UNIVERSIDAD VERACRUZANA

FACULTAD DE ESTADÍSTICA E INFORMÁTICA

“CONSULTA POR PALABRAS CLAVE A BASES DE DATOS ESTRUCTURADAS: EL ENFOQUE ‘KWERYDB’ ”

MODALIDAD:
TESIS

COMO REQUISITO PARCIAL PARA OBTENER EL GRADO DE:
LICENCIADO EN INFORMÁTICA

PRESENTA:
LUIS ENRIQUE CONTRERAS VARELA

DIRECTOR:
DR. EDGARD IVÁN BENÍTEZ GUERRERO

CODIRECTORA:

DRA. MARÍA DEL CARMEN MEZURA GODOY

XALAPA, VER. 11 DE FEBRERO 2015

Universidad Veracruzana
Facultad de Estadística e Informática

A QUIEN CORRESPONDA:

Facultad de Estadística e Informática

Dirección
Av. Xalapa esq. Ávila Camacho S/N
Col. Obrero Campesina
CP 91020
Xalapa de Enríquez
Veracruz, México

Teléfonos
(228) 8421700
Exts.14155, 14250
14108, 14106
Fax
(228) 8149990

Internet
fei@uv.mx
http://www.uv.mx/fei
@fei_uv
feiuv
feiuv

Toda vez que el **C. LUIS ENRIQUE CONTRERAS VARELA** de la Licenciatura en Informática ha reunido la aprobación del Director **DR. EDGAR IVÁN BENÍTEZ GUERRERO**, co-director **DRA. MARÍA DEL CARMEN MEZURA GODOY** y Sinodal **MTRO. JORGE GIBRÁN HERNÁNDEZ CALDERÓN** del trabajo recepcional en modalidad de **Tesis** intitulado **"Consulta por palabras clave a bases de datos estructuradas: el enfoque "KweryDB""**, así como la entrega de los CD's con el aval correspondiente, la Academia a mi cargo **autoriza** la colocación del (los) archivos digitales contenidos en el CD en el repositorio de la Facultad de Estadística e Informática así como la impresión de dicho trabajo.

A petición del interesado y para los fines que a el mismo convengan, se extiende la presente en la ciudad de Xalapa, Ver., a los 9 días del mes de febrero de dos mil quince.

Dra. María Karen Cortés Verdín
Coordinador de Academia de
Experiencia Recepcional y Servicio Social

Facultad de Estadística e Informática
Xalapa, Ver.

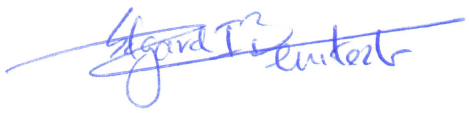
Dra. María Karen Cortés Verdín
Coordinadora de la Academia de
Experiencia Recepcional y Servicio Social
Facultad de Estadística e Informática
Universidad Veracruzana
PRESENTE

Después de revisar el trabajo en modalidad de **tesis**, intitulado "**Consulta por palabras clave a bases de datos estructuradas: el enfoque "KweryDB"**" presentado por el **C. Luis Enrique Contreras Varela,** con número de matrícula **S10011014**, considero que reúne los requisitos de fondo y forma necesarios para sustentar el examen de experiencia recepcional correspondiente, por lo cual doy mi voto aprobatorio.

Sin otro particular quedo a sus órdenes.

Xalapa Enríquez. Veracruz a 9 de Febrero de 2014.

Atentamente



Dr. Edgard Iván Benítez Guerrero
Director de tesis

Dra. María Karen Cortés Verdín
Coordinadora de la Academia de
Experiencia Recepcional y Servicio Social
Facultad de Estadística e Informática
Universidad Veracruzana
PRESENTE

Después de revisar el trabajo en modalidad de **tesis**, intitulado "**Consulta por palabras clave a bases de datos estructuradas: el enfoque 'KweryDB'** " presentado por el **C. Luis Enrique Contreras Varela,** con número de matrícula **S10011014**, considero que reúne los requisitos de fondo y forma necesarios para sustentar el examen de experiencia recepcional correspondiente, por lo cual doy mi voto aprobatorio.

Sin otro particular quedo a sus órdenes.

Xalapa Enríquez. Veracruz a 9 de Febrero de 2014.

Atentamente

Dra. María del Carmen Mezura Godoy
Codirectora de tesis

Dra. María Karen Cortés Verdín
Coordinadora de la Academia de
Experiencia Recepcional y Servicio Social
Facultad de Estadística e Informática
Universidad Veracruzana
PRESENTE

Después de revisar el trabajo en modalidad de **tesis**, intitulado "**Consulta por palabras clave a bases de datos estructuradas: el enfoque "KweryDB"**" presentado por el **C. Luis Enrique Contreras Varela,** con número de matrícula **S10011014**, considero que reúne los requisitos de fondo y forma necesarios para sustentar el examen de experiencia recepcional correspondiente, por lo cual doy mi voto aprobatorio.

Sin otro particular quedo a sus órdenes.

Xalapa Enríquez. Veracruz a 9 de Febrero de 2014.

Atentamente

Mtro. Jorge Gibrán Hernández Calderón
Sinodal

Resumen

Las bases de datos son el almacén de datos de múltiples organizaciones a lo largo del mundo, las cuales día tras día se ven inmersas en una creación masiva de información. A pesar del gran poder que estas herramientas ponen a disposición nuestra, el acceso a bases de datos se encuentra restringido a únicamente personas con la formación tecnológica adecuada. El resto debe conformarse con una comunicación a traves de formularios web u otra interfaz de usuario final que no permite cuestionar a la base de datos libremente.

En este sentido diversos trabajos ofrecen una alternativa, consultas por palabras clave. Este tipo de interacción ha probado ser altamente efectivo en ambientes web (gestores de búsqueda, por ejemplo) y promete ser una forma sencilla de aproximarse a las bases de datos sin necesidad de conocer el esquema ni un lenguaje de consulta como SQL.

Existen diversos tipos de sistemas que ofrecen la posibilidad de cuestionar a una base de datos con sólo un conjunto de palabras clave, los cuales podrían agruparse dentro de dos categorías: sistemas basados en grafos y sistemas basados en Redes Candidatas. Los primeros vacían el contenido de la base de datos en un grafo en el cuál se realizan las búsquedas, mientras que los sitemas basados en redes candidatas hacen uso del esquema de base de datos para generar una consulta SQL.

Ambos tipos de sistemas, a pesar de otorgar una respuesta de la base de datos con únicamente palabras clave, no son lo suficientemente sensibles para determinar cuál es la mejor respuesta a la intención del usuario, pues sólo consideran la longitud de la cadena de joins necesaria para conectar las palabras clave en la base de datos.

Este documento propone una alternativa de consula utilizando palabras clave, llamada *KweryDB*. Ésta reúne característias destacadas de otros sistemas de la literatura y también agrega nuevas métricas, como el número de tablas involucradas en la interpretación, la cantidad de valores del índice que están presentes, junto con las coincidencias de éstos; donde dos o más palabras clave hacen referencia al mismo valor del índice o a la misma tupla de la base de datos. Todo esto para mejorar la interpretación y desambiguación de la consulta, y así aproximarse a la intención del usuario y obtener mejores resultados.

Abstract

Organizations around the world daily create massive amounts of data that are stored in databases. Traditionally, access to these databases is aimed only to people with a technology background. Other kind of users must interact with databases using web forms or another user interfaces that do not allow to query the database freely.

In this context some works have proposed an alternative, querying interfaces using keywords. This form of communication has demonstrated its effectiveness in web environments (like search engines, for example) and it can be an option to query structured databases. In this way the user could query a database without knowing its schema or master a query language like SQL.

There are many types of systems which permit to query the database through a set of keywords, which can be classified in two groups: based on graphs and based on Candidate Networks. Graph-based systems copy the database content to a graph on which search algorithms are executed to find the result. On the other hand, Candidate Network-based ones use the database schema to generate a SQL query.

Despite of providing a response through only a set of keywords, both types are not sensitive enough to determine the best answer, as they only consider the length of the join's chain which is needed to connect all the keywords in the database.

This paper propose an alternative to querying databases based on keywords, called *kweryDB*. It gather features from previous works in the literature, and it also adds new metrics, such as the number of tables involved in the interpretation, the count of index values that are present along with its coincidences; where two or more keywords refer to the same index value or database tuple. All this to improve query interpretation and disambiguation, and therefore get better results.

Índice

Índice de figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Antecedentes

La necesidad de almacenar y organizar grandes volúmenes de información se hizo presente desde hace cientos de años. Antiguamente las instituciones gestionaban sus datos de manera manual, mediante documentos físicos bien organizados y ubicados en sitios estratégicos. De esta manera en universidades, hospitales, bibliotecas y empresas se crearon las primeras bases de datos. En ellas, actividades como realizar cambios en varios archivos, modificar la estructura de los datos o encontrar información específica eran tareas laboriosas y demandantes de tiempo.

No fue hasta 1960, década en que las computadoras se introdujeron al sector empresarial, que aparecieron los primeros sistemas manejadores de bases de datos (o DBMS) para facilitar la labor administrativa en cuanto a información. Gracias a los esfuerzos por separado de CODASYL, organización responsable de la creación del lenguaje de programación COBOL, e IBM se desarrollaron los dos modelos de bases de datos más importantes de la época: el modelo de *red* y *jerárquico*, respectivamente, los cuales se mantuvieron vigentes antes de la aparición del modelo relacional hacia el año 1970. A dichos modelos se les conoce como navigacionales debido a que para realizar consultas los datos debían ser recorridos mediante las relaciones presentes en ellos. Un problema de estos sistemas era la cohesión entre las relaciones conceptuales y la ubicación física de los datos, resultando en falta de abstracción e independencia de datos (Elmasri, 2008).

La aparición del modelo relacional separó con éxito el almacenamiento de datos de su representación. Edgar F. Codd, su creador, se apoyó sobre fuertes fundamentos matemáticos que facilitaron la representación de los datos y la formulación de consultas (Elmasri, 2008). Así, se originó un modelo en el cual el usuario se enfrenta únicamente a la estructura lógica o esquema de base de datos, sin la nece-

sidad de conocer la ubicación subyacente de los datos. Lo anterior propició que características deseables como el control de redundancia, integridad de los datos, escalamiento y seguridad de transacciones se consiguieran con el modelo relacional.

La primera implementación de un RDBMS (Sistema Manejador de Bases de Datos Relacionales), System-R en 1974 a manos de IBM, utilizaba un lenguaje de consulta llamado SEQUEL (siglas para Structured English QUEry Language, que más tarde sería llamado SQL). SEQUEL comprendía facilidades de manipulación y definición de datos, expresadas en palabras en inglés. De este modo, usuarios no programadores fueron capaces de crear e introducir consultas a la base de datos, a diferencia del modelo navigacional donde la interfaz era programática (Elmasri, 2008). Finalmente en 1986 el estándar SQL fue publicado y adoptado por los DMBS de la época y posteriores como el método de acceso definitivo. El lenguaje de consultas ha sufrido algunas modificaciones y mejoras, pero en esencia se preserva igual; como una interfaz basada en álgebra y cálculo relacional que permite la creación, consulta y actualización de bases de datos relacionales.

A pesar de las numerosas posibilidades que brinda SQL para recuperar información de las bases de datos, se ha mostrado incapaz de adaptarse a nuevos requerimientos y tipos de usuario que se inclinan a una comunicación más flexible, y no a las restricciones sintácticas que implica el uso de este lenguaje de consulta.

1.2. Definición del Problema

Las bases de datos están presentes en todo tipo de aplicaciones y la mayoría son accedidas a través del lenguaje de consulta estándar, SQL. Este, a pesar de proporcionar los cánones adecuados para expresar consultas, es una interfaz relativamente de “bajo nivel” en el sentido de ser necesario, por un lado, conocer el esquema de la base de datos de la cual se recabará la información, y por otro, tener conocimientos sobre el lenguaje de consulta.

Para algunos sistemas y usuarios la interfaz tradicional es la adecuada. No obstante, en modernas aplicaciones cuyos esquemas y datos fluctúan constantemente, son complejos o desconocidos, o cuyo público objetivo es un grupo de usuarios sin formación tecnológica que desean acceder a la información desde dispositivos móviles (los cuales no cuentan con el espacio para redactar una sentencia larga), se requieren nuevos mecanismos de acceso que faciliten la interacción directa con la base de datos de manera flexible.

Como alternativa al anterior tipo de interacción, algunos sistemas se decantan por una comunicación en lenguaje natural (Li et al., 2007), bajo la forma de oraciones redactadas a criterio de los usuarios, cuya

semántica es evaluada y traducida posteriormente a una consulta equivalente en SQL. El problema con el procesamiento de lenguaje natural es su complejidad de cómputo, aunado a resultados con un porcentaje considerable de imprecisión. Cuestiones sensibles en aplicaciones donde el tiempo de respuesta debe ser el mínimo y los resultados exactos.

Un enfoque simplificado se observa en los actuales motores de búsqueda, los cuales soportan consultas a partir de palabras clave. Bajo este paradigma, se omite la evaluación de construcciones gramaticales y, al igual que en las interfaces de consulta en lenguaje natural, no se somete a la rigurosidad del esquema de base de datos. Al introducir las palabras adecuadas cualquier usuario es capaz de cuestionar a la base de datos sin conocer su estructura interna. Por ejemplo, dada la base de datos que corresponde al esquema mostrado en la Figura 2.5, en un sistema por palabras clave la consulta *John Activity* podría sustituir a la consulta en *SQL select * from Activity where Name="John" and Name in (select Name from Activity)* (Sarda y Jain, 2001)).

Member

Name	City	Age
John	BO-3492	45
Mary	BO-3492	31
Tom	NY-1308	40

Activity

Name	Sport
John	Running
John	Biking
Mary	Rowing
Mary	Swimming
Mary	Running
Tom	Boxing

City

City	City Code	Location	City Information site
Boston	BO-3492	Massachusetts	Boston.msn.com
Chicago	CH-2330	Illinois	Chicago.msn.com
New York	NY-1308	New York	Newyork.msn.com
Atlanta	AT-6040	Georgia	Atlanta.msn.com
Birmingham	BM-4343	Alabama	Birmingham.msn.com

Figura 1.1: Ejemplo esquema de B.D. Member-Activity-City

La técnica anterior nació en el área de Recuperación de Información (Information Retrieval, en inglés), la cual consiste en encontrar datos en un entorno de información no estructurada (Manning et al., 2008), enfoque totalmente diferente al del orden y relación entre datos que poseen las bases de datos.

Algunos científicos, como Agrawal et al. (2002) o Bhalotia et al. (2002), desarrollaron herramientas que soportan consultas por palabras clave en bases de datos relacionales. En sus estudios, las palabras clave son interpretadas ya sea como tablas, atributos o valores de atributo en una tupla y son conectadas para obtener la sentencia SQL correspondiente. A pesar de que no todos los trabajos existentes con respecto a la búsqueda por palabras clave en bases de datos presentan las mismas técnicas, ellos han conseguido dar una solución parcial al problema; respondiendo a las consultas con una lista de las me-

jores *k* posibles opciones (top-k). Lo anterior permite realizar consultas a usuarios sin conocimiento del esquema de la base de datos y sin destreza en el manejo de SQL con la consecuencia de tener que escoger manualmente la respuesta deseada, pues este tipo de sistemas emite una lista en vez del resultado que espera el usuario.

Otro problema que presentan reside en la interpretación de las palabras clave. Aunque efectivamente se identifican tanto metadatos (tablas y atributos) como valores al analizar cada término de la consulta, la selección de las interpretación entre todas las posibles no es la más adecuada pues sólo se considera cómo unir las tablas con el objetivo de otorgar la respuesta más corta y no precisamente la más significativa para el usuario.

Bajo este escenario, el presente documento propone una solución a los problemas encontrados en los sistemas de búsqueda en bases de datos por palabras clave a través del desarrollo de un sistema, llamado *KweryDB*, que reúne las mejores características de dichos trabajos, e implementa algunas técnicas de autoría propia para poder responder a una consulta con sólo una consulta SQL que refleje la intención del usuario.

1.3. Preguntas de Investigación

- ¿Qué se necesita para desarrollar una interfaz de consulta a bases de datos mediante palabras clave?
- ¿En qué sentido se puede evaluar la intención del usuario al introducir una consulta?
- ¿Qué variables y en qué porcentaje son útiles para identificar la mejor respuesta?
- ¿Cuáles son los limites de un sistema de búsqueda por palabras clave en bases de datos como el que se plantea?

1.4. Objetivos

1.4.1. General

Proponer una alternativa de interfaz para bases de datos mediante el uso palabras clave que, integrando las características más destacadas de los sistemas del estado del arte, realice una interpretación de la consulta suficiente para emitir un único resultado que satisfaga la intención del usuario al introducir la consulta.

1.4.2. Específicos

- Analizar los trabajos existentes para determinar sus ventajas e inconvenientes
- Definir un mecanismo de búsqueda por palabras clave en bases de datos estructuradas que combine las ventajas de los trabajos actuales y minimice sus inconvenientes
- Validar la solución experimentalmente mediante la creación de un prototipo

1.5. Justificación

La relevancia de conseguir los anteriores objetivos es trascendental; hasta la fecha las bases de datos adolecen de un mecanismo de acceso para no expertos pues SQL es una interfaz muy útil pero poco flexible para nuevas aplicaciones, las cuales demandan una comunicación más sencilla entre usuario y base de datos. Diversos científicos han abordado el problema aplicando conocimientos y técnicas de otras ramas de la computación, proponiendo soluciones parciales. Sin embargo los resultados no son totalmente satisfactorios, y aún quedan muchos puntos por explorar, particularmente la interpretación de las palabras clave o la selección de una única respuesta de acuerdo a la intención del usuario.

Concretamente este proyecto pretende responder si es posible crear una interfaz de consulta a bases de datos mediante palabras clave y que ésta arroje resultados exactos. Se retoman estudios relacionados de diferentes autores, con especial atención a la interpretación de la consulta y la desambiguación de la misma para mejorar el proceso de consulta. Los resultados y conclusiones de este proyecto de investigación serán fructíferos en el ámbito científico ya que aborda la problemática desde una perspectiva nueva, como para desarrolladores de software y usuarios, siendo ellos los potenciales benefactores del mecanismo y la herramienta de software elaborada.

1.6. Alcances y Limitaciones

La intención principal de esta investigación es proponer una solución al problema planteado anteriormente, una alternativa a SQL como interfaz de consulta utilizando palabras clave. Además, como punto secundario, elaborar un prototipo de software que compruebe los hallazgos teóricos. En el desarrollo de dicho prototipo, el énfasis se hace sobre la evaluación de las posibles interpretaciones en las consultas y la satisfacción del usuario.

Debido a la relevancia y actualidad del tema a estudiar, existen diversos artículos científicos que abordan el problema y proponen soluciones interesantes a úste. Por estas razones la labor de investigación

nos invita a revisar la literatura existente relacionada con la problemática; consultas a base de datos por palabras clave, Information Retrieval, etc. Desafortunadamente no todos los repositorios científicos distribuyen su contenido gratuitamente, y esto puede convertirse en un impedimento si en algún momento de la investigación se necesitan nuevas fuentes de información que requieran pagos y no se cuente con acceso por medio de la universidad, pues este proyecto no cuenta con recursos económicos.

1.7. Metodología

El proyecto de investigación se condujo sobre una línea metodológica de tres fases principales. Como primer punto se completó una exhaustiva recopilación y revisión de propuestas referentes al objeto de estudio; consulta por palabras clave a bases de datos.

En la segunda fase, el proyecto abordó la problemática de manera práctica, en busca de proponer una solución. Incluyendo técnicas y métodos, provenientes de los estudios investigados en la primera etapa, que fueron considerados pertinentes y que además, en combinación, responden adecuada y suficientemente a consultas por palabras clave. Además las mejores características de los sistemas revisados se agregaron algoritmos y elementos propios, que se describirán mas adelante. La solución expresa las relaciones entre las estructuras de datos necesarias para el procesamiento de las consultas, además de las inherentes a la base de datos; el mecanismo y la secuencia del procesamiento de consultas en la solución planteada, donde cada paso fue claramente expuesto, paso a paso ilustrando el funcionamiento de todos los pasos y elementos involucrados en el proceso de consulta.

Como tercer fase, se ejecutó la experimentación correspondiente a la propuesta de solución. Dichas pruebas se realizaron mediante un prototipo de software, el cual fue desarrollado para fines de investigación únicamente. De acuerdo a la solución propuesta se diseñaron los experimentos, evaluando elementos como la completitud de los resultados, la interpretación de la consulta y el tiempo de respuesta.

1.8. Estructura del documento

Este documento presenta el desarrollo de una propuesta de un sistema como alternativa al uso de SQL para consultar bases de datos, en concreto mediante una comunicación con palabras clave. Conforme el lector recorra las páginas de esta tesis encontrará la siguiente estructura:

- **Capítulo 2: Trabajos Relacionados**

  Se abordan los trabajos relacionados a la problemática, consulta de bases de datos con solo palabras clave, analizando las propuestas más destacadas de la literatura. En este capítulo se discuten las

ventajas y desventajas de cada propuesta en particular, como de las técnicas que implementan los sistemas de su clase. Al terminar la lectura de este capítulo el lector tendrá una perspectiva mejor del estado del arte respecto a la busqueda en BD por palabras clave.

- **Capítulo 3: KweryDB**

  En este apartado se describe detalladamente cada elemento del centro de la investigación, la propuesta de un sistema de búsqueda por palabras clave para bases de datos. Se expresa la definición formal de la propuesta para después describir con detalle la secuencia de operación del mismo, además se plantea desde el comienzo una consulta que ilustrará cada módulo. Paso a paso la amalgama de técnicas escogidas del estado del arte más las contribuciones de este trabajo se verán reflejadas en interpretaciones adecuadas de las consultas; y cuando éste no sea el caso, también será reportado.

- **Capítulo 4: Validación Experimental**

  Posteriormente se somete al prototipo del sistema a una serie de pruebas, las cuales demostrarán la valía del trabajo. Se busca evaluar antes cualquier otra cuestión la completitud de resultados, pues es el objetivo principal del trabajo responde a la pregunta de investigación de si un sistema de búsqueda en bases de datos puede responder a consultas por palabras clave con la respuesta exacta, es decir la que corresponde a la expectativa del usuario.

- **Capítulo 5: Conclusiones**

  Por último se hace un recuento de las contribuciones de este trabajo al campo científico, poniendo en relieve tanto las ventajas del mismo como sus deficiencias. Éstas últimas darán pauta a nuevos trabajos de investigación que continuen la consecución del objetivo y amplien las posibilidades de la propuesta computacional descrita a lo largo de este documento

Capítulo 2

Trabajos Relacionados

El acceso a las bases de datos se encuentra restringido a únicamente los poseedores de los conocimientos adecuados: expertos en la materia que manejen el lenguaje de consulta con destreza (por ejemplo, SQL) y además conozcan la estructura de la base de datos a consultar, descrita por el esquema de base de datos. Por este motivo, diversos investigadores han propuesto soluciones al problema de cuestionar bases de datos de un modo más simple que el proporcionando por consultas en SQL, principalmente inspirados en técnicas de otra disciplina de la computación, la Búsqueda de Información o IR (acrónimo de Information Retrieval).

