados de relevancia entre $[0, 1]$, provenientes de la evaluación que realiza el sistema y el usuario de la calidad de las tuplas recuperadas.

En este modelo difuso, la formulación de la *exhaustividad y precisión* se realiza en función de la evaluación que realiza el sistema y el usuario, de la relevancia de cada tupla recuperada en la búsqueda que se realiza. Para cada tupla recuperada, el sistema le da una evaluación s_i y el usuario una evaluación u_i. Como corresponde en lógica difusa, para determinar el *AND* entre ambos valores, se calcula la función mínimo entre ellos.

Con estos precedentes, ambas medidas en este modelo quedarían expresadas de la siguiente forma:

$$Exhaustividad\ Difusa = \frac{\sum_i min(s_i, u_i)}{\sum_i (u_i)}$$

$$Precisión\ Difusa = \frac{\sum_i min(s_i, u_i)}{\sum_i (s_i)}$$

donde:

s_i: significa la evaluación que el sistema da a la tupla i.

u_i: significa la evaluación que el usuario da a la tupla i.

Para el caso de la *exhaustividad difusa* se calcula como el cociente entre la sumatoria de cada mínimo calculado entre la valoración del sistema y el usuario de cada tupla i, entre la sumatoria de la valoración del usuario de cada tupla i. Por su parte, la *precisión difusa* se define como el cociente entre la sumatoria de cada mínimo calculado entre la valoración del sistema y el usuario de cada tupla i, entre la sumatoria de la valoración del sistema de cada tupla i.

La interpretación de ambas medidas en este caso tiene el mismo significado que en el modelo booleano. Para el caso de la *exhaustividad difusa*, se pondera la opinión del usuario sobre la relevancia de las tuplas recuperadas. De esta forma prima su criterio sobre la relevancia de las tuplas recuperadas. En el caso de la precisión ocurre lo contrario, y se determina dicho valor en función de la valoración que realiza el sistema de la relevancia de las tuplas recuperadas.

En el contexto que nos ocupa, de consultas semánticas sobre atributos textuales en bases de datos, se pueden utilizar ambas medidas. En este caso, sus valores pueden ser calculados para los diferentes operadores tradicionales de búsqueda de textos en bases de datos relacionales, como el operador de igualdad (=) y el operador *Like*. También se pueden calcular estos valores sobre los resultados obtenidos utilizando el modelo propuesto en esta memoria. Esto podría servir como elemento comparativo de nuestro modelo, con respecto a la forma tradicional de resolver consultas sobre atributos textuales en bases de datos.

Al aplicar estas medidas en el entorno de consultas semánticas sobre textos cortos en bases de datos, será necesario utilizar las formas antes descritas para calcular las tuplas relevantes, que serán retornadas al usuario como respuesta a una consulta determinada.

Para el caso de los ejemplos que serán implementados en este capítulo, se realizarán dichos ejemplos utilizando la relevancia objetiva, en sus dos modalidades, booleana y difusa. Como se comentó anteriormente, el cálculo de relevancia por el modelo difuso incluye la subjetividad de la opinión del usuario (u_i) para dar el valor de relevancia de una tupla determinada.

En el cálculo de ambas medidas utilizando el modelo de *relevancia objetiva difusa*, no es necesario separar la forma en que se calcula la relevancia, para un sólo término o una frase. En este caso, en la *relevancia subjetiva* que da el usuario de la calidad

de la tupla retornada, y la evaluación que de ella da también el sistema, va implícita la calidad del acoplamiento entre la frase que se busca y cada tupla analizada.

En este modelo difuso, el cálculo del valor de u_i, como ya se mencionó anteriormente, se basa en la opinión del usuario de la calidad de la tupla retornada. Para el caso de s_i, sí se hace necesario separar su estudio cuando se busca una frase parcialmente incluida, o una frase completamente incluida. Debido a que en el primer caso el cálculo de s_i se realiza utilizando la operación índice de acoplamiento débil y en el segundo caso se utilza el índice de acoplamiento fuerte. Ambos índices pueden ser calculados por las dos variantes discutidas en el capítulo 3: por el máximo y el promedio.

Teniendo en cuenta estas ideas se puede resumir, que en este modelo de *relevancia objetiva difusa* el cálculo de la exhaustividad y la precisión para los operadores de igualdad y *Like*, y para el modelo propuesto, se utilizará la *relevancia subjetiva* que el usuario de a cada tupla y para s_i se utilizará el índice de acoplamiento según si se busca parte de la frase o la frase completa.

En el próximo apartado, se muestran tres ejemplos prácticos donde se calculan la *exhaustividad* y la *precisión* utilizando el modelo de *relevancia objetiva* y *objetiva difusa*. Como se comentó anteriormente, se utilizará para ello, los datos de la tabla 6.1.

7.3. Ejemplos prácticos del cálculo de exhaustividad y precisión para los modelos booleano y difuso

A continuación, se muestran tres ejemplos con los casos típicos discutidos anteriormente para el caso de la relevancia objetiva. En dichos ejemplos, se calculará también la *exhaustividad* y *precisión* utilizando las expresiones difusas. Los tres casos son:

1. *Cuando se busca un sólo término.*
2. *Cuando se busca más de un término y se quiere recuperar la tupla sólo si contiene todos los términos buscados.* Sería equivalente a la búsqueda de varios términos con un operador lógico *AND*.

3. *Cuando se busca más de un término y se quiere recuperar la tupla si contiene al menos uno de los términos que se buscan.* Sería equivalente a la búsqueda de varios términos con un operador lógico *OR*.

La estructura de dichos ejemplos es la misma en los tres casos: recuperar ciertos términos sobre el atributo *Nombre Postre* de la tabla de la figura 6.1. Se darán los términos de la consulta y a continuación por cada uno de los operadores antes discutidos y el modelo propuesto, se darán los resultados de las medidas *exhaustividad* y *precisión* con modelo de *relevancia objetiva* y *objetiva difusa*.

Ejemplo 28

Seleccionar las tuplas que contienen el término 'fresas'.

Los resultados del cálculo de ambas medidas, utilizando los dos operadores y el modelo propuestos, son los que se muestran a continuación para cada tipo de *relevancia*.

- ***Modelo booleano***

En este caso, al ser un sólo término sobre el que se hace la búsqueda, para calcular los términos relevantes y recuperados, se utilizan los criterios dados en la tabla 7.1. Para el caso del operador *Like,* se utiliza el patrón ***%fresas %*** (*asumiendo una variante de Like en todos los ejemplos que recupere el término sin importar mayúsculas o minúsculas*).

Operador de igualdad	**Operador Like**	**Modelo propuesto**
N_{Relv}= 4 N_{Rec}=0	N_{Relv}= 4 N_{Rec}=4	N_{Relv}= 4 N_{Rec}=4
exhaustividad = 0	*exhaustividad* = 1	*exhaustividad* = 1
precisión = 0	*precisión* = 1	*precisión* = 1

Como se observa en el ejemplo 28, para el caso del operador de igualdad la *exhaustividad* es 0, debido a que no recupera ninguna de las 4 tuplas que son relevantes, ya que contienen el término *fresas*. Esto se debe a que en las 4 ocasiones en que aparece dicho término en las tuplas no aparece solo, para que fuera recuperado por el operador de igualdad. Para este operador la *precisión* es 0.

Para el caso del operador *Like,* como es un sólo término el que se busca, el patrón especificado en la búsqueda encuentra el término en todas las posiciones en que se encuentre en las tuplas. En este caso, el valor de ambas medidas es 1, ya que se recuperan todas las tuplas que son relevantes. De forma idéntica ocurre con el modelo propuesto, que utilizando la operación *acoplamiento_debil* recupera las 4 tuplas relevantes que contienen el término *fresas*.

- ***Modelo difuso***

Para el caso del cálculo de la valoración del sistema y del usuario, para obtener la *exhaustividad* y *precisión* difusas, se realizará siguiendo los criterios establecidos anteriormente. En los tres ejemplos se utilizará el índice que responda al interés de recuperar todos los términos de la consulta, o solo algunos de ellos. En todos los caso se utilizará la variante de calcular el índice por la vía del máximo.

Operador de igualdad	**Operador Like**	**Modelo propuesto**
s_i= {0,0,0,0,0,0,0,0,0} u_i={0.6,0.8,0,0,0.8,0,0,0,0.5}	s_i= {0.5,0.5,0,0,0.5,0,0,0,0.25} u_i={0.6,0.8,0,0,0.8,0,0,0,0.5}	s_i= {0.5,0.5,0,0,0.5,0,0,0,0.25} u_i={0.6,0.8,0,0,0.8,0,0,0,0.5}
exhaustividad difusa **0**	*exhaustividad difusa* 1.75 / 2.7 = **0.64**	*exhaustividad difusa* 1.75 / 2.7 = **0.64**
precisión difusa **0**	*precisión difusa* 1.75 / 1.75 = **1**	*precisión difusa* 1.75 / 1.75 = **1**

En el caso de los tres ejemplos donde se calculan la *exhaustividad* y *precisión* difusas, se ha puesto para cada operador, un conjunto con los valores correspondientes a s_i y u_i. Aparece un valor para cada una de las nueve tuplas que contiene la tabla de la figura 6.1. Para determinar el mínimo de s_i con u_i, será necesario hacer coincidir los valores de cada posición de dicho conjunto.

Como se puede observar en el ejemplo 28, para el caso del operador igualdad, dado que la consulta buscando el término *fresas* no retorna ninguna tupla, su valor de s_i en todos los casos será 0, y por ende, tanto la exhaustividad como la precisión difusas toman valor cero.

Con el operador *Like,* la sumatoria de los mínimos entre s_i y u_i da 1.75. Para el cálculo de la *exhaustividad difusa*, se divide este valor entre la sumatoria de los u_i

que en este caso también es 2.7, obteniendo valor 0.64. Para calcular la *precisión difusa* se divide también la sumatoria de los mínimos entre s_i y u_i(1.75) entre la sumatoria de los s_i que es 1.75, obteniendo valor 1.

Para el caso del modelo propuesto, los resultados obtenidos son idénticos a los calculados con el operador *Like*. Esto se debe a que se busca un sólo término y el operador *Like*, al igual que nuestro modelo, es capaz de encontrarlo en cualquier posición dentro del valor de la tupla.

Ejemplo 29

Seleccionar las tuplas que contienen completamente la frase 'Tarta de chocolate'.

En este caso, al ser una frase lo que se recupera e interesa que se retornen las tuplas que la contienen completamente, para calcular los términos relevantes y recuperados, se utilizan las filas uno y tres de la tabla 7.2 respectivamente. Para el caso del operador *Like*, se utiliza el patrón *% **Tarta de chocolate** %* . Los resultados del cálculo de ambas medidas, utilizando los dos operadores y el modelo propuesto, son los que se muestran a continuación para cada tipo de *relevancia*.

- **Modelo booleano**

Operador de igualdad	**Operador Like**	**Modelo propuesto**
N_{Relv}= *2* N_{Rec}=*1*	N_{Relv}= *2* N_{Rec}=*1*	N_{Relv}= *2* N_{Rec}=*2*
exhaustividad = 0.5	*exhaustividad* = 0.5	*exhaustividad* = 1
precisión = 1	*precisión* = 1	*precisión* = 1

En la solución de este ejemplo, se ha tomado la tupla *Tarta de nata y Chocolate* como relevante, ya que contiene todos los términos que se están recuperando, aunque no en el mismo orden. Como se discutió anteriormente, para este caso particular, todas las tuplas que contengan todos los términos que se están buscando aunque no aparezcan en el mismo orden, son relevantes.

En este ejemplo 29, para el caso del operador de igualdad, la *exhaustividad* es 0.5, debido a que recupera la única tupla que contiene la frase *Tarta de Chocolate* completamente, y sin que contenga ningún otro término. Para el caso de la *precisión* toma valor 1, pues la tupla recuperada es relevante.

Para el caso del operador *Like* con el patrón de búsqueda antes mencionado, se obtienen resultados idénticos a los obtenidos con el operador de igualdad. Para este caso, la tupla *Tarta de nata y Chocolate* no es recuperada dado que no se encuentra en el mismo orden en que se introduce en la frase que se consulta.

Con el modelo propuesto los resultados son superiores. En este caso, se recupera la tupla *Tarta de nata y Chocolate,* y por tanto la la *exhaustividad* y la precisión toman valor 1. En este caso, se debe a que la operación *acoplamiento_fuerte* utilizada en este caso, recupera ambas tuplas, dado que no tiene en cuenta el orden en el que aparecen los términos en dichas tuplas.

- **Modelo difuso**

En este caso como se requiere que todos los términos que se buscan en la frase aparezcan en la tupla recuperada, se utilizará el índice de acoplamiento fuerte para el cálculo de s_i.

Operador de igualdad	**Operador Like**	**Modelo propuesto**
s_i = {0,0,0,1,0,0,0,0,0} u_i = {0,0.5,0.5,1,0,0,0.8,0.5,0.3}	s_i = {0,0,0,1,0,0,0,0,0} u_i = {0,0.5,0.5,1,0,0,0.8,0.5,0.3}	s_i = {0,0,0,1,0,0,0.66,0,0 } u_i = {0,0.5,0.5,1,0,0,0.8,0.5,0.3}
exhaustividad difusa 1 / 3.6 = **0.27**	*exhaustividad difusa* 1 / 3.6 = **0.27**	*exhaustividad difusa* 1.66 / 3.6 = **0.46**
precisión difusa 1/ 1 = **1**	*precisión difusa* 1/ 1 = **1**	*precisión difusa* 1.66 / 1.66 = **1**

En este ejemplo, para el caso del operador igualdad, la consulta buscando la frase *Tarta de chocolate* sólo retorna la tupla de la fila 4. Como la frase buscada se encuentra completamente incluida en dicha tupla se le asigna valor 1 para el parámetro s_i; al resto de las tuplas se le asigna valor 0. Para el caso de los valores de u_i, se realiza una valoración por el usuario acorde con el contenido de las tuplas y la frase que se busca.

La sumatoria de los mínimos entre s_i y u_i da 1. Para el cálculo de la *exhaustividad difusa* se divide este valor entre la sumatoria de los u_i que en este caso es

3.6, obteniendo valor 0.27. Para calcular la *precisión difusa* se divide también la sumatoria de los mínimos entre s_i y u_i(1) entre la sumatoria de los s_i que es 1, obteniendo valor 1. Para el caso del operador *Like,* dado que el patrón de búsqueda no es otro que la frase completa en cualquier posición, y sólo una tupla cumple esta restricción, los valores obtenidos son ídem a los obtenidos con el operador de igualdad.

Para el caso del modelo propuesto, la sumatoria de los mínimos entre s_i y u_i da 1.66. Para el cálculo de la *exhaustividad difusa* se divide este valor entre la sumatoria de los u_i que en este caso es 3.6, obteniendo valor 0.46. Para calcular la *precisión difusa* se divide también la sumatoria de los mínimos entre s_i y u_i(1.66), entre la sumatoria de los s_i que es 1.66, obteniendo valor 1.

Como se observa, en el caso del modelo propuesto se obtiene casi el doble de *exhaustividad difusa*, debido a la forma en que se obtienen los s_i. En este caso, se obtienen mejores resultados dado que no es obligatorio encontrar la frase con los términos en el mismo orden en que los expresa el usuario, por eso se recupera una tupla más que con los otros operadores. Para el caso de la precisión se obtiene el mismo valor 1 que en los operadores tradicionales.

Ejemplo 30

Seleccionar las tuplas que contienen al menos parcialmente la frase 'Fresas con nata'.

En este caso, al ser una frase lo que se recupera e interesa que se retornen las tuplas que la contienen al menos parcialmente, para calcular los términos relevantes y recuperados, se utilizan las filas dos y cuatro de la tabla 7.2 respectivamente. Para el caso del operador *Like,* se utiliza el patrón *%nata %*. Al utilizar este patrón, se está asumiendo que se recuperarán tanto las tuplas que contengan los términos *Fresas* y *nata*, como cualquier otra combinación que contenga el término *nata*. Esto puede ir en detrimento de la relevancia de los términos recuperados.

Los resultados del cálculo de ambas medidas, utilizando los dos operadores y el modelo propuesto, son los que se muestran a continuación para cada tipo de *relevancia.*

- **Modelo booleano**

Operador de igualdad	**Operador Like**	**Modelo propuesto**
N_{Relv}= 7 N_{Rec}=1	N_{Relv}= 7 N_{Rec}=6	N_{Relv}= 7 N_{Rec}=7
exhaustividad = 0.14	*exhaustividad* = 0.85	*exhaustividad* = 1
precisión = 1	*precisión* = 1	*precisión* = 1

En la solución de este ejemplo 30, se han tomado las 7 tuplas que contienen los términos *Fresas* y/o *nata* como relevantes, ya que contienen al menos uno de los términos que se están recuperando. Como se discutió anteriormente, para este caso particular, todas las tuplas que contengan al menos uno de los términos que se están buscando aunque no aparezcan en el mismo orden, son relevantes.

Como se observa en el ejemplo anterior, para el caso del operador de igualdad la *exhaustividad* es 0.14, debido a que recupera la única tupla que contiene la frase *Fresas con nata* completamente, y sin que contenga ningún otro término. Para el caso de la *precisión* toma valor 1, pues la tupla recuperada es relevante.

Para el caso del operador *Like* con el patrón de búsqueda antes mencionado, se recuperan 6 tuplas que contienen el término *nata*. Esto hace que la *exhaustividad* tome valor 0.85 y la *precisión* tome valor 1, pues los términos que se recuperan son relevantes.

Con el modelo propuesto los resultados también son superiores. En este caso se recupera la *tupla Tarta de fresas,* que no se obtiene ni con el operador *Like*, y por tanto la *exhaustividad* y la *precisión* toman valor 1. En este caso, se debe a que la operación *acoplamiento_ debil* utilizada en este caso, recupera esta tupla, dado que no tiene en cuenta el orden en que aparecen los términos en ella.

- **Modelo difuso**

En este caso como se requiere que los términos que se buscan en la frase aparezcan al menos parcialmente incluidos en la tupla recuperada, se utilizará el índice de acoplamiento débil para el cálculo de s_i.

Operador de igualdad	**Operador Like**	**Modelo propuesto**
s_i = {0,0,0,0,1,0,0,0,0} u_i = {0.8,0.4,0,0,1,0.5,0.3,0.5,0.6}	s_i = {1,0,0,0,1,0.5,0.33,0.5,0.5} u_i = {0.8,0.4,0,0,1,0.5,0.3,0.5,0.6}	s_i = {1,0.5,0,0,1,0.5,0.33,0.5,0.5} u_i = {0.8,0.4,0,0,1,0.5,0.3,0.5,0.6}
exhaustividad difusa 1 / 4.1 = **0.24**	*exhaustividad difusa* 3.6 / 4.1 = **0.87**	*exhaustividad difusa* 4 / 4.1 = **0.97**
precisión difusa 1 / 1 = **1**	*precisión difusa* 3.6 / 3.83 = **0.93**	*precisión difusa* 4 / 4.33 = **0.92**

En este ejemplo, para el caso del operador igualdad, la consulta buscando la frase *Fresas con nata* sólo retorna la tupla de la fila 5. De aquí que esa tupla sea a la única que se le asigne valor 1 para el parámetro s_i; al resto de las tuplas se le asigna valor 0. Para el caso de los valores de u_i, se realiza una valoración por el usuario acorde con el contenido de las tuplas y la frase que se busca. La sumatoria de los mínimos entre s_i y u_i da 1. Para el cálculo de la *exhaustividad difusa,* se divide este valor entre la sumatoria de los u_i que en este caso es 4.1, obteniendo valor 0.24. Para calcular la *precisión difusa* se divide también la sumatoria de los mínimos entre s_i y u_i(1) entre la sumatoria de los s_i que es 1, obteniendo valor 1.

Para el caso del operador *Like,* con el patrón de búsqueda %*nata* % se recuperan todas las tuplas que contengan este término. Por dichas tuplas, s_i toma los valores que aparecen en la tabla anterior. Para el caso de los valores de u_i se realiza una valoración por el usuario acorde con el contenido de las tuplas y la frase que se busca. La sumatoria de los mínimos entre s_i y u_i da 3.6. Para el cálculo de la *exhaustividad difusa* se divide este valor entre la sumatoria de los u_i que en este caso es 4.1, obteniendo valor 0.87. Para calcular la *precisión difusa* se divide también la sumatoria de los mínimos entre s_i y u_i(3.6) entre la sumatoria de los s_i que es 3.83, obteniendo valor 0.93.

Para el caso del modelo propuesto, la sumatoria de los mínimos entre s_i y u_i da 4. Para el cálculo de la *exhaustividad difusa* se divide este valor entre la sumatoria de los u_i que en este caso es 4.1, obteniendo valor 0.97. Para calcular la *precisión difusa* se divide también la sumatoria de los mínimos entre s_i y u_i(4) entre la sumatoria de los s_i que es 4.33, obteniendo valor 0.92.

Como se observa en el caso del modelo propuesto, se obtiene también una mayor *exhaustividad difusa*, debido a la forma en que se obtienen los s_i. En este caso,

se obtienen mejores resultados, dado que no es obligatorio encontrar la frase con los términos en el mismo orden en que los expresa el usuario; por eso se recupera una tupla más que con los otros operadores. Para el caso de la precisión, es prácticamente la misma que con el operador *Like*.

De forma general, el comportamiento del sistema propuesto es similar en la mayoría de los ejemplos, donde se obtiene una mayor *exhaustividad difusa*, e igual o ligeramente menos *precisión difusa*.

En la siguiente sección se realiza un conjunto de experimentos, con vistas a demostrar las mejoras introducidas por la solución propuesta en la presente memoria, con respecto al tratamiento de atributos textuales en bases de datos. Para ello, se utilizarán los conjuntos experimentales definidos en el capítulo 5, sobre datos hospitalarios.

7.4. Evaluación del modelo sobre un caso real: la base de datos médica

A continuación, se muestran varios ejemplos de consultas posibles a través del cliente de consulta implementado. En dichos ejemplos, se medirá la eficiencia del sistema calculando la *exhaustividad* y la *precisión,* utilizando el modelo de relevancia objetiva. Para ello, se muestran ejemplos que incluyen los casos discutidos en la sección anterior, considerando el modelo booleano.