2.1. Búsqueda de Información

La Búsqueda de Información (IR, por el término en Inglés) se encarga de encontrar datos en un entorno de información no estructurada que satisfaga una consulta dentro de grandes colecciones de datos (Manning et al., 2008). Las palabras *información no estructurada* hacen referencia a datos que no tienen una estructura clara que sea fácil de analizar automáticamente por computadora, escenario totalmente diferente al presentado por las bases de datos cuya información posee metadatos que denotan la naturaleza de los datos y sus relaciones.

A pesar de la disimilitud de los objetos de estudio de la Búsqueda de Información y de las Bases de datos, las propuestas en el estado del arte para consulta de bases de datos han introducido elementos de IR que aseguran una comunicación más sencilla. En una escueta descripción, la búsqueda por palabras clave proveniente de la Búsqueda de Información o IR consiste en construir un índice invertido de ocurrencia de términos, estructura sobre la cual se realizará la búsqueda. Dicho elemento almacena qué palabra o término se encuentra en cuál documento de la colección, expresados en el diccionario de términos y sus publicaciones respectivamente (véase la Figura 2.1). Sobre ése índice se realizará la búsqueda de las

palabras claves proporcionadas, obteniendo documentos donde se encuentren dichas palabras clave. La presentación de los resultados se expresa como un número *n* de resultados; documentos jerarquizados por algún criterio. Al proceso anterior se le conoce comúnmente como ranking. La manera más usual de jerarquizar los resultados de la búsqueda es aplicar una comparación de frecuencia de los términos sobre la frecuencia de cada término en el índice invertido, mejor conocido como Tf-idf (siglas para Term Frequecy-Inverse Index Frequency).

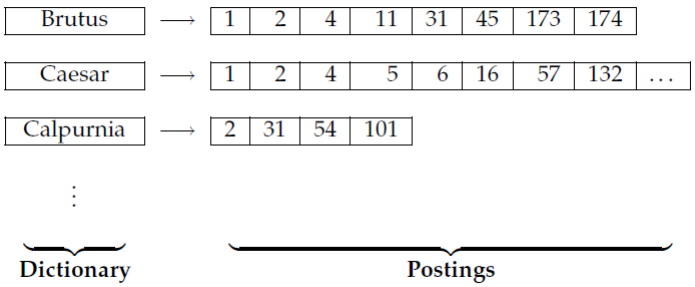


Figura 2.1: Ejemplo de un índice invertido de ocurrencias

Habiendo encontrado éxito al proporcionar interfaces con éste paradigma en el ambiente web, puede pensarse que el incluir la capacidad de consulta por palabras clave en las bases de datos implicaría únicamente una adaptación directa de las técnicas de IR sobre las bases de datos; es decir, añadir un índice de términos que haga referencia a los valores de los atributos textuales de la base de datos y mostrar como resultado las tuplas que contienen dicha palabra clave. omo puede suponerse, esa no es la salida esperada al cuestionar a una base de datos relacional, cuya unidad de resultados son tablas, generadas por alguna operación del cálculo relacional, y no tuplas aisladas. Desafortunadamente ese traslado de técnicas de IR no puede realizarse de un modo tan sencillo, porque la naturaleza de la información en ambos enfoques es totalmente distinta. La riqueza de metadatos que poseen las bases de datos hace inapropiado el método de IR (Chaudhuri y Das, 2009) pues segregarían a la valiosa información estructural (esquema de base de datos) de sus búsquedas y solo se consideraría la presencia de los términos de la consulta en la base de datos.

Este tópico no ha pasado desapercibido por la comunidad científica, que también ha presentado interés en desarrollar una interfaz de búsqueda de información sin importar la índole de su fuente de datos, ya sea estructurada (e.g., bases de datos), semi-estructurada (e.g., XML) o sin estructura (e.g., páginas web, texto plano, etc.), buscando procesar datos de múltiples y heterogéneas fuentes de datos que potencialmente responderían mejor a una consulta (Li et al., 2008). Cabe destacar que el presente trabajo no se encuentra bajo este enfoque, sino únicamente orientado a un tipo de fuente de datos estructurada en específico que son bases de datos relacionales.

Expuesta la influencia de las técnicas de Búsqueda de Información en el problema en cuestión, es posible comenzar a describir los fundamentos de los sistemas de consulta por palabras clave para bases de datos desarrollados hasta el momento.

2.2. Búsqueda por palabras clave en bases de datos

En general los sistemas, a pesar de tener sus propias características, tienen como entrada una consulta que consiste en una serie de palabras clave $Q = k_1, k_2, .., k_n$; que pueden referirse a entidades, atributos o valores. De ellas se elaborará una consulta a una base de datos relacional D. Como salida, una lista jerarquizada (ranked) en orden descendente de relevancia de resultados, conjuntos de tuplas conectadas vía joins de llaves foráneas. La respuesta debe contener todas las palabras clave proporcionadas (Sayyadian et al., 2007). De acuerdo con (Chaudhuri y Das, 2009), el proceso de consulta por palabras clave engloba dos actividades principales:

- **Conversión de palabras a consultas en SQL**.- Este paso requiere una traducción de una consulta por palabras clave a un conjunto de consultas candidatas en SQL, tomando en cuenta tanto el contenido como el esquema de la base de datos.

- **Jerarquización automática de resultados (ranking)**.- Es la tarea de determinar automáticamente el orden de las tuplas resultantes de cada consulta SQL identificada en el paso previo. Lo anterior asiste efectivamente al usuario a navegar entre grandes conjuntos de resultados, ayudándolo a enfocarse en las tuplas más relevantes.

Las soluciones a la búsqueda por palabras clave en bases de datos cubren las actividades nombradas, con distintos enfoques dependiendo de sus métodos y estructuras de datos usadas para realizar dichos pasos, (Chaudhuri y Das, 2009) las clasifica en:

- **Sistemas Basados en Grafos**.- En estos sistemas la base de datos es transformada a un grafo o árbol donde los nodos son tuplas que están relacionadas de distintas maneras por arcos (típicamente relaciones FK-PK).

- **Sistemas Basados en SQL o redes candidatas**.- Bajo este enfoque, cada consulta es traducida a un conjunto de consultas en SQL.

- **Sistemas mixtos**.- Estos sistemas varían en métodos, y aprovechan las ventajas de los dos anteriores.

En las siguientes secciones se describirán los paradigmas de solución para realizar consultas por palabras clave en bases de datos, exponiendo el problema; las estructuras de datos necesarias, su modo de interpretar las palabras clave y la jerarquización de resultados; las ventajas y desventajas que acarrea cada enfoque además de analizar los sistemas del estado del arte encontrados en la literatura.

2.3. Sistemas basados en grafos

2.3.1. Generalidades

La idea principal de este enfoque es vaciar el contenido de la base de datos en un árbol o grafo donde cada nodo o vértice es una tupla de la base de datos y cada arco equivale a una relación join de llave foránea-llave primaria entre los dos nodos (tuplas) que conecta. En algunos sistemas se consideran otro tipo de relaciones (e.g. dependencia funcional). Además del grafo de la base de datos es necesario contar con un índice, al estilo IR, que indique qué términos contienen los nodos del grafo y a cuál tupla de la base de datos hace referencia. Con estas estructuras se espera que a cada término de búsqueda en la consulta le corresponda un conjunto de nodos relevantes (Bhalotia et al., 2002). A partir de dichos conjuntos se fabrica un sub-grafo que conecta a los nodos entre sí, siendo éste una respuesta a la consulta. Para procesar consultas es necesario acuñar pesos o costos a los elementos del grafo de datos, especialmente a los arcos entre nodos, puesto que sus pesos serán consumidos por los algoritmos de construcción del sub-grafo esperado como respuesta y posteriormente la función de jerarquización de resultados.

Por ejemplo, dada una consulta por palabras clave $\{Yu, graph, search, vldb\}$ del grafo de la figura 2.2, se obtienen cuatro grupos de nodos que pertenecen al grafo de la base de datos y corresponden a cada palabra clave: $VYu = \{a_5, a_7\}$, $Vgraph = \{p_4, p_6\}$, $Vsearch = \{p_1, p_2, p_3, p_4, p_5, p_6, p_9\}$, y $Vvldb = \{p_1, p_2, p_3, p_6\}$. Los dos mejores resultados serían los sub-grafos compuestos por los nodos $\{a_5, p_3, p_4\}$ y $\{a_7, p_7, p_6\}$ (Li et al., 2011).

Este tipo de sistemas entonces buscan encontrar el mejor (o mejores, considerando que la mayoría de sistemas buscan los k mejores, o top-k) sub-árbol que contenga nodos que evoquen a las palabras clave proporcionadas como entrada. Por este motivo, algunos autores ((Bhalotia et al., 2002), (Li et al., 2011)) hacen alusión al problema de árboles Steiner mínimos, en el que se busca obtener un subgrafo con el mínimo peso que contenga un subconjunto de vértices (nodos) (Kou et al., 1981).

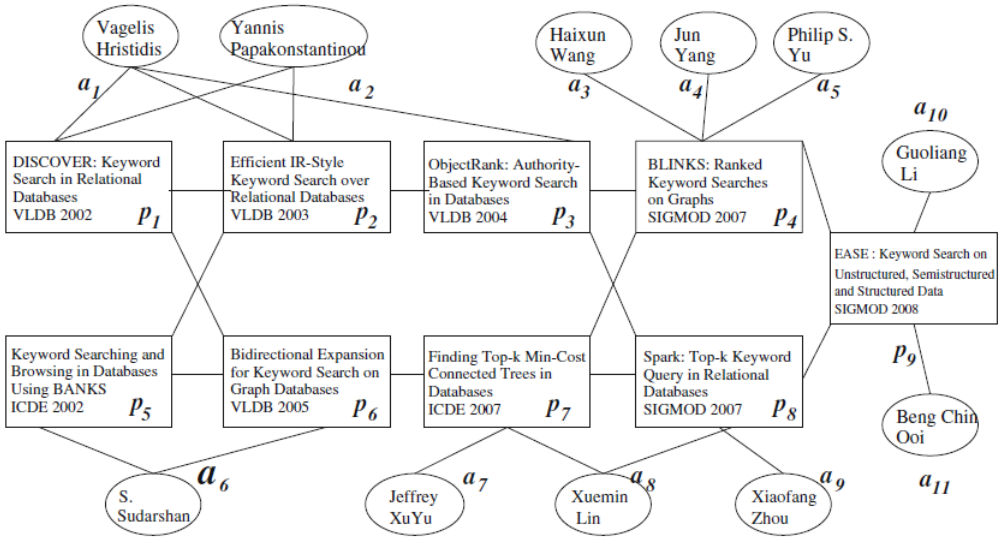


Figura 2.2: Ejemplo grafo de datos

2.3.2. Árbol Steiner, resultado de una consulta por palabras clave

Sea $G = (V, E, d)$ un grafo con un conjunto de nodos V, conectados por un grupo de arcos E que posean una función no-negativa de costo d (métrica). Cada árbol en G que contenga cierto conjunto de nodos $S \subset V$ es un árbol Steiner, a los nodos del conjunto S se les conoce como puntos Steiner. La característica especial de los árboles Steiner es la posibilidad de contener nodos que no formen parte del conjunto S. En la figura 2.3.f se muestra un ejemplo de un árbol Steiner, a partir del grafo dado en la figura 2.3.a. Se obtiene el árbol Steiner mínimo del conjunto de vértices $S = \{v_1, v_2, v_3, v_4\}$, expresado en la figura 2.3.f. Nótese el uso de los nodos $v5$,$v6$ y $v9$, los cuales no pertenecen al conjunto S, característica que diferencia al problema del árbol Steiner mínimo del problema del árbol de expansión mínimo, pues éste solo puede contener los nodos dados en el subconjunto S.

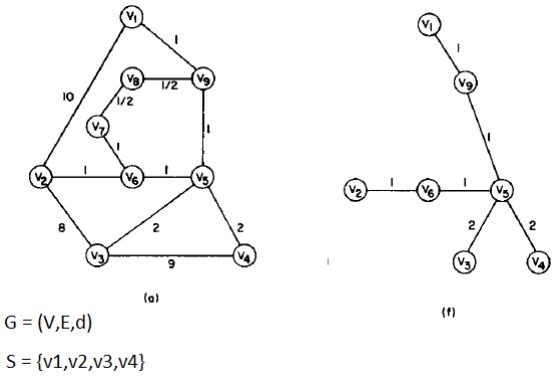


Figura 2.3: Ejemplo de árbol Steiner

Los sistemas de consulta basados en grafos consideran como respuesta un árbol. Éste debe contener enlaces que comiencen desde su nodo raíz y lleguen hasta cada nodo donde se encuentren las palabras clave, es decir un árbol Steiner. Por lo tanto, el problema central del enfoque basado en grafos consiste en obtener el árbol mínimo o el top-k de árboles Steiner mínimos (óptimos) a partir de los nodos que contienen las palabras clave con mayor relevancia (dependiendo del criterio de jerarquización de cada sistema), recuperados a partir del índice invertido del que se habló anteriormente. Lo anterior es un proceso difícil puesto que, el cálculo del árbol Steiner mínimo es un problema NP-Completo, clase de complejidad computacional de la que no se conoce algoritmo eficiente, ni tampoco la posibilidad de crear uno (Cormen et al., 2001). En el peor caso, un problema NP-completo, se resolvería en $O(n^k)$ donde n es el número de variables y k la constante del algoritmo aproximado, en otras palabras, tiempo polinómico. Para los sistemas de consulta basados en grafos las entradas son el número de nodos que tiene el grafo. Lo anterior indica que el tiempo de respuesta, que es una medida de eficiencia, se incrementaría de manera considerable si se consulta una base de datos grande. Esto sin contar el hecho de que los resultados siempre serán aproximados.

Los sistemas basados en grafos aquí presentados incluyen su propio algoritmo de búsqueda aproximada del árbol Steiner mínimo. Por ejemplo, en (Bhalotia et al., 2002) ,se realiza una búsqueda invertida comenzando con los nodos hoja en donde están las palabras clave, mediante una copia del algoritmo del camino más corto de Dijkstra hasta hallar el nodo más cercano que se conecte a los demás.

2.3.3. CSTREES: árboles Steiner compactos

En (Li et al., 2011) el concepto de árboles Steiner evoluciona para dar una mejor solución al paradigma top-k de consulta. Utilizando algoritmos para obtener el árbol Steiner mínimo es posible dar respuesta a consultas bajo el paradigma top-k. El resultado es llamado árbol Steiner de grupo mínimo, pues obtiene el árbol Steiner mínimo a partir de grupos de nodos, que pertenecen al grafo de datos. Cada grupo es el conjunto de nodos que contienen la misma palabra clave. Así, *"Encontrar la mejor respuesta de una consulta por palabras clave puede traducirse al problema de encontrar el árbol Steiner de grupo mínimo en un grafo de base de datos"* (Bhalotia et al., 2002).

La diferencia del enfoque de (Li et al., 2011) con el de sus precursores radica en la evaluación de la relevancia de los nodos n un sentido más amplio que en sistemas como BANKS (Bhalotia et al., 2002). La relevancia de un nodo v es vista como la presencia de un término en v o la posibilidad de llegar a un nodo que posea dicho término a través del nodo v. Si un nodo cumple alguna de las dos condiciones respecto a alguna palabra clave se toma como relevante y, en caso contrario, como irrelevante. A partir del análisis de relevancia de un nodo respecto a un término, se registra en el índice invertido el camino

más corto de un nodo a otro donde se encuentra el término buscado, llamado ruta Voronoi. La inclusión de rutas Voronoi en el índice se realiza durante la publicación, eliminando el cálculo de los caminos más cortos entre nodos durante las consultas pero incrementando el tiempo de publicación y el tamaño del índice invertido (además de una ligera modificación en su estructura, véase la figura 2.4). Un CSTREE es unl árbol que incluye un nodo por cada grupo de nodos (del grafo de datos) que contiene una de las palabra clave proveniente de la consulta, o en su defecto posee una ruta Voronoi a uno de esos nodos. Por lo anterior un CSTREE tiene un nodo o una ruta voronoi correspondiente a cada palabra clave que se encontró base de datos.

Term	Node	EdgeWeight	NodeWeight	Voronoi-path
Graph	p_4	0	$\omega(p_4)+ln2*ln10$	p_4
Graph	p_6	0	$\omega(p_6)+ln2*ln10$	p_6
...	...		...	...
Top-k	p_4	1	$\omega(p_4)+\frac{1}{5}*ln2*ln10$	p_4-p_7
...	...	...	...	...
Yu	p_7	1	$\omega(p_7)+ln2*ln10$	p_7-a_7
Yu	p_4	1	$\omega(p_4)+ln2*ln10$	p_4-a_5
...	...	...	...	...

Figura 2.4: Estrucura del índice invertido en CSTREE

En resumen, dado un nodo centro en el grafo de datos y una consulta por palabras clave, pueden existir muchos árboles Steiner de grupo mínimos pero solo un CSTREE tomando ese nodo como centro. La diferencia entre ellos es que un CSTREE agrupa a todos los MGST (árboles Steiner de grupo mínimo) que comparten el mismo nodo central. Por lo tanto de un CSTREE es posible obtener los MGST que posean dicho nodo central, resultando ideal para el paradigma de consulta top-k donde, al encontrar un solo CSTREE, se puede responder a una consulta por palabras clave en el paradigma top-k, únicamente descomponiendo dicho CSTREE a muchos árboles Steiner de grupo mínimo.

2.3.4. Jerarquización

Se le llama ranking o jerarquización al ordenamiento de los resultados a partir de ciertos criterios. Esto tiene la intención de que el usuario visualice las mejores respuestas al inicio y, en caso de no encontrar la salida esperada, pueda navegar entre los resultados para, eventualmente, hallar la respuesta a la verdadera intención de la consulta.

En IR el usuario espera navegar lo mínimo entre los resultados de sus consultas para encontrar la información deseada. Este requerimiento implica que las funciones diseñadas para jerarquizar los resultados en orden descendente de importancia realicen su trabajo efectivamente y de manera muy rápida, en ocasiones sobre conjuntos de datos grandes. La métrica clásica de la Búsqueda de Información para

esta tarea se llama $tf - idf$ que es en esencia el cociente del número de ocurrencias de un término (term frequency) entre la cantidad de veces que aparece dicho término en el índice invertido (index frequency). Cabe destacar que en la práctica una función de jerarquización tan sensible a frecuencias de términos disparejas o palabras poco relevantes, etc., como la simple división de estos dos valores, resultaría en un ranking desequilibrado donde las primeras opciones estarían sujetas a anomalías como las mencionadas anteriormente. Para contrarrestar estas situaciones, la función $tf - idf$ se ajusta logarítmicamente de modo que las calificaciones tengan la dispersión adecuada y no alteren de modo exagerado los resultados.

El proceso de jerarquización en este tipo de sistemas de consulta es muy similar al usado en IR. Realizando cambios lógicos a dicha función, debido la fuente de datos diferente y al tipo de resultado (árboles Steiner), se garantiza una jerarquización adecuada. El primer cambio es el uso de pesos dentro de la función de jerarquización para los elementos que forman el grafo de datos (arcos y nodos). Los sistemas basados en grafos hacen énfasis en que *no todos los términos que contienen una palabra clave son igual de importantes para la consulta* (Li et al., 2011), pero aplican criterios distintos para designar pesos tanto a nodos como a arcos. BANKS (Bhalotia et al., 2002) toma como medida de relevancia de un nodo n el número de nodos que tienen un arco direccionado hacia dicho nodo n, mientras que los autores de (Li et al., 2011) consideran la aridad del nodo en cuestión y la de los nodos que se conectan con una palabra clave, haciendo más rica la función $tf - idf$. En cuanto a los pesos de los arcos (Bhalotia et al., 2002) hace una diferencia clara entre los arcos directos (que indican una relación de FK a PK) de los invertidos (de PK a FK), asignándoles una calificación más negativa a los enlaces en sentido inverso. El escenario es diferente para sistemas como (Li et al., 2011), pues toman por igual a los arcos entre nodos sean directos e indirectos (algunos sistemas visualizan el grafo de datos como un grafo no dirigido).

Finalmente los sistemas orientados a grafos hasta aquí revisados no evalúan la intención de la consulta. Esto suma importancia a la jerarquización, pues ésta debe solventar dicha carencia de modo que la intencionalidad de las palabras clave dadas es intercambiada por la relevancia entre los términos y la posibilidad de escoger una respuesta entre el top-k de resultados.

Mecanismos de aprendizaje para modificación de calificaciones en las funciones de ranking, que aseguren un ajuste progresivo de la jerarquización de resultados a la intención de las consultas del usuario, son características que se encuentran entre los tópicos de vanguardia de investigación en IR. Incluso nuevos sistemas de consulta a bases de datos por palabras clave utilizan estas técnicas en su secuencia de operación (por ejemplo SAFE (Orsi et al., 2011)).

2.3.5. Desventajas de los sistemas basados en grafos

Recapitulando, los sistemas basados en grafos necesitan de dos estructuras de datos adicionales a la base de datos para ejecutar búsquedas por palabras clave: un grafo de datos y un índice invertido. Es notorio que para poseer el grafo de datos es necesario obtener la información de la base de datos. Los creadores de estos sistemas han llamado a dicho proceso, junto con el de crear el índice invertido que relaciona cada palabra clave con una o más tuplas de la base de datos y uno o más nodos del grafo, con el término ”publicación”. Un proceso proporcionalmente tan largo y pesado como lo sea la base de datos. En principio la publicación implica un problema de consistencia de datos, ya que se realiza copiando el contenido y la estructura de la base de datos en un momento específico, y cambios en alguno de estos dos puntos condicionarían al sistema a realizar otra publicación desde cero. BANKS (Bhalotia et al., 2002) es un sistema que presenta el anterior problema. Otros sistemas más modernos como CSTREE (Li et al., 2011) implementan actualización dinámica, tanto para actualización de los datos como también para cambios en el esquema.

Respecto a las limitaciones de consulta en comparación con sus semejantes basados en redes candidatas es necesario considerar lo siguiente. Al consultar un sistema basado en grafos las respuestas obtenidas serán siempre árboles Steiner que conecten de algún modo a los nodos que posean las palabras clave del grafo de datos, de tal suerte que se pierde la posibilidad de realizar consultas típicas de bases de datos, como obtener todas las tuplas a partir del nombre de una tabla o recuperar solo ciertos datos de las tablas involucradas en una consulta, por ejemplo. Los resultados arrojados serán siempre parte del contenido de la base de datos sin dar formato a la salida.

Otro elemento importante a considerar en los sistemas basados en grafos resulta en la residencia del grafo de datos en la memoria principal. Por ejemplo, BANKS (Bhalotia et al., 2002) asume que el grafo cabe en memoria, argumentando que éste solo contiene el identificador de tupla. Para un grafo de datos con 100 000 nodos y 300 000 arcos, se realiza un gasto de memoria de 120 MB, aproximadamente. Afortunadamente la capacidad de almacenamiento de memoria principal en las computadoras actuales, que ronda entre 4GB y 16GB, permite a los investigadores dejar de lado esta situación. Aunque es evidente que se desperdicia el poder y la eficiencia que otorgan los manejadores de bases de datos, pues el contenido es copiado y la base de datos toma un rol poco participativo en el proceso de consulta.

2.4. Sistemas Basados en SQL o redes candidatas

2.4.1. Generalidades

Al igual que los sistemas orientados a grafos, los sistemas basados en redes candidatas también tienen como base el modelo de índice invertido, así comom la búsqueda y la jerarquización de resultados, propios de IR. La diferencia principal de los sistemas basados en SQL o redes candidatas es que estos abordan el problema sin la necesidad de un grafo de datos, y por consecuente, de navegarlo hasta encontrar los resultados. Se aprovecha la información estructural del esquema de base de datos, convertida en un grafo (llamado grafo de esquema pues solo contiene metadatos), y se utiliza en combinación con un índice invertido que indica qué tupla de cuál tabla contiene cierto término. De manera que el único reto bajo este enfoque es decidir el camino de joins entre las relaciones del grafo de esquema, conocido en la literatura como red candidata, que mejor responda a la consulta (o un número k de redes candidatas para la mayoría de los sistemas de este estilo, pues consideran una búsqueda top-k). Una vez que se tiene la red candidata y las llaves de las tuplas que hacen referencia a las palabras clave, obtenidas a partir del índice invertido, es posible cuestionar a la base de datos en el lenguaje de consulta nativo (por ejemplo, SQL). Con este enfoque se puede escalar a bases de datos muy grandes para las cuales almacenar el grafo en memoria tal vez no sea posible, además de poder continuar sin la necesidad de actualizar o reconstruir representaciones intermedias. *En vez de buscar rutas en un grafo de base de datos, se generan y ejecutan consultas SQL, tarea en la que los DBMS son muy eficientes* (Sarda y Jain, 2001).

Formalmente el problema se define de la siguiente manera. Considere una base de datos con n tablas o relaciones $R_1, R_2, \cdots, R_n$, donde cada relación de R_i tiene m_i atributos $a^i_1, \cdots, a^i_m i$. El **grafo del esquema** G es un grafo dirigido que captura las relaciones de llave foránea-llave primaria en el esquema de base de datos (la dirección la indica el sentido de la relación $FK - PK$). G contiene un nodo R_i por cada relación R_i de la base de datos y un arco $Ri \rightarrow Rj$ por cada relación de llave foránea a llave primaria, expresadas por un conjunto de atributos $(a^i_{b1}, \cdots, a^i_{bl})$ de Ri a un conjunto de atributos $(a^j_{b1}, \cdots, a^j_{bl})$ de Rj, donde $a^i_b k \equiv a^j_b k$. G_u es la versión no direccionada del grafo del esquema G. Una **red de join de tuplas** j es un árbol de tuplas donde por cada par de tuplas adyacentes t_i, $t_j \in j$, donde $t_i \in R_i$ y $t_j \in R_j$, existe un arco (R_i, R_j) en G_u y $(t_i \; JOIN \; t_j) \; \in \; (R_i \; JOIN \; R_j)$. Una **consulta por palabras clave** es un conjunto de palabras clave $k_1, \cdots, k_m$.

El resultado de una consulta por palabras clave es un conjunto de todas las posibles redes de join de tuplas que sean totales, donde cada palabra clave está contenida en al menos una tupla de la red de join, y mínimas, de forma que no se puede quitar ninguna tupla de la red de join y continuar siendo una

red de join de tuplas total. Una red de join de tuplas es el resultado final de una consulta y se abstrae a dos conceptos que están dados en función de los conjuntos de tuplas (tablas de la base de datos) libres: aquellas que poseen tuplas con alguna de las palabras clave o no. **Una red de join de conjuntos de tuplas** J es un árbol de conjuntos de tuplas, tablas de la base de datos, donde para cada R_K^i, R_M^j en J hay un arco $R_i - R_j$ en el grafo del esquema no direccionado G_u. Finalmente, una red candidata C es una red de join de conjuntos de tuplas, donde la instancia I de la base de datos posea una red de join de tuplas $M \in C$ y ninguna tupla $t \in M$ que provenga de un conjunto de tuplas libre $F \in C$ ontenga palabras clave (Hristidis y Papakonstantinou, 2002).

Esta definición es clara pues delimita la fabricación del grafo que corresponde al esquema de base de datos y se enumeran las entradas y salidas de sistemas creados bajo este enfoque. La secuencia de operación de los sistemas basados en redes candidatas no se encuentra indicada en la anterior definición, de cualquier modo la publicación del índice, cálculo de redes candidatas, jerarquización de redes candidatas, traducción y ejecución en SQL, son constantes en sistemas basados en redes candidatas como DBXplorer (Agrawal et al., 2002), DISCOVER (Hristidis y Papakonstantinou, 2002), Kite (Sayyadian et al., 2007) y SQAK (Tata y Lohman, 2008).

A simple vista no se encuentran muchas diferencias de los pasos de ejecución de este enfoque en comparación con sus semejantes basados en grafos. No obstante, una diferencia notoria es el cálculo de redes candidatas en vez de un árbol Steiner. Cabe destacar que la generación de la red candidata mínima es, al igual que el problema del árbol Steiner mínimo, un problema NP-completo, lo que hace necesario un algoritmo heurístico que sopese esta característica y ofrezca buenos resultados. A diferencia de los sistemas basados en grafos, dicho algoritmo realiza la búsqueda de una red candidata solo sobre el grafo del esquema. Como regla general, un grafo de esquema será siempre mucho más pequeño que un grafo de datos que represente todo el contenido de una base de datos, pues éste solo contemplará los metadatos.

Algunos de estos sistemas incluyen características especiales y muy interesantes, como el uso de un vocabulario para sinónimos de términos y elementos del esquema de base de datos (Sarda y Jain, 2001), soporte a múltiples bases de datos heterogéneas induciendo relaciones no explicitas entre las bases de datos (Sayyadian et al., 2007), generación de resultados intermedios durante la ejecución de las sentencias SQL de cada red candidata, soporte de consultas con funciones específicas (por ejemplo promedio, conteo, etc.) (Tata y Lohman, 2008), etc.

A continuación se describen los aspectos más destacados en las dos actividades más importantes del proceso de consulta; indización y jerarquización de resultados, más una enumeración de ventajas y desventajas de los sistemas basados en redes candidatas.

2.4.2. Indización

Como se describió anteriormente, los sistemas de consulta por palabras clave han sido altamente influenciados por las técnicas del área de Búsqueda de Información (IR), las cuales consumen un índice de ocurrencias de términos para proporcionar resultados. Los sistemas orientados a redes candidatas no son la excepción, sosteniendo gran parte del proceso de consulta sobre el índice invertido. Sin embargo, a diferencia del utilizado en sus semejantes basados en grafos, en éstos se requiere almacenar como mínimo la tabla y atributo al que pertenece cada palabra clave, de modo que se aproveche la información estructural de la base de datos y consecuentemente las capacidades de consulta de los DBMS.

Diversas propuestas incluyen novedades al índice típico para este tipo de sistemas. DBXplorer (Agrawal et al., 2002) es el caso más notorio ya que desarrolló dos tipos de índice en función de la granularidad: por columnas o celdas, a los que llama índices *Pub-Col* o *Pub-Cell*. En el índice por columnas, por cada palabra clave se almacenan las columnas que la contienen (mediante un identificador único de columna para toda la base de datos), estableciendo una estructura *tabla.columna - palabra clave*. La otra opción que se plantea es crear un índice basado en celdas, en el cuál se guarda el conjunto de celdas de la base de datos (con un identificador *tabla.columna.id* por cada celda o valor) donde se encuentra cada término. La motivación para posibilitar la implementación de un índice invertido basado en columnas o celdas reside en las variaciones de tres elementos (requerimientos de tamaño y espacio, desempeño de la búsqueda por palabras clave y mantenimiento de la tabla de símbolos) cuyos distintos valores indicarán el tipo de índice a utilizar dependiendo de los objetivos del sistema de consulta.

Son notorias las ventajas del índice por columnas o Pub-Col. En primer lugar, el índice invertido y las consultas a realizar reducen su tamaño considerablemente puesto que nunca existirán más columnas que celdas en una base de datos no vacía, y por tanto al cuestionar al índice con palabras clave se retornan sentencias SQL más cortas que hacen referencia solo a las columnas que evocan a los términos de la consulta. En DBXplorer (Agrawal et al., 2002) se argumenta que debido a las deficiencias en cuanto al manejo de índices por los manejadores de bases de datos en la época (año 2002) donde los índices de texto no estaban disponibles en la mayoría de DBMS, estos índices debían complementarse o sustituirse por los basados en celdas. Hoy en día, sistemas manejadores de bases de datos como SQL Server, MySQL, Oracle, cuentan con soporte a búsqueda de texto completo a través de índices.

2.4.3. Interpretación de la consulta

Los sistemas basados en redes candidatas no poseen una estructura extra donde se almacene el contenido de la base de datos, únicamente cuentan con un índice invertido que indica qué términos se encuen-

tran en qué sitio de la base de datos, además de la información estructural de la base de datos. De modo que la elaboración y el uso de estas estructuras cobran vital importancia, ya que no solo se consumen para denotar dónde se encuentra cada palabra clave en la base de datos, sino también para realizar una interpretación de la intención de la consulta por palabras clave.

La interpretación de la consulta añade variables al proceso global y apoya a conseguir mejores resultados, es un paso que junto con la recuperación de información del índice invertido da un mayor sentido a cada palabra clave. En la mayoría de sistemas se consideran búsquedas top-k de resultados, por tanto los que presentan esta cualidad realizan múltiples interpretaciones para una sola consulta SQL. Los creadores de SQAK (Tata y Lohman, 2008) llaman **Interpretación Candidata** a la interpretación de una consulta por palabras clave, debido a que se generan múltiples interpretaciones potenciales para una misma consulta en función del contexto del usuario, del esquema y del contenido de la base de datos. Una Interpretación Candidata o *CI* se define de la siguiente manera:

Dada una base de datos D que contenga un conjunto de tablas T, cada una con un conjunto columnas $C(t_i)$, una Interpretación Candidata $S = (C, a, F, w)$ donde:

- $C = \{(c_i^j, p) | c_i^j$ es la columna numero j de la tabla i y p es un predicado sobre $c_i^j\}$
- $a \in C$
- F es una función de agregado (por ejemplo la función promedio o máximo).
- $w \in C \cup \phi$, es el nodo que precede a la palabra reservada "with" (opcional).

La figura 2.5 muestra un esquema de base de datos de una universidad con las tablas faculty, department, courses, section, student y enrollment. Dada una consulta *"John num courses"* con la intención de conocer el número de cursos que *John* ha tomado, una potencial respuesta sería *({Student.name [=John], Courses.courseid}, Courses.courseid, count, $w = \phi$)*. Siendo el primer elemento *({Student.name [=John], Courses.courseid}* el conjunto de tablas y atributos referenciados en la consulta (con o sin predicado, en este caso *"John"* es el valor del atributo name de la tabla *Student*); puesto que *"John"* pertenece a la tabla student y la palabra clave *"courses"* evoca a la relación de la base de datos con el mismo nombre. El segundo miembro del conjunto respuesta *Course.courseid* indica la tabla base, el punto de inicio en la creación de la futura red candidata. Por último se encuentra la principal atracción de esta propuesta (Tata y Lohman, 2008), que son funciones agregadas, en este caso la función *count*. La interpretación candidata de la consulta anterior no posee un nodo *w*, como el descrito en la definición, ya que no contiene la palabra *"with"*, que indica un predicado sobre el resultado de la operación SQL group by.

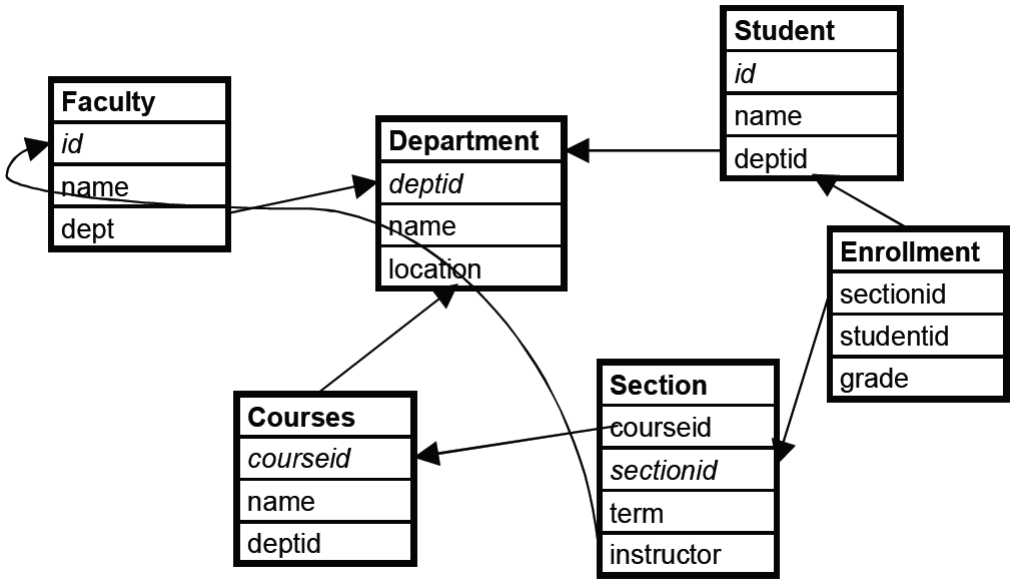


Figura 2.5: Ejemplo esquema de B.D. de una universidad

Cabe destacar que existen más sistemas que realizan una interpretación de la consulta por palabras clave previa a la elaboración de las redes candidatas. Mragyati (Sarda y Jain, 2001) por ejemplo, primero consulta las palabras clave en una base de datos extra llamada vocabulario, donde se almacenan sinónimos para los elementos del esquema de base de datos (tablas y atributos) y también de los distintos términos almacenados como valores en la base de datos. La creación del vocabulario se realiza durante la publicación. La anterior, en coacción con el índice invertido, recupera una interpretación adecuada del sentido de la palabra clave dada, ya sea a partir de las palabras textuales o de los sinónimos previamente asignados. Una vez que el sistema conoce cuál es el sentido de cada palabra clave dada, se verifica cuál es el patrón (o combinación de patrones) que responde mejor a la consulta. En Mragyati (Sarda y Jain, 2001) existen tres patrones de consulta establecidos, que pueden encadenarse para realizar cuestionamientos más elaborados; (R,A,V), (R,A) y (R), donde R se refiere a las tablas o relaciones, A indica atributos de la relación R, y finalmente V a un valor en el atributo A de la tabla R. De acuerdo con esta propuesta, las palabras clave ya interpretadas se ajustarán a un patrón o varios en combinación, y mediante la sustitución de variables en sentencias SQL prefabricadas que corresponden a los patrones se obtiene una consulta SQL que alude a las palabras clave que se proporcionaron como entrada.

2.4.4. Generación de Redes candidatas

Ya exista un módulo de interpretación de consultas o simplemente se verifique la coincidencia de texto de cada palabra clave en el índice invertido (como en DBXplorer (Agrawal et al., 2002), DISCO-

VER (Hristidis y Papakonstantinou, 2002) o Kite (Sayyadian et al., 2007)), los sistemas basados en redes candidatas utilizan un índice invertido para localizar las palabras clave. El paso siguiente en el proceso de consulta resulta en construir las redes candidatas posibles para las palabras clave proporcionadas, con el principio de encontrar el camino más significativo entre las tablas que se vincularon con los términos dados en la consulta. Como se mencionó en párrafos anteriores, la consecución de las mejores redes candidatas es un problema NP, lo cual exige utilizar un algoritmo aproximado para obtener el top-k de redes candidatas. Todos los sistemas basados en redes candidatas aquí mencionados (excepto Mragyati (Sarda y Jain, 2001) puesto a que está orientado una traducción más directa a SQL, en razón de sus patrones predefinidos) presentan uno con algunas diferencias importantes. A continuación se presenta una breve reseña de las diferentes soluciones al problema.

En DBXplorer (Agrawal et al., 2002) se toma el grafo del esquema como un grafo no dirigido y se enumeran árboles de joins que cumplan con dos criterios:

- Las hojas pertenecen al conjunto de tablas de la base de datos que contienen al menos una palabra clave de la consulta
- En total, las hojas contienen todas las palabras clave de la consulta.

La estrategia que permite enumerar las redes candidatas, o árboles de joins, consta de dos procesos. Primero, progresivamente se quitan del grafo G todos los nodos blancos, es decir que no pertenecen a una tabla que contenga una palabra clave, lo que resulta en un grafo G'. Posteriormente se generan la redes candidatas mediante una búsqueda en anchura, carente de una heurística sofisticada que mejore los resultados del problema.

Similar a la anterior es la generación de redes candidatas en SQAK (Tata y Lohman, 2008) puesto que las redes candidatas resultantes también deben ser mínimas y totales, pero además de estos requisitos se suma una condición llamada *claridad de nodo*. Un nodo n es claro si este no posee más de un arco dirigido hacia él. Otra diferencia importante en comparación con el mecanismo de DBXplorer (Agrawal et al., 2002) en torno a generación de redes candidatas es que se alimenta de las Interpretaciones Candidatas previamente encontradas; éstas indican las tablas importantes a consultar, ya sean libres o que posean palabras clave, y también la función de agregación que se aplicará para responder a la consulta. El algoritmo comienza con una copia de una solución parcial, $temp$, inicializada con solo el nodo de la función de agregación (ej., promedio). Luego se localiza el nodo de la Interpretación candidata con la distancia más corta en el esquema de base de datos. La ruta entre $temp$ y el nodo escogido de la CI se añaden a $temp$ si dicho nodo cumple la condición de claridad. De este modo, el algoritmo añade en cada iteración nodos de la Interpretación Candidata a la solución parcial, conforme sea la distancia a $temp$.

En caso de no poder continuar, se remueve el último nodo añadido en conjunto con su ruta y se procede a agregar los nodos restantes de la *CI*. Una vez evaluados todos los nodos de *CI* el algoritmo termina y se obtiene una red candidata.

Una propuesta mucho más compleja e interesante se presenta en DISCOVER (Hristidis y Papakonstantinou, 2002), un sistema que pese a haber sido desarrollado en el mismo año que DBXplorer (Agrawal et al., 2002), extiende en buena medida su mecanismo de generación de redes candidatas mediante una mejor evaluación de la validez de la consulta, más el uso de resultados intermedios que reducen el número de accesos a la base de datos. *"El reto clave es evitar la generación de redes de join de conjuntos de tuplas redundantes"*, tal requisito se traduce a obtener solo redes candidatas mínimas y totales; el menor número de joins posible entre las tablas que contienen a las palabras clave y la inclusión de todas las palabras clave repartidas en las hojas del sub-árbol de esquema o red candidata; respectivamente.

Para obtener redes candidatas con estas características es necesario aplicar dos criterios de exclusión de redes candidatas durante el proceso de creación. El primer criterio remueve las redes candidatas con subárboles con la forma $R^K - S^L - R^M$ donde existen las tablas R y S y también un arco $R \rightarrow S$ en el grafo de esquema. La razón para suprimir las redes candidatas con secciones bajo este formato es que el mantenerlas rompería con la condición de minimalidad de las redes candidatas de DISCOVER (Hristidis y Papakonstantinou, 2002), pues al realizar la operación join de un sub-árbol $R^K - S^L - R^M$ uno de los nodos se incluiría más de una vez. La segunda regla para aceptar una red candidata indica que los nodos vacíos, sin ninguna palabra clave, de cada red candidata únicamente puede ser visitada una vez. Estos deben ser nodos de paso ya que no contribuyen directamente a la consulta, solo conectan a las tablas que sí contienen los términos. Aplicando estas restricciones se asegura también conseguir redes candidatas de n nodos, uno por cada palabra clave dada en la consulta.

Aunado a la purga de redes candidatas, DISCOVER (Hristidis y Papakonstantinou, 2002) presenta una atracción que mejora el desempeño de todo el proceso de consulta; la **generación de resultados intermedios**. Alimentado por un conjunto con k número de redes candidatas mínimas y totales, el sistema planea la secuencia de consultas SQL a la base de datos con la intención de reducir el tiempo perdido en operaciones de entrada y salida. La idea básica del algoritmo planificador es que, iterativamente, encuentre operaciones de join entre dos tablas que se repitan en el conjunto de redes candidatas y por consiguiente las sustituya para realizarlas sólo una vez.

Por otra parte se encuentra KITE (Sayyadian et al., 2007), un sistema diferente del resto de los expuestos en este capítulo, ya que está diseñado para consultar múltiples bases de datos heterogéneas que no están explícitamente relacionadas pero en esencia lo están. En concreto identifica relaciones aproximadas entre tablas de diferentes bases de datos al analizar los datos de ambas tablas y los esquemas

de las bases de datos a conectar (para mayor información véase (Huhtala et al., 1999)). De tal suerte que la generación de redes candidatas tendrá como resultado árboles con nodos referentes a cualquier tabla del conjunto de las bases de datos publicadas. La creación de redes candidatas en KITE (Sayyadian et al., 2007) se resume en las siguientes actividades.

- **Conjuntos de tupla**: El primer paso es obtener los conjuntos de tuplas (o tablas). Se busca en cada tabla *R* del conjunto de bases de datos *D*, usando los índices de todas las bases de datos, las tuplas que contienen palabras claves de *Q*.

- **Grafo de conjuntos de tupla**: Se obtiene consumiendo los conjuntos de tuplas previamente calculados, incluyendo las relaciones FK no explícitas descubiertas, y los esquemas de las bases de datos.

- **Redes candidatas**: Si el grafo no excede el tamaño límite se considera una red candidata. Es necesario que las redes candidatas se subordinen a este primer filtro, ya que al contemplar más de una base de datos como en las otras propuestas, éstas tienen la tendencia a ser de tamaños poco manejables.

- **Redes candidatas condensadas**: En el entorno multi-base de datos que estudia KITE (Sayyadian et al., 2007) el número de redes candidatas *CN* puede crecer exponencialmente en la misma medida que aumenta el número de Bases de Datos, lo que torna la búsqueda top-k muy ineficiente. Por esta razón se incluye n la propuesta una agrupación de redes candidatas; muchas redes candidatas contienen los mismos conjuntos de tuplas con solo diferentes joins asociados y deben ser unificadas para reducir el número de redes candidatas y, por consiguiente, minimizar también la ejecución de consultas en la base de datos.

2.4.5. Jerarquización

Hasta este punto es notorio que el uso de un índice es indispensable en cualquier sistema de consulta por palabras clave a bases de datos, sea basado en grafos, en redes candidatas o de otro tipo. El primer paso en todos ellos es conseguir la ubicación de cada palabra clave a partir del índice. En el particular caso del enfoque basado en redes candidatas, del índice se espera obtener un conjunto de tuplas no vacío por cada palabra clave en la consulta.

Gracias a la influencia de la Búsqueda de Información en este tipo de sistemas, la respuesta típica de una consulta a base de datos, que son conjuntos de resultados desplegados tabularmente (o Result Sets), ha sido sustituida por el formato top-k de resultados, donde cada respuesta es un conjunto de tuplas o

registros de la base de datos unidos a través de operaciones de join. Posteriormente se argumentará por qué este formato de salida, en conjunción con la pobre (o nula) evaluación de la verdadera intención de las consultas, tiende a ser ambiguo y poco apropiado en relación con el resultado que se obtendría ejecutando una consulta SQL que materialice a su semejante por palabras clave, con la misma intención que el usuario al momento que introdujo su consulta.