Todos los ejemplos que se muestran a continuación comienzan con la definición de la consulta, seguido del conjunto experimental sobre el que se obtienen los resultados. Posteriormente, se implementa la solución utilizando los operadores de igualdad (=) y *Like*. Finalmente, dentro del ejemplo se muestra la solución utilizando nuestro modelo. Para el caso de los ejemplos que retornan un conjunto de tuplas, sólo se pone la cantidad de tuplas retornadas para ganar en legibilidad en las tablas en que se muestran dichos ejemplos.

7.4.1. Exhaustividad y precisión en el modelo booleano

En este apartado se muestran diferentes ejemplos de consultas realizadas sobre casos reales en una base de datos médica. En dichos ejemplos, se utilizarán diferentes

operaciones que fueron introducidas en el modelo propuesto. Utilizando el modelo booleano, se definirán los ejemplos para los diferentes casos que se han discutido que se pueden presentar bajo este modelo booleano. En la explicación de los diferentes ejemplos, se destaca además, las ventajas que brinda el modelo propuesto, con respecto a los operadores tradicionales para consultar atributos textuales en bases de datos.

A continuación se muestran dos ejemplos de consultas que se realizan bajo este modelo de relevancia, cuando se intenta recuperar un sólo término.

7.4.1.1. Evaluación de consultas con un sólo término

Ejemplo 31

Determinar la cantidad de pacientes a los que se les ha diagnosticado ' ABSCESO'.

Conjunto experimental: *Conjunto1* **Número de tuplas relevantes:** 438
Solución con el operador de igualdad (=) tradicional
Select count(*) **AS** Cantidad **From** tintervenciones **Where** diagnostico = 'ABSCESO'; *Cantidad de tuplas recuperadas:* 0 —> *exhaustividad* = 0 y *precisión* = 0
Solución con el operador de comparación tradicional en textos (Like)
Select count(*) **AS** Cantidad **From** tintervenciones **Where** diagnostico Like '%ABSCESO%'; *Cantidad de tuplas recuperadas:* 307 —> *exhaustividad* = 0.70 y *precisión* = 1
Solución con el modelo propuesto
Select count(*) **AS** Cantidad **From** tintervenciones **Where** *acoplamiento_ debilTDA* (TDAdiagnostico, 'ABSCESO', TRUE) = TRUE; *Cantidad de tuplas recuperadas:* 438 —> *exhaustividad* = 1 y *precisión* = 1

Tabla 7.3: Comparativa de operadores en modelo booleano (Ejemplo 31).

Como se observa en el ejemplo de la tabla 7.3, se intenta recuperar el número de pacientes a los que se les ha diagnosticado un absceso. Para el caso del operador de igualdad no se retorna ninguna tupla. Esto se debe a que este operador busca las tuplas que contienen exactamente esta palabra, y en este caso ninguna tupla de la base de datos cumple esta condición. Para este operador, la *exhaustividad* y la *precisión* toman valor 0, ya que no se recupera ninguna tupla.

Con el operador *Like* se ha especificado el patrón de búsqueda *%ABSCESO %*, que significa encontrar todas las tuplas que contienen el término *ABSCESO*, sin importar la posición que ella ocupe al inicio, medio o final de dicha tupla. En este caso, se recuperan 307 tuplas de pacientes que han sido diagnosticados con un absceso. En el caso de este operador, vale destacar que según sea la calidad del patrón de búsqueda que se utilice, así será la calidad de los resultados obtenidos. Para este operador, la *exhaustividad* toma un valor de 0.70 (*307/438*) y la *precisión* toma valor 1.

Para el caso de la solución implementada utilizando nuestro modelo, como se observa en la tabla 7.3, se obtienen 438 tuplas de pacientes que han sido diagnosticados con un *ABSCESO*. Este resultado es considerablemente superior al de los dos operadores tradicionales. En este caso se utiliza el método *acoplamiento_ debilTDA* para localizar todas las tuplas que contienen el término que se desea encontrar. Para nuestro modelo, como se recuperan todas las tuplas que son relevantes, la *exhaustividad* y la *precisión* toman valor 1.

La función *acoplamiento_ debilTDA* recibe tres parámetros: el primero es el atributo sobre el que se realiza la búsqueda, en este caso *TDAdiagnostico*, que se corresponde con la representación que se obtiene para el atributo *diagnóstico* de la base de datos original. El segundo parámetro es la lista de términos que se busca, en este caso el término *ABSCESO*. Finalmente, se le pasa a la función un atributo booleano donde se le indica si se le aplica o no el proceso de limpieza descrito en capítulos anteriores a la frase que se está buscando antes de realizar la consulta. El resto de las funciones invocadas en los siguientes ejemplos tienen los mismos parámetros con el mismo significado.

En el caso particular de este ejemplo, la mejoría obtenida por nuestra solución viene dada porque recupera el término que se busca en todas las formas diferentes en que aparece en la base de datos. A las 307 veces que aparece escrito correctamente (*ABSCESO*), se le suman las 125 veces que aparece escrito como *ABCESO* y las 6 veces que aparece escrito como *ASBCESO*.

Teniendo en cuenta esta recuperación de sinónimos que hace nuestro modelo, siempre se recuperará la máxima cantidad de términos relevantes; de aquí que siempre, tanto la exhaustividad *como la* precisión *toman valor 1. Para ello, se asume que en el fichero de sinónimos están las diferentes formas en que se puede encontrar escrita un mismo término en la base de datos, y que además se tienen agrupados diferentes términos que aunque tienen diferencia sintáctica, significan lo mismo semánticamente. De esta forma, todas las tuplas que sean relevantes en una consulta determinada, el sistema es capaz de recuperarlas con este tratamiento semántico, mediante la operación correcta para dar la respuesta.*

De forma general, en todos los ejemplos implementados este es el patrón que se repite, donde los resultados obtenidos con el operador de igualdad son nulos, con el operador *Like* considerablemente mejores, y con el modelo propuesto, mejores aún.

En el siguiente ejemplo que aparece en la tabla 7.4, se muestra una consulta que basa su respuesta en la semántica de los términos, además de en su sintaxis. Para ello, se recuperan términos que semánticamente tienen el mismo significado aunque no se escriben de igual forma. El sistema realiza la solución utilizando su fichero de sinónimos, para encontrar los términos que semánticamente significan lo mismo.

Ejemplo 32

Determinar el número de pacientes que han llegado a Urgencias con una 'FRACTURA'.

Como se observa en el ejemplo de la tabla 7.4, se intenta recuperar el número de pacientes que han llegado urgencias con una 'FRACTURA'. Para el caso del operador de igualdad, no se retorna ninguna tupla. Esto se debe a que este operador busca las tuplas que contienen exactamente este término, y en este caso ninguna tupla de la base de datos cumple esta condición. Debido a esto, *la exhaustividad* y la *precisión* toman valor 0.

Con el operador *Like,* se ha especificado el patrón de búsqueda *%FRACTURA %,* que significa encontrar todas las tuplas que contienen el término *FRACTURA*, sin importar la posición que ella ocupe al inicio, medio o final de dicha tupla. En este caso se recuperan 46 tuplas que contienen el término que se busca. Debido a esto la *exhaustividad* toma valor 0.92 *(46/50)* y la *precisión* toma valor 1.

Conjunto experimental: Conjunto3 **Cantidad de tuplas relevantes:** 50
Solución con el operador de igualdad (=) tradicional
Select count(*) **From** turgencias **Where** motivo = 'FRACTURA'; *Cantidad de tuplas recuperadas:* 0 —> *exhaustividad* = 0 y *precisión* = 0
Solución con el operador de comparación tradicional en textos (Like)
Select count(*) **From** turgencias **Where** motivo Like '%FRACTURA%'; *Cantidad de tuplas recuperadas:* 46 —> *exhaustividad* = 0.92 y *precisión* = 1
Solución con el modelo propuesto
Select count(*) **From** turgencias **Where** motivo **Where** acoplamiento_debilTDA (TDAmotivo, 'FRACTURA', TRUE) = TRUE; *Cantidad de tuplas recuperadas:* 50 —> *exhaustividad* = 1 y *precisión* = 1

Tabla 7.4: Comparativas de operadores en modelo booleano (Ejemplo 32).

Para el caso de la solución implementada utilizando nuestro modelo, como se observa se obtienen 50 tuplas. En este caso, se recuperan 4 tuplas más que con el operador *Like*. Esto se debe a que se recuperan las tuplas que contienen el término *ROTURA*, que según el conocimiento de expertos, tiene el mismo significado semántico. De esta forma, el usuario recibe información de todas las tuplas que contienen el término que está buscando más las que, aunque no se escriban igual, tienen el mismo significado y pueden resultar de su interés.

En este caso, se utilizó la función *acoplamiento_ debilTDA* para localizar todas las tuplas que contienen el término que se desea encontrar. Como se recuperan todas las tuplas que son relevantes, la *exhaustividad* y la *precisión* toman valor 1.

Una vez discutidos estos dos ejemplos de consultas con un sólo término, a continuación se verán ejemplos de consultas con más de un término cuando interesa recuperar la frase completa.

7.4.1.2. Evaluación de consultas que contengan la frase completa

Ejemplo 33

Conjunto experimental: Conjunto3 **Cantidad de tuplas relevantes:** 6
Solución con el operador de igualdad (=) tradicional
Select numero, fnac **From** turgencias **Where** motivo = 'FRACTURA CADERA DERECHA'; *Cantidad de tuplas recuperadas:* 0 —> *exhaustividad* = 0 y *precisión* = 0
Solución con el operador de comparación tradicional en textos (Like)
Select numero, fnac **From** turgencias **Where** motivo Like '%FRACTURA CADERA DERECHA%'; *Cantidad de tuplas recuperadas:* 0 —> *exhaustividad* = 0 y *precisión* = 0
Solución con el modelo propuesto
Select numero, fnac **From** turgencias **Where** motivo **Where** acoplamiento_fuerteTDA (TDAmotivo, 'FRACTURA, CADERA, DERECHA', TRUE) = TRUE; *Cantidad de tuplas recuperadas:* 6 —> *exhaustividad* = 1 y *precisión* = 1

Tabla 7.5: Comparativas de operadores en modelo booleano (Ejemplo 33).

Determinar el número y la fecha de nacimiento de los pacientes que llegan a Urgencias con 'FRACTURA CADERA DERECHA'.

Como se observa en el ejemplo de la tabla 7.5, se intenta recuperar el número y la fecha de nacimiento de los pacientes que llegan a urgencia con una fractura de cadera derecha. Para el caso del operador de igualdad no se retorna ninguna tupla. Esto se debe a que este operador busca las tuplas que contienen exactamente esta frase, y en este caso ninguna tupla de la base de datos cumple esta condición. Debido a esto *la exhaustividad* y la *precisión* toman valor 0.

Con el operador *Like,* se ha especificado el patrón de búsqueda *%FRACTURA CADERA DERECHA %,* que significa encontrar todas las tuplas que contienen la frase *FRACTURA CADERA DERECHA*, sin importar la posición que ella ocupe al inicio, medio o final de dicha tupla. En este caso, no se recupera ninguna tupla pues no aparece el patrón antes descrito en ninguna tupla de la base de datos. Debido a esto *la exhaustividad* y la *precisión* toman valor 0.

Para el caso de la solución implementada utilizando nuestro modelo, como se observa se obtienen 6 tuplas. Este resultado es superior al de los dos operadores tradicionales. Como se puede observar, se utiliza el método *acoplamiento_fuerteTDA* para localizar todas las tuplas que contienen los tres términos que se desea encontrar. Para nuestro modelo, como se recuperan todas las tuplas que son relevantes, la *exhaustividad* y la *precisión* toman valor 1.

Ejemplo 34

Determinar la cantidad de urgencias que se han recibido, cuyo juicio se acopla totalmente a 'DOLOR ABDOMINAL'con índice mayor que 0.2.

En la tabla 7.6 aparecen los resultados que se obtienen en este ejemplo 34. Como se observa, los resultados son análogos al ejemplo anterior. En este caso utilizando la función *indice_ acoplamiento_fuerteTDA*.

Conjunto experimental: Conjunto4 **Cantidad de tuplas relevantes:** 436
Solución con el operador de igualdad (=) tradicional
No es posible responder a esta consulta con el operador de igualdad *Cantidad de tuplas recuperadas:* 0 —> *exhaustividad* = 0 y *precisión* = 0
Solución con el operador de comparación tradicional en textos (Like)
No es posible responder a esta consulta con el operador Like *Cantidad de tuplas recuperadas:* 0 —> *exhaustividad* = 0 y *precisión* = 0
Solución con el modelo propuesto
Select count(*) **AS** Cantidad **From** turgencias **Where** indice_acoplamiento_fuerteTDA (TDAjuicio, 'DOLOR, ABDOMINAL', TRUE) >0.2; *Cantidad de tuplas recuperadas:* 436 —> *exhaustividad* = 1 y *precisión* = 1

Tabla 7.6: Comparativas de operadores en modelo booleano (Ejemplo 34).

De forma similar al apartado anterior, en el siguiente, se muestran ejemplos, de consultas con más de un término, cuando no interesa recuperar la frase completa, si no, tuplas con al menos un términos de los que conforman dicha frase.

Conjunto experimental: *Conjunto2* **Cantidad de tuplas relevantes:** 122
Solución con el operador de igualdad (=) tradicional
Select numero, fecha **From** tintervenciones **Where** ipropuesta = 'PIE IZQUIERDO'; *Cantidad de tuplas recuperadas:* 0 —> *exhaustividad* = 0 y *precisión* = 0
Solución con el operador de comparación tradicional en textos (Like)
***Con patrón de búsqueda:* %PIE IZQUIERDO %** **Select** numero, fecha **From** tintervenciones **Where** ipropuesta Like '%PIE IZQUIERDO %'; *Cantidad de tuplas recuperadas:* 31 —> *exhaustividad* = 0.25 y *precisión* = 1 ***Con patrón de búsqueda:* %PIE IZQ %** **Select** numero, fecha **From** tintervenciones **Where** ipropuesta Like '%PIE IZQ %'; *Cantidad de tuplas recuperadas:* 68 —> *exhaustividad* = 0.55 y *precisión* = 1
Solución con el modelo propuesto
Select numero, fecha **From** tintervenciones **Where** *acoplamiento_ debilTDA* (TDAipropuesta, 'PIE, IZQUIERDO', TRUE) = TRUE; *Cantidad de tuplas recuperadas:* 122 —> *exhaustividad* = 1 y *precisión* = 1

Tabla 7.7: Comparativas de operadores en modelo booleano (Ejemplo 35).

7.4.1.3. Evaluación de consultas que contengan parcialmente los términos

Ejemplo 35

Obtener el número de paciente y la fecha en que se le ha propuesto la intervención, si se le ha propuesto una intervención en su 'PIE IZQUIERDO'.

Como se observa en el ejemplo de la tabla 7.7, se intenta recuperar el número de paciente y la fecha en que se le ha propuesto la intervención, si se le ha propuesto una intervención en su pie izquierdo. Para el caso del operador de igualdad, no se retorna ninguna tupla. Esto se debe a que este operador busca las tuplas que contienen exactamente esta frase, y en este caso ninguna tupla de la base de datos

cumple esta condición. Debido a esto, *la exhaustividad* y la *precisión* toman valor 0.

Con el operador *Like,* como se ha especificado más de un término de búsqueda, se han especificado dos patrones diferentes para discutir la diferencia entre ellos. Como se comentó anteriormente, de la calidad del patrón que sea capaz de construir el usuario, depende la cantidad de información que puede ser recuperada.

El patrón de búsqueda *%PIE IZQUIERDO %,* significa encontrar todas las tuplas que contienen la frase *PIE IZQUIERDO*, sin importar la posición que ella ocupe al inicio, medio o final de dicha tupla. En este caso, se recuperan 31 tuplas de pacientes a los que se les ha diagnosticado alguna intervención en el pie izquierdo. Para este patrón de búsqueda del *Like,* la *exhaustividad* toma un valor de 0.26 (*31/122*) y la *precisión* toma valor 1.

El patrón de búsqueda *%PIE IZQ %,* implica encontrar todas las tuplas que contienen la frase *PIE IZQ*, sin importar la posición que ella ocupe al inicio, medio o final de dicha tupla. En este caso se recuperan 68 tuplas de pacientes a los que se les ha diagnosticado alguna intervención en el pie izquierdo. Para este patrón de búsqueda del *Like,* la *exhaustividad* toma un valor de 0.55 (*68/122*) y la *precisión* toma valor 1.

Para el caso de la solución implementada utilizando nuestro modelo, como se observa se obtienen 122 tuplas. Este resultado es considerablemente superior al de los dos operadores tradicionales. Como se puede observar, se utiliza el método *acoplamiento_ debilTDA* para localizar todas las tuplas que contienen al menos un término de los que se desea encontrar. Para nuestro modelo, como se recuperan todas las tuplas que son relevantes, la *exhaustividad* y la *precisión* toman valor 1.

Ejemplo 36

Determinar la cantidad de urgencias que se han recibido, cuyo juicio se acopla parcialmente a 'DOLOR ABDOMINAL'con índice mayor que 0.4.

Como se observa en el ejemplo de la tabla 7.8, se intenta recuperar la cantidad de urgencias que se han recibido, cuyo juicio se acopla parcialmente a dolor abdominal con índice mayor que 0.4.

Conjunto experimental: Conjunto4 **Cantidad de tuplas relevantes:** 1741
Solución con el operador de igualdad (=) tradicional
No es posible responder a esta consulta con el operador de igualdad *Cantidad de tuplas recuperadas:* 0 —> *exhaustividad* = 0 y *precisión* = 0
Solución con el operador de comparación tradicional en textos (Like)
No es posible responder a esta consulta con el operador Like *Cantidad de tuplas recuperadas:* 0 —> *exhaustividad* = 0 y *precisión* = 0
Solución con el modelo propuesto
Select count(*) **AS** Cantidad **From** turgencias **Where** indice_acoplamiento_debilTDA (TDAjuicio, 'DOLOR, ABDOMINAL', TRUE) >0.4; *Cantidad de tuplas recuperadas:* 1741 —> *exhaustividad* = 1 y *precisión* = 1

Tabla 7.8: Comparativas de operadores en modelo booleano (Ejemplo 36).

En este caso, este tipo de consultas no puede ser resuelta ni con el operador de igualdad ni con el operador *Like*. Esto se debe a que con ninguno de los dos operadores se puede determinar el índice de acoplamiento parcial de la frase que se busca con las tuplas de la base de datos. Debido a esto, en ambos casos la *exhaustividad* y la *precisión* toman valor 0.

Para el caso de la solución implementada utilizando nuestro modelo, como se observa se obtienen 1741 tuplas. Se utiliza el método *indice_ acoplamiento_ debilTDA* para localizar todas las tuplas que contienen los dos términos con un índice mayor que 0.4. En este caso, se recuperan todas las tuplas que son relevantes y por ello la *exhaustividad* y la *precisión* toman valor 1.

7.4.2. Exhaustividad y precisión en el modelo difuso

A continuación se desarrollan algunos ejemplos para detallar el cálculo de le *exhaustividad* y *precisión difusas* utilizando operaciones del modelo propuesto. Para ello, se discutirán algunas variantes que aparecen derivadas de la flexibilidad de

dicho modelo. Hay que tener en cuenta, no obstante, que el cálculo automático de la relevancia difusa subjetiva del usuario no es viable, por el tamaño de la base de datos. Por esta razón, nosotros hemos incluido ejemplos en los que el número de tuplas resultante sí se pueden evaluar, quedando pendiente como trabajo futuro una propuesta de cálculo de u_i de forma, al menos, semiautomática.

Con el siguiente ejemplo, se discuten las posibles variantes a utilizar con nuestro modelo, para realizar el cálculo de las medidas. También se detallará el proceso del cálculo de forma automática, de la *evaluación del sistema* (s_i) que puede ser realizado de tres formas:

1. Presencia / ausencia de los términos de la consulta en las tuplas de la base de datos (modelo booleano).
2. Utilizando la operación *indice_ acoplamiento_ fuerteTDA*. Dicha operación encuentra el grado de acoplamiento de la frase buscada con el TDA de cada tupla, cuando interesa encontrar todos los términos buscados en dichas tuplas.
3. Utilizando la operación *indice_ acoplamiento_ debilTDA*. Dicha operación encuentra el grado de acoplamiento de la frase buscada con el TDA de cada tupla, cuando interesa encontrar al menos un término de los buscados en dichas tuplas.

Ejemplo 37

Seleccionar las tuplas que contienen la frase 'fractura femur izquierdo'.

Para calcular el s_i correspondiente a buscar la frase '*fractura femur izquierdo*' en una tupla que tuviera la siguiente estructura:

Tupla original = '*fractura abierta de femur izq + herida del brazo izquier*'

Tupla modificada = '*fractura abierta femur izquierdo + herida brazo izquierdo*'

TDA = $\{\{fractura, abierta, femur, izquierdo\}, \{herida, brazo, izquierdo\}\}$

aparecen las tres variantes siguientes:

1. *Cálculo del acoplamiento mediante presencia/ausencia (modelo booleano):* De esta forma se obtiene el valor de s_i a partir de calcular el máximo entre los

valores t_1 y t_2. El valor t_1 se refiere al grado de acoplamiento de la frase buscada, con el primer conjunto del TDA ($\{fractura, abierta, femur, izquierdo\}$). Por su parte, el valor t_2 se refiere al grado de acoplamiento de dicha frase, con el segundo conjunto del TDA ($\{herida, brazo, izquierdo\}$). Dichos acoplamientos son calculados realizando la intersección de cada término de la frase buscada, con el conjunto correspondiente del TDA. Si el término de la frase buscada se encuentra en el conjunto del TDA, se le da valor 1 de acoplamiento; si no está, se le asigna valor 0. Cada t_i será calculado como el valor mínimo entre todos los acoplamientos calculados entre cada término por separado de la frase buscada y el conjunto i del TDA.