Por este motivo, es necesario que los sistemas, incapaces de determinar la consulta deseada por el usuario, reordenen la lista de resultados de manera descendente de acuerdo a su relevancia (cuyo concepto cambia ligeramente entre los sistemas), delegando parte de la tarea de desambiguación a otro actor del sistema: el usuario. Éste observará el conjunto de resultados ofrecidos y escogerá el que mejor le parezca. Cabe destacar que la tendencia de los sistemas es desarrollar mejores herramientas que aseguren buenos resultados bajo el paradigma top-k, atacando la problemática de una manera más cercana a la Búsqueda de Información que al objetivo de este proyecto, el cual busca obtener un resultado exacto para cada consulta, a través de la identificación de la intención real de cada consulta y su posterior conversión a SQL. A continuación un recuento de las políticas de ranking entre algunas soluciones basadas en redes candidatas.

MragyatiSarda y Jain (2001) es un sistema que aborda el problema desde una perspectiva sencilla y directa: traducir un conjunto de palabras clave a una consulta SQL. Esta tarea es realizada efectivamente, permitiendo hacer referencia con cada palabra clave a entidades, atributos o valores y, además, considerando sinónimos para dichos elementos proporcionados por el administrador durante la publicación de la base de datos. La secuencia de operación en esta propuesta es la más sencilla de todos los sistemas, debido a que la solución reside más en el paso de palabras clave a consultas SQL que en técnicas para un mejor despliegue de resultados top-k. Sin embargo, a pesar de que Mragyati no cuenta con jerarquización de resultados en un sentido estricto **de las palabras clave se obtiene solo una consulta SQL** y de dicha consulta se ejecuta un ordenamiento jerárquico sobre las tuplas resultantes, basado en el número de tuplas que hacen referencia a cada registro. Siendo R el conjunto de tuplas a desplegar, el esquema de la base de datos indica si la llave primaria K es una llave foránea de otra tabla S. Las tuplas en R están ordenadas, de manera descendente, por la suma de tuplas en S que hacen referencia a cada tupla de R . Esto se consigue mediante la ejecución de la siguiente consulta:

Select R., C as (select count(*) from S where S.F = R.K) from R order by desc C*

Existen también otras propuestas de jerarquización simplistas. DBXplorer (Agrawal et al., 2002) y DISCOVER (Hristidis y Papakonstantinou, 2002), por ejemplo, abordan la jerarquización en razón de simplicidad, bajo la premisa de que *"operaciones de join entre muchas tablas son difíciles de comprender"*, algo similar al ranking en IR donde los documentos que contienen los términos buscados más

cercanos entre ellos son mejor calificados (Agrawal et al., 2002). La magnitud de relevancia es inversamente proporcional al número de joins; organizados del resultado que implique menos operaciones de join al mayor. De igual manera, en SQAK (Tata y Lohman, 2008) el objetivo es encontrar la respuesta más simple. La calificación de una Red Candidata es la suma de los pesos de nodos y arcos; al peso de cada arco se le asigna automáticamente la unidad, mientras que el de cada nodo es determinado a través de un algoritmo de coincidencia de texto aproximado durante la interpretación. Este identifica si una palabra, esté o no escrita correctamente, coincide con un elemento de la base de datos (tablas o atributos) asignando una calificación entre $[1, \infty)$; sí sobrepasa cierto umbral se considera como posible coincidencia. De esta manera, la Red Candidata con el peso menor es seleccionada como la mejor alternativa para una Interpretación Candidata.

Al otro extremo se encuentran sistemas con una jerarquización más complicada que propicie mejores resultados en el paradigma top-k. Tal es el caso de Kite(Sayyadian et al., 2007). En él, debido a estar enfocado a bases de datos heterogéneas, la jerarquización no solo se encarga de seleccionar la red candidata más relevante si no también de evaluar la validez de las relaciones FK-PK descubiertas entre tablas de diferentes bases de datos. Así pues, la función de calificación incluye, además de los factores estándar de jerarquización en este tipo de sistemas, número de joins, ocurrencias en el índice y en la base de datos, evaluación de confianza entre operaciones de join. Asignando total confianza a joins entre tablas de la misma base de datos y menor a las operaciones de unión entre tablas de diferentes bases de datos.

2.4.6. Desventajas

Los sistemas de consulta basados en redes candidatas presentan un mejor aprovechamiento de la información estructural de la base de datos y del DBMS, una ventaja sustancial en comparación con el enfoque de grafos. A pesar de ello, este paradigma no ofrece una solución óptima. En primer lugar, los resultados desplegados por estos sistemas son difíciles de visualizar, ya que se presentan como un conjunto de tuplas unidas por joins, sin proyectar las columnas importantes sino mostrando todas de cada tabla; produciendo en ocasiones resultados muy largos y poco amigables que contrastan con el paradigma de consulta por palabras clave, el cual trata de facilitar la interacción con las bases de datos. En segundo, el proceso de cálculo de redes candidatas, que al ser también un cálculo del árbol Steiner mínimo, es un problema NP-completo y requiere un gasto considerable de tiempo. A pesar de ello, la demora para realizar la búsqueda es mucho menor que en los sistemas basados en grafos, puesto que la entrada del algoritmo es el grafo del esquema de la base de datos, el cual será siempre más pequeño que el grafo de datos. Ambos problemas podrían ser solucionados a partir de la evaluación de la intención de cada consulta que el usuario introduce al sistema a través de información extra proveniente del contexto, como

se argumentará más adelante.

2.5. Conclusiones

A lo largo de este capítulo se han comentado las distintas propuestas que dan solución al problema que apunta este proyecto, consultar por palabras clave a bases de datos, pasando por diversos sistemas ubicados dentro de las dos clases proporcionadas (Chaudhuri y Das, 2009); sistemas basados en grafos y en redes candidatas. Los primeros, sistemas basados en grafos, abordan la problemática volcando el contenido de la base de datos en un grafo donde las tuplas se representan como nodos, y éstos son conectados por arcos que indican las relaciones de llave foránea-llave primaria. Posteriormente, se calcula el sub-árbol mínimo que contenga los términos buscados, obtenidos mediante el uso de un índice que mantiene el registro de las parejas tupla-nodo del grafo. La salida de este tipo de sistemas consiste en un despliegue top-k de respuestas organizadas de acuerdo a la relevancia de cada sub-grafo resultante, computada de manera distinta en cada propuesta.

Por otro lado, los sistemas que se encuentran bajo el enfoque de redes candidatas hacen uso importante del esquema y del manejador de la base de datos, de tal suerte que una consulta por palabras clave pueda ser traducida efectivamente a una consulta SQL. Para realizar este proceso de traducción es necesario identificar el camino de joins que debe realizarse para conectar las tuplas que posean los términos. De igual manera que con los sistemas basados en grafos, estos jerarquizan los resultados en función del criterio de relevancia implementado en cada propuesta, donde el criterio más influyente es la cercanía de los términos buscados.

En ambos enfoques se observa una gran influencia de la Búsqueda de Información. Esto se hace evidente por el uso del índice invertido, en el cual se localiza cada término y su ubicación en la base de datos (identificador de tupla), así como también por el posterior despliegue del top-k de resultados ordenados por relevancia. A pesar de ser una alternativa a consultar una base de datos a través de SQL, aún se requiere que el usuario escoja una de todas las opciones como la correcta. Tal interacción difiere con el objetivo del trabajo, pues en vez delegar tal responsabilidad al usuario se desea que el sistema sea capaz de responder con una única salida, el resultado exacto a una consulta.

Considerando el anterior requisito y las distintas propuestas consultadas, el paradigma basado en redes candidatas aparece como la opción más viable por distintos motivos:

- El cálculo de redes candidatas es más rápido que el de su semejante (Arboles Steiner de Grupo Mínimo) en los sistemas basados en grafos como BANKS (Bhalotia et al., 2002) o CSTREE (Li et al., 2011), ya que su tamaño máximo es el mismo que el del esquema de BD, a diferencia del

grafo de datos donde el tamaño es igual al número de tuplas en la base de datos.

- Para obtener el mejor resultado es necesario interpretar correctamente la consulta por palabras clave. En el paradigma de Redes Candidatas tal requisito es indispensable, ya que los términos se pueden referir tanto a los metadatos como al contenido de la base de datos. Mientras que en los sistemas basados en grafos, la búsqueda recae únicamente sobre el contenido de la base de datos, por lo que no es necesario intrepretar los términos si no sólo buscarlos en el índice.

- En un sistema basado en grafos, el índice invertido tiene una entrada por cada nodo en el grafo de datos, es decir, una entrada por tupla en la base de datos. Aunque la mayoría de sistemas basados en redes candidatas presentan el mismo problema, en DBXplorer (Agrawal et al., 2002) se plantea la posibilidad de resumir el índice invertido en función de los distintos valores y no de los identificadores de tupla.

Por estos motivos la propuesta de solución se encuentra alineada al paradigma de Redes Candidatas, e intenta crear un híbrido con algunas de las características de este tipo de sistemas que mejor se adaptan a los objetivos de la investigación. Específicamente basada en dos sistemas: Mragyati (Sarda y Jain, 2001) y DBXplorer (Agrawal et al., 2002). Del primero se ha tomado el proceso de consulta en general ya que procede de una manera muy sencilla, donde la interpretación de la consulta y la generación de SQL son muy claras. De este modo es posible incluir nuevos módulos con nuevas funcionalidades que extienden su funcionalidad, la cual no contempla el mejor camino de joins para conectar entre los terminos buscados, ni una evaluación exahustiva de todas las posibilidades, por nombrar algunos problemas de Mragyati (Sarda y Jain, 2001). De DBXplorer (Agrawal et al., 2002) se toman dos elementos, especialmente su índice Pub-Col o basado en columnas como se comentó anteriormente.

En la siguientes páginas se describirá detalladamente *KweryDB*, la propuesta de este trabajo con respecto a consultas por palabras clave en bases de datos.

Capítulo 3

KweryDB

El presente trabajo propone una solución al problema en cuestión, consultar una base de datos mediante palabras clave para obtener la consulta SQL correspondiente a la intención del usuario. Para conseguir tal objetivo se ha creado un sistema que permite tal interacción (véase la figura 3.1). Tal propuesta lleva por nombre *KweryDB*, un juego de palabras con las abreviaciones de *keywords* y *database*, términos en inglés para *palabras claves* y *base de datos* respectivamente, y el verbo *query* que significa consultar, de modo que el nombre de la aplicación evoca su función; consutar por palabras clave a bases de datos.

El sistema ha sido implementado en un prototipo (observe la figura 3.2) que se abordará, junto con el desarrollo de un ejemplo, en el resto del capítulo.

3.1. Arquitectura de KweryDB

KweryDB contiene tres módulos principales:

- **Publicador de Base de Datos:** Para poder responder a cualquier consulta el sistema debe conocer la BD, es decir almacenar cierta información de ésta. A tal proceso se le conoce publicación. Para conseguir un desempeño adecuado se requiere que este proceso consuma la menor cantidad de recursos posibles y que sea de fácil mantenimiento, ya que probablemente las estructuras de datos más importantes en el sistema son el esquema de la base de datos (incluyendo los metadatos) y el índice invertido.

  A diferencia de sistemas que necesitan realizar una nueva publicación desde cero cada vez que la BD sufre algún cambio, se propone el manejo de un índice por columnas como el que presenta DBXplorer (Agrawal et al., 2002); donde existe una entrada por cada distinta palabra (no todos los valores se consideran palabras, como es el caso de *"EC2N 11HN"*) en cada atributo de las

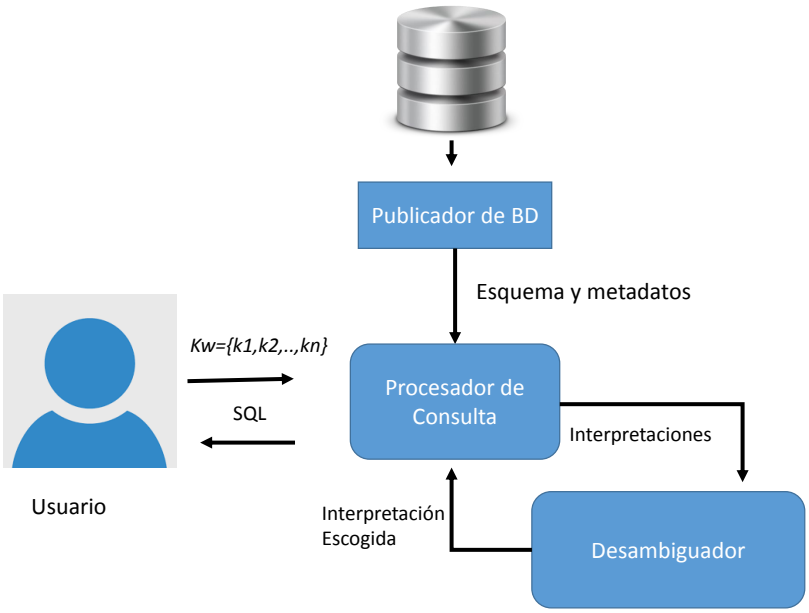


Figura 3.1: Arquitectura del sistema sin contexto

tablas de la BD. También se incluyen disparadores que aseguran un índice invertido actualizado en cualquier momento.

- **Procesador de Consultas:** En este módulo se encarga de coordinar todos los procesos necesarios para la traducción de las palabras clave a su respectiva consulta SQL. Específicamente el procesador de consulta comienza separando las palabras clave para búscarles todas sus posibles interpretaciones, después solicita al módulo desambiguador la mejor opción y finalmente a partir del esquema de base de datos, la interpretación de cada palabra clave y su red candidata (obtenida también a través al módulo Desambiguador); ensambla la consulta SQL.

- **Desambiguador:** Como se ha mencionado anteriormente la desambiguación de la consulta es indispensable si se desea otorgar un único resultado exacto. El módulo Desambiguador se encarga de tal tarea y consta de cuatro pasos:

  1. ***Combinación de Interpretaciones:*** El primero es obtener todas las combinaciones entre las distintas interpretaciones de cada palabra clave.

  2. ***Primera Evaluación:*** De cada combinación se realiza una evaluación, a través de una fórmula, semejante a las tf-idf en IR (Manning et al., 2008), la cual considera la cantidad de tuplas del índice, el número de tablas y la cantidad de palabras clave que no pudieron interpretarse.

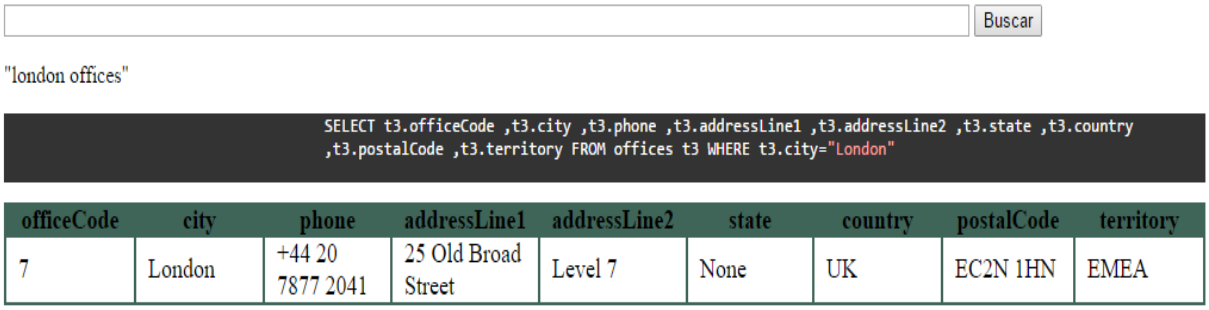


Figura 3.2: Captura de pantalla de la interfaz de consulta

Mientras más bajas sean estas variables mejor calificación tendrá cada posibilidad.

3. ***Cálculo de Redes Candidatas:*** Para conectar a las tuplas que contienen las palabras clave se necesita calcular su Red Candidata; es decir el árbol Steiner del esquema de base de datos donde los nodos a conectar, o puntos Steiner, son el grupo de tablas que contiene las tuplas donde residen los términos buscados.

   Si bien el cálculo de árboles Steiner es un proceso complejo en cómputo (es un problema NP-completo según Tata y Lohman (2008)), tener que calcular uno nuevo desde el inicio por cada combinación de interpretaciones resulta en un gasto de recursos mucho mayor, ya que el número de combinaciones crece exponencialmente. Afortunadamente existen dos situaciones que permiten calcular únicamente una porción reducida del total de posibilidades. El primer motivo es que el grafo en donde se calcula el Steiner Tree es el esquema de la base datos, por lo que las palabras clave pueden estar en la misma tabla y aún sí ser interpretadas como diferentes (un atributo y otro, valores distintos, etc), de modo que las coincidencias serán recurrentes y por ello al computar la Red Candidata de una posiblidad se pueden calcular las de otras. Por el otro lado al haber dado calificación a cada posibilidad, se asume que la respuesta a la consulta por palabras clave debería estar en las posibilidades mejor puntuadas.

   Por lo anterior se decidió que el sistema sólo calculara la red candidata de las tres opciones con la mejor calificación que incluyeran distintas tablas.

4. ***Segunda Evaluación:*** Una vez que se han calculado los tres mejores árboles Steiner, es necesario asignarle a cada combinación de interpretaciones su respectiva red candidata para realizar una segunda evaluación, añadiendo como variable el tamaño del Steiner Tree (número de nodos o tablas incluídas). Si no es posible asignarle una red candidata de las calculadas a una posibiblidad se descarta.

Después de la segunda evaluación se tiene un conjunto de posibilidades nuevamente jerarquizadas, ahora con un nuevo criterio que contempla la cantidad de joins necesarios para unir los términos buscados; mientras menor sea el tamaño de la red candidata más cercanos son los elementos y mejor la calificación. De todas las posibilidades se toma la mejor y sobre ella se realiza la generación de SQL en el Procesador de Consultas. Si más de una opción posee la misma puntuación se realiza una selección aleatoria, puesto que no se tiene otra manera de proceder. En los siguientes capítulos se hará énfasis en esta situación y se comentarán posibles soluciones.

3.2. Publicación de la Base de Datos

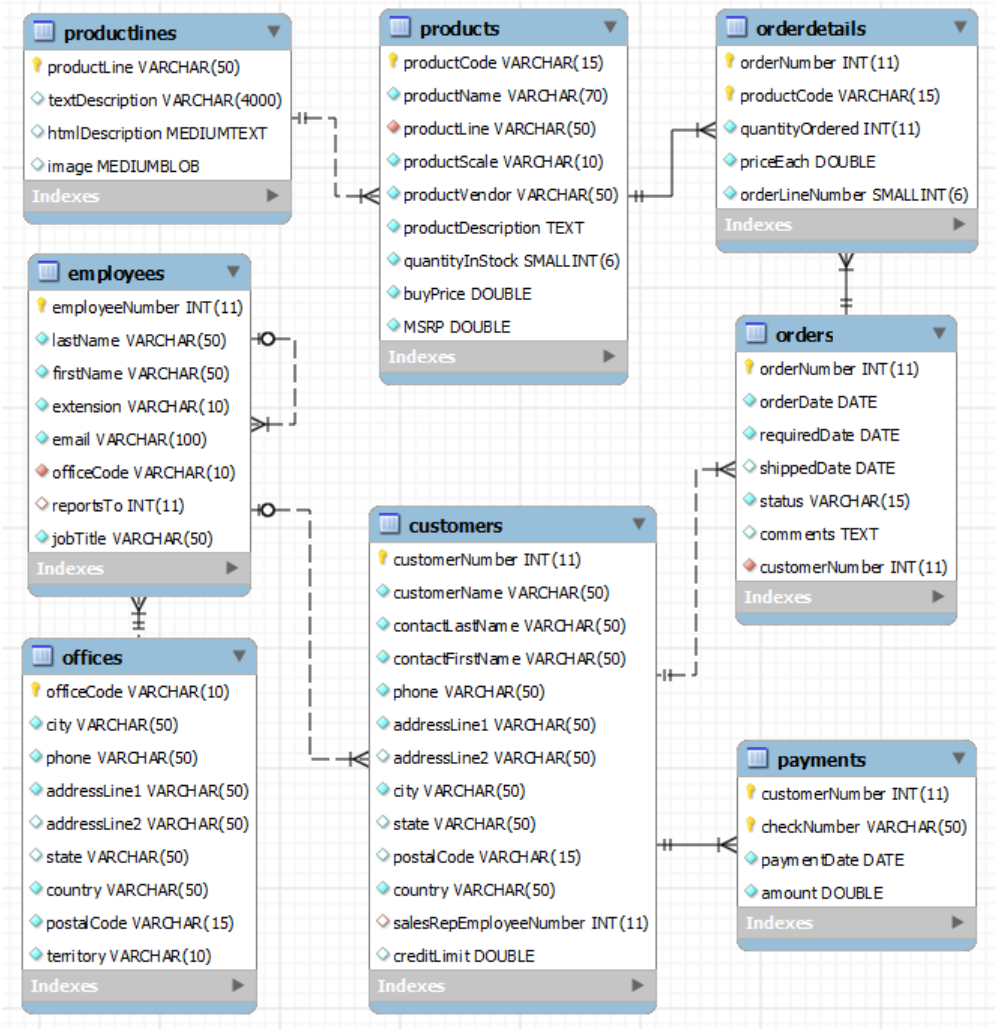


Figura 3.3: Base de Datos tomada como ejemplo, de Doe (2015)

Tómese como ejemplo para toda la descripción de *KweryDB* una base de datos como la de la figura

3.3. En ella se almacena información de una empresa internacional de venta de vehículos y contienen información acerca de empleados, clientes, productos, oficinas, órdenes de productos, etc. Se escogió esta base de datos por motivos de sencillez, ya se usará de manera ilustrativa durante el resto del capítulo.

Consideremos como consulta de ejemplo la siguiente. Se desea conocer qué *clientes provenientes de los estados unidos* adquirieron un automovil *Chevrolet Camaro del año 1969*. La consulta por palabras clave podría aparecer como la siguiente:

```
‘‘1969 camaro customers usa’’
```

En adelante se resolverá pogresivamente la consulta anterior, de manera paralela a la descripción detallada de los procesos y algoritmos que presenta el sistema. Aunque se ilustrará con otros ejemplos cuando la consulta “1969 camaro customers usa” no pueda aplicarse a la situación que se explica.

Cuando el usuario cuestione al sistema lo hará únicamente con palabras clave y probablemente no conozca la estructura de la base de datos ni las relaciones entre los datos de manera exacta. Evidentemente es imposible construir una consulta SQL, la salida deseada, sin saber nada acerca de la base datos; se debe indicar de qué tablas se recavara la información, los predicados de atributos-valor e incluso el modo en que se presentarán los datos (la proyección de atributos); por ello es necesario, como primera operación en el sistema, proporcionar al sistema la información necesaria de la base de datos y almacenarla en ciertas estructuras de datos. A este proceso se le conoce como publicación de Base de Datos.

3.2.1. Esquema y Metadatos

El esquema de base de datos y sus metadatos son indispensables para el procesamiento de consultas; pues sopesan el desconocimiento de la estructura de la base de datos por parte de los usarios, permiten la interpretación de palabras clave como metadatos (tablas y atributos), son requisito para construir el grafo de esquema, etc. De modo que sin esta información es imposible el funcionamiento del sistema.

El primer paso para publicar la base de datos es iniciar una conexión con la base de datos, una vez establecida la conexión se realizan las consultas necesarias para obtener *tablas*, *atributos*, *restricciones de llave foránea* y *llave primaria* (véase el apéndice A.1 para ver las consultas SQL utilizadas). Esta información se almacena en memoria principal ya que será consultada frecuentemente.

El siguiente paso es crear el grafo del esquema de base de datos, necesario para consumir el algoritmo de árbol Steiner (Kou et al., 1981) mínimo que se abordará más tarde. Tal grafo únicamente necesita el nombre o un identificador de cada tabla pues el resto de la información no se utiliza para obtener el árbol Steiner. La construcción del grafo se realiza conectando las tablas con las que tenga una restricción de llave foránea. Por lo tanto habrá el mismo numero de nodos que tablas en la base de datos, conectados por

arcos que simbolizan las restricciones de llave foránea aunque no se almacene en el grafo información alguna de éstas.

Cabe destacar que en *KweryDB* se ha optado por ver el grafo como no dirido, ya que así los árboles Steiner serán más pequeños pues el camino más corto entre dos tablas distintas no será el mismo si la dirección de los arcos está dada en función de las restricciones FK-PK (es poco probable que una tabla A sea referenciada en B y que B esté referenciada en A).

3.2.2. Índice Invertido

El índice invertido de ocurrencias es la estructura de datos básica en los sistemas de consulta por palabras clave a BD, sean basados en grafos o, como el que aquí se presenta, en redes candidatas. Su función es relacionar los términos con su ubicación en la base de datos para que al comparar las palabras clave de entrada con los términos del índice se pueda interpretar como el valor de cierta tupla en una tabla y atributo específico. Con esta información, y con las interpretaciones a elementos del esquema, es posible determinar en dónde está cada punto de intéres en la consulta (aunque no cómo conectarlos, eso se abordará más tarde).

La mayoría de los sistemas basados en redes candidatas implementan el índice relacionando cada término que aparece en la base de datos con la tupla donde aparece. Como puede suponerse, tal manera de poblar el índice resultaría en número elevado de entradas; y por consiguente, una búsqueda poco eficiente en escenarios con bases de datos muy grandes. Además podrían existir muchas entradas del mismo término si en la base de datos tal palabra aparece en tuplas diferentes.

3.2.3. Estructura e idea general

En este trabajo se propone un diseño del índice invertido similar al de DBXplorer(Agrawal et al., 2002), el cual presenta una compresión en relación a las anteriores alternativas, además de aprovechar las ventajas de la búsqueda de texto (o Full Text Serch), la cual permite cuestionar a la base de datos con cadenas aproximadas. El índice registra una entrada por cada término distinto en los atributos de tipo textual (text, varchar, etc) de todas las tablas de la base de datos de tres atributos: la palabra clave, la tabla y el atributo (véase el apéndice para ver la declaración de creación).

Se asume que en las bases de datos un mismo término puede ocurrir repetidas veces, incluso varias veces en un mismo atributo (de la misma tabla). En tal caso, si la palabra clave se encuentra en la tupla A, B o C es irrelevante para la construcción de la consulta SQL, pues en vez de expresar un predicado con la llave primaria (*offices.officeCode=1*) se realiza con el término y el atributo (*offices.country=USA*).

Si la consulta es únicamente esa palabra clave, el resultado incluirá a la tupla A, B y C. Pero si tiene otras palabras que obliguen una operación join entre tablas u otros predicados que filtren el resultado el usuario recibirá un Result Set con la tupla que cumplió con estos requisitos (esta situación se retoma en el siguiente apartado).

Por ejemplo, en la tabla *offices* de la base de datos de la figura 3.3 existen tres entradas de oficinas ubicadas en Estados Unidos. En un índice "clásico" existirían tres entradas (sólo contemplando las ocurrencias en la tabla *offices*), en cambio en un índice Pub-Col, como es bautizado en DBXplorer(Agrawal et al., 2002), sólo existiría un registro de USA en el índice.

Una entrada en un índice Pub-Col índica que existe uno o más elementos en el atributo y la tabla determinados, mientras que un registro de los índices invertidos "clásicos" indica la ocurrencia de un término en la tupla especificada.

3.2.3.1. Llenado del índice invertido

Una vez construído el índice la siguiente tarea es llenarlo con los distintos términos presentes en la base de datos. Es claro que no todos los valores de la base de datos son palabras clave potenciales; cifras, fechas o incluso un objeto binario (o BLOB) no deberían de ser incluídos en el índice. Para asegurar un índice donde cada entrada sea útil y pueda ser eventualmente usada para como interpretación de las palabras clave del usuario, se debe analizar cada término antes de buscarle un lugar en la base de datos.

Los criterio establecidos en el sistema para determinar si una cadena de texto es una palabra clave son los siguientes:

1. **Más letras que números:** No se consideran cadenas que tengan un número mayor o igual a la mitad de letras.

2. **1/5 caracteres es vocal:** Uno de cada cinco caracteres debe ser vocal. En la mayoría de los idiomas al menos se necesita de una vocal para construir las palabras más pequeñas.

3. **1/10 caracteres es delimitador:** Uno de cada diez caracteres debe ser un delimitador (elementos que separan las palabras como espacios en blanco, comas, guiones, etc.)

Con estos criterios, es posible decir si es conveniente agregar un registro en el índice para cada término, pero no es todo lo necesario para comenzar a insertar tuplas en el índice. Debido a la especial estructura del índice Pub-Col, las operaciones básicas: insertar, borrar o actualizar no garantizan un cambio en el índice, por el contrario es posible que se realice cualquiera de estas operaciones y el índice se mantenga estable sin sufrir cambios. Es menester establecer un mecanismo que garantice el mantenimiento

y servicio actualizado del índice. En este trabajo se optó por dar solución al problema desde la base de datos con procedimientos almacenados que determinan cuando se debe borrar, agregar o modificar un registro del índice; en conjunción con disparadores que invoquen a los procedimientos almacenados cada vez que exista un cambio en los datos (véase el apéndice A.2 para consultar el procedimiento almacenado para agregar al índice).

La publicación culmina cuando se han procesado todos los valores de texto que cumplieron con los criterios para ser tomado en cuenta como palabras claves, de modo que el índice y el esquema de BD puedan utilizarse para interpretar las consultas del usuario.

En la base de datos del ejemplo, figura 3.3, existen 3864 tuplas. Después de publicar la BD en el sistema, el índice invertido registró 800 tuplas. Un muestra clara de las ventajas de usar un índice Pub-Col es notorio en la tabla 3.2, por ejemplo en la entrada con *Id* igual a *654* indica que en la tabla *offices* y el attributo *country* existe una o más veces el valor *"USA"*; es decir hay al menos una oficina ubicada en los Estados Unidos. Si se realiza una consulta para conocer cuántas oficinas existen en Estados Unidos, se descubrirá que existen 3 oficinas americanas (véase la tabla 3.1); por lo que el índice ha ahorrado dos espacios. Visualizar al índice bajo esta óptica permite que la propuesta sea escalable a bases de datos más grandes.

3.3. Interpretación y Desambiguación de Consultas

Una vez que se ha publicado la base de datos, el sistema conoce la información suficiente de la base de datos para procesar consultas pues se tiene acceso a la estructura y contenido de la base datos mediante el esquema y el índice invertido, respectivamente. El siguiente paso resulta en seleccionar los elementos adecuados, sean éstos del esquema (tablas o atributos) o del índice, para que al conectarse respondan a la consulta del usuario; donde cada palabra clave tiene una interpretación, y juntas satisfacen la intención del usuario al ingresar la consulta.

Desafortunadamente, debido a la falta de expresividad de las consultas por palabras clave, encontrar el significado correcto de cada término es una tarea complicada puesto que una sola palabra puede ser tratada distinto dependiendo del resto de palabras clave.

Considere el siguiente ejemplo, se ha ingresado la consulta **"city offices"** con la intención de obtener la lista de ciudades de las oficinas de la empresa. Al tratar de darle tal interpretación, el sistema descubre que hay cuatro posibilidades para *"city"*: los atributos *city* en las tablas *customer* y *offices*, y los valores del índice *"Makati City"* y *"Motor City Art Classics"*, donde el primero es el nombre de la ciudad de uno o más clientes y el segundo es el nombre del proveedor de uno o más productos (véase la figura 3.4).

officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
1	San Fran...	+1 650 21...	100 Market St...	Suite 300	CA	USA	940...	NA
2	Boston	+1 215...	1550 Court...	Suite 102	MA	USA	021...	NA
3	NYC	+1 212...	523 East 53rd St..	apt. 5A	NY	USA	100...	NA
4	Paris	+33 14 ...	43 Rue Jouffroy ...			France	75...	EMEA
5	Tokyo	+81 33...	4-1 Kioicho	Chiyoda-ku		Japan	102...	Japan
6	Sydney	+61 2 ...	5-11 Wentworth Av..	Floor #2		Aus...	NS...	APAC
7	London	+44 20...	25 Old Broad St...	Level 7		UK	EC...	EMEA

Tabla 3.1: Contenido de la tabla *offices*

Id	Palabra Clave	Tabla	Atributo
...	...	...	...
650	San Francisco	offices	city
651	100 Market Street	offices	addressLine1
652	Suite 300	offices	addressLine2
653	CA	offices	state
654	USA	offices	country
654	NA	offices	territory
656	Boston	offices	city
657	1550 Court Place	offices	addressLine1
658	Suite 102	offices	addressLine2
659	MA	offices	state
660	NYC	offices	city
661	523 East 53rd Street	offices	addressLine1
...	...	...	...

Tabla 3.2: Índice invertido en la base de datos de ejemplo (Figura 3.3)

Es evidente que en este sencillo ejemplo la opción correcta sería escoger como interpretación de *"city"* el atributo con el mismo nombre en la tabla *offices*, pues la palabra *"offices"* hace referencia a la tabla de oficinas. Pero en consultas más complicadas tal selección resulta considerablemente más complicada, como se abordará posteriormente.

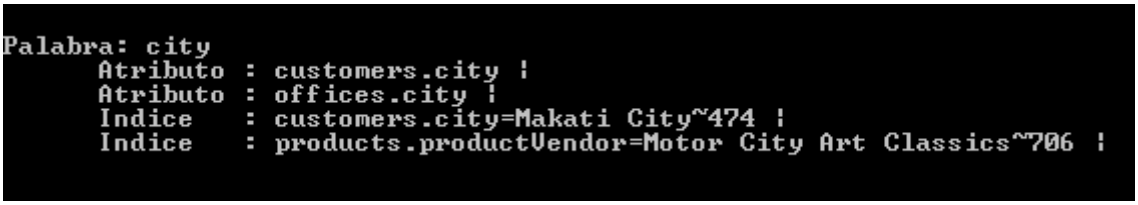


Figura 3.4: Interpretaciones para la palabra clave *"city"*

3.3.1. Interpretaciones Potenciales

Mencionada ya la importancia de la interpretación de la consulta en la búsqueda por palabras clave, es menester esclarecer las posibilidades de interpretación contempladas en este trabajo. La idea básica

es encontrar todas las posibles interpretaciones para cada palabra clave, obtener todas las combinaciones entre ellas y, a partir distintos criterios, escoger la mejor opción.

Cada una las interpretaciones de una palabra clave será un atributo, una tabla o un registro del índice. Obtenerlas se podría resumir a realizar dos operaciones para todos las términos en la consulta.

1. **Buscar en el esquema:** Escudriñar el esquema de la BD, para encontrar si la palabra clave puede referirse a tablas o atributos

2. **Buscar en el índice:** Buscar en el índice el término en cuestión con una consulta SQL como la siguiente:

```
SELECT * FROM indice WHERE MATCH(keyword) AGAINST("palabra
    clave");
```

Se puede apreciar el observar el proceso detallado en el algoritmo 1, el cual indica el trato que se le da una consulta; separando sus términos y asignándoles una Interpretación, sean éstas como tablas, atributos o valores del índice invertido; para obtener una instancia de Desambiguador que concentre todas las interpretaciones de cada palabra clave posibles.

Después de ejecutar el algoritmo, o de encontrar todas la interpretaciones de las palabras clave, se necesita obtener todas las combinaciones posibles, es decir el producto cartesiano entre todos los conjuntos de interpretaciones de cada palabra clave que tienen al menos una interpretación .

Retomemos la consulta que se planteó junto con la base de datos de la figura 3.3, ***"1969 camaro customers usa"***, la cual prentende conocer a los clientes estadounidenses que compraron un automóvil modelo Camaro del año 1969.

Para interpretar tal consulta el primer paso es buscar todas las posibilidades de términos. Como puede apreciarse en la figura 3.5, existen muchas opciones para cada palabra clave; *"1969"* tiene 5, *"camaro"* tiene 2, *"customers"* solo 1 y *"usa"* presenta 2. Para determinar el significado de cada término, éste debe ser analizado en relación a la interpretación del resto de palabras clave; por lo que es ineludible calcular el producto cartesiano y así poder evaluar cada posible combinación. En este ejemplo, el producto cartesiano produce 20 combinaciones de interpretaciones (tabla 3.3).

En el siguiente apartado se describe el mecanismo de selección de interpretación candidata de la consulta, abordando los criterios establecidos y continuando con el ejemplo de la consulta anterior.

Algoritmo 1: Interpretación de Consulta

Entrada: Consulta por palabras clave $Q = \{k_1, k_2, \ldots, k_i\}$
Salida: Instancia de Desambiguador que concentra todas las interpretaciones de palabras clave

```
1  desambiguador ← nuevo Desambiguador();
2  para cada palabra clave k ∈ Q hacer
       /* Buscar interpretaciones potenciales en metadatos                    */
3      para cada tabla t ∈ T hacer
4          si k=t.nombre entonces          /* Agregar interpretación como la tabla t */
5              i ← nueva Interpretación(t,k);
6              desambiguador.agregar Interpretación(i);
7          para cada columna c ∈ C(t) hacer
8              si k=c.nombre entonces      /* Agregar interpretación como la columna c */
9                  i ← nueva Interpretación(t,c,k);
10                 desambiguador.agregar Interpretación(i);
11         fin
12     fin
       /* Buscar interpretaciones pot. en el índice invertido con la consulta de texto:  */
       /* SELECT * FROM indice WHERE MATCH(keyword) AGAINST("k");                        */
13     ResulSet ← Consulta de k en Índice Invertido;
14     si |ResultSet| > 0 entonces
15         para cada entrada del índice e ∈ ResultSet hacer
               /* Agregar int. como la entrada del índice e                  */
16             t ← e["tabla"];
17             a ← e["atributo"];
18             id ← e["identificador"];
19             p ← e["palabra clave"];
20             i ← nueva Interpretación(t,a,p,id,k);
21             desambiguador.agregar Interpretación(i);
22         fin
23     fin
24 fin
25 devolver desambiguador;
```

3.3.2. Primera Evaluación

Hasta este punto, se ha calculado el conjunto de posibles interpretaciones de la consulta y se dispone a determinar cuál de ellas es la indicada. En el específico caso de la consulta "1969 camaro customers usa" existen 20 posibles interpretaciones donde solo una refleja la intención del usuario; la opción número 17 de la tabla 3.3. En ella las palabras clave *"customers"* y *"usa"* se relacionan a la tabla *customers* y al valor ***"usa"***, a su vez de la tabla *customers*. Con tal interpretación se asegura la primera parte de la consulta, recuperar la información de los clientes estadounidenses. Del mismo modo el sentido escogido de los términos *"1969"* y *"camaro"*, interpretadas como el mismo registro del índice, con el valor ***"1969 Chevrolet Camaro"*** identifican correctamente la porción restante correspondiente al automóvil *camaro*

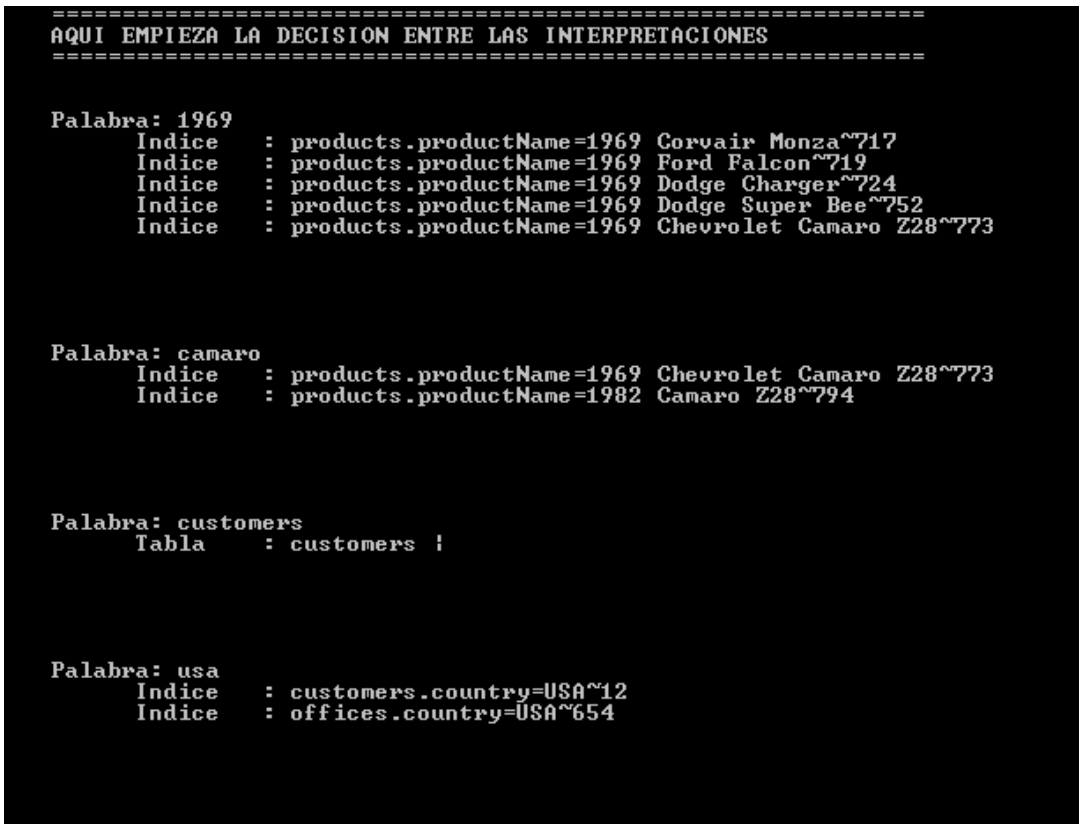


Figura 3.5: Interpretaciones de cada palabra clave en *"1969 camaro customers usa"*

	1969	camaro	customers	usa
1	"1969 Corvair Monza"	"1969 Chevrolet Camaro Z28"	customers	"USA" (customers)
2	"1969 Corvair Monza"	"1969 Chevrolet Camaro Z28"	customers	"USA" (offices)
3	"1969 Corvair Monza"	"1982 Camaro Z28"	customers	"USA" (customers)
4	"1969 Corvair Monza"	"1982 Camaro Z28"	customers	"USA" (offices)
5	"1969 Ford Falcon"	"1969 Chevrolet Camaro Z28"	customers	"USA" (customers)
6	"1969 Ford Falcon"	"1969 Chevrolet Camaro Z28"	customers	"USA" (offices)
7	"1969 Ford Falcon"	"1982 Camaro Z28"	customers	"USA" (customers)
8	"1969 Ford Falcon"	"1982 Camaro Z28"	customers	"USA" (offices)
9	"1969 Dodge Charger"	"1969 Chevrolet Camaro Z28"	customers	"USA" (customers)
10	"1969 Dodge Charger"	"1969 Chevrolet Camaro Z28"	customers	"USA" (offices)
11	"1969 Dodge Charger"	"1982 Camaro Z28"	customers	"USA" (customers)
12	"1969 Dodge Charger"	"1982 Camaro Z28"	customers	"USA" (offices)
13	"1969 Dodge Super Bee"	"1969 Chevrolet Camaro Z28"	customers	"USA" (customers)
14	"1969 Dodge Super Bee"	"1969 Chevrolet Camaro Z28"	customers	"USA" (offices)
15	"1969 Dodge Super Bee"	"1982 Camaro Z28"	customers	"USA" (customers)
16	"1969 Dodge Super Bee"	"1982 Camaro Z28"	customers	"USA" (offices)
17	"1969 Chevrolet Camaro Z28"	"1969 Chevrolet Camaro Z28"	customers	"USA" (customers)
18	"1969 Chevrolet Camaro Z28"	"1969 Chevrolet Camaro Z28"	customers	"USA" (offices)
19	"1969 Chevrolet Camaro Z28"	"1982 Camaro Z28"	customers	"USA" (customers)
20	"1969 Chevrolet Camaro Z28"	"1982 Camaro Z28"	customers	"USA" (offices)

Tabla 3.3: Producto cartesiano de interpretaciones para la consulta *"1969 camaro customers usa"*

modelo 1969.

Escoger una de las interpretaciones entre las 20 disponibles es claramente sencillo si se conoce la intención del usuario. Pero tal información no es fácil de determinar, puesto que el único valor de entrada que proporciona el usuario es un conjunto de palabras clave cuya intención es implícita para el usuario pero no para el sistema. Se necesita evaluar las posibilidades a partir de ciertos criterios que se consideran favorables, por ejemplo menor número de registros en el resultado, menor número de tablas involucradas en relación con la cantidad de palabras clave, la imposibilidad de interpretar ciertos términos de la consulta,etc. En otras propuestas como DBXplorer (Agrawal et al., 2002) o DISCOVER (Hristidis y Papakonstantinou, 2002) el único criterio de relevancia es el número de joins que se necesitan para conectar las palabras clave.

La evaluación que se plantea en este trabajo va un poco más allá realizando dos evaluaciones; la primera sobre todas las posibles interpretaciones de la consulta sin contemplar el árbol Steiner, o Red Candidata como se le conoce en los sistemas homónimos, y toma en cuenta dos situaciones: **tablas involucradas** y **registros del índice invertido**. A continuación se pretenta la fórmula de calificación de interpretaciones de la consulta, obtenidas a través del producto cartesiano de la interpretación de cada palabra clave de manera individual, y posteriormente se comenta la evaluación de los criterios considerados refiriéndose al ejemplo que se ha abordado hasta ahora.

A continuación se encuentra el algoritmo 2, el cuál resume el proceso de desambiguación de Interpretaciones Candidatas. Éste describe la secuencia de operaciones necesarias para pasar de un conjunto de interpretaciones por cada palabra clave a la selección de una sola interpretación entre el resto de opciones del producto cartesiano de las interpretaciones de cada término. El algoritmo omite la descripción detallada de las evaluaciones y de la generación de Redes Candiadatas por razones de espacio ya que

estos tres elementos se revisarán detenidamente en las siguientes páginas.