2. *Cálculo del acoplamiento mediante el indice_ acoplamiento_ fuerteTDA:* Como se discutió anteriormente, existen dos formas de determinar el valor del índice de acoplamiento fuerte. La primera es determinando el índice de acoplamiento de una frase con todos los conjuntos del TDA, siempre que dicha frase aparece completamente incluida en el conjunto con el que se calcula su acoplamiento. La segunda es determinando el máximo grado de acoplamiento que se obtiene para cada conjunto que forma el TDA, si la frase buscada aparece completamente incluida en el conjunto con el que se calcula su acoplamiento. Teniendo esto en cuenta, se puede calcular s_i a partir de estas dos formas de determinar el índice de acoplamiento fuerte. En este caso, no se calcula el acoplamiento de cada término de la frase con cada conjunto del TDA, si no que se calcula el acoplamiento de la frase completa con cada conjunto del TDA, tal y como define el modelo esta operación.

3. *Cálculo del acoplamiento mediante el indice_ acoplamiento_ debilTDA:* Como se discutió anteriormente, existen dos formas de determinar el valor del índice de acoplamiento débil. La primera es determinando el índice de acoplamiento de una frase con todos los conjuntos del TDA, si dicha frase aparece parcialmente incluida en el conjunto con el que se calcula su acoplamiento. La segunda es determinando el máximo grado de acoplamiento que se obtiene para cada conjunto que forma el TDA, si la frase buscada aparece parcialmente incluida en el conjunto con el que se calcula su acoplamiento. Teniendo esto en cuenta, se puede calcular s_i a partir de estas dos formas de determinar el índice de acoplamiento débil. En este caso no se calcula el acoplamiento de cada término de la frase con cada conjunto del TDA, si no que se calcula, el acoplamiento de la frase completa con cada conjunto del TDA, tal y como define el modelo esta operación.

A continuación se detalla cómo se calcula s_i en las tres variantes antes descritas.

- **Utilizando el modelo booleano para el cálculo del grado de acoplamiento**

Calculando el acoplamiento de la frase buscada con el primer conjunto del TDA:

$t_1 = min(\{fractura\} \cap \{fractura, abierta, femur, izquierdo\},$
$\{femur\} \cap \{fractura, abierta, femur, izquierdo\}, \{izquierdo\} \cap \{fractura, abierta, femur, izquierdo\})$

$t_1 = min(1, 1, 1) = 1$

Calculando el acoplamiento de la frase buscada con el segundo conjunto del TDA:

$t_2 = min(\{fractura\} \cap \{herida, brazo, izquierdo\},$
$\{femur\} \cap \{herida, brazo, izquierdo\}, \{izquierdo\} \cap \{herida, brazo, izquierdo\})$

$t_2 = min(0, 0, 1) = 0$

Calculando s_i en función de t_1 y t_2:

$s_i = max(t_1, t_2) = max(1, 0) = 1$

Como se observa, por esta vía de cálculo, el s_i calculado toma valor 1. Este resultado es correcto con lo expresado anteriormente: que esta forma de cálculo se puede utilizar cuando interesa encontrar la frase buscada completamente en el TDA de la tupla (*operación AND entre todos los términos de la frase buscada*). Como se observa, la frase buscada se acopla completamente con el primer conjunto del TDA, de aquí que s_i tome valor 1.

A continuación se realiza el mismo cálculo, ahora utilizando las dos variantes que usa la función *indice_ acoplamiento_ fuerteTDA*.

- **Utilizando la operación *indice_ acoplamiento_ fuerteTDA* obteniendo el índice de acoplamiento con todos los conjuntos del TDA**

En todos los ejemplos que se discutirán, para ganar en legibilidad, la operación *indice_ acoplamiento_ fuerteTDA* se sustituirá por el término *ind_ acopl_ fuerte.*

Calculando el acoplamiento de la frase buscada con el primer conjunto del TDA:

$t_1 = ind\_acopl\_fuerte(\{fractura, abierta, femur, izquierdo\},$
$'fractura, femur, izquierdo', true)$

t_1= *3/4 = 0.75*

Calculando el acoplamiento de la frase buscada con el segundo conjunto del TDA:

$t_2 = min(ind\_acopl\_fuerte(\{herida, brazo, izquierdo\},$
$'fractura, femur, izquierdo', true)$

t_2= *0 (no se acopla completamente la frase buscada con el conjunto del TDA)*

Calculando s_i en función de t_1 y t_2:

s_i= *(3/4 + 0)/2 = 0.375*

Como se observa, por esta vía de cálculo, el s_i calculado toma valor 0.375. Este resultado es correcto con lo expresado anteriormente: que esta forma de cálculo se puede utilizar cuando interesa encontrar la frase buscada completamente incluida en el TDA de la tupla. Como se observa, la frase buscada se acopla completamente con uno de los conjuntos que forman el TDA con un índice de acoplamiento de 3/4 pero no se acopla completamente con el otro conjunto del TDA. Esto hace que el resultado final sea menor, pues se está calculando el índice de acoplamiento fuerte con los dos conjuntos del TDA.

A continuación se utiliza la misma operación índice de acoplamiento fuerte, pero con la modalidad donde se calcula dicho índice como el máximo entre los acoplamientos de la frase buscada, con cada conjunto que forman el TDA.

- **Utilizando la operación *indice_ acoplamiento_ fuerteTDA* obteniendo el índice máximo de acoplamiento con todos los conjuntos del TDA**

En este caso el cálculo de t_1 y t_2 son ídem a los del caso anterior de este mismo índice, sólo varía la forma de calcular s_i. Por ende:

t_1= *3/4 = 0.75*

t_2= *0 (no se acopla completamente la frase buscada con el conjunto del TDA)*

Calculando s_i en función de t_1 y t_2:

$s_i = max(3/4, 0) = 0{,}75$

Como se observa, por esta vía de cálculo, el s_i calculado toma valor 0.75. Este resultado es correcto con lo expresado anteriormente: que esta forma de cálculo se puede utilizar cuando interesa encontrar la frase buscada completamente incluida en el TDA de la tupla. Como se observa, la frase buscada se acopla completamente con uno de los conjuntos que forman el TDA con un índice de acoplamiento de 3/4 pero no se acopla completamente con el otro conjunto del TDA. Esto hace que al calcular s_i buscando conocer el mayor grado de acoplamiento que tiene con todos los conjuntos que forman el TDA, el resultado final sea mayor que en el caso anterior. De esta forma, se está retornando el mayor índice de acoplamiento, de la frase con cada uno de los conjuntos que forma en TDA, o sea, el índice sobre el conjunto con que mejor se acopla completamente la frase buscada.

A continuación se realiza el mismo cálculo, ahora utilizando las dos variantes que usa la función *indice_ acoplamiento_ debilTDA*.

- **Utilizando la operación *indice_ acoplamiento_ debilTDA* obteniendo el índice de acoplamiento con todos los conjuntos del TDA**

En todos los ejemplos que se discutirán, para ganar en legibilidad, la operación *indice_ acoplamiento_ debilTDA* se sustituirá por el término *ind_ acopl_ debil*.

Calculando el acoplamiento de la frase buscada con el primer conjunto del TDA:

$t_1 = ind\_acopl\_debil(\{fractura, abierta, femur, izquierdo\},' fractura, femur, izquierdo', true)$

t_1= 3/4 = 0.75

Calculando el acoplamiento de la frase buscada con el segundo conjunto del TDA:

$t_2 = min(ind\_acopl\_debil(\{herida, brazo, izquierdo\},' fractura, femur, izquierdo', true)$

t_2= 1/3 = 0.33 (se acopla completamente la frase buscada con el conjunto del TDA)

Calculando s_i en función de t_1 y t_2:

s_i= (3/4 + 1/3)/2 = 0.54

Como se observa, por esta vía de cálculo, el s_i calculado toma valor 0.54. Este resultado es superior al obtenido por similar vía, para le índice de acoplamiento fuerte, pues en esta variante se tiene en cuenta el acoplamiento parcial que ocurre

entre la frase buscada y el segundo conjunto del TDA a través del término ' *izquierdo* '. Algo consecuente con lo expresado: que esta forma de cálculo se puede utilizar cuando interesa encontrar la frase buscada parcialmente incluida en el TDA de la tupla. En este caso, como se busca el acoplamiento general con todos los conjuntos que forman el TDA, al dividir por 2 que es la cantidad de conjuntos, el índice general de acoplamiento disminuye.

A continuación, se utiliza la misma operación índice de acoplamiento débil, pero con la modalidad donde se calcula dicho índice como el máximo entre los acoplamientos de la frase buscada, con cada conjunto que forman el TDA.

- **Utilizando la operación *indice_ acoplamiento_ debilTDA* obteniendo el índice máximo de acoplamiento con todos los conjuntos del TDA**

En este caso, el cálculo de t_1 y t_2 son ídem a los del caso anterior de este mismo índice, sólo varía la forma de calcular s_i. Por ende:

$t_1 = 3/4 = 0.75$

$t_2 = 1/3 = 0.33$

Calculando s_i en función de t_1 y t_2:

$s_i = max(3/4, 1/3) = 0{,}75$

Como se observa, por esta vía de cálculo, el s_i calculado toma valor 0.75. Este valor es el mismo que obtiene el índice de acoplamiento fuerte para esta vía de cálculo con el máximo grado de acoplamiento. Esto ocurre así, pues en este ejemplo, la frase buscada está completamente incluida en uno de los conjuntos del TDA.

En este caso, la frase buscada se acopla completamente con uno de los conjuntos que forman el TDA con un índice de acoplamiento de 3/4, y parcialmente con el otro conjunto con un índice de 1/3. Esto hace que al calcular s_i buscando conocer el mayor grado de acoplamiento que tiene con todos los conjuntos que forman el TDA, el resultado final sea 3/4. De esta forma, se está retornando el mayor índice de acoplamiento, de la frase con cada uno de los conjuntos que forma en TDA, o sea, el índice sobre el conjunto con que mejor se acopla completamente la frase buscada.

Hasta aquí, se ha ejemplificado el cálculo del grado de acoplamiento que brinda el sistema (s_i), por las tres variantes que aparecen para su cálculo. A continuación, se resumen los casos posibles de consultas por el usuario, y la variante del cálculo de s_i que será utilizada para resolver cada caso.

Como se discutió anteriormente los tres casos posibles son:

1. *Consultas con un sólo término:* Para este tipo de consulta, interesa recuperar las tuplas que contengan el término que se busca. Para determinar el valor de $s_i$se utilizará en este caso el índice de acoplamiento fuerte, en las dos variantes discutidas anteriormente.

2. *Consultas que contengan la frase completa:* Para este tipo de consulta, interesa recuperar las tuplas que contengan completamente la frase buscada. Es del tipo de consultas representadas por un operador *AND* entre todos los términos que aparecen en la consulta. Para determinar el valor de s_i se utilizará en este caso el índice de acoplamiento fuerte, en las dos variantes discutidas anteriormente.

3. *Consultas que contengan parcialmente los términos:* Para este tipo de consulta, interesa recuperar las tuplas que contengan al menos un términos de los que contiene la frase buscada. Es del tipo de consultas representadas por un operador *OR* entre todos los términos que aparecen en la consulta. Para determinar el valor de $s_i$se utilizará en este caso el índice de acoplamiento débil, en las dos variantes discutidas anteriormente.

Para el caso de todos los ejemplos que restan en esta sección, se utilizará como conjunto experimental el *Conjunto2*, referente al atributo *IPropuesta* de la tabla *TIntervenciones* en la base de datos médica.

Las estructura del ejemplo para cada uno de los tres tipos de consultas posible, será calcular s_i por las dos variantes antes mencionadas, para cada operador y el modelo propuesto. Al final de cada variante de cálculo de s_i, se muestra una tabla comparativa de la *exhaustividad* y la *precisión* difusas por cada operador.

7.4.2.1. Evaluación de consultas con un sólo término

A continuación, se formulará una consulta donde se recupera un sólo término, y se muestra el cálculo del s_i, y *la exhaustividad* y *precisión* difusas, para los operadores

de igualdad y *Like*, y con el modelo propuesto. Para el cálculo de s_i, serán utilizadas las dos variantes discutidas para el cálculo del índice de acoplamiento fuerte.

Ejemplo 38

Seleccionar las tuplas que contienen el término 'muscular'.

Para calcular los s_i correspondientes a buscar el término '*muscular*', se asume que existen sólo las dos tuplas siguiente que contienen el término buscado:

	Tupla modificada	**TDA**
Tupla 1	*Cura + drenaje absceso muscular*	$\{\{Cura\}, \{drenaje, absceso, muscular\}\}$
Tupla 2	*Biopsia muscular*	$\{Biopsia, muscular\}$

A continuación, se realiza el mismo cálculo de s_i, para cada una de las dos variantes que usa la operación *indice_ acoplamiento_ fuerteTDA*. También se realiza el cálculo para los operadores de igualdad y *Like*, y con el modelo propuesto.

Variante 1: *Utilizando la operación* indice_acoplamiento_fuerteTDA *obteniendo el índice de acoplamiento con todos los conjuntos del TDA.*

- ***Operador de igualdad:***

Para el operador de igualdad ninguna de las dos tuplas serían retornadas, dado que no contienen únicamente el término *muscular*. Por esto, tanto la *exhaustividad* como la *precisión* difusas, serían igual a 0.

- ***Operador Like y modelo propuesto***

En este caso, como la consulta es de un sólo término, el operador *Like* utilizando el patrón de búsqueda %*muscular* % tiene el mismo comportamiento que el modelo propuesto, por eso se agrupan sus cálculos. En este caso se realizará el cálculo por el modelo propuesto.

Calculando el valor de $s_1$correspondiente al acoplamiento del término buscado con el TDA de la primera tupla:

Calculando el acoplamiento del término buscado con el primer conjunto del TDA:

$t_1 = ind\_acopl\_fuerte(\{Cura\},'muscular', true)$

t_1= *0 (el término no se encuentra incluido en el TDA)*

Calculando el acoplamiento del término buscado con el segundo conjunto del TDA:

$t_2 = min(ind\_acopl\_fuerte(\{drenaje, absceso, muscular\},'muscular', true)$

t_2= *1/3 (se acopla completamente la frase buscada con el conjunto del TDA)*

Calculando s_1 en función de t_1 y t_2:

s_1= *(0 + 1/3)/2 = 0.16*

Calculando el valor de s_2*correspondiente al acoplamiento del término buscado con el TDA de la segunda tupla:*

Calculando el acoplamiento del término buscado el TDA:

$t_1 = ind\_acopl\_fuerte(\{Biopsia, muscular\},'muscular', true)$

t_1= *1/2 = 0.5 (el término se encuentra incluido en el TDA)*

Como el TDA tiene un sólo conjunto, el valor de s_2 es el mismo que el de t_1, por ende s_2 toma valor 0.5.

A continuación, se utiliza la misma operación índice de acoplamiento fuerte, pero con la modalidad donde se calcula dicho índice como el máximo entre los acoplamientos de la frase buscada, con cada conjunto que forman el TDA.

Variante 2: *Utilizando la operación* indice_acoplamiento_fuerteTDA *obteniendo el índice máximo de acoplamiento con todos los conjuntos del TDA.*

En este caso el cálculo de t_1 y t_2 son ídem a los del caso anterior de este mismo índice, sólo varía la forma de calcular s_i. Por ende:

Para la primera tupla:

t_1= *0*

t_2= *1/3*

Calculando s_1 en función de t_1 y t_2:

$s_1 = max(0, 1/3) = 1/3 = 0.33$

Operador de igualdad	**Operador Like**	**Modelo propuesto**
No se recupera ninguna tupla	s_i= *{0.16,0.5}* u_i=*{0.3,0.5}*	s_i= *{0.16,0.5}* u_i=*{0.3,0.5}*
exhaustividad difusa **0**	*exhaustividad difusa* 0.66/ 0.8 = **0.82**	*exhaustividad difusa* 0.66/ 0.8 = **0.82**
precisión difusa **0**	*precisión difusa* 0.66/ 0.66 = **1**	*precisión difusa* 0.66/ 0.66 = **1**

Tabla 7.9: *Exhaustividad* y *relevancia* difusas en consultas con un sólo término y cálculo de s_i con índice de acoplamiento fuerte promediado.

Operador de igualdad	**Operador Like**	**Modelo propuesto**
No se recupera ninguna tupla	s_i= *{0.33,0.5}* u_i=*{0.3,0.5}*	s_i= *{0.33,0.5}* u_i=*{0.3,0.5}*
exhaustividad difusa **0**	*exhaustividad difusa* 0.8/ 0.8 = **1**	*exhaustividad difusa* 0.8/ 0.8 = **1**
precisión difusa **0**	*precisión difusa* 0.8/ 0.83 = 0.96	*precisión difusa* 0.8/ 0.83 = 0.96

Tabla 7.10: *Exhaustividad* y *relevancia* difusas en consultas con un sólo término y cálculo de s_i con máximo índice de acoplamiento fuerte.

Para la segunda tupla:

t_1= *1/2*

Como el TDA tiene un sólo conjunto, el valor de s_2 es el mismo que el de t_1, por ende s_2 toma valor 0.5.

Asumiendo que el valor de relevancia que le da el usuario (u_i) a la primera tupla es de $u_1 = 0.3$ y que para la segunda tupla es $u_2 = 0.5$, a continuación se muestran las tablas 7.9 y 7.10, resultantes por ambas vías de calcular s_i. En dichas tablas se muestra el cálculo de la *exhaustividad* y la *precisión* difusas para cada operador y el modelo propuesto.

Un primer análisis de las tablas antes mencionadas, muestra claramente la desventaja del operador de igualdad para el trabajo con atributos textuales en base de datos. Dado que el término buscado aparece acompañado de otros, en ambas tuplas, dicho operador no recupera ninguna de las tuplas analizadas. En el caso particular de este tipo de consultas sobre un sólo término, el modelo propuesto tiene exactamente el mismo comportamiento que el operador *Like* con el patrón de búsqueda *%término %* . Como ya se ha comentado, dicho patrón encuentra las tuplas que contienen el término buscado en cualquier posición.

Del análisis comparativo de los resultados entre ambas tablas, se puede observar cómo cuando se calculan los s_i teniendo en cuenta el promedio del valor que aporta cada conjunto del TDA, la *exhaustividad difusa* es menor que por la variante en la que s_i es calculado teniendo en cuenta el máximo de los valores que aporta cada conjunto del TDA. Esto se debe a que, cuando se hace con la primera variante, el valor de s_i da un valor menor que el de u_i para esa tupla. Por ello, al calcular la sumatoria del $min(s_i, u_i)$, este valor da menor que por la otra variante, ya que en ésta, el s_i es calculado promediando los valores de t_i. De esta forma, siempre s_i será menor que en el otro caso, en el que se calcula el máximo entre los valores t_i para obtener dicho coeficiente s_i.

De forma general, se puede decir que cuando se utiliza para el cálculo de s_i la variante de calcular el índice de acoplamiento fuerte, utilizando el promedio de los t_i se obtiene mayor precisión y menor exhaustividad difusa. Por el contrario, cuando se calcula s_i a partir del máximo valor de t_i se obtiene mayor exhaustividad, pero mejor precisión. Este resultado es muy coherente, pues en la primera de las dos variantes, el resultado es más preciso, dado que la forma en que se calcula s_i es más cercana a la valoración general que realiza el usuario de toda la tupla. Esto, dado que tiene en cuenta todo los conjuntos del TDA.

Para el otro caso, la precisión es menor pues se da el criterio del mejor acoplamiento de la frase buscada con un sólo conjunto del TDA (*el que de el máximo valor de* t_i), y se descarta la información que puede aparecer en otro de los conjuntos de dicho TDA. Este resultado es lógico y coherente también.

En el próximo apartado se realiza un análisis similar al del aparto anterior, pero ahora con consultas de más de un término donde interesa que aparezca la frase completa.

7.4.2.2. Evaluación de consultas que contengan la frase completa

A continuación se formulará una consulta donde se recuperan varios términos, y se muestran los valores de s_i (calculados de la misma forma que en apartado anterior), y *la exhaustividad* y *precisión* difusas, para los operadores de igualdad y *Like*, y con el modelo propuesto. Para el cálculo de s_i, serán utilizadas las dos variantes discutidas para el cálculo del índice de acoplamiento fuerte. Se utiliza este índice porque es el que garantiza que la frase buscada aparezca completamente incluida en las tuplas recuperadas. Esto sería el equivalente a cuando el usuario introduce varios términos unidos por el operador lógico *AND*.

Ejemplo 39

Seleccionar las tuplas que contienen la frase Ámputacion dedo'.

Para calcular los s_i correspondientes a buscar la frase '*Amputacion dedo*', se asume que existen sólo las dos tuplas siguientes que contienen los términos buscados:

	Tupla modificada	**TDA**
Tupla 1	*Amputacion dedo*	$\{Amputacion, dedo\}$
Tupla 2	*Amputacion total dedo*	$\{Amputacion, total, dedo\}$

A continuación se realiza el mismo cálculo de s_i, *exhaustividad y precisión* difusas, para cada una de las dos variantes que usa la operación *indice_ acoplamiento_ fuerteTDA*. También se realiza el cálculo para los operadores de igualdad y *Like*, y con el modelo propuesto.

Variante 1: *Utilizando la operación* indice_acoplamiento_fuerteTDA *obteniendo el índice de acoplamiento con todos los conjuntos del TDA.*

- ***Operador de igualdad:***

En este caso, el operador de igualdad recupera la primera tupla y la segunda no, por ello, los valores de s_i en este caso serían $s_1 = 1$ y $s_2 = 0$.

- ***Operador Like***

Con el operador *Like*, utilizando el patrón de búsqueda %*Amputación dedo* % recupera la primera tupla y la segunda no; por ello, los valores de s_i en este caso serían s_1= 1 y $s_2 = 0$.