Algoritmo 2: Desambiguación y Selección de Consulta

Entrada: instancia de Desambiguador
Salida: Interpretación Candidata con mejor calificación

```
   /* Producto Cartesiano de todas las interpretaciones                    */
1  IC ← I_1 × ··· I_|Q| = {(i_1, ..., i_|Q| : i_j ∈ I)};
2  CT ← {};                                  // Conjunto de tablas involucradas distintas
3  para cada interpretación cadidata ic ∈ IC hacer
4  |  primera Evaluación();
   |  /* Agregar conjuntos de tablas inv. mayores a 1 no repetidos       */
5  |  si no. de tablas involucradas |TI(ic)| > 1  y  TI(ic) ∉ CT entonces
6  |  |  agregar TI(ic) a CT;
7  |  fin
8  fin
9  ordenar(IC);
10 RC ← {};
   /* Obtener la Red C. del top-3 de c. de tablas inv. en CT               */
11 para cada c. de tablas involucradas {ct_i ∈ CT | 0 ≤ i ≤ 3} hacer
12 |  rc ← Red Candidata Mínima(ct_i);
13 |  agregar {ct_i, rc} a RC;
14 fin
15 para cada interpretación cadidata ic ∈ IC hacer
   |  /* asignar Red Candidata correspondiente a cada ic                  */
16 |  para cada Red Candidata rc ∈ RC hacer
17 |  |  si rc[0]=TI(ic) entonces
18 |  |  |  ic.rc ← rc[1];
19 |  |  fin
20 |  fin
21 |  segunda Evaluación();
22 fin
23 reordenar(IC);
24 mejores ← {};                          // Escoger el conjunto con la mejor calificación
25 mejores ← ic_j|ic_j ∈ IC, máx_{1≤j≤|Q|} calificación(ic)};
26 ie ← {};
   /* si hay más de 1 escoger al azar                                      */
27 si |mejores| > 1 entonces
28 |  ie ← aleatorio(mejores) ;
29 en otro caso
30 |  ie ← mejores[0] ;
31 fin
32 devolver ie;
```

3.3.2.1. Fórmula de Calificación, primera evaluación

A continuación se encuentra la fórmula de calificación para la primera evaluación, la cual puntúa una Interpretación Candidata *IC* en función de los criterios presentados en la tabla 3.4. Al terminar la primera

evaluación de la Interpretación Candidata *IC* las calificaciones denotaran cómo procederá el sistema.

Criterio	Porcentaje de Influencia
Tablas involucradas	37.5 %
Concidencias en tuplas de la BD o del índice	37.5 %
Interpretaciones como valores del índice	25 %

Tabla 3.4: Porcentaje de Influencia de los Criterios en la Primera Evaluación

$$\text{calificación(IC)} = \textit{cal. tablas} \cdot (0{,}375) + \textit{cal. índice} \cdot (0{,}25) + \textit{cal. tuplas BD} \cdot (0{,}375)$$

3.3.2.2. Tablas involucradas

Las distintas palabras clave, interpretadas como tablas, atributos o valores del índice; deben ser conectadas para construir una consulta SQL. Tal unión se realiza a través de operaciones join entre las tablas involucradas. La cadena de operaciones join debe ser lo más pequeña posible, pues *"operaciones join entre muchas tablas son difíciles de comprender"*Hristidis y Papakonstantinou (2002) y mientras más reducida sea la secuencia de joins entre tablas requeridas la información estará más cerca, o dicho de otro modo tendrá una relación más directa. Lo cual es una característica positiva.

Tal como sugiere DBXplorer, *"El reto clave es evitar la generación de redes de unión de conjuntos de tuplas redundantes"* (Agrawal et al., 2002), y mientras menos tablas sean dadas como puntos Steiner es más probable que el tamaño del sub-árbol resultante sea menor.

Por otro lado, el anteponer interpretaciones de la consulta con un menor número de tablas también apoya a que la selección de las interpretaciones de las palabras clave coincidan entre sí en la misma tabla. Si se recordamos el sentido con que se planteó la consulta *"1969 camaro customers usa"* para la base de datos de la figura 3.3, las palabras clave *"1969"* y *"camaro"* se interpretaron como un registro del índice que hace referencia al valor *"1969 Chevrolet Camaro"* en el atributo *productName* de la tabla *products*; por lo que se puede decir que se encuentran en la misma tabla. Sucede algo semejante con el resto de términos en la consulta, pues *"customers"* se interpretó como la tabla con el mismo nombre y *"usa"* como una entrada del índice que señala a la tabla *"customers"* como su ubicación, resultando en otra coincidencia entre tablas.

En este ejemplo sólo existen dos posibles conjuntos de tablas involucradas: *products* y *customers*, y *products*, *customers* y *offices*. Por lo tanto el sistema debe escoger la primera opción con el menor número de tablas, dos. Correspondiendo así el sentido con que se ideó la consulta, ya que la otra alternativa contiene información de las oficinas, y esto no figura en la intención original.

Concretamente se califica el número de tablas en la interpretación de la consulta en relación a la cantidad de palabras clave ingresadas. Éste criterio tiene un peso del 37.5 % de la primera evaluación y se calcula con la siguiente fórmula.

$$\textit{calificación tablas(IC)} = \begin{cases} \frac{|Q|\text{-}|TI(ic)|}{|Q|} & \text{si número de tablas inv.} |TI(ic)| > 1 \\ 0 & \text{si número de tablas } |TI(ic)| = 0 \end{cases}$$

Donde $|TI(ic)|$ es el número de tablas en la interpretación candidata ic, y $|Q|$ la cantidad de palabras clave en la consulta original. Al realizar tal evaluación las calificaciones más cercanas a cero serán juzgadas como mejores.

Refiriéndose al ejemplo, la consulta "1969 camaro customers usa" tiene **4 palabras clave** de las que se identificaron las tablas *products y customers*, es decir **2 tablas**. Por lo tanto la calificación dada al segmento de la interpretación referente a las tablas sería de 0,1875, una calificación muy positiva que ha premiado las coincidencias antes mencionadas.

$$\frac{|Q| - tablas}{|Q|} \cdot (0{,}375) = \frac{4-2}{4} \cdot (0{,}375) = 0{,}1875$$

3.3.2.3. Registros del índice

La inclusión de los registros dentro de la fórmula de calificación no es tan sencilla como la del número de tablas involucradas debido a la naturaleza de su origen, el índice invertido. Al haber "resumido" por columas el índice, se pierde la posibilidad de contar las ocurrencias de una tupla de la BD en una interpretación, lo que podría verse como una métrica análoga a la función *tf-idf* en la rama de Information Retrieval (Manning et al., 2008).

En vez de que el índice nos señale la tupla en la que se encuentra el término buscado, éste *nos da a conocer que en cierta tabla y atributo existe al menos una tupla con la palabra clave que se busca*. Por ejemplo, al procesar la palabra *"usa"* de la consulta ilustrativa, se encuentran dos registros del índice que contienen tal término(véase la figura 3.5); ellos indican que existen oficinas en estados unidos y que hay clientes norteamericanos.

La manera en que el sistema responde con la tupla que se desea a la consulta es a través del filtrado por predicados; intersecciones de las tablas conectadas por operaciones join que juntas desechan las tuplas que no cumplieron tales condiciones. Imaginando que un usuario desea conocer las oficinas ubicadas en Nueva York e ingresa la consulta "offices nyc" el sistema retornará la tupla deseada. Como puede verse en la tabla 3.6, la consulta SQL no contiene ningún predicado referente al identificador de tupla por los

motivos antes comentados en cambio obtuvo la respuesta filtrando por valor.

"offices nyc"

```
SELECT t3.officeCode ,t3.city, t3.phone, t3.addressLine1, t3.
    addressLine2, t3.state, t3.country, t3.postalCode, t3.territory
    FROM offices t3 WHERE t3.city="NYC";
```

officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
3	NYC	+1 212 5...	523 East 53rd St...	apt. 5A	NY	USA	10022	NA

Tabla 3.6: Consulta SQL y ResultSet para la consulta *"offices nyc"*

Habiendo puesto en claro el uso del índice, es necesario mencionar que la identificación de interpretaciones a una misma tupla de la base de datos (no del ínidice invertido) debe realizarse de una manera diferente, pues el índice no presenta información acerca de las tuplas y sí de los valores de atributo. La manera directa de identificar que se "habla" de la misma tupla de la base de datos resulta en contar las coincidencias de entradas del índice en la interpretación de la consula. Por ejemplo, en la consulta "1969 camaro customers usa" la interpretación número 17, que posteriormente se alzará como la mejor, presenta la misma entrada del índice *{773,"1969 Chevrolet Camaro Z28",products,productName}* relacionada con dos palabras clave distintas; *"1969" y "camaro"*, por lo que se consideran referencias a la misma tupla de la base de datos.

El anterior es el caso sencillo, puesto que no require un análisis meticuloso, ¿Qué sucede cuando dos palabras clave distintas hacen referencia a la misma tupla en la base de datos, pero no al mismo registro del índice invertido? El sistema debería de ser capaz de reconorcerlo e incluirlo en la función de calificación.

Considérese el trato dado a la consulta *"Gerard Bondur"*, que busca a una persona con ese nombre. Como puede verse en la figura 3.6, el sistema relaciona la palabra *"Bondur"* con la única entrada del registro posible. Por el otro lado, al tratar de interpretar *"Gerard"* se encuentra con dos registros del índice que referencian a la misma tabla (*employees*) pero diferente atributo; por lo que sólo con el criterio del menor número de tablas sería imposible elegir entre uno de ellos.

Para solucionar tal predicamento, este trabajo propone aprovechar el conocimiento del esquema de base de datos y el orden en que se publicó la misma para que con esa información se deduzca cuándo dos entradas del índice distintas hacen referencia a la misma tupla de la base de datos. Debido a que el llenado del índice se realiza tabla por tabla; navegando tupla por tupla, y valor de atributo textual tras valor de atributo textual (en anchura); se puede ver una secuencia en el índice invertido que refleja la estructura de cada tabla.

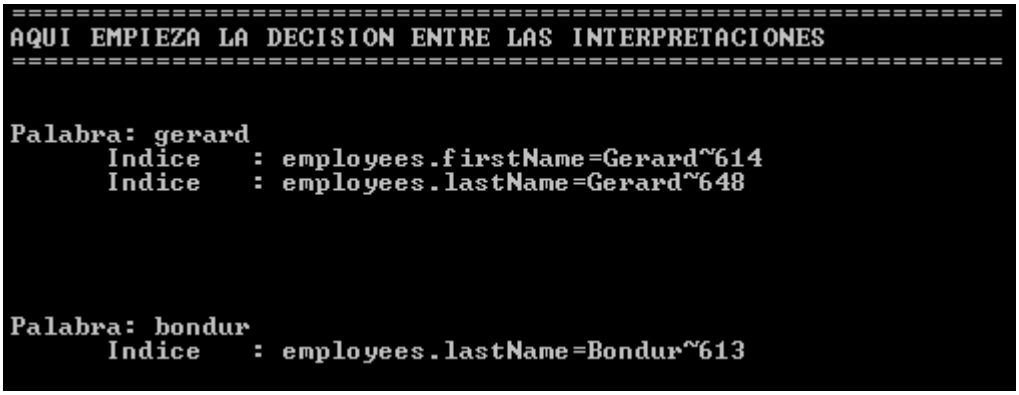


Figura 3.6: Interpretaciones de la consulta *"gerard bondur"*

Para ilustrar lo anterior se puede observar la tabla 3.2, cuya porción del índice invertido preserva la secuencia de atributos *city, addressLine1, addressLine2, state, country, territory* (tales atributos tienen tipos de dato textuales y por esto son los únicos presentes en el índice, véase la figura 3.3) de la tabla *offices*.

Con tal información es posible denotar el comienzo y final las tuplas en la base de datos, en la porción del índice mostrada en la tabla 3.2 se consideran como una misma tupla las entradas con un identificador en los intervalos *[650,654]* y *[655,659]*. Cabe destacar que la segunda tupla de offices encontrada en el índice no incluyó ni el atributo *country* ni *territory*, puesto que poseer el mismo valor que en la tupla anterior el sistema no las incluye. Aún así como se ha respetado la secuencia hasta antes de esos atributos es posible identificar un registro de la tabla *offices* desde la entrada 655 a la 659 del índice.

Conociendo la secuencia esperada de atributos que identifican una tupla de la base de datos, lo único necesario es comparar la sucesión en el índice, y si ésta es coherente desde dos entradas del índice que están relacionadas con dos palabras clave distintas de la consulta; se cuentan como la misma tupla. Retomando la consulta "gerard bondur", la cual evidencia tal situación, se observa en la figura 3.6 que existen dos interpretaciones para *"gerard"* y sólo una para *"bondur"*, por lo tanto al calcular el producto cartesiano se obtienen solo dos opciones (véase la tabla 3.7).

La primera opción presenta los registros del índice con identificadores 614 y 613, los cuales se encuentran uno seguido del otro y hacen referencia a la tabla *employees*. Con esos datos uno puede remitirse al esquema de base de datos para encontrar los atributos de texto, únicos candidatos a tener registros en el índice, y encontrar la secuencia en la que se agregaron los valores al índice. Tal sucesión de atributos es la siguiente : *lastName, firstName, extension, email, officeCode, jobTitle*.

Gracias a que el fragmento *lastName, firstName* coincide con los atributos de la primera opción y, más importante, a la diferencia de los identificadores del índice (1, que indica que un valor se encuentra

después del otro) se puede decir que ambos describen a la misma tupla de la BD. Se da una mejor calificación a la primera opción ya que no se puede deducir coincidencia de tuplas en la segunda probabilidad (segunda entrada de la tabla 3.7), visible por dos motivos; están separados por 35 registros del índice (sólo hay 8 atributos en la tabla *employees* y de ellos son considerados 6, con tipos de dato textuales) y al ser entradas del índice que indican una misma tabla y atributo, es imposible que una tupla tenga dos valores distintos en el mismo atributo.

	gerard	bondur
1	employees.firstName="Gerard" 614	employees.lastName="Bondur" 613
2	employees.lastName="Gerard" 648	employees.lastName="Bondur" 614

Tabla 3.7: Producto cartesiano de interpretaciones para la consulta *"gerard bondur"*

Como última consideración conciernente al número de registros del índice y las tuplas a las que éstos hacen referencia, se debe mencionar que es necesario calificar la semejanza entre las palabras clave y los valores del índice invertido. Debido a que el llenado del índice invertido, por razones de desempeño, se realiza consumiendo el procedimiento almacenado *agregar* (véase en el apéndice A.2 la declaración de creación del procedimiento *agregar*), el cual ingresa una entrada al índice por cada valor textual distinto en cada atributo de la base de datos, los valores del atributo *Palabra Clave* no son "palabras" en un sentido estricto, si no un cojunto de caracteres que presentó las características contempladas en el apartado 3.2.3.1; y tal conjunto de letras puede ser una sola palabra o un conjunto de éstas. Ante tal situación el sistema calcula la posibildad de referirse a una entrada del índice, con una sencilla premisa:

Si el valor del índice **y** es igual al término en turno **x** la semejanza entre **x** y **y** es de **0.001**, en caso contrario es igual a **1** menos el cociente de 1 entre el número de palabras que contiene la entrada del índice.

$$semejanza(x,y) = \begin{cases} 0{,}001 & \text{si } x = y \\ 1 - \frac{1}{|y|)} & \text{si } x \neq y \end{cases}$$

Como puede observarse se otorgan valores más pequeños mientras más "semejantes" son la palabra clave y su interpretación como valor del índice, esto se debe a que la calificación de la interpretación está planteada en función de la especificidad, mientras menor sea el conjunto de tuplas de la base de datos distintas que se encuentren en una interpretación se asume que la respuesta es más específica. En el primer caso donde un término x es igual a un valor del índice y, el caso preferenciado, se asigna *0.001* en vez de *0* únicamente para evitar que la fórmula de calificación

Con la función de semejanza es posible que cada interpretación de las palabra clave aporte el puntaje correspondiente a la porción del valor del índice justo; es decir la fracción de texto presente en el valor del índice. De este modo la suma de las semejanzas entre el término y el valor del índice interpretado apoyarán a decidir qué tan conveniente es decantarse por una interpretación u otra. Es común que el usuario haga referencia al mismo valor del índice con las mismas palabras clave y si éste es el caso el cálculo de semejanza apoya su selección. Por el contrario, si existen múltiples posibilidades de interpretación para un término, y una de ellas hace alusión a un registro del índice con una semenjanza baja, el resto de palabras clave deberán hacer referencia a tal entrada del índice para que tal interpretación sea escogida.

Si se dirige al ejemplo especificado en la figura **??** (con la consulta *"1969 camaro customers usa"*), se observan las dos posibilidades de evaluación de semejanza; para la palabra "usa" se identifica el registro del índice *{12,"USA",customers,country}* y debido a que tanto la palabra clave como el valor del índice son iguales la semenza es igual 0,001. Por el otro lado, la entrada del índice *{773,"1969 Chevrolet Camaro Z28",products,productName}* posee 4 palabras, de modo que la semejanza de los términos "1969" y "camaro", con el valor del índice , será de 0,75.

$$1 - \frac{1}{4} = \frac{3}{4} = 0{,}75$$

.

Materializando las reflexiones hechas acerca del índice y de la coincidencia de tuplas de la BD e interpretaciones de distintas palabras clave, hay dos sub-calificaciones para esos dos valores. La primera, con el *25 %* de influencia en la primera evaluación, se refiere al número de registros del índice invertido en una posible interpretación de la consulta, tomando en cuenta la semejanza de los términos con los valores del índice y normalizado en función del número de palabras clave. La segunda fórmula, con el *37.5 %* restante en la calificación global de la primera evaluación, corresponde al número de tuplas de la base de datos a los que refiere la consulta, como se explicó anteriormente.

$$\textit{calificación registros índice(IC)} = \begin{cases} \frac{|Q|\text{-}i}{|Q|} & \text{si número de reg. del índice } i > 1 \\ 0 & \text{si número de reg. del índice } i = 0 \end{cases}$$

$$\textit{calificación tuplas de la BD} = \begin{cases} \frac{|Q|\text{-}tup}{|Q|} & \text{si número de tuplas de la BD } tup > 1 \\ 0 & \text{si número de tuplas de la BD } tup = 0 \end{cases}$$

Donde i es el la suma de semejanzas entre todos los registros del índice obtenidos (en una combi-

nación de interpretaciones y sus palabras claves correspondientes), incluyendo repeticiones. Y *tup* es la suma las semejanzas de valores del índice distintos:

- El valor de semejanza entre un valor del índice y una palabra clave se añadirá una vez, si se encuentra interpretación a otras palabras clave que se hayan entendido como el mismo valor del índice éstas ya no se incluirán.
- Si dos interpretaciones a dos palabras clave distintas como valores del índice distinto aluden a la misma tupla de la base de datos solo se incluye el valor de semejanza de uno de ellos.

se calcula la semenjanza de un valor del índice si solo una palabra clave e de tuplas de la base de datos deducidos a partir de la secuencia del índice y $|Q|$ es el número de palabras clave ingresadas como entrada al sistema.

En la consulta con la que se ha analizado la secuencia de operación del sistema, *"1969 camaro customers usa"*, cuya mejor interpretación (fila 17 de la tabla 3.3) presenta 3 registros del índice; en 2 ocasiones la entrada *"1969 Chevrolet Camaro"* con un valor de 0,75, y *"USA"* con una semejanza de 0,001. La calificación de registros del índice es igual a 0,156 gracias a que intervienen .

$$\text{i = 0.75+0.75+0.001}$$

$$\text{cal. índice } = \frac{|Q| - i}{|Q|} \cdot 0{,}25 = \frac{4 - 1{,}51}{4} \cdot 0{,}25 = 0{,}156$$

Mientras que la puntuación correspondiente a la cantidad de tuplas de la base de datos es igual a 0,304 se debe a la entrada que aparece dos veces, con un valor de 0,75 sólo se incluye una vez.

$$\text{tup = 0.75+0.001}$$

$$\text{cal. tuplas BD} = \frac{|Q| - tup}{|Q|} \cdot 0{,}375 = \frac{4 - 0{,}751}{4} \cdot 0{,}375 = 0{,}304$$

3.3.2.4. Primera calificación total

La consulta introducida en la figura **??**, *"1969 camaro customers usa"*, pudo ser calificada efectivamente. El proceso se describió para la mejor de todas las posibles interpretaciones de la tabla 3.3, la opción número 17 (visible en la tabla C.4 del apéndice C donde se encuentra un concentrado de calificaciones de cada opción de la anterior consulta).

Tal calificación es la suma de las evaluaciones de los criterios establecidos; en función del número de tablas, de registros del índice y tuplas de la BD involucrados.

$$\text{calificación}(IC_{17}) = \textit{cal. tablas} + \textit{cal. índice} + \textit{cal. tuplas BD}$$

$$\text{calificación}(IC_{17}) = 0{,}1875 + 0{,}1556 + 0{,}3046 = 0{,}6477$$

3.3.3. Generación de Redes Candidatas

En secciones anteriores se ha analizado con detenimiento la propuesta del trabajo respecto a la interpretación y la evaluación parcial de consultas por palabras clave. Este primer filtro no es suficiente, puesto que aún no se tiene conocimiento de cómo conectar los elementos deseados como respuesta; sólo se conocen las mejores interpretaciones de los términos introducidos por el usuario.

Es imposible contruir la consulta SQL, salida esperada, sin saber cómo vincular cada porción de la respuesta que ha identificado cada palabra clave. En una interpretación candidata, a cada palabra clave de la consulta se le otorga una interpretación como tabla, atributo o valor del índice; donde un atributo sólo puede pertencer a una tabla, y los valores del índice invertido apuntan a su ubicación en una tabla y atributo específicos. De modo sólo es necesario contemplar las tablas denotadas en cada interpretación para encontrar el camino que asocie dichas interpretaciones.

Para que tal camino sea trazado se necesita calcular la cadena de operaciones join entre las tablas de cada interpretación de palabra clave. Al igual que el resto de sistemas presentados en el apartado de Trabajos Existentes alineados al paradigma de Redes Candidatas (sección 2.4); se aborda el problema de generación de redes candidatas como la obtención del árbol Steiner mínimo del grafo de esquema, que tiene como puntos Steiner el conjunto de tablas involucradas en una interpretación candidata. En otras palabras la tarea es encontrar el sub-grafo del esquema de BD que conecte los extremos encontrados durante el proceso de interpretación, donde dichos extremos están ubicados en una tabla. De tal modo que el sub-grafo resultante sea usado como mapa para construir la consulta SQL.

3.3.3.1. Reuso de las redes candidatas generadas

Como se mencionó antes el cómputo de un árbol steiner es un problema complicado, pues pertenece a la clase de complejidad NP-completo, lo que implica una inversión considerable de tiempo para calcular una red candidata. Para dar respuesta a la consulta se deberían contemplar todas las interpretaciones candidatas, o al menos las que se suponen mejores.

He aquí donde la primera evaluación cobra sentido; todas las posibilidades se han ordenado respondiendo a tres calificaciones individuales: número de tablas involucradas, número de interpretaciones como valores del índice y número de coincidencias en valores del índice o tuplas de la BD. Los anteriores

son criterios que se consideran positivos ya que se prefieren resultados cortos y con un menor número de tuplas, que aseguren especificidad e una interpretación global coherente.

Si en vez de ejecutar el algoritmo del árbol Steiner mínimo el mismo número de veces que interpretaciones candidatas, se realiza únicamente con el top-3 de conjuntos de tablas distintos en el conjunto de interpretaciones candidatas (con más de una tabla). Es posible incluir un nuevo criterio en la puntuación de interpretaciones candidatas: *la longitud de la red candidata*.

Para la consulta *"1969 camaro customers usa"* se encontraron 20 diferentes interpretaciones tomando en cuenta las distintas interpretaciones aisladas de los términos de entrada y procesando su producto cartesianos (ver la tabla 3.3. Si se calculara la red candidata de cada una de las posibilidades se haría un desperdicio en cómputo y tiempo pues a pesar de que existen 20 posibilidades de interpretación sólo existen 2 conjuntos diferentes de tablas involucradas: *{customers, products}* y *{customers, offices, products}*. De este modo se reduce el número de 20 redes candidatas a 2.

En consultas con un mayor número de conjuntos distintos de tablas involucradas se calculará únicamente el top-3 de redes candidatas, usando los mejores 3 conjuntos de tablas involucradas como puntos Steiner (entrada del algoritmo).

En seguida se encuentra la descripción del algoritmo utilizado para obtener el árbol Steiner mínimo, o Red Candidata, y en el siguiente apartado con la continuación del ejemplo.

3.3.3.2. Algoritmo de Steiner Tree mínimo

En el sistema propuesto se implementó el algoritmo de L. Kou, G. Markowsky y L. Berman (1981), el cual propone una secuencia sencilla para obtener el árbol Steiner mínimo de un grafo G para un conjunto de nodos S. Tal algoritmo ha sido simplemente incrustado en esta étapa del proceso de consulta. A continuación se presenta la definición del problema del algoritmo antes mencionado, sustituyendo los

nombres de las variables para empatar con la problemática; consulta por palabras clave a BD:

Algoritmo 3: Obtener Red Candidata mínima

Entrada: Grafo del Esquema $E = (T, J, d)$, Conjunto de tablas $TI \subseteq T$
Salida: Red Candidata (o Árbol Steiner mínimo) de TI en E

```
1  para cada combinación de nodos {t_i, t_j} ∈ TI hacer
2  |  Calcular el camino más corto de t_i a t_j en E;
3  fin
4  Crear un grafo R;
5  para cada nodo t ∈ TI hacer
6  |  Agregar t a R;
7  |  para cada nodo n|(n ≠ t), n ∈ TI hacer
8  |  |  Agregar n a R;
9  |  |  Unir t y n con un arco;
10 |  |  Asingar la distancia del camino más corto entre t y n como costo de d(t, n);
11 |  fin
12 fin
13 Buscar el Árbol de Spanning Mínimo S de R;
   /* Si existe más de uno, tomar uno arbitrariamente                    */
14 para cada arco a = {t_i, t_j} ∈ S hacer
15 |  Sustituir a por el camino más corto entre {t_i, t_j};
16 fin
17 para cada nodo t ∈ S hacer
18 |  si t ∉ TI y t no es puente entre dos nodos a ∈ TI y b ∈ TI entonces
19 |  |  remover t del grafo S ;
20 |  fin
21 fin
22 devolver Árbol S;
```

En el algoritmo 3 se describe el proceso necesario para obtener la red candidata mínima a partir de un cojunto de tablas TI. Tal proceso puede resumirse en cuatro etapas básicas:

1. **Caminos más cortos:** Consumiendo el algoritmo de Dijkstra, se obtienen los caminos más cortos de todas combinaciones de tablas de TI sobre el esquema de base de datos.

2. **Resumir en un grafo:** Se crea un grafo R con un nodo por cada tabla en TI, éstos se unen por arcos con costos iguales al camino más corto entre dichos nodos. Se puede decir que R "resume" los caminos caminos más cortos calculados en el paso anterior.

3. **Árbol de Spanning Mínimo:** De R se obtiene el árbol de Spanning mínimo, el cual busca el árbol que contenga todos los nodos en un grafo y que la suma de sus costos de distancia sea la menor o igual a la(s) menor(es).

4. **Sustitución por caminos más cortos:** Por último se descartan los nodos no contribuyentes, es decir aquellos cuyas tablas no forman parte del conjunto TI ni son usados como puente para

conectar tablas del conjunto TI.

3.3.3.3. Ejemplo

Retomando la consulta *"1969 camaro customers usa"*, utilizada como ejemplo durante todo el capítulo, y el progreso hasta ahora descrito.

Se han conseguido interpretar las palabras clave por separado, posteriormente a través del producto cartesiando de las interpretaciones de cada término de manera individual se obtuvieron 20 posibilidades y por último se asignó calificicación a cada una de ellas; priorizando criterios como el número de tablas, de interpretaciones como valores del índice invertido y la cantidad de concidencias (véase la tabla C.4 para ver las opciones y sus calificaciones desglozadas).

Como se comento antes, para obtener las redes candidatas únicamente necesitan el conjunto de tablas involucradas en la interpretación. En este caso existen 20 interpretaciones pero sólo 2 conjuntos diferentes de tablas involucradas: *{customers, products}* y *{customers, offices, products}*. Por lo que sólo se debe ejecutar dos veces el algoritmo.

Por razones de espacio se mostrará la secuencia para obtener el árbol Steiner mínimo del conjunto *{customers,products}*, pues es el que pertenece a la opción número 17; la opción con la mejor puntuación después de la primera evaluación. A partir de ahora se hará referencia a las tablas del esquema usando las acotaciones indicadas en la tabla 3.8.

acotación	tabla
pl	productLines
pr	products
od	orderDetails
or	orders
c	customers
pa	payments
e	employees
of	offices

Tabla 3.8: Acotaciones para el grafo del Esquema

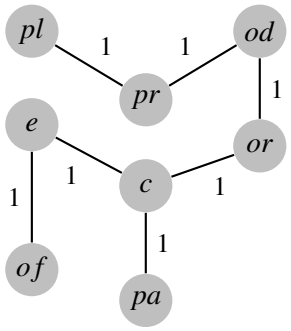


Figura 3.7: Grafo del Esquema de Base de datos

1. **Caminos más Cortos**

   El primer paso para conseguir la red candidata es obtener los caminos más cortos entre todas las combinaciones de la tablas en TI. Como se menciono anteriormente, se aplicará el algoritmo para el conjunto $TI = \{customers, products\}$ sobre el esquema E visible en la figura 3.7.

   El camino más corto de un nodo a otro en un grafo se resuelve mediante el algoritmo homónimo de

Dijkstra (para más información consulte **?**); el cual, como su nombre lo indica, retorna la sucesión de nodos conectados por arcos cuya suma de pesos resulta menor.

En el ejemplo sólo se debe calcular el caminos más corto de *c-pr*, pero si se calculara la segunda opción (recordando que existen dos grupos de tablas involucradas distintas para todas las Interpretaciones Candidatas) sería necesario calcular tres: {*c-of, c-pr, of-pr*} (véase el apéndice C).

Acotación	Ruta	Distancia
c-pr	c-or-od-pr	3

Tabla 3.9: Acotaciones grafo caminos más cortos

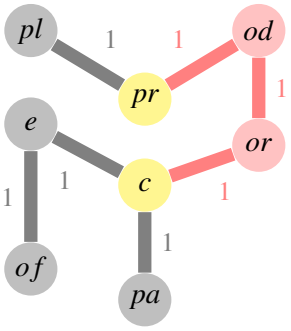


Figura 3.8: Caminos más corto (ejemplo $TI = \{customers, products\}$)

2. **Resumir en un Grafo**

Lo siguiente en la consecución del árbol Steiner mínimo o Red Candidata mínima, consiste en crear un nuevo grafo *R* a partir de los caminos más cortos calculados en el paso anterior. Se añade un nodo por tabla en *TI* y un arco por cada camino más corto calculado que tiene el costo de dicha ruta.

En el ejemplo $TI = \{customers, products\}$, únicamente se obtuvo el camino más corto entre las tablas*customers* y *products*, la cual tiene una distancia de 3. La figura 3.9 muestra el grafo resumido que servirá de entrada al algoritmo del árbol de Spanning Mínimo usado en el siguiente paso.

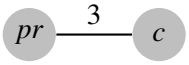


Figura 3.9: Grafo Resumido y MST (ejemplo $TI = \{customers, products\}$)

3. **Árbol de Spanning Mínimo**

Una vez que se han resumido las tablas en *TI* y los caminos más cortos es posible escojer el árbol $S \subseteq R$ que contenga todos los nodos y tenga el menor peso. Como es evidente en el ejemplo esta situación no sucede ya que sólo existe una opción, por lo que no tiene ningún sentido calcular el árbol de Spanning mínimo. El grafo resumido de la figura 3.9 **ya es un árbol de Spanning mínimo** de sí mismo.

Diríjase al Apéndice B.2. para observar la obtención del árbol de Spanning Mínimo para $TI = \{customers, products, offices\}$ en la cual se ejemplifica mejor este paso.

4. **Sustitución por caminos más cortos**

Finalmente el algoritmo debe sustituir cada arco entre los nodos del árbol de Spanning Mínimo S por el camino más corto correspondiente. En el ejemplo basta con sustituir el arco que conecta *pr* y *c* con una distancia de tres, por el camino más corto entre dichas tablas: *c-or-od-pr*. De modo que la red candida en el ejemplo es igual a la que se presenta en el grafo de la figura 3.10.

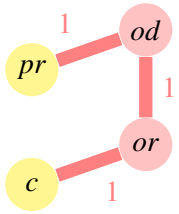


Figura 3.10: Red Candidata (ejemplo $TI = \{customers, products\}$)

3.3.4. Segunda Evaluación y Selección de Interpretación

Llegados a este punto se asume que el sistema posee los elementos suficientes para escoger una Interpretación Candidata, incluyendo el árbol Steiner dentro de tal selección. Es muy importante considerar este último elemento, tanto así que en muchos Sistemas de Búsqueda por palabras clave es el único criterio de jerarquización; se ordenan las top-k respuestas en función de la longitud de la Red Candidata o Árbol Steiner, según sea la clase de sistema.

Así mismo, este trabajo sumará a la primera evaluación el criterio de longitud de Red Candidata, de tal suerte que la calificación deberá ser reajustada (véase la tabla 3.10). Debido a que no es el factor más importante se le otorga un porcentaje de influencia del 16 %; otros criterios como las coincidencias de tuplas o el número de tablas son preferidos ya que ellos analizan coincidencias en las interpretaciones de las palabras clave. Si más de una palabra clave se refiere a lo mismo debe no es casualidad (y si lo es un buen sistema de búsqueda debería tener las herramientas para detectarlo).

A continuación se encuentra la función de calificación la cual introduce la función de longitud de la red candidata:

$$calificación = tablas \cdot (0{,}24) + índice \cdot (0{,}16) + tuplas\ BD \cdot (0{,}24) + Red\ Candidata \cdot (0{,}16)$$

$$\textit{calificación Red Candidata} = \frac{|T(BD)| - long(rc)}{|T(BD)|}$$

$$long(rc) = \begin{cases} |rc| & \text{si hay más de una tabla en la Red Candidata } |rc| > 1 \\ 1 & \text{si número de tablas en TI(ic) fue igual a 1 } |TI(ic)| = 1 \end{cases}$$

Criterio	Porcentaje de Influencia
Tablas involucradas	24 %
Concidencias en tuplas de la BD o del índice	24 %
Interpretaciones como valores del índice	16 %
Longitud de Red Candidata	16 %

Tabla 3.10: Porcentaje de Influencia de los Criterios en la Segunda Evaluación

Donde $|T(BD)|$ es el número de tablas en la base de datos, $long(rc)$ es una función que retorna la longitud de una Red Candidata *rc*: si el conjunto de tablas involucradas TI de la Interpretación candidata *ic* en turno contiene más de un elemento la longitud de la red Canidata será igual al número de nodos del Árbol Steiner mínimo; en caso contrario, gracias a la ausencia de red candidata, la longitud será igual a 1. Lo anterior se debe a que si una Interpretación candidata solo incluye una tabla no es necesario calcular la Red Candidata mínima, pues no hay necesidad de encontrar una cadena de operaciones join.

En el ejemplo, específicamente la Interpretación Candidata número 17 (cuadro 3.3) de la consulta *"1969 camaro customers usa"*, se calculó la Red Candidata expresada en la figura 3.10, la cual contiene 4 nodos: *{products, orderDetails, orders, customers}*. De modo que la longitud del árbol Steiner o Red candidadta es de 4 y por consiguiente la calificación otorgada al rubro correspondiente es de $\frac{T(BD)-long(RC)}{T(BD)} = \frac{8-4}{8} = 0{,}5$. Ya que existen 8 tablas en la base de datos de ejemplo (observe la figura 3.3 en la página 34), variable que normaliza la longitud de la red candidata a un rango entre [0,1].

Con este nuevo valor se recalcula la calificación general; que, con una puntuación de 0,49406, aún después de la segunda evaluación sigue siendo la más alta entre todas las interpretaciones candidatas (véase el apéndice C) y por consiguiente **es escogida como la Interpretación Deseada**.

$$\textit{calificación} = (\textit{tablas} \cdot 0{,}24) + (\textit{índice} \cdot 0{,}16) + (\textit{tuplas BD} \cdot 0{,}24) + (\textit{Red Candidata} \cdot 0{,}16)$$

$$\textit{calificación} = (0{,}5 \cdot 0{,}24) + (0{,}6225 \cdot 0{,}16) + (0{,}8122 \cdot 0{,}24) + (0{,}05 \cdot 0{,}16)$$

$$\textit{calificación} = 0{,}1201 + 0{,}0996 + 0{,}1944 + 0{,}08 = 0{,}49406$$

3.4. Procesamiento de Consultas

Anteriormente se han descrito los componentes de *KweryDB*, un sistema de consulta por palabras clave en bases de datos. Éstos comprenden la publicación de la BD, interpretación de consultas; con una primera evaluación, generación de redes candidatas mínimas, una segunda evaluación y finalmente la selección de interpretación(observe el algoritmo 4).

Algoritmo 4: Procesador de Consultas

Entrada: Consulta por palabras clave $Q = \{k_1, k_2, \ldots, k_i\}$
Salida: Consulta SQL
Datos: Conexión con una Base De Datos *BD*

```
1 desambiguador=Interpretar Consulta(Q);
2 ie = desambiguador.seleccionar Interpretación();
3 SQL= Genera SQL(ie);
4 devolver SQL;
```

La teoría se ha demostrado al ilustrar el trato dado a la consulta *"1969 camaro customers usa"* en la base de datos presentada en la figura **??**, conduciendo al lector en la secuencia de operación del sistema.

3.4.1. Generación de SQL

Con lo expresado hasta el momento, el proceso de consulta no ha finalizado; aún resta un último paso antes de poder responder la consulta con una sentencia SQL; pues hasta el momento la respuesta sería una Interpretación Candidata (incluyendo el árbol Steiner o Red Candidata). Esta estructura de datos no puede ser pasada a un DBMS para obtener una salida tabular, lo cual es parte del requerimiento, pues usuarios sin conocimiento de la BD ni de SQL deberían poder cuestionar al sistema y visualizar una respuesta comprensible para ellos.

Considerando lo anterior, el sistema se dispone a construir una consulta SQL consumiendo dos elementos: una Interpretación Candidadata, la que tuvo mejor calificación, y su correspondiente Red Candidata Mínima. Con ambas estructuras es posible rellenar una plantilla de consultas SQL con el siguiente formato:

```
SELECT <atributos> FROM <tablas> WHERE <predicados de FK-PK> AND <
   predicados de valores>;
```

Con este molde de consultas únicamente es necesario tomar las partes indicadas de las estructuras de datos antes mencionadas e insertarlas en la plantilla. A continuación se resume tal proceso (véase el algoritmo 5 en la página 88 para mayor detalle):

- **Tablas:**

  En una consulta SQL es crucial denotar las tablas de donde obtendrá la información, pues encon-

trar los demás elementos de la consulta SQL depende de las tablas de origen correctas, y de ser necesario, enlazarlas con operaciones de unión (el sistema utiliza la notación con comas ‘,’de SQL) de modo que las tuplas de diferentes tablas se encuentren conectadas.

Las tablas se encuentran en la *Red Candidata Mínima* de la interpretación seleccionada y todas ellas sustituirán al segmento $< tablas >$ de la plantilla. Si no se calculó una Red Candidata Mínima para la Interpretación escogida *ie* significa que la interpretación sólo incluye una tabla, y ésta se obtiene del conjunto de tablas involucradas $TI(ie)$

- **Atributos:**

  Es importante mencionar que no siempre todas las tablas que conforman la red candidata serán proyectadas, pues existen elementos intermedios que sólo restan claridad a la respuesta tabular. Incluso no todas las tablas donde se hayaron las palabras clave verán sus atributos proyectados. Se busco entregar al usuario una respuesta breve que contenga los elementos buscados, y si es conveniente otros los atributos de aquellas tablas interpretadas.

  La secuencia es simple: *siempre se proyectan los atributos que fueron solicitadas explícitamente*, y *atributos cuyas tablas fueron textualmente interpretadas*; por ejemplo todas los atributos de la tabla *customers* en la consulta *“1969 camaro customers usa”*. También se proyectan *los atributos donde se encuentran las palabras clave interpretadas gracias al índice invertido*, como es el caso de *customers.country o products.productName*; por ser los atributos que alojan los registros encotrados gracias a los términos de la consulta *{“1969”, “customers”, “usa”}*. Puede llamarse a este tipo de atributos, **atributos explícitos**.

  Con lo anterior el sistema se asegura de proporcionar al menos todo lo que fue relacionado con los términos de entrada. En el caso de los atributos de las interpretaciones de palabra clave como tabla, no se puede discrimiar ningún atributo puesto que no hay información útil para inferir si el usuario desea toda la información de la tabla o ciertas partes.

  Pero, con los atributos que no cumplen con ninguna de las dos condiciones anteriores, es decir los atributos no explícitas, la determinación es diferente. Imagine que se utilizará el operador * en vez de proyectar al menos las atributos explícitos, el resultado podría ser excesivamente largo, y el resto de atributos de las tablas en interpretaciones textuales de atributo, o en interpretaciones como valores del índice se incluirían y podrían ser una carga visual al usuario.

  Con esta óptica el sistema determina si se deben incluir o no dependiendo de la “longitud” de la consulta; realizándolo cuando la connsulta es corta, o en su defecto, sin proyectas las *atributos*

no explícitos. Si la red candidata mínima de la interpretación de consulta candidata seleccionada *incluye más de dos tablas únicamente se proyectarán las atributos explícitas, en caso contrario se incluyen todas*.

Finalmente el conjunto de atributos proyectados se implanta en la plantilla remplazando el segmento $< atributos >$.

En el ejemplo se proyectarían únicamente los atributos explícitos ya que se supero el umbral para incluir al resto, tales atributoss son [*products.productName* y todas los atributos de la tabla*customers* (ya que ella incluye el atributo *country* explícito gracias al término "usa").

- **Predicados FK-PK:**

  Los predicados de llave foránea-llave primaria son creados de una manera muy simple, iterando la red candidata se inserta un predicado "$t_1.FK = t2.PK$" por cada pareja $t_i, t_j \in RC(ie)$ más un operador " AND " entre cada dos predicados. Por ejemplo, para la consulta *"1969 camaro customers usa"*, cuya interpretación seleccionada contiene las tablas *{ products, orderDetails, orders, customers}* en su red candiadata 3.10, se incluiría la cadena de predicados siguiente "orderDetails.productCode = products.productCode AND orderDetails.orderNumber = orders.orderNumber AND orders.customerNumber = customers.customerNumber".

- **Predicados Valores:**

  Finalmente se agregan los predicados que filtran horizontalmente el resultado, sustituyendo el segmento $< predicadosdevalores >$.

  Debido al uso de un índice invertido por columnas como el que sugiere DBXplorer (Agrawal et al., 2002) los fitros se hacen con predicados del estilo *atributo="palabra clave del índice"*, muy similar a como lo hacen los usuarios de SQL. Por ejemplo en la consulta "offices nyc", que busca las oficinas ubicadas en New York City (véase la tabla 3.6 de la página 47) realiza el filtrado con el predicado *offices.city="NYC"*. Tal filtro es el mismo que un usuario con conocimientos de SQL realizaría, es poco probable que se sustituya el anterior predicado por *offices.officeCode=3* ya que códigos y dígitos son difíciles de memorizar. Concretamente se agrega a la consulta un predicado de valor por cada palabra clave con interpretación como valor del índice invertido, ya que una entrada del índice señala la tabla y atributo donde se encuentra el valor. Cabe destacar que el sistema tiene la capacidad de reconocer cuándo debe de usarse un operador AND u OR al añadir predicados; **si dos predicados tienen las mismas tabla y atributo** no pueden hacer referencia a la misma tupla por lo que **se conectan con un operador OR**, que permitirá mostrar ambas tuplas

en el ResultSet final. **Al caso contrario**, el más usual, y **le corresponde un operador AND**

Para consulta "1969 camaro customers usa" se inclyen predicados para el auto y el país; *customers.country="USA" AND products.productName="1969 Chevrolet Camaro Z28"*.

Juntando los elementos recién mencionados, la consulta a la que se ha referido el documento durante la descripción de *KweryDB*, "1969 camaro customers usa", tiene como resultado la consulta siguiente:

```
SELECT t8.productName, t0.* FROM customers t0, products t8,
orderdetails t4,orders t5 WHERE t4.productCode=t8.productCode
AND t4.orderNumber=t5.orderNumber AND
t5.customerNumber=t0.customerNumber AND t0.country="USA"
AND t8.productName="1969 Chevrolet Camaro Z28"}
```

Como puede verse el sistema realiza una traducción directa, buscando que la salida tabular de un DMBS al procesar la consulta SQL sea clara y lo más corta posible. Y que aún así, incluya todos elementos que fueron interpretados (con mejor calificación).

Una de las ventajas de este sistema es que ofrece como respuesta consultas SQL en función de los términos y no de los identidificadores de tupla, como lo hacen algunos de los sistemas analizados en el apartado de Trabajos Existentes. Esto se debe en gran medida al índice por columnas, el cual obliga a basarse en un filtrado por los valores deseados y no por llaves primarias de las tuplas (ya que no se encuentran en el índice).

3.5. Conclusiones

A lo largo de este capítulo se describió el enfoque de **KweryDB** para consultas por palabras clave en bases de datos estructuradas. Se comentó ampliamente la arquitectura, la cual consiste en tres módulos principales: *Procesador de Consultas, Interpretación y Desambiguación, y Generación de SQL*. Además, cada uno de los módulos fue explicado proporcionando el algoritmo y la descripción correspondiente, e ilustrado con desarrollo de la consulta de ejemplo *"1969 camaro customers usa"*.

Recapitulando la secuencia de operación en *KweryDB*, el primer paso que se toma para obtener una consulta SQL de conjunto de palabras clave es la **interpretación** de cada uno de los términos. Cada palabra clave en la consulta puede tener varias interpretaciones distintas, sean como tablas, atributos o valores de la base de datos. Ya que tales interpretaciones son individuales es necesario obtener todas las combinaciones posibles entre los distintos sentidos dados a los términos de la consulta, es decir obtener el producto cartesiano de todas las interpretaciones de palabras clave. Con esto se conocen todas las posibles respuestas a la consulta del usuario, a cada elemento de este conjunto se le llama *Interpretación Candidata*.

Posteriormente es necesario determinar la respuesta correcta entre todas las interpretaciones candidatas. KweryDB realiza una evaluación en dos partes bajo la optica de calcular el menor número de Redes Candidatas (o árbol Steiner) y llegar más rápido al resultado. La primera evaluación consiste está conformada por tres puntajes específicos: calificación de tablas involucradas, valores del índice intrepretados y coincidencias en interpreteaciones. Con estas variables es posible favorecer a las opciones con menor número de tablas y de valores del índice, especialmente si las interpretaciones de dos o más palabras de la consulta indican el mismo valor del índice o tupla de la base de datos.