Cabe destacar, que si el usuario fuera capaz de construir un patrón de búsqueda mejor, los resultados se podrían igualar, para este ejemplo, con los resultados que logra el modelo propuesto. En cualquier caso, esto dependería de los conocimientos de lenguaje SQL *del usuario y su habilidad para construir dicho patrón. En este sentido, el modelo propuesto es considerablemente superior, pues el usuario puede expresar los términos para sus consultas sin importar el orden, y se recuperan todas las tuplas que los contiene. Esto, aunque en dichas tuplas no aparezcan en el mismo orden en que los introduce el usuario.*

- ***Modelo propuesto***

Mediante el modelo propuesto, explotando esta posibilidad comentada anteriormente, de recuperar las tuplas aunque no se encuentren en el mismo orden los términos que en la frase buscada, se recuperan ambas tuplas. Debido a esto, los valores de s_i en este caso serían s_1= 1 y $s_2 = 2/3$.

Variante 2: *Utilizando la operación* indice_acoplamiento_fuerteTDA *obteniendo el índice máximo de acoplamiento con todos los conjuntos del TDA.*

En este caso, como el TDA contiene un sólo conjunto, los valores de s_i serán los mismos que los calculados con la variante 1.

Asumiendo que el valor de relevancia que le da el usuario (u_i) a la primera tupla es de $u_1 = 1$ y que para la segunda tupla es $u_2 = 0.8$, a continuación se muestra la tabla 7.11, que resulta por ambas vías de calcular s_i. En dicha tabla se muestra el cálculo de la *exhaustividad* y la *precisión* difusas para cada operador y el modelo propuesto.

Como se comentó, para el caso de este ejemplo, por ambas vías de calcular s_i se obtienen idénticos resultados, esto debido a que el TDA contiene un sólo conjunto;

Operador de igualdad	**Operador Like**	**Modelo propuesto**
s_i= {1,0} u_i={1,0.8}	s_i= {1,0} u_i={1,0.8}	s_i= {1,0.66} u_i={1,0.8}
exhaustividad difusa 1/ 1.8 = **0.55**	*exhaustividad difusa* 1/ 1.8 = **0.55**	*exhaustividad difusa* 1.66/ 1.8 = 0.92
precisión difusa 1/1 = **1**	*precisión difusa* 1/1 = **1**	*precisión difusa* 1.66/1.66 = 1

Tabla 7.11: *Exhaustividad* y *relevancia* difusas en consultas para buscar todos los términos y cálculo de s_i con índice de acoplamiento fuerte promediado y con valor máximo.

por ende, el resultado de obtener el promedio y el máximo del único valor de t_i que hay para cada tupla será el mismo.

En este caso los operadores de igualdad y *Like* recuperan sólo una de las dos tuplas posibles. Esto hace que tengan una exhaustividad de 0.55, y la precisión de la que recuperan es 1, pues es exactamente la frase que se busca.

Para el caso del modelo propuesto, recupera las dos tuplas, aunque en el caso de la segunda tupla, el término *total* aparece entre los dos términos que se buscan, el sistema es capaz de recuperar dicha tupla. Por ello, obtiene una exhaustividad de 0.92. Dicho valor es correcto, porque para que fuera 1, la frase se tendría que haber acoplado exactamente con las dos tuplas, y no es el caso. Para la segunda tupla, se acopla con dos de los tres términos que ella contiene. Para el caso de la precisión toma valor 1, dado que las dos tuplas retornadas contienen completamente la frase buscada.

Como se puede observar, para el caso de la precisión tiene el mismo valor 1, lo que indica la calidad de las tuplas retornadas. Para el caso de la exhaustividad, se nota cómo el modelo propuesto es notablemente superior.

Como se dijo anteriormente en este capítulo, esta es la tendencia general que se observa en todos los experimentos realizados, el sistema propuesto recupera más tuplas que los operadores tradicionales, con igual o ligeramente menos precisión. Indiscutiblemente esto es una ventaja porque lograr recuperar muchas más tuplas que los operadores tradiciones y mantiene casi el mismo nivel de precisión que éstos.

En el próximo apartado se realiza un análisis en consultas de más de un término pero donde interesa que aparezca al menos un término de la frase buscada.

7.4.2.3. Evaluación de consultas que contengan parcialmente los términos

A continuación se formulará una consulta donde se recuperan varios términos. En este caso, sólo interesa que al menos uno de los términos especificados en la consulta aparezca en las tuplas recuperadas. Se muestran los valores de s_i (de forma análoga a como se hizo en apartado anterior), y *la exhaustividad* y *precisión* difusas, para los operadores de igualdad y *Like*, y con el modelo propuesto. Para el cálculo de s_i, serán utilizadas las dos variantes discutidas para el cálculo del índice de acoplamiento débil.

Como se discutió anteriormente, se utiliza este índice porque es el que garantiza que la frase buscada aparezca al menos parcialmente incluida en las tuplas recuperadas. Esto sería el equivalente a cuando el usuario introduce varios términos unidos por el operador lógico *OR*. Cuando al menos uno aparece en la tupla con que se comparada, se retorna dicha tupla como recuperada.

Ejemplo 40

Seleccionar las tuplas que contienen al menos parcialmente la frase 'Protesis total cadera derecha'.

Para calcular los s_i correspondientes a buscar la frase '*Protesis total cadera derecha*', se asume que existen sólo las dos tuplas siguientes que contienen los términos buscados:

	Tupla modificada	**TDA**
Tupla 1	*Protesis mano derecha*	$\{Protesis, mano, derecha\}$
Tupla 2	*Amputacion total dedo*	$\{Amputacion, total, dedo\}$

A continuación se realiza el mismo cálculo de s_i, *exhaustividad y precisión* difusas, para cada una de las dos variantes que usa la operación *indice_ acoplamiento_ debilTDA*. También se realiza el cálculo para los operadores de igualdad y *Like* y con el modelo propuesto.

Variante 1: *Utilizando la operación* indice_acoplamiento_debilTDA *obteniendo el índice de acoplamiento con todos los conjuntos del TDA.*

- ***Operador de igualdad:***

Para el operador de igualdad, ninguna de las dos tuplas serían retornadas, dado que no contienen únicamente la frase buscada. Por esto, tanto la *exhaustividad* como la *precisión* difusas, serían igual a 0.

- ***Operador Like***

Con el operador *Like,* utilizando el patrón de búsqueda (*%Protesis total cadera derecha %)* no se recupera ninguna de las dos tuplas. Por esto, tanto la *exhaustividad* como la *precisión* difusas, serían igual a 0.

Como se comentó anteriormente, con un patrón de búsqueda más flexible, se podrían recuperar tuplas en la base de datos. Este operador Like tiene la dificultad que mientras más términos aparecen en la frase, más complejo se hace de definir el patrón, como ocurre en este ejemplo. Pues si se especificara un patrón de búsqueda como (ipropuesta Like %Protesis % OR ipropuesta Like %cadera derecha %) se recuperarían una gran cantidad de tuplas que no tienen la información semántica que está buscando el usuario, pues saldrían todos los tipos de 'protesis'y todos los tipos de intervenciones que se hayan realizado en la 'cadera derecha'.

Esta situación la previene nuestro sistema como se ha explicado, pues el usuario sólo tiene que elegir la operación correcta del modelo que responde a su consulta, y es el sistema el que se encarga de brindarle la mejor respuesta a su consulta.

- ***Modelo propuesto***

Mediante el modelo propuesto, explotando esta posibilidad comentada anteriormente de recuperar las tuplas aunque no se encuentren en el mismo orden los términos que en la frase buscada, se recuperan ambas tuplas. Debido a esto, los valores de s_i calculando un índice de acoplamiento débil serían $s_1 = 2/3$ y $s_2 = 1/3$.

Variante 2: *Utilizando la operación* indice_acoplamiento_debilTDA *obteniendo el índice máximo de acoplamiento con todos los conjuntos del TDA.*

En este caso, como el TDA contiene un sólo conjunto, los valores de s_i serán los mismos que los calculados con la variante 1.

Asumiendo que el valor de relevancia que le da el usuario (u_i) a la primera tupla es de $u_1 = 0.3$ y que para la segunda tupla es $u_2 = 0.1$, a continuación se muestra

Operador de igualdad	**Operador Like**	**Modelo propuesto**
No se recupera ninguna tupla	*No se recupera ninguna tupla*	s_i= *{0.66,0.33}* u_i ={*0.3,0.1*}
exhaustividad difusa **0**	*exhaustividad difusa* **0**	*exhaustividad difusa* 0.4/ 0.4 = **1**
precisión difusa **0**	*precisión difusa* **0**	*precisión difusa* 0.4/ 0.99 = **0.40**

Tabla 7.12: *Exhaustividad* y *relevancia* difusas en consultas para buscar algunos términos y cálculo de s_i con índice de acoplamiento fuerte promediado y con valor máximo.

la tabla 7.12, que resulta por ambas vías de calcular s_i. En dicha tabla se muestra el cálculo de la *exhaustividad* y la *precisión* difusas para cada operador y el modelo propuesto.

Como se comentó, para el caso de este ejemplo, por ambas vías de calcular s_i se obtienen idénticos resultados; esto es debido a que el TDA contiene un sólo conjunto, y por ende el resultado de obtener el promedio y el máximo del único valor de t_i que hay para cada tupla, dará el mismo resultado.

Este ejemplo, dado que la frase recuperada es más compleja, ni el operador de igualdad y ni el operador *Like con el patrón de búsqueda especificado,* recuperan ninguna de las dos tuplas. Esto hace que para ambos operadores el valor de la exhaustividad y la precisión sea 0.

Para el caso del modelo propuesto, recupera las dos tuplas, por lo que se ha venido comentando de que en este caso no importa el orden en que se encuentren los términos en la consulta. En este ejemplo en que las tuplas que contienen términos no tienen mucha relación semántica con los términos de la búsqueda realizada, la evaluación del usuario a cada tupla es muy baja. Debido a esto, en ambas tuplas, la evaluación que realiza el sistema en cada caso es superior a la del usuario. Esto hace que la precisión obtenida sea tan baja. Algo correcto también, pues en realidad las tuplas retornadas por el sistema no tienen la calidad esperada por el usuario.

La exhaustividad toma valor 1 ya que coinciden la sumatoria de los mínimos entre s_i y u_i con la sumatoria de u_i; esto indica que en el AND que se calcula al determinar el mínimo entre ambos valores ha primado la baja valoración que realiza el usuario de las tuplas recuperadas.

7.5. Conclusiones

A lo largo de este capítulo se han definido y aplicado los conceptos de *relevancia booleana y difusa*, para a partir de ellos calcular la *exhaustividad y precisión booleana y difusa* respectivamente. Se han dado los criterios para el cálculo de dichas medidas para ambos modelos de relevancia, así como para los diferentes tipos de consultas que se pueden especificar por parte del usuario. También se han desarrollado ejemplos prácticos que muestran el cálculo de dichas medidas sobre diferentes ejemplos de consultas.

El análisis de los resultados obtenidos en la experimentación que se desarrolla en este capítulo se resume en los siguientes aspectos:

- Los resultados en las consultas con el operador Like dependerá de los conocimientos de lenguaje *SQL* del usuario y su habilidad para construir el patrón de búsqueda. En este sentido, el modelo propuesto es considerablemente superior, pues el usuario puede expresar los términos para sus consultas sin importar el orden, y se recuperan todas las tuplas que los contiene. Este operador tiene además la dificultad que mientras más términos aparecen en la frase, más complejo se hace de definir el patrón. Especialmente en frases largas, el modelo propuesto obtiene mejores resultados.

- En el caso del modelo booleano, teniendo en cuenta el tratamiento de sinonimia que realiza nuestro modelo, siempre se recupera la máxima cantidad de términos relevantes; de aquí que siempre, tanto la *exhaustividad* como la *precisión* toman valor 1. Para ello, se asume que en el fichero de sinónimos están las diferentes formas en que se puede encontrar escrita un mismo término en la base de datos.

- En el caso del modelo difuso, la tendencia general en que se observa en todos los experimentos realizados, es que el sistema propuesto recupera más tuplas que los operadores tradicionales, con igual o ligeramente menos precisión.

Este hecho constituye sin dudas, la mejor prueba de la superioridad del modelo sobre los operadores tradicionales del trabajo con atributos textuales en bases de datos.

- De forma general, los resultados obtenidos por el sistema para el caso de la exhaustividad son, cuando menos, iguales a los operadores tradicionales (para frases de un solo término), y en la mayoría de los casos superiores. Para el caso de la precisión, en muchos casos son resultados iguales o superiores y, en otros, ligeramente inferior cuando se obtiene una mayor relevancia, al recuperar tuplas que los otros operadores no son capaces de obtener.

En el siguiente capítulo se recogen las conclusiones generales de este trabajo, así como, las líneas de trabajo futuras que de él se derivan.

Capítulo 8

Conclusiones y trabajos futuros

El presente capítulo nos servirá para resumir los objetivos que se han logrado y que han sido descritos en la presente memoria. A continuación de estos, se exponen ciertas líneas futuras que extenderán los resultados obtenidos y aquí presentados.

8.1. Conclusiones

El objetivo fundamental planteado al comienzo de este trabajo es la obtención de una forma de representación del conocimiento que mantenga la semántica de los atributos textuales en una base de datos, a través de técnicas de Minería de Textos. Además, lograr la implementación de dicha estructura a través de un TDA, que permita manejar dichos atributos textuales como el resto de los atributos de la base de datos.

De forma general, para conseguir estos objetivos, se ha profundizado en el estudio de las técnicas y herramientas necesarias para la definición del modelo propuesto y su implementación. Se ha realizado la formulación matemática de las estructuras y operaciones que componen dicho modelo. También se ha logrado su implementación en un TDA como lo propone el estándar SQL:99. Además, se ha logrado su aplicación sobre atributos textuales en una base de datos médica. Finalmente, se han desarrollado las herramientas de software necesarias que permiten de forma automática la obtención del modelo propuesto, y la realización de consultas semánticas sobre atributos textuales en bases de datos.

Es de destacar, que la solución obtenida sobrepasa las necesidades del problema de investigación, pues se ha obtenido una forma de representación del conocimiento, que puede ser aplicada en otros contextos y ayudar a resolver problemas diversos sobre atributos textuales. Se ha dado un conjunto de operaciones que permite capturar la semántica de las frases contenidas en atributos textuales en base de datos, y se han realizado las extensiones a un SGBD para dar soporte a consultas semánticas sobre dichas estructuras. De forma más detallada, la labor realizada queda descrita a continuación:

1. *Se ha obtenido el modelo abstracto de una nueva Forma Intermedia de Representación para atributos textuales en bases de datos.*

   - Dicho modelo se basa en las estructuras matemáticas definidas como conjuntos-AP y estructuras-AP. Los conjuntos-AP son obtenidos a través del cálculo de los itemsets maximales, siguiendo el algoritmo Apriori, proveniente del área de extracción de Reglas de Asociación dentro de la Minería de Textos. La estructura-AP es definida a partir de la colección de conjuntos-AP que se obtienen. Ambas estructuras tienen forma reticular.
   - Para el caso de los conjuntos-AP, además de su definición formal, se han introducido las operaciones inclusión de conjuntos-AP, subconjunto-AP inducido y superconjunto-AP inducido. Estas operaciones serán la base para otras más generales que se definen en las estructuras-AP y para realizar búsquedas sobre los datos almacenados.
   - Para el caso de la estructura-AP, además de su definición formal, se han introducido las operaciones inclusión de estructuras-AP, unión e intersección de estructuras-AP, subestructura-AP inducida y superestructura-AP inducida. También, con la idea de determinar el tipo de acoplamiento y el grado con que ocurre de determinados conjuntos sobre esta estructura, se definieron las operaciones de acoplamiento fuerte y débil, e índice de acoplamiento fuerte y débil respectivamente.

2. *Se ha realizado la implementación del modelo obtenido.*

   - Para ello, se ha definido una Metodología que desde el punto de vista del modelado conceptual, parte del modelo abstracto y realiza su implementación en un lenguaje de definición de objetos como OQL. A

continuación, tomando como referencia las interfaces que se definen en este lenguaje, se realiza su implementación a través de un TDA en el SQL:99 estándar. Una vez en este modelo, se da la estrategia a tener en cuenta para su implementación en un Modelo de Datos Relacional Orientado a Objetos. Finalmente, se realiza su implementación en un TDA en PostgreSQL, con algunas variaciones debido a limitaciones que presenta dicho gestor, para la implementación de este tipo de datos.

- Se han dado dos alternativas de almacenamiento de los TDA que conforman los conjuntos-AP y estructuras-AP: almacenar sólo el conjunto generador de las estructuras y almacenar el conjunto generador expandido hasta las hojas. Aquí se han discutido las ventajas y desventajas de cada modelo y se ha definido la primera de las dos variantes como la utilizada en la implementación realizada.

3. *Se ha planteado la arquitectura del sistema y se ha definido una metodología de desarrollo, con vistas a la obtención de la estructura-AP global y las estructuras-AP inducidas para cada tupla de la base de datos.*

   - Dentro de la arquitectura del sistema se han definido sus componentes fundamentales y la relación que existe entre ellos. Aquí se ha discutido en detalle el módulo de Preprocesamiento y obtención de forma intermedia. Este módulo es el que se encarga de la limpieza de datos y del cálculo de los itemsets maximales, que serán los que formen los conjuntos-AP; con ellos se obtiene la estructura-AP global del atributo textual procesado.
   - Se han definido los pasos que componen la metodología para, partiendo del atributo textual, obtener la estructura-AP inducida que le corresponde en la base de datos. A partir de dicha metodología, se han realizado todos sus pasos de forma práctica sobre los conjuntos experimentales de datos definidos.
   - Utilizando dichos conjuntos experimentales se han mostrado diversos ejemplos y estadísticas que demuestran las mejoras obtenidas, después de la realización del proceso de limpieza de datos. También se ha descrito en detalle el algoritmo de limpieza de datos implementado.
   - Se ha mostrado cómo se realiza la obtención de la estructura-AP global de conocimiento. Para ello, se ha descrito la implementación del

algoritmo Apriori, se ha mostrado el algoritmo de obtención de los itemsets maximales implementado, y se han dado ejemplos de retículos de conjuntos-AP y estructuras- AP obtenidos sobre los datos experimentales.

- Se ha obtenido la estructura-AP inducida para cada tupla de la base de datos. Aquí se ha descrito el proceso de intersección entre el valor de una tupla y la estructura-AP y se ha dado el algoritmo implementado con este fin. También se han puesto ejemplos de casos posibles que ocurren en la intersección. Además se ponen ejemplos y se discuten las estadísticas obtenidas en dicho proceso.

4. *Se han definido y ejemplificado los tipos de consultas posibles sobre el sistema, comparando el modelo propuesto con los operadores tradicionales para consultas con atributos textuales en bases de datos. Además, se ha realizado la validación del sistema, utilizando las medidas de exhaustividad y precisión provenientes del área de la Recuperación de Información.*

   - Del análisis de las estructuras obtenidas para representar el modelo, y de las características de los atributos textuales, se han definido y puesto ejemplos de los tres tipos de consultas posibles sobre el sistema. Dichas consultas pueden ser del tipo: consultas sobre la estructura-AP, consultas sobre el TDA inducido de cada tupla y consultas posibles por el diseñador.
   - Se ha demostrado a través de ejemplos experimentales, las ventajas del modelo propuesto sobre los operadores tradicionales en bases de datos relacionales. En casi todas las consultas que son posibles utilizando dichos operadores, nuestro modelo recupera una mayor cantidad de tuplas, además que permite la definición de un grupo de operaciones que no son posibles en este modelo de bases de datos.
   - Se han aplicado los conceptos de relevancia booleana y difusa para, a partir de ellos definir la exhaustividad y precisión booleana y difusa respectivamente. A partir de estos conceptos, se han desarrollado ejemplos prácticos que muestran el cálculo de dichas medidas sobre diferentes ejemplos de consultas.
   - Con los ejemplos desarrollados, se ha demostrado que los resultados obtenidos por el sistema para el caso de la exhaustividad son, cuando menos, iguales a los operadores tradicionales, y en la mayoría de

los casos superiores. Para el caso de la precisión, en muchos casos son resultados iguales o superiores, y en otros ligeremente inferior cuando se obtiene una mayor relevancia, al recuperar tuplas que los otros operadores no son capaces de obtener.

5. *Se han implementado las herramientas necesarias tanto para la realización de forma automática de la implementación del modelo, como para la realización de consultas semánticas sobre atributos textuales en bases de datos.*

   - La herramienta implementada *Text Mining Tool V1.0* , se encarga de realizar las tareas correspondientes al módulo de Preprocesamiento y obtención de Forma Intermedia. Para ello realiza la limpieza de los datos, la obtención de los itemset maximales, conjuntos-AP y estructura-AP. También se encarga de obtener la estructura-AP inducida correspondiente a cada tupla de la base de datos. Todos estos procesos poseen varios parámetros de entrada y salida que son configurables en dicha aplicación. Para los procesos de limpieza de datos y obtención de la estructura-AP inducida, el sistema devuelve las estadísticas correspondientes a ambos procesos.

   - La herramienta implementada Query Maker V1.0 es el cliente de consulta que posibilita al usuario la realización de consultas semánticas sobre atributos textuales en la base de datos. De esta herramienta, se ha mostrado su arquitectura, interfaz y funcionalidades que implementa. A partir de los resultados obtenidos en las consultas realizadas usando las estructuras del modelo obtenido, se ha realizado una comparativa de los resultados obtenidos con operadores tradicionales sobre atributos textuales. Dicha comparativa demuestra las ventajas del modelo obtenido. También se han realizado las extensiones correspondientes en PostgreSQL con el objetivo de implementar las operaciones del modelo. Además se han dotados las funciones implementadas en este gestor de un grupo de bondades que flexibilizan y hacen más fructíferas las consultas del usuario.

8.2. Trabajos futuros

El trabajo descrito en esta memoria abre todo un conjunto de ideas sobre las que continuar la investigación. Algunas de ellas no han sido abordadas por estar fuera de los objetivos inicialmente planteados, y otras han aparecido como consecuencia de las aportaciones aquí realizadas. A continuación, presentamos algunas de las líneas que hemos considerado más interesantes de cara a futuras aportaciones.

1. *Profundizar en el estudio del rendimiento del sistema propuesto:*

   - Estudio de las condiciones donde se valida la hipótesis. Con la idea de hacer la experimentación sobre otros tipos de textos (documentos textuales completos, textos menos estructurados que los utilizados, etc.), datos provenientes de otras bases de datos más allá del entorno hospitalario (resúmenes de artículos, correos electrónicos, etc).
   - Estudio en profundidad de las medidas de *exhaustividad* y *precisión*, para evaluar la calidad de la información brindada por el sistema. Establecer una forma de cálculo lo más automatizada posible de la relevancia subjetiva del usuario.

2. *Problemas no tratados en el modelo:*

   - Modificación de la base de datos y su implicación en la base de conocimientos extraída. Con la idea de resolver el problema del impacto de agregar o eliminar información en la base de datos, y que se recalculen de forma automática las estructuras de la base de conocimiento que dependen de la frecuencia de aparición de los términos.
   - Fusión de información y mezcla de bases de datos. Utilizar las operaciones de unión e intersección para lograr repoblar las estructuras, y si es posible, unificar vocabularios distintos en uno solo (unión de estructuras-AP).

3. *Uso del modelo en OLAP y Data Warehouse:*

   - Creación de un modelo multidimensional, donde una o más dimensiones sean atributos textuales, representadas utilizando el modelo propuesto.

4. *Desde el punto de vista de extensión del modelo abstracto:*

   - Introducción de incertidumbre y soporte en las estructuras del modelo y sus operaciones. Permitiría plantear el modelo de forma difusa y utilizar el soporte de los itemsets en la definición de las operaciones.
   - Definición de una secuencia-AP. Permitiría la introducción de restricciones en la semántica. Entre ellas, extracción de frases ordenadas y formulación de estructuras basadas en secuencia. Para el contexto médico particularmente, se pudiera utilizar diccionarios estándares como el que es suministrado por el ICD (International Classification of Diseases) (National-Center-for-Health-Statistics, 2007).

Apéndice A

El modelo Relacional Orientado a Objetos: Implementación en PostgreSQL

En este apéndice se realiza un compendio de las características fundamentales del modelo de bases de datos Relacional Orientado a Objetos. Para ello, se dan algunos elementos que motivaron su aparición y las características que ha incorporado dicho modelo para resolver las limitaciones del Modelo Relacional puro.

Además, se discuten las características del SGBD utilizado en la implementación del modelo obtenido en esta memoria, el PostgreSQL. Este gestor, como hemos dicho anteriormente, implementa este modelo de datos Relacional Orientado a Objetos. También se darán las características y ventajas de su lenguaje procedural el PL/pgSQL.

A.1. Modelo Relacional Orientado a Objetos

Los SGBD relacionales son los dominantes dentro del mundo de las bases de datos. Los sistemas de bases de datos orientado a objetos han ido alcanzando áreas más amplias de aplicación en sistemas financieros y de telecomunicaciones (Internet). La mayoría de las grandes empresas son muy reacias a cambiar el modelo de datos relacional, que han utilizado históricamente. El modelo relacional ha sido suficiente

para las aplicaciones tradicionales comerciales o de negocios; estas aplicaciones se caracterizan por manejar datos muy simples en grandes volúmenes. Datos simples generalmente incluyen datos alfanuméricos que, con bastante precisión y facilidad, pueden ser representados en una computadora.

Estas aplicaciones han evolucionado en cuanto a sus necesidades de manipulación, de almacenamiento y análisis de datos históricos. En este entorno, la tecnología de base de datos se comienza a utilizar en aplicaciones que ya no son del ámbito del procesamiento de datos "tradicional". Esto trae consigo la necesidad de almacenar y consultar objetos complejos, con datos provenientes de aplicaciones de diseño de ingeniería, datos de monitoreo del ciclo de producción en manufacturas, inclusión de audio, video, texto, etc.

La forma más obvia de abordar estas cuestiones es dotar al modelo relacional de todos los elementos necesarios para tratar con este tipo de problemas. Estas son las ideas básicas que subyacen en el desarrollo de los nuevos sistemas relacionales.

Existen distintos modelos cuyas características dependen de la forma y el grado en que se haya extendido el modelo relacional. (Informix, IBM, Oracle, PostgreSQL). Todos los modelos comparten el concepto de tabla y el lenguaje de consulta, incluyen algún concepto de "objeto" y permiten almacenar conjuntamente datos y métodos a través del concepto de "disparador" o procedimiento. Se ha utilizado una terminología muy diversa y se ha impuesto la terminología "Relacional Orientado a Objetos". Los lenguajes asociados a estos modelos son extensiones de SQL. Existe un estándar de SQL para este modelo, el SQL:99. Dicho estándar incorpora las nuevas funcionalidades disponibles en este modelo, como permitir la herencia, los tipos colecciones, etc.

El modelo Relacional Orientado a Objetos da soporte completo al modelo relacional. Hace un uso extensivo de elementos de orientación a objetos tales como: encapsulamiento, herencia, polimorfismo, soporte para tipos de datos que permiten almacenar objetos complejos, sin la primera forma normal e identidad de objetos. Como se comentó, utiliza SQL:99 como lenguaje de datos para soportar estas características.

Entre las ventajas de utilizar un SGBD con este modelo están: la posibilidad de reutilizar y compartir estructuras definidas previamente. Se pueden asociar a los tipos almacenados funcionalidades que antes se deban asociar con las aplicaciones. También se pueden crear librerías. Además, permite preservar el cuerpo del conocimiento y la experiencia obtenida en el desarrollo de aplicaciones relacionales.

En el mundo de las bases de datos, los programadores se han encontrado frente a alternativas de Sistemas de Gestión de Bases de Datos (SGBD) no comerciales que constituyen alternativas nada despreciables frente a los gigantes históricos establecidos en este sector como Oracle, Informix, Microsoft SQL Server, Borland Interbase, entre otros. Éstos, en gran medida, ofrecen productos destinados a la gestión de grandes cantidades de información a precios normalmente inalcanzables para una empresa media, no rentables para desarrollo de proyectos de investigación y sin hablar de un usuario doméstico.

Entre las características deseables de un software de base de datos se tienen las siguientes:

- Confiable.
- Rápido.
- Robusto.
- Escalable.
- De libre distribución.
- Gratuito.
- Sin límite de usuarios.
- Soportado por multitud de plataformas, entre ellas Windows y Linux.

Los gestores de bases de datos libres son desarrollados generalmente por Universidades o por grupos de voluntarios. Algunos de estos productos tienen una gran calidad y dan soporte a la inmensa mayoría de las características antes mencionadas como deseables en un SGBD. También en muchos casos, superan las cualidades de sus competidores comerciales; dichos productos normalmente van dirigidos a un mercado de pequeñas y medianas empresas y grupos de investigación y desarrollo.

Entre estos productos gratuitos y de gran calidad, encontramos a PostgreSQL como un SGBD con modelo de datos Relacional Orientado a Objetos. Dicho gestor está dirigido a satisfacer la necesidades de los programadores de Bases de Datos tradicionales, así como a los que deseen darle una mayor robustez a sus aplicaciones explotando sus potencialidades de programación en el servidor de Bases de Datos.

Para ello, presenta un poderoso lenguaje de consulta PL/pgSQL y/o utilizando algunas características de la orientación a objetos.

En la siguiente sección se realiza una introducción a PostgreSQL. También se dan algunos de los elementos que lo caracterizan como SGBD con modelo Relacional Orientado a Objetos.

A.2. Introducción a PostgreSQL

PostgreSQL está basado en POSTGRES, Versión 4.2, desarrollado en el departamento de Ciencias de la Computación de Berkeley, en la Universidad de California y se coloca en la categoría de las Bases de Datos conocidas como objeto-relacionales de código abierto (open source) y es la más avanzada en la actualidad, años de pruebas y miles de usuarios en todo en mundo así lo demuestran. Ha generado algunas características que son propias del mundo de las bases de datos orientadas a objetos.

Esto ha llevado a que, algunas Bases de Datos comerciales hallan incorporado recientemente estas ventajas en las que PostgreSQL fue pionera. Como hemos comentado, PostgreSQL es software libre. Concretamente está liberado bajo la licencia BSD (Berkeley Software Distribution), lo que significa que cualquiera puede disponer de su código fuente, modificarlo a voluntad y redistribuirlo libremente, es más, la licencia BSD permite redistribuir el código modificado o no como software cerrado, en contraposición a la licencia GPL (General Public License) que fuerza a que las modificaciones sean publicadas también bajo la GPL, es decir, siempre tienen que ser publicadas como código abierto.

PostgreSQL además de ser libre es gratuito y se puede descargar libremente de su Sitio Web (PostgreSQL-Global-Devel-Group, 2008a) para multitud de plataformas. La versión oficial actual de PostgreSQL es la 8.3.1, liberada el 17 de Marzo de 2008. A continuación, se detallan las características fundamentales de este gestor de base de datos.

A.2.1. Características de PostgreSQL

Como ya se comentó anteriormente, PostgreSQL está considerado como la base de datos de código abierto más avanzada del mundo. PostgreSQL proporciona un

gran número de características que normalmente sólo se encontraban en las bases de datos comerciales tales como DB2 u Oracle.

El modelo de datos de PostgreSQL está basado en el modelo relacional, pero proporciona objetos (una clase es una tabla, una fila es una instancia y una columna es un atributo), identificadores de objetos (único para cada instancia), objetos compuestos, herencia múltiple, sobrecarga de funciones o métodos, y versiones. Además, incluye disparadores, control de concurrencia multi-versión, soporte multi-usuario, transacciones, optimización de consultas, el tipo *array* entre otros.

- **Extensibilidad dirigida por catálogos:** En el catálogo de PostgreSQL además de guardar la información sobre las tablas, y columnas que constituyen la base de datos como hacen los SGBD tradicionales, se almacena mucha más información como son los tipos, las funciones, los operadores y los métodos de acceso (P.M, 1991). Esta información, que aparece a los usuarios en forma de clases, puede ser modificada por el usuario permitiendo así la extensibilidad del sistema.

  El optimizador y el procesador de consultas son conducidos por tablas, de forma que todas las extensiones definidas por el usuario y empleadas en una consulta pueden ser cargadas en un proceso servidor de PostgreSQL en tiempo de ejecución. El usuario puede especificar un fichero de código objeto que implementa un nuevo tipo o función y PostgreSQL lo cargará dinámicamente. El mecanismo de carga dinámica empleada será el del sistema operativo subyacente.

- **Incorporación de nuevos métodos de acceso**: Básicamente, un método de acceso PostgreSQL es una colección de trece funciones C que hacen operaciones a nivel de registro, tales como traer el siguiente registro, insertar un nuevo registro, eliminarlo, etc. Para añadir un nuevo método de acceso, lo que un usuario debe hacer es proporcionar implementaciones para cada una de estas funciones y crear una colección de entradas en el catálogo del sistema (Stonebraker y Rowe, 1986).

- **Soporte_SQL_Comprensivo:** PostgreSQL soporta la especificación SQL99 e incluye características avanzadas tales como las uniones (joins) SQL92.

- **Integridad Referencial:** PostgreSQL soporta integridad referencial, la cuál es utilizada para garantizar la validez de los datos de la base de datos.

- **API Flexible:** La flexibilidad del API de PostgreSQL ha permitido a los vendedores proporcionar soporte al desarrollo fácilmente para el SGBD PostgreSQL. Estas interfaces incluyen Object Pascal, Python, Perl, PHP, ODBC, Java/JDBC, Ruby, TCL, C/C++, y Pike.

- **Lenguajes Procedurales:** PostgreSQL tiene soporte para lenguajes procedurales internos, incluyendo un lenguaje nativo denominado PL/pgSQL. Este lenguaje es comparable al lenguaje procedural de Oracle, PL/SQL. Otra ventaja de PostgreSQL es su habilidad para usar Perl, Python, o TCL como lenguaje procedural embebido.

- **MVCC:** MVCC o Control de Concurrencia Multi-Versión (Multi-Version Concurrency Control), es la tecnología que PostgreSQL usa para evitar bloqueos innecesarios. Comparando esta característica con otros SGBD con capacidades SQL, tal como MySQL o Microsoft Access, tenemos que en estos últimos hay ocasiones en las una lectura tiene que esperar para acceder a información de la base de datos. La espera está provocada por usuarios que están escribiendo en la base de datos. Resumiendo, el lector está bloqueado por los escritores que están actualizando registros.

  Mediante el uso de MVCC, PostgreSQL evita este problema por completo. MVCC está considerado mejor que el bloqueo a nivel de fila porque un lector nunca es bloqueado por un escritor. En su lugar, PostgreSQL mantiene una ruta a todas las transacciones realizadas por los usuarios de la base de datos. PostgreSQL es capaz entonces de manejar los registros sin necesidad de que los usuarios tengan que esperar a que los registros estén disponibles.

- **Cliente/Servidor:** PostgreSQL usa una arquitectura proceso-por-usuario cliente/servidor. Esta es similar al método del Apache 1.3.x para manejar procesos. Hay un proceso maestro que se ramifica para proporcionar conexiones adicionales para cada cliente que intente conectar a PostgreSQL.

- **Write Ahead Logging (WAL):** La característica de PostgreSQL conocida como Write Ahead Logging incrementa la dependencia de la base de datos al registro de cambios antes de que estos sean escritos en la base de datos. Esto garantiza que en el hipotético caso de que la base de datos se caiga, existirá un registro de las transacciones a partir del cual podremos restaurar la base de datos. Esto puede ser enormemente beneficioso en el caso de caída ya que, cualesquiera cambios que no fueron escritos en la base de datos, pueden

ser recuperados usando el dato que fue previamente registrado. Una vez el sistema ha quedado restaurado, un usuario puede continuar trabajando desde el punto en que lo dejó cuando cayó la base de datos.

Debemos decir que a pesar poseer todas estas excelentes características, una de las principales desventajas de PostgreSQL es la carencia de una buena interfaz gráfica que permita un manejo más rápido y sencillo de la base de datos como sí la poseen por ejemplo SQL Server y Oracle. Las herramientas gráficas diseñadas para PostgreSQL como *pgAccess* y *pgAdmin* están todavía lejos de poseer las prestaciones de sus similares exhibidas por Oracle y SQL Server.

Resumiendo estas características tenemos:

- *Características Operacionales :*

- Transacciones (Transactions).
- Disparadores (Triggers).
- Restricciones (Constrains).
- Replicación (Replication).
- Backup y Recuperación (Backup & Recovery).
- Reglas (Rules).
- Procedimientos Almacenados/Funciones (Stored Procedures/Functions).
- Integridad Referencial.
- Outer Joins.
- Sintaxis ANSI SQL 89, 92 y 99.
- Logging.
- Extensivo y programable.
- Orientado a Objetos
- Características sofisticadas de integridad de datos.
- Tipos de datos y funciones definidos por el usuario.

- Cliente/servidor, entre otros.

- Límites de la una base de datos en PostgreSQL:

- Máximo tamaño de una base de datos: ilimitado, solo limitado por la capacidad de almacenamiento del hardware.
- Máximo tamaño de una tabla: hasta 64 Tb (terabytes).
- Máximo tamaño de un campo: 1Gb.
- Máxima cantidad de tuplas o registros: ilimitado.
- Máxima cantidad de columnas en un tabla: hasta 1600.
- Máxima cantidad de índices por tabla: ilimitado.

A.2.2. El lenguaje PL/pgSQL

Los objetivos de diseño para PL/pgSQL fueron crear un lenguaje procedural cargable que:

- Pudiera ser usado para crear funciones y disparadores.
- Añadiera estructuras de control al lenguaje SQL.
- Pudiera realizar computaciones complejas.
- Heredase todos los tipos definidos por el usuario, funciones y operadores.
- Pudiera ser definido para ser validado por el servidor.
- Fuese sencillo de utilizar.

El gestor de llamadas PL/pgSQL interpreta el código fuente en texto plano de la función y produce un árbol de instrucciones binarias internas la primera vez que ésta es llamada (dentro de cualquier proceso principal). El árbol de instrucciones traduce toda la estructura de la sentencia PL/pgSQL, pero las expresiones individuales de SQL y las consultas SQL usadas en la función no son traducidas inmediatamente.

Cuando cada expresión y/o consulta SQL es usada por primera vez en una función, el intérprete PL/pgSQL crea un plan de ejecución preparado (usando las funciones del gestor SPI SPI_prepare y SPI_saveplan). Las siguientes visitas a dicha expresión y/o consulta reutilizan en plan de ejecución preparado. Así, una función con código condicional que contiene muchas sentencias para los cuales el plan de ejecución podría ser requerido sólo preparará y almacenará aquellos planes que realmente sean usados durante el tiempo de vida de la conexión a la base de datos.

Esto puede, consecuentemente, reducir la cantidad de tiempo requerida para interpretar y generar planes de consultas para las sentencias en una función de lenguaje procedural. Una desventaja es que los errores en una expresión o consulta específica pueden no ser detectados hasta que no se llegue a esa parte de la función en la ejecución. Una vez que PL/pgSQL ha realizado un plan de ejecución para una determinada consulta en una función, éste reusará dicho plan durante toda la vida de la conexión a la base de datos. Esto significa normalmente una ganancia en el rendimiento, pero puede causar algunos problemas si se altera dinámicamente su esquema de base de datos.

Excepto para el caso de conversión de entrada/salida y funciones de cálculo para tipos definidos por el usuario, cualquier cosa que pueda ser definida en funciones del lenguaje C también pueden serlo con PL/pgSQL. Es posible crear complejas funciones de computación condicionales y más tarde usarlas para definir operadores o usarlas en índices funcionales.

- **Ventajas de usar PL/pgSQL**

Mayor Rendimiento: SQL es el lenguaje que PostgreSQL (y la mayoría del resto de bases de datos relacionales) usa como lenguaje de consultas.

Es portable y fácil de aprender: Pero cada sentencia SQL debe ser ejecutado individualmente por el servidor de bases de datos. Esto significa que una aplicación cliente debe enviar cada consulta al servidor de bases de datos, esperar a que se procese, recibir el resultado, realizar alguna computación, y luego enviar otras consultas al servidor. Todo esto incurre en una comunicación entre procesos y también puede sobrecargar la red si dicho cliente se encuentra en una máquina distinta al servidor de bases de datos.

Con PL/pgSQL puede agrupar un grupo de operaciones y una serie de consultas dentro del servidor de bases de datos, teniendo así la potencia de un lenguaje

procedural y la sencillez de uso del SQL, pero ahorrando una gran cantidad de tiempo porque no tiene la sobrecarga de una comunicación cliente/servidor. Esto puede redundar en un considerable aumento del rendimiento.

Soporte SQL: PL/pgSQL añade a la potencia de un lenguaje procedural la flexibilidad y la sencillez del SQL. Con PL/pgSQL puede usar todos los tipos de datos, columnas, operadores y funciones de SQL. Portabilidad: Debido a que las funciones PL/pgSQL corren dentro de PostgreSQL, estas funciones funcionarán en cualquier plataforma donde PostgreSQL corra. Así se podrá reusar el código y reducir costos de desarrollo.

Apéndice B

Extracción de Reglas de Asociación: Implementación del algoritmo Apriori

El presente apéndice está dedicado a dar algunos detalles de las técnicas de Minería de Datos utilizadas en esta memoria. Concretamente se discute sobre las Reglas de Asociación, como técnica que permite la obtención de los itemset frecuentes, que son la base de las estructuras de la Forma Intermedia propuesta en esta memoria.

Se darán detalles del algoritmo Apriori utilizado, con la finalidad de la obtención de los itemset frecuentes. También se darán los detalles de la aplicación creada para automatizar todos los procesos planteados en la metodología propuesta anteriormente para la obtención de la Forma Intermedia y el TDA correspondiente a cada tupla de la base de datos.

B.1. Reglas de Asociación

La Minería de Datos es un campo que se basa en el uso de técnicas y herramientas para la búsqueda de información, en grandes volúmenes de datos, surgidos por la acumulación histórica de información en las organizaciones. Esta tarea ha hecho converger los campos de estadística, bases de datos e inteligencia artificial. Dentro

de las técnicas utilizadas en Minería de Datos, las Reglas de Asociación han aparecido como una de las más populares. Ellas son particularmente importantes para establecer relaciones entre los datos en grandes bases de datos.

Dado un conjunto de items, las reglas de asociación describen cómo varias combinaciones de items están apareciendo juntas en los mismos itemsets. El primer ejemplo relevante en este campo fue el que descubrió una gran cadena estadounidense de supermercados, *Wal-Mart*, que realizó a finales de los años 90 un análisis de los hábitos de compra de sus clientes. Sorprendentemente, descubrieron una correlación estadísticamente significativa entre las compras de pañales y cerveza: *los viernes por la tarde, los hombres entre 25 y 35 años que compraban cerveza también compraban pañales*. El objetivo es encontrar regularidades en los comportamientos de los clientes dentro de términos de combinaciones de productos que son comprados muchas veces en su conjunto, o sea reglas que reflejen relaciones entre los atributos presentes en los datos.

El descubrimiento de reglas de asociación busca relaciones o afinidades entre los itemsets. Un itemset se define como cualquier combinación formada por dos o más items diferentes de todos los items disponibles.

Podemos definir a I como un itemset y a T como un conjunto de transacciones con items en I , ambos, conjuntos finitos. Considerando dos itemsets $I_1, I_2 \subseteq I$, donde $I_1 \cap I_2 \neq \emptyset$; la regla $I_1 \rightarrow I_2$ es una regla de implicación que significa que la aparición del itemset I_1 implica la aparición del itemset I_2en el conjunto de transacciones T. Los itemsets I_1 y I_2son nombrados *antecedente* y *consecuente* de la regla respectivamente.

Las medidas más utilizadas para describir las relaciones entre antecedente y consecuente son el *soporte*$(supp)$, y la *confianza*$(conf)$, los cuales son valores numéricos.

El soporte de un itemset $I_0 \subseteq I$ es:

$$supp(I_0, T) = \frac{|\{t \in T | I_0 \subseteq t\}|}{|T|}$$

y significa la probabilidad de que una transacción T contenga I_0. El soporte de una regla de asociación $I_1 \longrightarrow I_2$ en T es:

$$supp(I_1 \longrightarrow I_2, T) = supp(I_1 \cup I_2)$$

y significa qué porcentaje de los atributos de una regla aparecen con valor positivo dentro de las transacciones de un conjunto de datos.

La confianza es:

$$conf(I_1 \longrightarrow I_2, T) = \frac{supp(I_1 \cup I_2)}{supp(I_1)} = \frac{Supp(I_1 \longrightarrow I_2)}{supp(I_1)}$$

y es la razón probabilística de I_2 con respecto a I_1, en otras palabras, es la cardinalidad relativa de I_2 con respecto a I_1.

Las técnicas utilizadas para poder encontrar las reglas de asociación, intentan descubrir reglas cuyo soporte y confianza es mayor o igual a dos umbrales, los cuales son determinados por el usuario, llamados *minsupp* y *minconf*, respectivamente. Tales reglas son llamadas reglas fuertes.

En (Agrawal et al., 1993) se propone el algoritmo APriori el cual ha sido ampliamente referenciado por varios autores por cuanto minimiza la operación más lenta del proceso de extracción de reglas de asociación, que es la medición de los datos. A continuación se dan algunos detalles de este algoritmo.

B.2. Algoritmo Apriori

En la literatura, aparecen diversos algoritmos para la extracción de reglas de asociación. Los más básicos son SETM (Houtsma y Swami, 1993) y AIS (Agrawal et al., 1993). Éste último fue adaptado posteriormente y dio lugar al algoritmo Apriori (presentado junto con una optimización del mismo, Apriori- TID, en (Agrawal y Srikant, 1994)), siendo el más conocido por su versatilidad y simplicidad. La mayoría de los enfoques siguientes parten del funcionamiento básico de éste.

El objetivo en todo algoritmo de búsqueda de reglas de asociación es encontrar todas las reglas que satisfacen con la condición de soporte y confianza mínimos dados por el usuario (*minsupp* y *minconf* respectivamente). Esto es necesario por cuanto si no es así la búsqueda se haría exhaustiva, encontrándose al final una explosión en el número de reglas generadas. Cuando se desea realizar la búsqueda en grandes bases de datos como sucede en Minería de Datos, se debe tratar de minimizar la cantidad de tiempo que se emplea en acceder las mismas , por cuanto estas operaciones de acceso a disco son por lo general las más lentas del proceso.

Un algoritmo como APriori satisface estos requerimientos y de hecho se ha convertido en referencia obligada en esta área, el mismo consta de dos pasos:

1. Encontrar todos los itemsets frecuentes, es decir todos aquellos subconjuntos de Ique cumplen con la condición $soporte >= minsupp$.

2. Usar los itemsets frecuentes obtenidos para generar reglas que cumplan con la condición $confianza >= minconf$.

La idea del algoritmo se basa en que si un conjunto de items cumple con la condición de soporte mínimo, entonces todo subconjunto de este también la cumplirá. En realidad se usa el contrarrecíproco, cuando se obtiene un conjunto de items se chequea si todos los subconjuntos de este medidos en la iteración anterior cumplían con la condición de mínimo soporte, si se encuentra alguno que no la cumple se puede concluir, "a priori", que dicho conjunto no la cumplirá y por tanto no es necesario medirlo. Esto evita mucha medición innecesaria, y con ello minimizar el tiempo total de acceso a la base de datos. En la tabla B.1 se muestra una variante del algoritmo Apriori presentada en (Serrano, 2003).

En este algoritmo, los itemsets se consideran ordenados por tamaño. Primero se calculan los 1-itemsets, después los 2-itemsets, etc. En cada iteración, se recorre la base de datos y se comprueba si los itemsets aparecen en ella con suficiente soporte (*para lo cual se fija un umbral mínimo al que se suele llamar minsupp*). Si no es así, se descartan. La función $CrearNivel\,(i, L)$ se usa para generar el siguiente conjunto de itemsets candidatos a ser frecuentes, siguiendo un resultado presentado en (Agrawal y Srikant, 1994) según el cual todos los subconjuntos propios de un itemset frecuente han de ser también frecuentes. De esta forma, se utiliza una especie de poda para eliminar de antemano aquellos candidatos que no van a ser frecuentes, ahorrándo espacio de memoria y tiempo de proceso.

A continuación en la próxima sección, se dan los detalles del software implementado para realizar las tareas discutidas anteriormente, que corresponden al Preprocesamiento y obtención de itemsets frecuentes utilizando el algoritmo Apriori antes descrito.

Entrada: I, un conjunto de items y T, un conjunto de transacciones sobre I.
Salida: F, conjunto de itemsets frecuentes.

1. *Definir un contador* c_i *para cada* $i \in I$
2. $L_1 \leftarrow \{\{i\} | i \in I\}$
3. $F \leftarrow \emptyset$
4. $l \leftarrow 1$
5. **MIENTRAS** $l <= |I|$ y $L_l \neq \emptyset$ **HACER**
6. **PARA CADA** $t \in T$ **HACER**
7. **PARA CADA** $I_* \in L_l$ **HACER**
8. **SI** $I_* \subseteq t$ **ENTONCES**
9. $c_{I_*} \leftarrow c_{I_*} + 1$
10. **FIN SI**
11. **FIN PARA**
12. **FIN PARA**
13. **PARA CADA** $I_* \in L_l$ **HACER**
14. **SI** $c_{I_*} < minsupp \times |T|$ **ENTONCES**
15. $L_l \leftarrow L_l \setminus \{I_*\}$
16. *Liberar la memoria usada por* cI_*
17. **FIN SI**
18. **FIN PARA**
19. $F \leftarrow F \cup L_l$
20. $L_{l+1} \leftarrow CrearNivel\,(l+1, L_l)$
21. $l \leftarrow l + 1$
22. **FIN MIENTRAS**
23. **DEVOLVER** F

Tabla B.1: Algoritmo Apriori para el cálculo de itemsets frecuentes.

B.3. Software utilizado

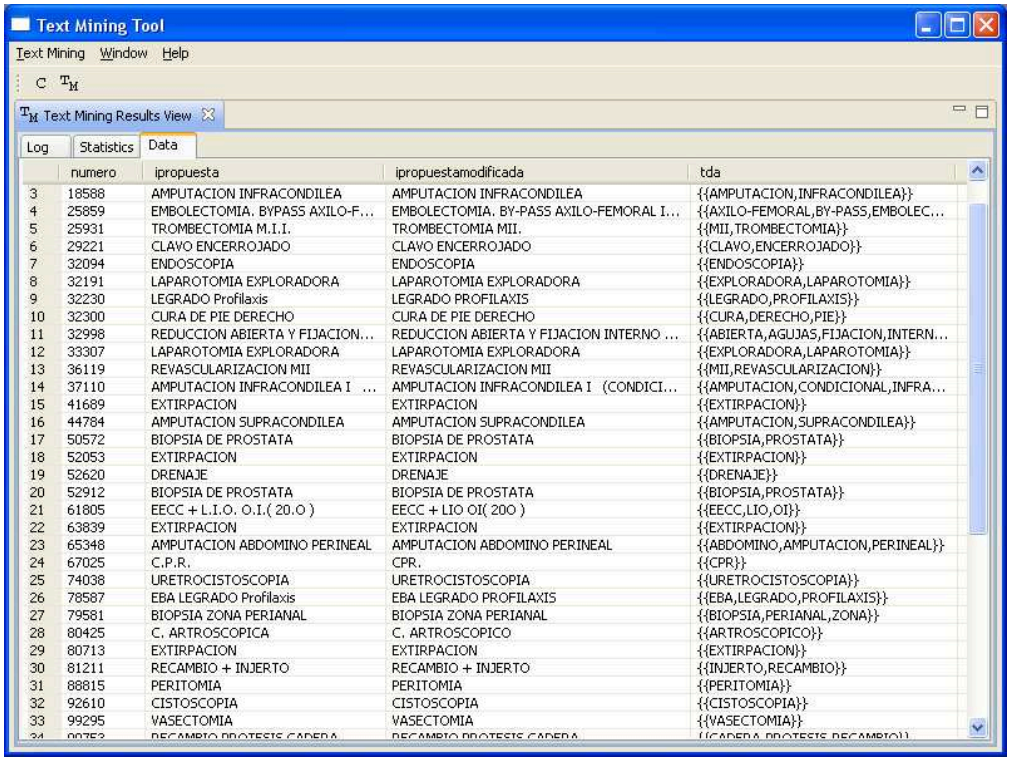
En esta sección se dan las ideas referentes a la interfaz y funcionalidades implementadas en la herramienta de Minería de Textos (Text Mining Tool v 1.0). Dicha herramienta es capaz de realizar de forma integrada, en una sola aplicación, todos los procesos de limpieza de datos, obtención de conjuntos-AP y estructura-AP y el proceso de intersección con el objetivo de obtener el TDA. También brinda al usuario las estadísticas de la limpieza e intersección, así como, un reporte de los datos del atributo procesado, el atributo después de la limpieza y su TDA correspondiente, sin necesidad de verlo en el SGBD utilizado.

B.3.1. Herramientas y tecnologías utilizadas en la implementación de Text Mining Tool v1.0

La herramienta ha sido implementada en lenguaje Java, tomando como IDE de desarrollo el Eclipse v3.3 y como gestor de base de datos PostgreSQL v8.2.1. Para el desarrollo de la interfaz visual se utilizó la biblioteca SWT (*Standard Widged Toolkit*) que incorpora Eclipse, utilizando tecnología RCP (*Rich Client Platform*). Esto posibilita la creación de futuros *plug in* para perfeccionar la herramienta sin necesidad de cambios además de generar el "*producto*", que sería el ejecutable de la aplicación final.

B.3.2. Interfaz Principal

La figura B.1 muestra la interfaz principal de la herramienta *Text Mining Tool V1.0* implementada. Desde dicha interfaz se pueden desencadenar los asistentes que guían la realización de los procesos fundamentales que implementa el sistema: solamente la limpieza de datos, o todo el proceso, desde la limpieza de datos hasta la obtención de la estructura-AP y el TDA asociado al atributo textual procesado. Como se observa, el sistema además es capaz de mostrar los diferentes resultados de dichos procesos: el fichero de *log* donde se recogen los resultados de todos los procesos realizados, las estadísticas del proceso de limpieza de datos y de intersección, así como, muestra los datos obtenidos para el TDA.



	numero	ipropuesta	ipropuestamodificada	tda
3	18588	AMPUTACION INFRACONDILEA	AMPUTACION INFRACONDILEA	{{AMPUTACION,INFRACONDILEA}}
4	25859	EMBOLECTOMIA. BYPASS AXILO-F...	EMBOLECTOMIA. BY-PASS AXILO-FEMORAL I...	{{AXILO-FEMORAL,BY-PASS,EMBOLEC...
5	25931	TROMBECTOMIA M.I.I.	TROMBECTOMIA MII.	{{MII,TROMBECTOMIA}}
6	29221	CLAVO ENCERROJADO	CLAVO ENCERROJADO	{{CLAVO,ENCERROJADO}}
7	32094	ENDOSCOPIA	ENDOSCOPIA	{{ENDOSCOPIA}}
8	32191	LAPAROTOMIA EXPLORADORA	LAPAROTOMIA EXPLORADORA	{{EXPLORADORA,LAPAROTOMIA}}
9	32230	LEGRADO Profilaxis	LEGRADO PROFILAXIS	{{LEGRADO,PROFILAXIS}}
10	32300	CURA DE PIE DERECHO	CURA DE PIE DERECHO	{{CURA,DERECHO,PIE}}
11	32998	REDUCCION ABIERTA Y FIJACION...	REDUCCION ABIERTA Y FIJACION INTERNO ...	{{ABIERTA,AGUJAS,FIJACION,INTERN...
12	33307	LAPAROTOMIA EXPLORADORA	LAPAROTOMIA EXPLORADORA	{{EXPLORADORA,LAPAROTOMIA}}
13	36119	REVASCULARIZACION MII	REVASCULARIZACION MII	{{MII,REVASCULARIZACION}}
14	37110	AMPUTACION INFRACONDILEA I ...	AMPUTACION INFRACONDILEA I (CONDICI...	{{AMPUTACION,CONDICIONAL,INFRA...
15	41689	EXTIRPACION	EXTIRPACION	{{EXTIRPACION}}
16	44784	AMPUTACION SUPRACONDILEA	AMPUTACION SUPRACONDILEA	{{AMPUTACION,SUPRACONDILEA}}
17	50572	BIOPSIA DE PROSTATA	BIOPSIA DE PROSTATA	{{BIOPSIA,PROSTATA}}
18	52053	EXTIRPACION	EXTIRPACION	{{EXTIRPACION}}
19	52620	DRENAJE	DRENAJE	{{DRENAJE}}
20	52912	BIOPSIA DE PROSTATA	BIOPSIA DE PROSTATA	{{BIOPSIA,PROSTATA}}
21	61805	EECC + L.I.O. O.I.(20.O)	EECC + LIO OI(20O)	{{EECC,LIO,OI}}
22	63839	EXTIRPACION	EXTIRPACION	{{EXTIRPACION}}
23	65348	AMPUTACION ABDOMINO PERINEAL	AMPUTACION ABDOMINO PERINEAL	{{ABDOMINO,AMPUTACION,PERINEAL}}
24	67025	C.P.R.	CPR.	{{CPR}}
25	74038	URETROCISTOSCOPIA	URETROCISTOSCOPIA	{{URETROCISTOSCOPIA}}
26	78587	EBA LEGRADO Profilaxis	EBA LEGRADO PROFILAXIS	{{EBA,LEGRADO,PROFILAXIS}}
27	79581	BIOPSIA ZONA PERIANAL	BIOPSIA ZONA PERIANAL	{{BIOPSIA,PERIANAL,ZONA}}
28	80425	C. ARTROSCOPICA	C. ARTROSCOPICO	{{ARTROSCOPICO}}
29	80713	EXTIRPACION	EXTIRPACION	{{EXTIRPACION}}
30	81211	RECAMBIO + INJERTO	RECAMBIO + INJERTO	{{INJERTO,RECAMBIO}}
31	88815	PERITOMIA	PERITOMIA	{{PERITOMIA}}
32	92610	CISTOSCOPIA	CISTOSCOPIA	{{CISTOSCOPIA}}
33	99295	VASECTOMIA	VASECTOMIA	{{VASECTOMIA}}

Figura B.1: Interfaz principal de la herramienta *Text Mining Tool V1.0*.

B.3.3. Ejemplo práctico de utilización de la herramienta implementada

Con el siguiente ejemplo se pretende ilustrar la flexibilidad con que ha sido diseñada la herramienta, mostrando todas las posibilidades y variantes de configuración. También se muestra la realización de las diferentes etapas que componen el proceso desde el atributo textual hasta la obtención de la estructura-AP y el TDA, así como la posibilidad que brinda de mostrar resultados y estadísticas al usuario, del procesamiento realizado.

Todas las operaciones que implementa el sistema han sido creadas a través de un asistente que guía al usuario en la selección de los parámetro de configuración que se requieren, por tanto, el usuario en cada momento podrá regresar a un paso anterior o siguiente, teniendo la posibilidad de rectificar cualquier acción antes de finalizar la operación que está realizando.

Como muestra la figura B.2, el proceso comienza con la posibilidad que se le brinda al usuario de seleccionar un fichero de configuración, si existe, para que pueda reutilizar una configuración que haya sido creada previamente, y de esta forma no tener que introducir en el sistema siempre todos los parámetros de configuración. Como se discutirá más adelante, antes de finalizar el asistente se le brinda al usuario la posibilidad de salvar la configuración que acaba de crear. Una vez que el usuario eligió una configuración, o si no lo desea, y quiere crear una nueva, seleccionando la opción "*Next*" pasará a la página siguiente del asistente. Además se le da la posibilidad de importar y exportar un fichero de configuración determinado, así como eliminar uno existente.

En la siguiente página del asistente (figura B.3), se le brinda la posibilidad al usuario de seleccionar los parámetros de conexión a la base de datos: servidor de base de datos, puerto por el que se realizará la conexión, nombre de la base de datos, usuario y contraseña. Una vez que ha seleccionado dichos parámetros, el usuario debe validar la conexión oprimiendo el botón "*Test*", si se puede establecer la conexión, se habilitará la posibilidad de pasar a la página siguiente, en caso contrario se le mostrará el mensaje de error correspondiente. Como se observa, en cada página del asistente el usuario tiene la posibilidad de cancelar la operación que está realizando mediante el botón "*Cancel*".

En la figura B.4 se muestra la página siguiente del asistente, en la que se recoge toda la información referente a la base de datos y los atributos necesarios para

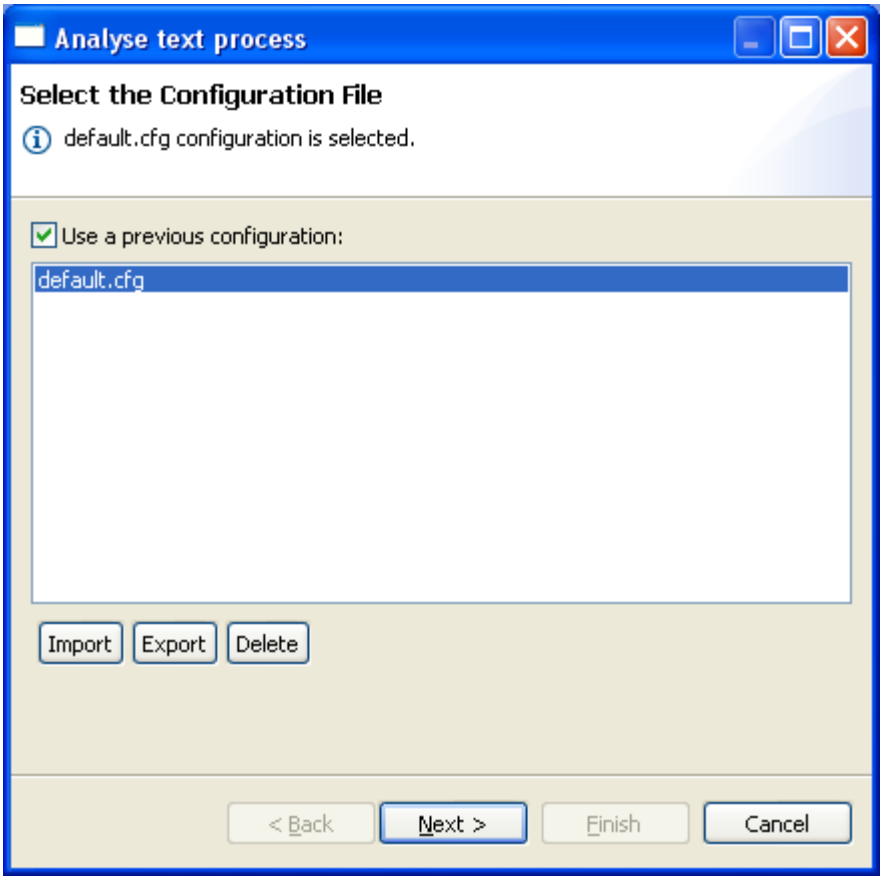


Figura B.2: Selección de un fichero de configuración existente.

realizar los procesos de limpieza de datos y Minería de Textos. Aquí se comienza seleccionando la tabla que contiene el atributo al que se le realizará el proceso de minería, también se seleccionan el atributo llave dentro de la tabla seleccionada, y el atributo textual que será la fuente de todo el proceso, además se selecciona el nombre del atributo que contendrá el TDA generado.

En esta página también se le da la posibilidad al usuario, de que aunque haya escogido realizar el proceso completo desde el atributo textual hasta el TDA, que si no lo entiende o desea, puede elegir entre realizar o no, el proceso de limpieza sobre el atributo fuente, marcando o desmarcando el *check box* "*Clean*". En caso que elija realizar la limpieza, debe especificar el atributo de la base de datos donde se escribirá el resultado de dicho proceso.

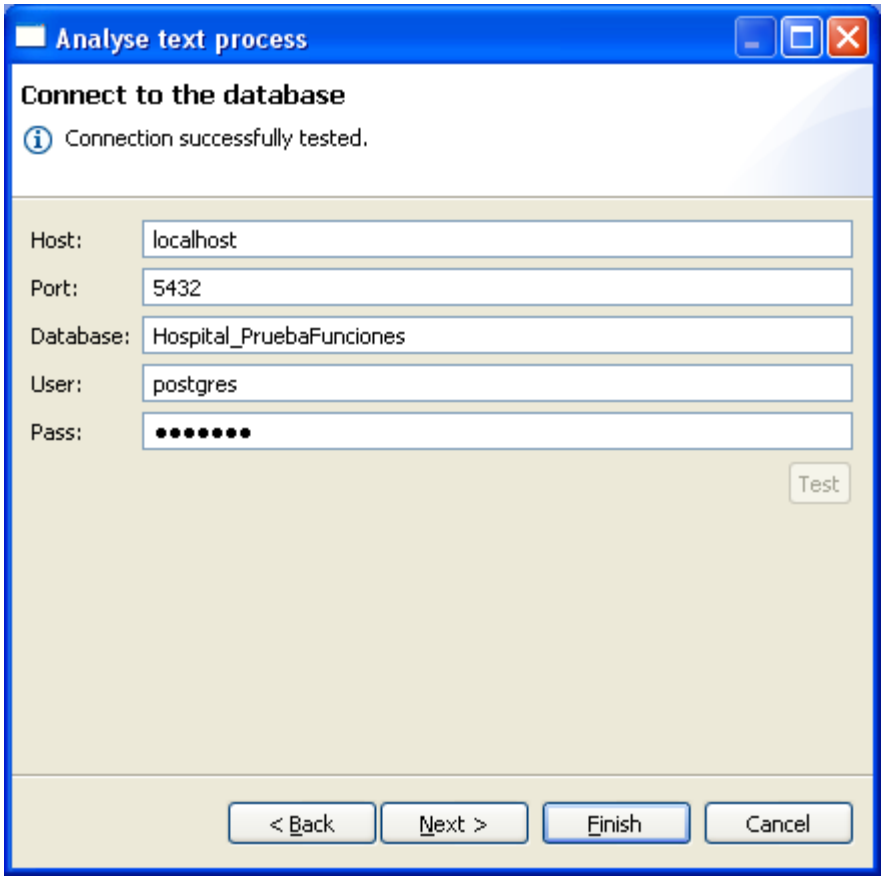


Figura B.3: Selección de los parámetros de conexión con la base de datos.

El la figura B.5 se muestra la siguiente página, una vez que se ha realizado con éxito la conexión al servidor de base de datos y seleccionado los atributos a procesar. En dicha página se le da la posibilidad al usuario de seleccionar los ficheros que son requeridos para realizar la limpieza de datos, y el proceso de Minería de Textos: el fichero de sinónimos, acrónimos y de las palabras de parada.

Por último antes de realizar el procesamiento del atributo textual, se le brinda la posibilidad al usuario, como se muestra en la figura B.6, de configurar las salidas que desea se le muestren como resultado del proceso y de salvar la configuración que ha elegido dentro de todo el proceso. Para ello puede elegir entre no salvar la configuración realizada, sobreescribir la que eligió inicialmente (solo en caso que eligiera una previamente), o salvarla como una nueva configuración.

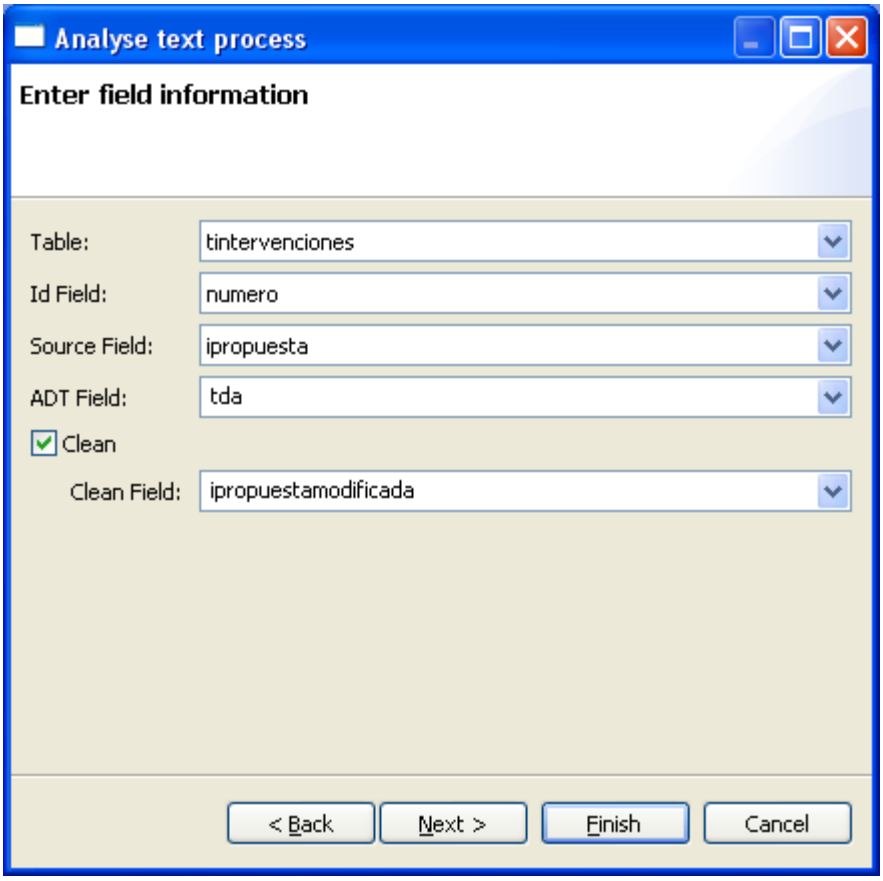


Figura B.4: Selección de atributos a procesar.

Para los resultados a mostrar del proceso realizado, se le brinda la posibilidad de mostrar el fichero de *log* que se crea, las estadísticas obtenidas del proceso de limpieza e intersección realizado (según sea el caso) y mostrar los datos que se almacenan en la base de datos, referentes al atributo procesado, el atributo resultante de la limpieza y su TDA correspondiente. Para el caso del fichero de *log*, se muestran los valores de los parámetros utilizados, así como el orden y estado en que va ocurriendo cada proceso.

Las estadísticas que se muestran para el caso de las referentes al proceso de limpieza son: la cantidad de sinónimos, acrónimos y puntos sustituidos, así como el total de cambios realizados. Para el caso de las estadísticas del proceso de intersección, de muestra la cantidad total de tuplas procesadas, la cantidad y porciento de ellas

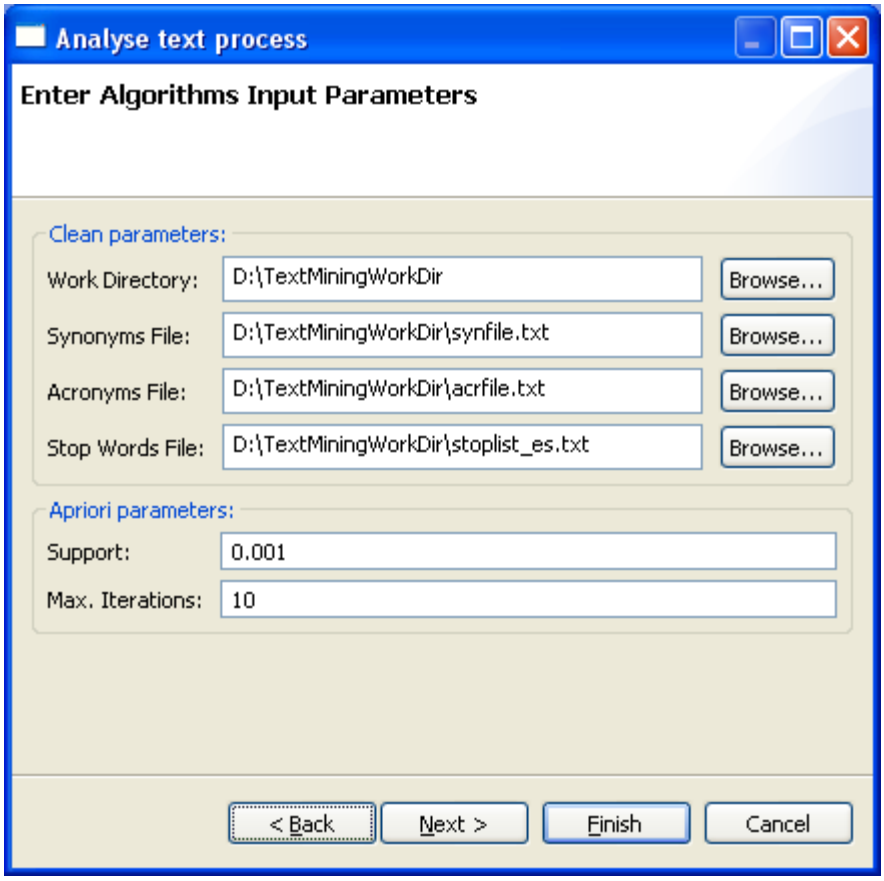


Figura B.5: Selección de los parámetros requeridos para la limpieza y el algoritmo Apriori.

que contienen palabras perdidas, y lo mismo para el total de palabras procesadas.

En las figuras B.7, B.8 y B.9 se muestran ejemplos de los resultados obtenidos en los ficheros de *log*, de estadísticas y los datos obtenidos respectivamente. Estos se mostrarán según los resultados seleccionados en la página del asistente que se muestra en la figura B.6.

Como se muestra en la figura B.9, el sistema brinda la posibilidad al usuario de ver la información resultante del proceso de minería realizado, para ello se muestra, para cada registro, el atributo llave de la tabla procesada, así como el atributo textual procesado antes y después de la limpieza, y finalmente el TDA asociado a

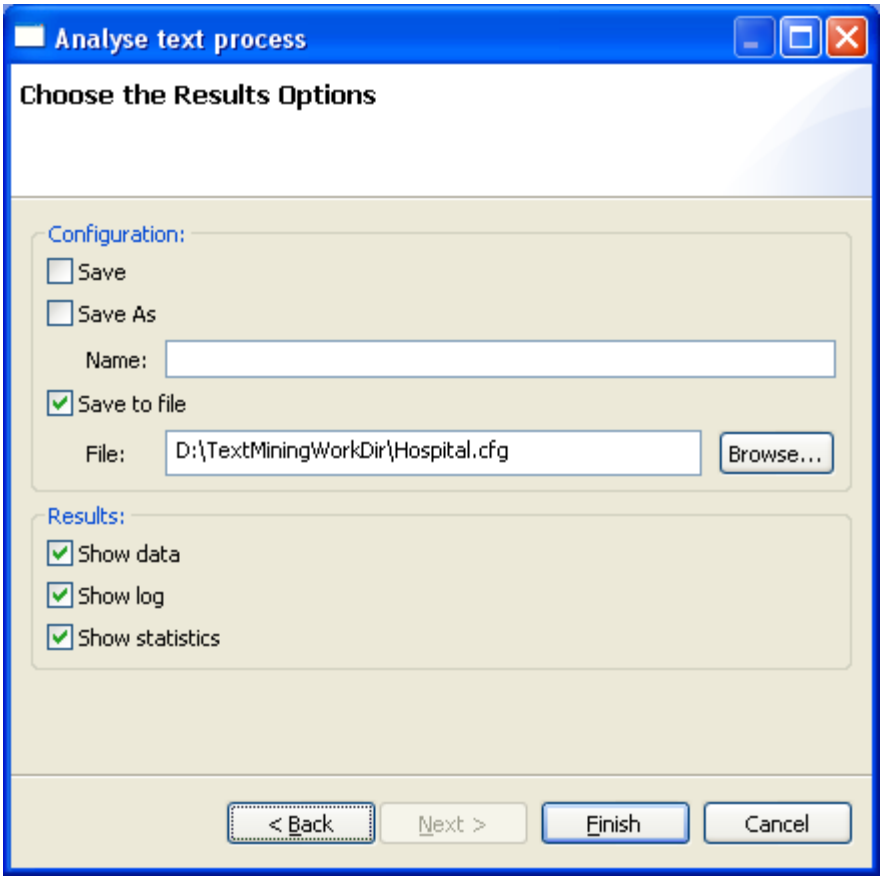


Figura B.6: Selección de opciones de salida.

dicho atributo. Con esta posibilidad el cliente no tiene que utilizar ninguna herramienta del SGBD utilizado para ver dichos resultados, pues el propio sistema se los muestra.

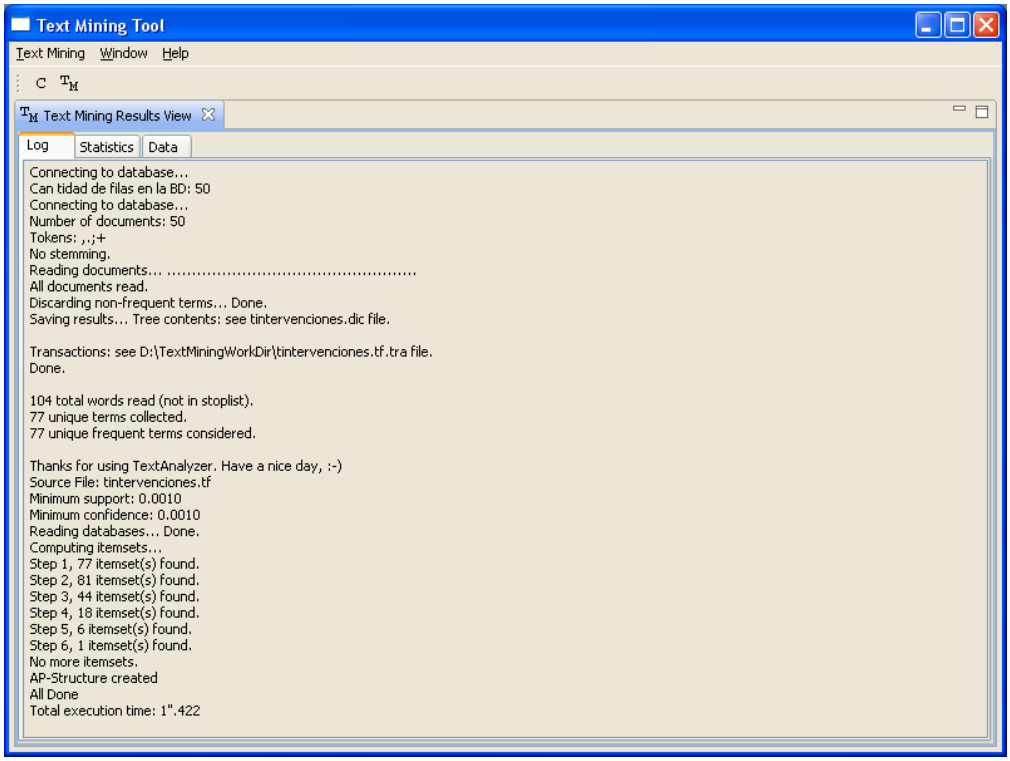


Figura B.7: *Log* de operaciones realizadas.

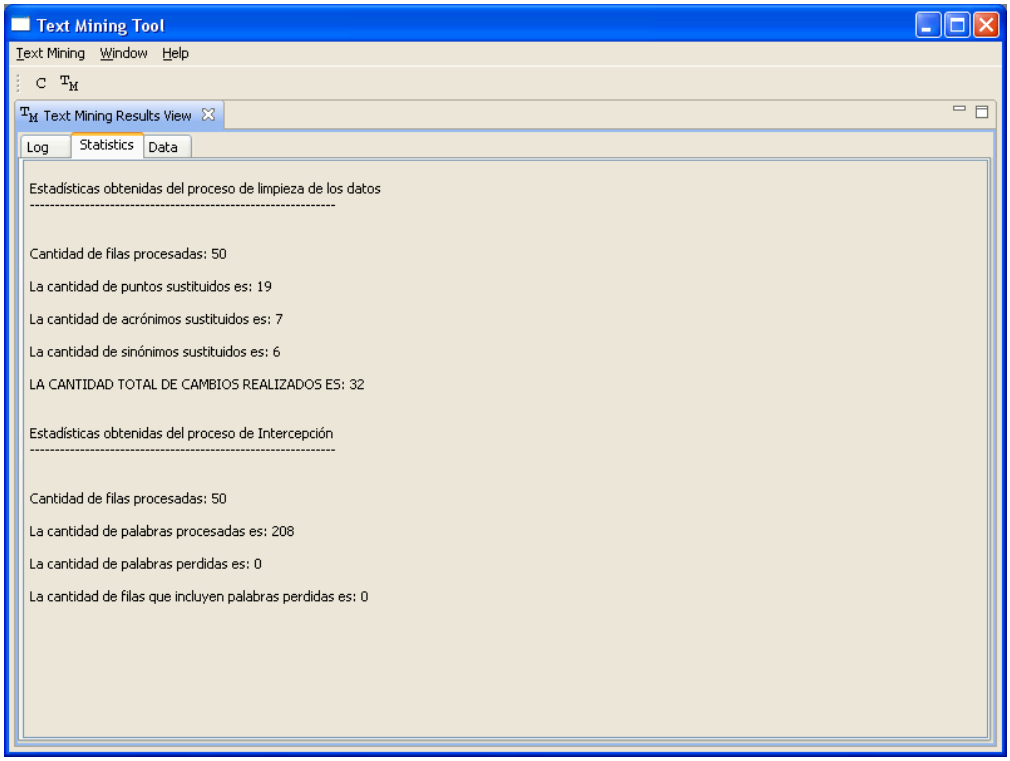
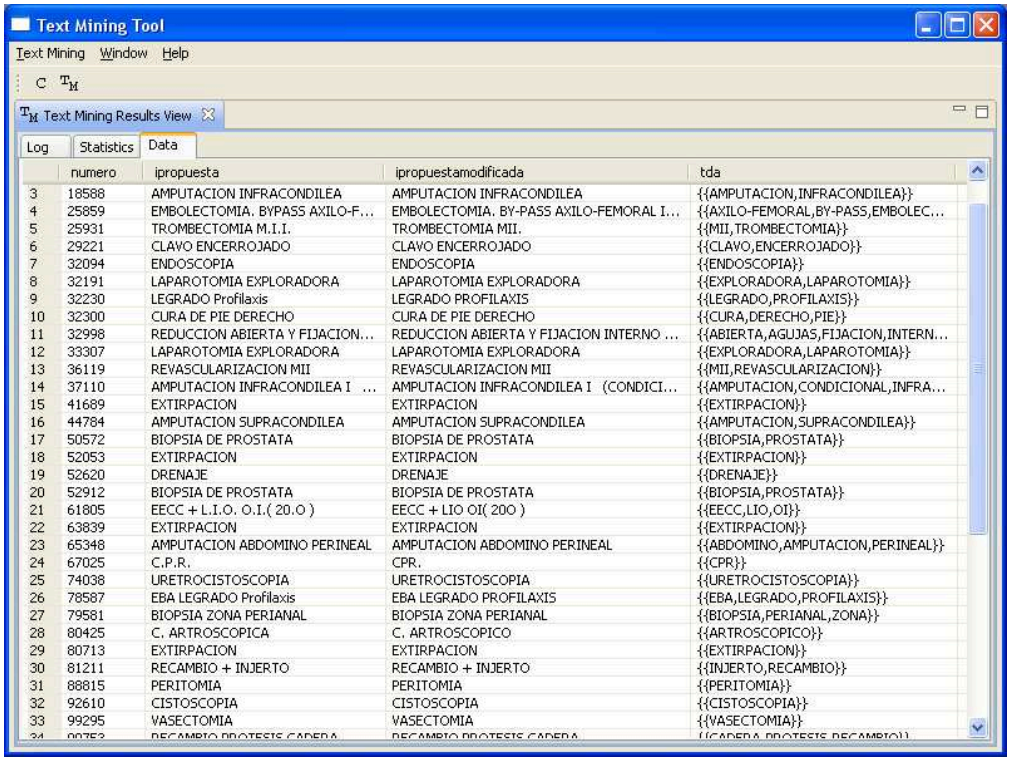


Figura B.8: Estadísticas obtenidas por el sistema.



	numero	ipropuesta	ipropuestamodificada	tda
3	18588	AMPUTACION INFRACONDILEA	AMPUTACION INFRACONDILEA	{{AMPUTACION,INFRACONDILEA}}
4	25859	EMBOLECTOMIA. BYPASS AXILO-F...	EMBOLECTOMIA. BY-PASS AXILO-FEMORAL I...	{{AXILO-FEMORAL,BY-PASS,EMBOLEC...
5	25931	TROMBECTOMIA M.I.I.	TROMBECTOMIA MII.	{{MII,TROMBECTOMIA}}
6	29221	CLAVO ENCERROJADO	CLAVO ENCERROJADO	{{CLAVO,ENCERROJADO}}
7	32094	ENDOSCOPIA	ENDOSCOPIA	{{ENDOSCOPIA}}
8	32191	LAPAROTOMIA EXPLORADORA	LAPAROTOMIA EXPLORADORA	{{EXPLORADORA,LAPAROTOMIA}}
9	32230	LEGRADO Profilaxis	LEGRADO PROFILAXIS	{{LEGRADO,PROFILAXIS}}
10	32300	CURA DE PIE DERECHO	CURA DE PIE DERECHO	{{CURA,DERECHO,PIE}}
11	32998	REDUCCION ABIERTA Y FIJACION...	REDUCCION ABIERTA Y FIJACION INTERNO ...	{{ABIERTA,AGUJAS,FIJACION,INTERN...
12	33307	LAPAROTOMIA EXPLORADORA	LAPAROTOMIA EXPLORADORA	{{EXPLORADORA,LAPAROTOMIA}}
13	36119	REVASCULARIZACION MII	REVASCULARIZACION MII	{{MII,REVASCULARIZACION}}
14	37110	AMPUTACION INFRACONDILEA I ...	AMPUTACION INFRACONDILEA I (CONDICI...	{{AMPUTACION,CONDICIONAL,INFRA...
15	41689	EXTIRPACION	EXTIRPACION	{{EXTIRPACION}}
16	44784	AMPUTACION SUPRACONDILEA	AMPUTACION SUPRACONDILEA	{{AMPUTACION,SUPRACONDILEA}}
17	50572	BIOPSIA DE PROSTATA	BIOPSIA DE PROSTATA	{{BIOPSIA,PROSTATA}}
18	52053	EXTIRPACION	EXTIRPACION	{{EXTIRPACION}}
19	52620	DRENAJE	DRENAJE	{{DRENAJE}}
20	52912	BIOPSIA DE PROSTATA	BIOPSIA DE PROSTATA	{{BIOPSIA,PROSTATA}}
21	61805	EECC + L.I.O. O.I.(20.O)	EECC + LIO OI(200)	{{EECC,LIO,OI}}
22	63839	EXTIRPACION	EXTIRPACION	{{EXTIRPACION}}
23	65348	AMPUTACION ABDOMINO PERINEAL	AMPUTACION ABDOMINO PERINEAL	{{ABDOMINO,AMPUTACION,PERINEAL}}
24	67025	C.P.R.	CPR.	{{CPR}}
25	74038	URETROCISTOSCOPIA	URETROCISTOSCOPIA	{{URETROCISTOSCOPIA}}
26	78587	EBA LEGRADO Profilaxis	EBA LEGRADO PROFILAXIS	{{EBA,LEGRADO,PROFILAXIS}}
27	79581	BIOPSIA ZONA PERIANAL	BIOPSIA ZONA PERIANAL	{{BIOPSIA,PERIANAL,ZONA}}
28	80425	C. ARTROSCOPICA	C. ARTROSCOPICO	{{ARTROSCOPICO}}
29	80713	EXTIRPACION	EXTIRPACION	{{EXTIRPACION}}
30	81211	RECAMBIO + INJERTO	RECAMBIO + INJERTO	{{INJERTO,RECAMBIO}}
31	88815	PERITOMIA	PERITOMIA	{{PERITOMIA}}
32	92610	CISTOSCOPIA	CISTOSCOPIA	{{CISTOSCOPIA}}
33	99295	VASECTOMIA	VASECTOMIA	{{VASECTOMIA}}

Figura B.9: Datos obtenidos por el sistema.

Bibliografía

Abbey, M., 2004. Oracle Database 10 G: Guía de Aprendizaje, primera Edición. McGraw-Hill/ Interamericana de España, S.A., Madrid, España.

Abiteboul, S., January 1997. Querying semi-structured data. En Proceedings of the International Conference on Databases Theory (ICDT), 1–18.

Abiteboul, S., Buneman, P., Suciu, D., 1999. Data on the Web: From Relations to Semiestructured Data and XML, firsts Edición. Morgan Kauffmann Publishing, San Francisco, USA.

Adar, E., May 1998. Hybrid-search and storage of semi-structured information. Master's thesis, MIT.

Agrawal, R., Hohn, S. C., 1996. Parallel mining of association rules. IEEE Transactions on Knowledge and Data Engineering.

Agrawal, R., Imielinski, T., Swami, A., 1993. Mining association rules between sets of items in large databases. En: Proceedings of the 1993 ACM SIGMOD Conference. Washington DC, USA.

Agrawal, R., Srikant, R., Sept 1994. Fast algorithms for mining association rules. En: Proceedings of VLDB. Santiago, Chile.

Anand, T., Kahn, G., 1993. Opportunity explorer: Navigating large databases using knowledge discovery templates. En: Proceedings of the 1993 workshop on Knowledge Discovery in Databases.

Ananthakrishna, R., Chaudhuri, S., Ganti, V., 2002. Eliminating fuzzy duplicates in data warehouses. En: Proceedings of the 28th VLDB Conference. Hong Kong, China.

Apweiler, R., Bairoch, A., Wu, C., Barker, W., Boeckmann, B., Ferro, S., et al., 2004. Uniprot: the universal protein knowledgebase. Vol. 32. pp. D115–D119.

Astrahan, M. M., Blasgen, M. W., Chamberlin, D. D., otros, 1976. System r: Relational approach to database management. TODS 1 (2), 97–137.

Baeza, R., Ribeiro, B., 1999. Modern information retrieval. ACM Press Addision-Wesley New York.

Berzal, F., Cubero, J., Marín, N., Serrano, J., 2001. Tbar: An efficient method for association rule mining in relational databases. Data and Knowledge Engineering 37 (1), 47–64.

Bruni, R., Sassano, A., 2001. Errors detection and correction in large scale data collecting. Advances of Intelligent Data Analysis, Lecture Notes in Computer Science 2189, Springer-Verlag.

Buell, D., Kraft, D., 1981. Performance measurement in a fuzzy retrieval environment. En: SIGIR. pp. 56–62.

Carey, M. J., Kiernan, J., Shanmugasundaram, J., Shekita, E. J., Subramanian, S. N., September 2000. Xperanto: Middleware for publishing object-relational data as xml documents. En: In Proc. of 26th Intl. Conf. on Very Large Data Bases. pp. 646–648.

Cattell, R., Barry., D., 2000. The Object Data Standard: ODMG 3.0. Morgan Kauffmann Publishers.

Celko, J., 2005. SQL for smarties. Advanced SQL Programming, 3rd Edición. Morgan Kaufmann, USA.

Christophides, V., Cluet, S., Simèon, J., 2000. On wrapping query languages and efficient XML integration. pp. 141–152.

Ciravegna, F., 2001. Challenges in information extraction from text for knowledge management. IEEE Intelligent Systems and Their Applications 16 (6), 88–90.

Codd, E., 1970. A relational model for large shared databanks. Communications ACM 13, 377–387.

Cooley, R., Mobasher, B., Srivastava, J., 1999. Data preparation for mining world wide web browsing patterns. Knowledge and information Systems 1 (1), 5–32.

Croft, W., 1995. What do people want from information retrieval? D-Lib Magazine, 22–44.

Dalvi, N., Suciu, D., 2004. Efficient query evaluation on probabilistic databases. En: Proceedings of the 30th VLDB Conference. pp. 864–875.

Dalvi, N., Suciu, D., 2005. Answering queries from statistics and probabilistic views. En: Proceedings of the 31st VLDB Conference. Trondheim, Norway.

Dandretta, G., Junio 2002. Web mining: Implementando técnicas de data mining en un servidor web. Technical Report, Universidad de Belgrano, Buenos Aires.

Davis, R. S., Kent, A. J., Ramamohanarao, K., Thom, J. A., Zobel, J., 1995. Atlas: A nested relational database system for text applications. Transactions on Knowledge and Data Engineering 7 (3), 454–470.

de Miguel, A., Piattini., M., 1999. Fundamentos y modelos de Bases de Datos, segunda Edición. Ediciones Ra-Ma, Madrid, Espanna.

DeFazio, S., Daoud, A., Smith, L. A., Srinivasan, J., 1995. Integrating IR and RDBMS Using Cooperative Indexing. En: Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM Press, Seattle, Washington, pp. 84–92.

Delgado, M., in Bautista, M. M., Sánchez, D., Vila, M., September 2002. Mining text data: Special features and patterns. En: Proceedings of Pattern Detection and Discovery: ESF Exploratory Workshop. Springer-Verlag Heidelberg.

Delgado, M., Sánchez, D., Vila, M. A., Marín, N., 2003. Fuzzy association rules: General model and application. IEEE Transactions on Fuzzy Systems 11 (2), 214–225.

Desai, B., Goyal, P., Sadri, F., 1987. Non first normal form universal relations: an application to information retrieval systems. Information Systems 12 (1), 49–55.

D.Lee, 2002. Query relaxation for xml model. Tesis Doctoral.

EMS-DMSolutions-Inc, 2008. Ems postgresql manager pro.
URL http://www.sqlmanager.net/en/products/studio/postgresql

Escobar, V. H., Diciembre 2007. Minería web de uso y perfiles de usuario: Aplicaciones con lógica difusa. Tesis Doctoral, Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Granada, Granada, España.

Fagin, R., June 1998. Fuzzy queries in multimedia database systems. En: PODS. pp. 1–10.

Fagin, R., Lotem, A., Naor, M., 2001. Optimal aggregation algorithms for middleware. En: Symposium on Principles of Database Systems.

Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, P., 1996. Advances in knowledge discovery and data mining. AAAI/MIT Press.

Fegaras, L., Elmasri, R., September 2001. Query engines for web-accessible xml data. En: Proceedings of the 27th International Conference on Very Large Data Bases. pp. 251–260.

Feldman, R., Aumann, Y., Amir, A., Klösgen, W., Zilberstien, A., 1997. Maximal association rules: a new tool for mining for keyword co-occurrences in document collections. En: Proceedings of the 3rd International Conference on Knowledge Discovery (KDD).

Feldman, R., Dagan, I., 1995. Knowledge discovery in textual databases (kdt). En: Proceedings of the First International Conference on Knowledge Discovery (KDD). Montreal, Canada, pp. 112–117.

Feldman, R., Fresko, M., Kinar, Y., Lindell, Y., Liphstat, O., Rajman, M., Schler, Y., Zamir, O., 1998. Text mining at the term level. En: Proceedings of the 2nd European Symposium of Principles of Data Mining and Knowledge Discovery. pp. 65–73.

Feldman, R., Hirsh, H., 1996. Exploiting background information in knowledge discovery from text. Journal of Intelligent Information Systems.

Fernandez, M., Tan, W., Suciu, D., May 2000. Silkroute: Trading between relations and xml. En: Proceedings of the 9th International World Wide Web Conference. pp. 723–745.

Frawley, W., Piatetsky-Shapiro, G., Matheus, C., 1992. Knowledge discovery in databases: An overview. SAAAI/MIT Press, 57–70.

Fuhr, N., Grossjohann, K., September 2001. Xirql: A query language for information retrieval in xml documents. En: Proceedings of the 24th ACM SIGIR Conference. pp. 172–180.

Fukuda, M., Ñanri, I., 2000. Mining from literary texts: Pattern discovery and similarity computation. Progress in Discovery Science 2000, 518–531.

Galhardas, H., Florescu, D., Shasha, D., 2001a. Declarative data cleaning: Language, model, and algorithms. En: VLDB 2001. pp. 371–380.
URL `citeseer.ist.psu.edu/451854.html`

Galhardas, H., Florescu, D., Shasha, D., Simon, E., 2000. An extensible framework for data cleaning. En: Proceedings of the 16th International Conference on Data Engineering ICDE-00. p. 312.

Galhardas, H., Florescu, D., Shasha, D., Simon, E., Saita, C., 2001b. Improving data cleaning quality using a data lineage facility. En: Design and Management of Data Warehouses. p. 3.

Gelbukh, A., Montes, M., López-López, A., 2002. Text mining at detail level using conceptual graphs. En Soft Computing in Information Retrieval: Techniques and Applications, Germany: Physica-Verlag 2393.

Goldman, R., Widom, J., 2000. WSQ/DSQ: A practical approach for combined querying of databases and the web. En: SIGMOD Conference. pp. 285–296.

Grossman, D., Frieder, O., Holmes, D., Roberts, D., 1997. Integrating structured data and text: A relational approach. Journal of the American Society for Information Science 48, 122–132.

Grossman, D. A., Driscoll, J. R., September 1992. Structuring text within a relation system. En: Proceedings of the 3rd Intl. Conf. on Database and Expert System Applications. pp. 72–77.

Gulutzan, P., Pelzer, T., 1999. SQL-99 Complete, Really. CMP Books, USA.

Hayashi, Y., Tomita, J., Kikui, G., July 2000. Searching text-rich xml documents with relevance ranking. En: Proceedings of the ACM SIGIR 2000 Workshop on XML and Information Retrieval.

Hearst, M., June 1999. Untangling text data mining. En: Proc. of the 37th Annul Meeting of the Association for Computational Linguistics (ACL99). University of Maryland, USA.

Hernández, J., 2002. Exploración de datos masivos (data mining). Transparencias de seminario. Master de Ingeniería del Software Dsic.
URL `http://www.dsic.upv/jorallo/master/curs.html`

Hiemstra, D., de Vries, A., Blok, H., van Keulen, M., Jonker, W., Kersten, M., September 2003. Cirquid: Complex information retrieval queries in a database. En: Proceedings of the VLDB PhD Workshop. Berlin, Germany.

Houtsma, M., Swami, A., October 1993. Set-oriented mining of association rules. Research Report RJ 9567, IBM Almaden Research Center, San Jose, California,.

IBM-Corp., 2008. Db2 9.5.
URL `http://www-306.ibm.com/software/data/db2/9/`

IBM-Corp, 2008. Ibm informix datablade.
URL `http://www-306.ibm.com/software/data/informix/blades/`

Iliopoulos, I., Tsoka, S., Andrade, M. A., et. al, 2003. Evaluation of annotation strategies using an entire genome sequence. Bioinformatics 19 (6), 717–726.

Iritano, S., Ruffolo, M., 2001. Managing the knowledge contained in electronic documents: a clustering method for text mining. En: DEXA Workshop.

Justicia, C., 2004. Formas intermedias de representacion en mineria de texto. Memoria para el Diploma de Estudios Avanzados, Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Granada, Granada, España.

Justicia, C., Bautista, M. M., Sánchez, D., Vila, M., September 2005. Text mining: Intermediate forms for knowledge representation. En: Proceedings of the conference Eusflat-2005. Barcelona, Spain, pp. 1082–1087.

Karanikas, H., Tjortjis, C., Theodoulidis, B., 2000. An approach to text mining using information extraction. En: Proceedings of the Knowledge Management Theory Applications Workshop, (KMTA 2000).

Klösgen, W., 1992. Problems for knowledge discovery in databases and their treatment in the statistics interpreter explora. International Journal for Intelligent Systems 7 (7), 649–673.

Klösgen, W., 1995. Efficient discovery of interesting statements. The Journal of Intelligent Information Systems 4 (1).

Kodratoff, Y., 1999. Knowledge discovery in texts: A definition and applications. Foundation of Intelligent Systems, Lectures Notes on Artificial Intelligence, Springer Verlag 1609.

Kodratoff, Y., 2001. Comparing machine learning and knowledge discovery in databases: An application to knowledge discovery in texts. Machine Learning and its Applications, Advanced Lectures. Lecture Notes in Computer Science Series 2049, Springer, 1–21.

Korfhage, R., 1997. Information Storage and Retrieval. John Wiley, New York.

Lahiri, T., Abiteboul, S., Widom, J., 2000. Ozone: Integrating structured and semistructured data. Lecture Notes in Computer Science 1949, 297+.

Lee, M., Ling, T., Lu, H., Teng, Y., 1999. Cleansing data for mining and warehousing. En: Database and Expert Systems Applications. pp. 751–760.

Lent, B., Agrawal, R., Srikant, R., 1997. Discovering trends in text databases. En: Proceedings of the 3rd International Conference on Knowledge Discovery (KDD).

Lynch, C. A., Stonebraker, M., August 1988. Extended user-defined indexing with application to textual databases. En: Proceedingd of the Fourteenth International Conference on Very Large Data Bases. pp. 306–317.

Mack, R., Hehenberger, M., 2002. Text-based knowledge discovery: search and mining of life-sciences documents. Drug Discovery Today 7 (11 (Supl.)), S89–S98.

Mack, R., Mukherjea, S., Soffer, A., Uramoto, N., Brown, E., Coden, A., Cooper, J., Inokuchi, A., Iyer, B., Mass, Y., Matsuzawa, H., Subramaniam, L. V., 2004. Text analytics for life science using the unstructured information management architecture. IBM Systems Journal 43 (3).

Manolescu, I., Florescu, D., Kossman, D., September 2001. Answering xml queries on heterogeneous data sources. En: Proceedings of the 27th International Conference on Very Large Data Bases. pp. 241–250.

Marín, N., Martín-Bautista, M., Prados, M., Vila, M., June 2006. Enhancing short text retrieval in databases. En: Proceedings of FQAS. Milan, Italy.

Martín-Bautista, M., Martínez-Folgoso, S., Vila, M., September 2008. A new semantic representation for short texts. En: To appear in Proceedings of the 10th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2008), LNCS Springer-Verlag. Turin, Italy.

Martín-Bautista, M., Prados, M., Vila, M., Martínez-Folgoso, S., July 2006. A knowledge representation for short texts based on frequent itemsets. En: Proceedings of IPMU. Paris, France.

Melton, J., 1998. Understanding SQL's Stored Procedures: A complite Guide to SQL/PSM. Morgan Kaufmann Publishers, San Francisco, USA.

Melton, J., 2003. Advanced SQL:1999 Understanding Object-Relational and other Advanced Features. Morgan Kaufmann Publishers, San Francisco, USA.

Melton, J., Eisenberg, A., May 2000. Understanding SQL and Java Together : A Guide to SQLJ, JDBC, and Related Technologies. Morgan Kaufmann, USA.

Melton, J., Simon, A., 2002. SQL:1999- Understanding Relational Language Components. Morgan Kaufmann Publishers, San Francisco, USA.

Microsoft-Corp., Junio 2006. Directivas de seguridad, administración operativa y comunicaciones. Microsoft Developer Network (MSDN).
URL `http://msdn.microsoft.com/es-es/library/ms978348.aspx`

Molina, H., J.Ullman, Widom, J., 2000. Database System Implementation, 1st Edición. Prentice-Hall, New Jersey, USA.

Müller, H., 2003. Semantic data cleansing in genome databases. En: VLDB PhD Workshop.

National-Center-for-Health-Statistics, August 2007. International classification of diseases. tenth revision (icd-10).
URL `http://www.cdc.gov/nchs/about/major/dvs/icd10des.htm`

O2-Tech-Inc, 2007. O2 objet oriented database.
URL `http://www.o2tech.com`

Oracle-Corp, 2007. Oracle database 11g.
URL `http://www.oracle.com/database/index.html`

Oracle-Corp, 2008. Oracle data mining. connectors & extensions.
URL *http : //www.oracle.com/solutions/business¡ntelligence/data − mining.html*

Oracle-Corp-XML, May 2008. Xml technology center.
URL `http://www.oracle.com/technology/tech/xml/index.html`

Paralic, J., May 2001. Knowledge discovery in databases. En: Proceedings of the 2nd Internal Scientific Conference of the Faculty of Electrical Enginering and Informatics, TU Kosice. Kosice, Slovakia, pp. 33–34.

Paralic, J., Bednar, P., 2002. Knowlelge discovery in texts supporting e-democracy. En 6th IEEE International Conference o Intelligent Engineering Systems,INES 2002, 327–332.

P.M, A., 1991. Implementation of extended indexes in postgres. Special Interest Group on Information Retrieval Forum 25 (1), 2–9.

PostgreSQL-Global-Devel-Group, 2008a. Postgresql.
URL `http://www.postgresql.org/`

PostgreSQL-Global-Devel-Group, 2008b. Postgresql documentation.
URL `http://www.postgresql.org/docs`

Prados, M., Peña, C., 2002. Sistemas de Información Hospitalarios: Organización y Gestión de Proyectos. EASAP, España.

Prados, M. A., C.Peña, M., Vila, M. A., Prados, M. B., May 2006. Generation and use of one ontology for intelligent information retrieval from electronic record histories. En: 8th International Conference on Enterprise Information Systems. pp. 565–571.

Prados, R. M., Peña, Y. C., 2004. Gestión del Conocimiento en el Ámbito Hospitalario. EASAP, España.

PRICE, J., 2004. ORACLE DATABASE 10G SQL, 1st Edición. MCGRAW-HILL, London, England.

Raghavan, S., Garcia-Molina, H., 2001. Integrating diverse information management systems: a brief survey. IEEE Data Engineering Bulletin 24 (4), 44–52.

Raghavan, S., Garcia-Molina, H., March 2003. Representing web graphs. En: In Proceedings of the IEEE International Conference on Data Engineering.

Rahm, E., Do, H. H., 2000. Data cleaning: Problems and current approaches. IEEE Data Engineering Bulletin 24 (4), 3–13.

Rajman, M., Besançon, R., October 1997. Text mining: Natural language techniques and text mining applications. En: Proceedings of the seventh IFIP 2.6 Working Conference on Database Semantics (DS-7). Chapam & Hall IFIP Proceedings serie.

Salton, G., 1989. Information Retrieval: Data Structure and Algorithms. Addison-Wesley, Reading, Massachussetts, USA.

Salton, G., 1991. Developments in automatic text retrieval. Science 253, 974–980.

Salton, G., Buckley, C., 1988. Term-weighting approaches in automatic text retrieval. Information Processing & Management 24 (5), 513–523.

Salton, G., McGill, M., 1983. Introduction to Modern Information Retrieval.

Saracevic, T., 1997. Relevance: A review of and a framework for the thinking on the notion in information science (paper review). En: Spark, K., Willet, P. (Eds.), Readings in Information Science. Morgan Kaufmann Publisher, San Francisco.

Schank, R., 1973. Identification of conceptualizations underlying natural language. Compputer Models of Thought and Language, W.H. Freeman Co., San Francisco, 187–247.

Schank, R., 1980. Language and memory. Cognitive Science 4.

Schmidt, A., Kersten, M. L., Windhouwer, M., Waas, F., 2000. Efficient relational storage and retrieval of xml documents. En: In WebDB (Informal Proceedings). pp. 47–52.

Serrano, J., Septiembre 2003. Fusión de conocimiento en bases de datos relacionales: medidas de agregación y resumen. Tesis Doctoral, Universidad de Granada, Granada, España.

Stonebraker, M., Rowe, L., 1986. The design of postgres. En: ACM SIGMOD Conference on Management of Data. Washington DC, USA.

Tan, A., 1999. Text mining: Promises and challenges. Pacific Asia Conference on Knowledge Discovery and Data Mining PAKDD99 workshop on knowledge Discovery from Advanced Databases, 63–70.

Terveen, L. G., Altman, B., Borgida, A., Halper, F., Kirk, T., Lazar, A., McGuinness, D. L., Brachman, R., Selfridge, P. G., Resnick, L. A., 1993. Integrated support for data archeology. International Journal of Intelligent and Cooperative Information Systems 2 (2), 159–185.

The-Eclipse-Foundation, 2008. Eclipse - an open development platform. URL `http://www.eclipse.org/`

Thuraisingham, B., 1999. Data mining: Technologies, techniques, tools and trends. CRC Press.

Ullman, J., Windom, J., 1997. A First Course in Database Systems, 1st Edición. Prentice-Hall, New Jersey, USA.

VanRjsbergen, C., 1979. Information retrieval. Butterworths, London Second Edition 7 (1), 2–10.

Vasanthkumar, S. R., Callan, J. P., Croft, W. B., 1996. Integrating inquery with an rdbms to support text retrieval. Bulletin of the Technical Committee on Data Engineering 19 (1), 24–33.

Vila, M., 2002. Modelos de datos avanzados: Bases de datos orientadas a objetos. Apuntes de clase.

Vila, M. A., Delgado, M., Sánchez, D., 2000. Acquisition of fuzzy association rules from decimal data. En: Barro S, Marín R, editors. Fuzzy logic in medicine, Physical-Verlag.

Wanson, D. R., 1986. Subjetive versus objetive relevance in bibliografic retrieval system. Library Quartely 56, 389 –398.

Whitemarsh-IS-Corp., 2000. Sql-mm standards: Sql multimedia and application packages. part 2: Full-text. ISO/IEC FDIS 13249 (2).
URL (`http://www.wiscorp.com/sqlfulltext.zip`)

Winkler, K., Spiliopoulou, M., 2001. Extraction of semantic xml dtds from texts using data mining techniques. En: Proceedings of the K-CAP 2001 Workshop Knowledge Markup & Semantic Annotation.

Witten, I. H., Moffat, A., Bell, T., 1999. Managing Gigabytes: Compressing and Indexing Documents and Images, 2nd Edición. Morgan Kauffmann Publishers, San Francisco, USA.

Wong, P., Cowley, W., Foote, H., Jurrus, E., Thomas, J., 2000. Visualizing sequential patterns for text mining. Proc. IEEE Information Visualization, 43–50.

Xu, F., Kurz, D., Piskorski, J., Shmeier, S., 2002. Term extraction and mining of term relations from unrestricted texts in the financial domain. En Proceedings of BIS 2002.

Yalagandula, P., Dahlin, M., 2004. A scalable distributed information management system. En: SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications. ACM, New York, NY, USA, pp. 379–390.

Zobel, J., Thom, J. A., Davis, R. S., September 1991. Efficiency of nested relational document database systems. En: In Proc. of the 17th Intl. Conf. on Very Large Data Bases. pp. 91–102.