Después de la primera evaluación se tienen indicios de cuál respuesta puede ser la indicada, pero aún no es posible traducir directamente a una consulta SQL. Se necesita conocer el camino de joins para conectar todas las palabras clave de la manera más cercana, es decir encontrar la sucesió de joins que vincula las tablas que contienen los términos de consulta del usuario. Lo anterior es llamado en la literatura, cálculo de redes candidata, y requiere otorgar a cada interpretación candidata un sub-árbol del esquema de base de datos donde se encuentre la manera más corta de conectar las tablas presentes en dicha interpretación. En KweryDB se consideró que a pesar de que en las interpretaciones candidatas las palabras clave se interpreten distinto es común que coincidan en tablas, y por consiguiente tengan la misma Red Candidata. Por este motivo sólo se computan las mejores tres (top-3) redes candidatas distintas, pues éstas pueden ser útiles no sólo a las mejores tres interpretaciones candidatas si no también a otras interpretaciones que hagan referencia a las mismas tablas involucradas en el top-3.

Este conjunto de interpretaciones candidatas jerarquizadas que ahora incluyen su correspondiente red candidata (una de las tres), pueden ser evaluadas por segunda ocasión añadiendo una nueva variable a la ecuación: longitud de la red candidata, es decir número de joins en la red candidata. Después de la segunda evaluación solo resta escoger la interpretación candidata con la mejor calificación para construir una consulta SQL con el formato `SELECT * FROM ...`. Cabe destacar que si hay un empate en las calificaciones, *KweryBD* seleccionará una de manera aleatoria pues las métricas son insuficientes en tales casos.

A continuación se encuentra la experimentación realizada sobre *KweryDB*, donde se retoman los planteamientos expresados a lo largo del capítulo y se contrasta el desempeño de un prototipo con otros sistemas de búsqueda por palabras clave en bases de datos estructuradas.

Capítulo 4

Validación Experimental

4.1. Introducción

El creciente aumento en la generación de información y la muda de la mayoría de organizaciones al plano web ha supuesto que las bases de datos sean el sostén de un intercambio de datos voráz, donde todo tipo de usuario es consumidor de la información que éstas ofertan. Tal escenario no es asequible a través de consultas SQL, pues es evidente que no todas las personas tienen una formación tecnológica.

En este documento se ha presentado una alternativa a la consulta de bases de datos a través de consultas en lenguaje SQL en un sentido distinto que el de otros autores: se desea obtener un resultado *exacto* que refleje la intención del usuario al cuestinonar el sistema, en vez de mostrar un conjunto *top-k* donde se espera que se encuentre la mejor opción. Antes de decidir construir un sistema con el anterior requisito fue necesario hacer una revisión exhaustiva de los trabajos en torno a la búsqueda en bases de datos por palabras claves. En concreto se revisarón sistemas de dos categorias: sistemas basados en grafos y en redes candidatas, como mencionan Chaudhuri y Das (2009).

De ambas clases se tomó la determinación para que la propuesta se alinease al paradigma de redes candidatas, implementando los elementos más destacados de los sistemas revisados; como el índice por columnas usado por Sarda y Jain (2001) y Agrawal et al. (2002); y añadiendo nuevas características que sopesaran las desventajas de los distintos sistemas, especialmente en lo que concierne a la interpretación y evaluación de la consulta, pues en la mayoría de sistemas el criterio de selección es únicamente el tamaño del árbol Steiner o de la Red Candidata, según sea la clase de sistema. Como se comentó anteriormente se proponé una interpretación en dos momentos distintos para reducir el número de redes candidatas a computar.

En las anteriores secciones se describieron con detenimiento los conceptos teóricos en los cuales está

cimentada la propuesta, se comentó la elaboración del prototipo que instancia tales aseveraciones. En adelante se abordarán los métodos utilizados para la comprobación y validación del prototipo.

Debido a situaciones de tiempo y compatibilidad no es posible comparar el desempeño del sistema propuesto contra el resto de sistemas analizados de la literatura. Existen muchas particularidades en los trabajos realizados que no permiten una evaluación en igualdad de condiciones.

El primer problema surge gracias al paradigma de sistema, como es de suponerse no sería correcto comparar un sistema de consulta por palabras clave basado en redes candidatas contra uno basado en grafos, por la sencilla razón de no poder usar el mismo origen de datos: uno utilizaría la base de datos mientras que el otro necesitaria la creación de un grafo de datos. Además, no es posible evaluar el prototipo contra sistemas como SQAK (Tata y Lohman, 2008) o Kite (Sayyadian et al., 2007), pues éstos incluyen características que a pesar de ser interesantes en el escenario de búsqueda por palabras clave; por ejemplo funciones de agregado o la posibilidad de consultar bases de datos heterogéneas; se encuentran fuera del alcance de esta investigación. Otra *situación* aparece gracias al tipo de índice invertido que se implementó en el sistema; por columnas, como el que aparece únicamente en Mragyati y DBXplorer. Como trabajo futuro podría realizarse una comparación del desempeño con sistemas que implementen otro tipo índice.

El sistema desarrollado se contrastará con **Mragyati** (Sarda y Jain, 2001). Este sistema tiene una secuencia de operación muy sencilla en la cual está inspirada la propuesta del trabajo, pero con algunos cambios importantes que se veran reflejados en los resultados. Cabe destacar que para demostrar la valía final del sistema desarrollado es necesario medirse con todo tipo de sistemas de consulta a bases de datos por palabras clave, pero en un primer momento es menester conocer los resultados del sistema en comparación con los trabajos más similares.

Tal evaluación se da en un sentido especial, el sistema se ha planteado de forma que a una consulta por palabras clave le corresponda otra consulta en SQL que satisfaga las expectativas del usuario. Lo anterior puede verse de dos maneras distintas: como la *experiencia del usuario* al utilizar el sistema; es decir, si considera que tanto la presentación de los resultados como la interpretación de las palabras clave fue la correcta para él. Y el otro modo, bajo la óptica más sencilla de evaluar, como *completitud de resultados*; donde el usuario espera un conjunto determinado de datos al escribir su consulta, llámese a éste *el conjunto dorado* (como una analogía a las clásicas fábulas de duendes, arcoíris y cofres de oro). De modo que el resultado debe ser igual, o acercarse considerablemente, al conjunto dorado para ser considerado como completo.

Se realizarán las pruebas comparando una serie de consultas y sus conjuntos dorados contra los resultados obtenidos en *KweryDB* y en *Mragyati* (Sarda y Jain, 2001). La idea de la que se parte es que

esta comparación brindará un primer parámetro del desempeño del sistema desarrollado, especialmente en términos de interpretación de la consulta. Se espera que el prototipo tenga mejores resultados que Mragyati, pues se puede decir que *KweryDB* es una versión extendida de ese sistema; al cual se le ha añadido una evaluación en dos momentos con más criterios de calificicación, y el algoritmo de cálculo de árbol Steiner mínimo para generar Redes Candidatas mínimas.

Del mismo modo se espera que las limitaciones de *KweryDB* salgan a flote y den pauta a trabajos futuros; palabras clave que no son interpretadas por no ser textualmente iguales a las referidas (por ejemplo sinónimos, conceptos génericos de un término, etc.), retricciones de llave primaria-llave foránea circulares (de una tabla a sí misma) o diferentes interpretaciones de palabras clave ubicadas a la misma distancia en el grafo de esquema son algunas de las múltiples deficiencias de este sistema (y la mayoría de trabajos revisados).

Finalmente se agregan a este capítulo observaciones referentes a los tiempos de respuesta y tamaño de las redes candidatas en las respuestas de ambos sistemas; con el objetivo de contrastar la evaluación principal, completitud de resultados, contra indicadores de desempeño y especificidad.

4.2. Materiales y Métodos

Para realizar la validación de *KweryDB* se ha decidido realizar una serie consultas por palabras clave sobre el sistema propuesto y el sistema Mragyati (Sarda y Jain, 2001) que evidencien las posibilidades y deficiencias de ambos trabajos, tal evaluación se da en razón de la completitud de resultados. Como métricas secundarias se registraron el tiempo en que los sistemas antes mencionados respondieron a las consultas anteriores (y a algunas otras), y la lonitud de la cad ena de joins (o tamaño de la Red Candidata) de cada respuesta.

Al igual que en la descripción, se usará la base de datos *classicmodels* (Doe, 2015) cuyo esquema puede observarse en la figura 3.3, tal base de datos contiene *3864 tuplas* repartidas en *8 tablas*. El índice invertido para la base de datos *classicmodels* presenta 800 registros con el modelo *{id, palabra clave, tabla, atributo}*.

Esta base de datos ofrece posibilidades interesantes para formular consulstas por palabras clave, ya que existen nombres iguales de atributos en distintas tablas, valores en el índice similares para distintas tablas y atributos, entre otras características delicadas que sin la interpretación adecuada podrían conducir a resultados inesperados.

A continuación se presenta la tabla 4.1, en ella se encuentran 10 consultas por palabras clave que serán procesadas en ambos sistemas con el fin de conocer las ventajas de uno y el otro, así como las

limitaciones de cualquiera de los dos sistemas al procesar una de estas consultas. Tal evaluación se realizará contemplando la presencia del conjunto esperado de resultados, o conjunto dorado, en relación a la intención detrás de las palabras clave de cada consulta. Lo anterior se encuentra registrado en la tabla 4.2, se observan las interpretaciones que se desean para cada palabra clave, ya que en algunas consultas existen muchas posibilidades de interetación (el número de posibles interpretaciones, obtenido mediante el producto cartesiano de las interpretaciones individuales de cada término, puede apreciarse en la tabla 4.1).

	Consulta	**Intención**	**Posibilidades ($\|IC\|$)**
1	*offices*	todas las oficinas de la empresa	1
2	*in process on hold orders*	todas las órdenes de productos en proceso o en espera	1
3	*1969 camaro customers usa*	Clientes estadounidenses que hayan comprado un camaro 1969	20
4	*Gerard Bondur*	Empleado con nombre Gerard Bondur	2
5	*Emea employes phone*	Teléfono de los empleados del territorio EMEA	4
6	*Products cancelled Paris offices*	Productos cancelados que vendieron los empleados de las oficinas de Paris	2
7	*Gerard Martin*	Empleado con nombre Martin Gerard	6
8	*uk employees jobtitle*	Cargos que poseen los empleados que trabajan en el Reino Unido (en oficinas)	2
9	*the queen mary customers*	Clientes que han comprado el producto “The Queen Mary”	3
10	*studio art models*	Productos del proveedor “Studio M Art Models”	12

Tabla 4.1: Consultas de prueba

Para la evaluación de tiempos de respuesta y tamaño de las redes candidatas se utilizaron las respuestas a las consultas de la tabla 4.1, más las de otro conjunto de 10 consultas (cuyos resultados específicos no son presentados por razones de espacio en el documento); un total de 20 consultas. Cabe destacar que en esta parte de la experimentación se desea conocer si *KweryDB* responde a una velocidad similar a la de Mragyati, considerando que implementa nuevas funciones que teóricamente mejorarían los resultados, pero a al mismo tiempo ralentizan el proceso de búsqueda; y también, saber si la evaluación favorece a redes candidatas más pequeñas, las cuales se presumen más puntuales, pues se asume que las

	Interpretaciones			Red Candidata
palabra clave	Tabla	Atributo	Valor	
1.- *offices*				
•*offices*	offices			{of}
2 .- *in procress on hold orders*				
•*in procress*	orders	status	"In Process"	{or}
•*on hold*	orders	status	"On Hold"	
•*orders*	orders			
3.- *1969 camaro customers usa*				
•*1969 camaro*	products	productName	"1969 Chevrolet Camaro Z28"	{pr, od, or, c}
•*customers*	customers			
•*usa*	customers	country	"USA"	
4.- *Gerard Bondur*				
•*Gerard*	employees	firstName	"Gerard"	{e}
•*Bondur*	employees	lastName	"Bondur"	
5.- *Emea employees phone*				
•*Emea*	offices	territory	"EMEA"	{e,of}
•*employees*	employees			
•*phone*	offices	phone		
6.- *products cancelled Paris offices*				
•*products*	products			{pr, od, or, c,e,of}
•*cancelled*	orders	status	"Cancelled"	
•*Paris*	offices	city	"Paris"	
•*offices*	offices			
7.- *Gerard Martin*				
•*Gerard*	employees	lastName	"Gerard"	{e}
•*Martin*	employees	firstName	"Martin"	
8.- *uk employees jobtitle*				
•*uk*	offices	country	"UK"	{e,of}
•*employees*	employees			
•*jobtitle*	employees	jobtitle		
9.- *the queen mary customers*				
•*the queen mary*	products	productname	"The Queen Mary"	{pr, od, or, c}
•*customers*	customers			
10.- *studio art models*				
• *studio art models*	products	productVendor	"Studio M Art Models"	{pr}

Tabla 4.2: Resultados Esperados para las consultas de la tabla 4.1

nuevas características han acotado el espectro de búsqueda para apuntar a los elementos de la base de datos deseados, y mientras menor sea el número se piensa que se hay más posibilidad de haber dado en los blancos.

Con los resultadas de dichas consultas, se obtuvo:

- *La media de segundos* que tardó en responder cada sistema, contando desde que comienza la interpretación de palabras clave hasta que la consulta SQL fue totalmente construida, agrupando las consultas *por el número de palabras clave* que contiene.

- *Tamaño promedio de red candidata en las consultas con n palabras clave.* El tamaño es igual al número de tablas en la red candidata o a la unidad cuando no se calculó una red candidata para la Interpretación Candidata seleccionada, es decir cuando en el resultado sólo intervenga una tabla.

A continuación se encuentra el resultado de las consultas de la tabla 4.1 al realizarse sobre el sistema desarrollado y, un sistema de consula basado en redes candidatas llamado Mragyati (Sarda y Jain, 2001). También se observan dos gráficos (véase la figura 4.2) que aluden al tiempo de respuesta en cada sistema con relación al número de palabras clave; y al tamaño de la red candidata, a su vez en función del número de términos.

Número	Consulta	KweryDB	Mragyati	SQL correcto
1	*offices*	✓	✓	`SELECT t3.* FROM offices t3`
2	*in process on hold orders*	✓	✓	`SELECT t5.* FROM orders t5 WHERE (t5.status="In Process" or t5.status="On Hold")`
3	*1969 camaro customers usa*	✓	×	`SELECT t8.productName, t0.* FROM customers t0, products t8, orderdetails t4, orders t5 WHERE t4.productCode=t8.productCode AND t4.orderNumber=t5.orderNumber AND t5.customerNumber=t0.customerNumber AND t0.country="USA" AND t8.productName="1969 Chevrolet Camaro Z28"`
4	*Gerard Bondur*	✓	✓	`SELECT t1.* FROM employees t1 WHERE t1.lastName="Bondur" AND t1.firstName="Gerard"`
5	*Emea employes phone*	✓	×	`SELECT t3.phone ,t1.* FROM employees t1,offices t3 WHERE t1.officeCode=t3.officeCode AND t3.territory="EMEA"`
6	*Products cancelled Paris offices*	✓	×	`SELECT t8.* ,t5.status ,t3.city FROM customers t0,employees t1,offices t3,orderdetails t4,orders t5,products t8 WHERE t0.salesRepEmployeeNumber=t1.employeeNumber AND t1.officeCode=t3.officeCode AND t4.productCode=t8.productCode AND t4.orderNumber=t5.orderNumber AND t5.customerNumber=t0.customerNumber AND t5.status="Cancelled" AND t3.city="Paris"`
7	*Gerard Martin*	✓	×	`SELECT t1.* FROM employees t1 WHERE t1.lastName="Gerard" AND t1.firstName="Martin"`
8	*uk employees jobtitle*	×	×	`SELECT t3.* ,t1.jobTitle FROM employees t1,offices t3 WHERE t1.officeCode=t3.officeCode AND t3.country="UK"`
9	*the queen mary customers*	✓	×	`SELECT t8.productName ,t0.* FROM customers t0,products t8,orderdetails t4,orders t5 WHERE t4.productCode=t8.productCode AND t4.orderNumber=t5.orderNumber AND t5.customerNumber=t0.customerNumber AND t8.productName="The Queen Mary"`
10	*studio art models*	✓	×	`SELECT t8.* FROM products t8 WHERE t8.productVendor="Studio M Art Models"`

Tabla 4.3: Resultados en el sistema propuesto y Mragyati

4.3. Análisis de Resultados

La mayoría de las consultas ha sido efectivamente procesada por el sistema aquí propuesto, consiguiendo tal éxito con base en los criterios de evaluación incluídos: *la cantidadad de registros del índice*

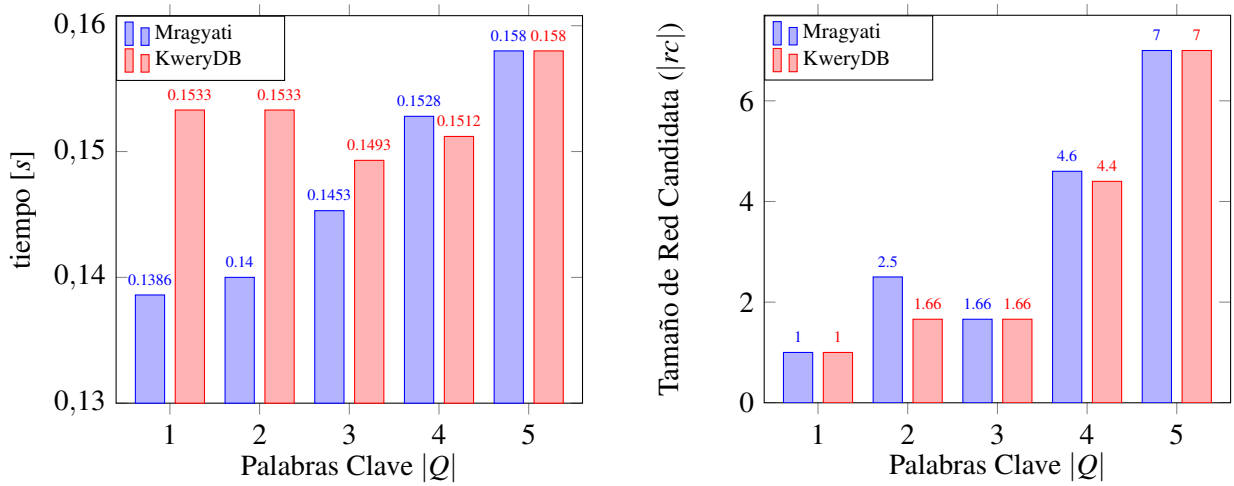


Figura 4.1: Promedio de Tiempo y tamaño de redes candidatas por no. de palabras clave en Mragyati y KweryDB

que se interpretaron en contraste con las coincidencias de tuplas, número de tablas involucradas en la interpretación y la longitud de la red candidata. En adelante se analizan los resultados de cada una de las consultas en específico, poniendo en relieve las ventajas de KweryDB en contra de Mragyati, así las limitaciones de ambos.

1. ***offices*.-** En esta sencilla consulta basta interpretar correctamente la palabra *office* como la tabla homónima. Como puede verse en la tabla 4.3 ambos sistemas responden con éxito a ella.

2. ***in process on hold orders*.-** La siguiente demuestra la flexibilidad de ambas propuestas para manejar semántica OR cuando es necesario. Ésta se identifica cuando dos registros del índice interpretados por distintas palabras clave apuntan a la misma tabla y atributo, por lo que no pueden referirse a la misma tupla. Utilizar una semántica AND excluiría de la salida ambas interpretaciones.

3. ***1969 camaro customers usa*.-** A lo largo de la descripción de los componentes de *KweryDB* se hizo referencia a esta consulta, la cual presenta algunas de las características más importantes que se implementaron en el sistema. Por ejemplo la coincidencia de interpretaciones del mismo registro del índice, gracias este conocimiento es posible segregar las 19 posibildades que son incorrectas (en el apéndice C se encuentra el desarrollo de esta consulta en detalle).

   Mragyati puede dar una interpretaciones a palabras clave donde el valor del índce contiene término correcto, por ejemplo para *"1969"* se podría considerar el registro *"1969 Corvair Monza"* como el adecuado. Lo anterior no está errado si se observa de manera aislada, pero cuando se consideran el resto de palabras clave es notorio que la selección de tal registro y no de *"1969 Chevrolet Camaro Z28"* (donde también encuentra la palabra *"camaro"*) no es el mejor. En cambio, la propuesta

de este trabajo es capaz de indentificar tal situación. Mragyati al únicamente implementar como medida de selección la longitud de la red Candidata; y puesto que sólo existen dos redes candidatas, {products,orderDetails,Orders,customers} y {products,orderDetails,Orders,customers,employees, offices}, seleccionará aleatoriamente una de las IC de la primera red candidata más pequeña.

4. ***Gerard Bondur*.-** Esta consulta podría parecer sencilla pero con un índice por columnas ello depende de un análisis más profundo que el realizado por Mragyati. En las pruebas ha respondido correctamente gracias que con sólo dos opciones con el mismo tamaño de red candidata (la unidad, pues se refiere a un empleado de la tabla *employees*) aleatoriamente se escogió la opción correcta.

   En el caso de la propuesta desarrollada en este trabajo, *KweryDB*, la situación es reconocible a partir de la comparación del índice invertido, la estructura de la tabla employees y las llaves primarias de los registros del índice, con estos datos es posible determinar que *"Gerard" y "Bondur"* hacen referencia a la misma tupla, pero en distinto atributo.

5. ***Emea employees phone*.-** Otro ejemplo del buen hacer del sistema de búsqueda de este trabajo se observa al procesar esta consulta, ya que gracias a la función de semejanza que identifica la porción interpretable de una palabra clave como cierto registro del índice invertido.

   Al analizar las opciones que existen para interpretar al término "Emea", el cual se refiere al territorio europeo donde laboran ciertas oficinas, se descubren dos posibilidades: la primera como un cargo de empleado como admministrador de ventas de el territorio EMEA (*"Sale Manager (EMEA)"*), la otra opción corresponde al terriorio antes comentado que aparece en la tabla *offices*. Por razones obvias, y la aclaración de la expectativa de la consulta, el no tener ninguna palabra que haga referencia a cargos hace sospechar que al sistema que la mejor opción sería decantarse por el registro de la tabla offices.

   Esta decisión también posibilita escoger el atributo teléfono (*phone*) de la tabla *offices* en vez de su semejante ubicada en la tabla *customers*, lo cuál tiene más sentido ya que el teléfono de la oficina de un empleado es su teléfono también, en cambio sería muy extraño que un empleado tuviera el mismo número telefónico que sus clientes.

   Mragyati no es capaz de identicar ninguno de los dos escenarios; pues no contempla si la palabra clave es exactamente igual al valor del índice invertido o es una sub-cadena, y por consiguiente no entra en la selección.

6. ***Products cancelled Paris offices*.-** Aquí se ejemplifica claramente la necesidad de aplicar más métricas a la selección que la longitud del Steiner Tree o de la Red Candidata. El prototipo de

KweryDB respondió adecuadamente a las expectativas expresadas en las tablas 4.1 y 4.2, encontrando los significados de cada palabra clave que se ajusten mejor al del resto de palabras claves.

En cambio Mragyati tiene como única medida el tamaño de la red candidata, por lo que al otorgale un sentido a la palabra *Paris* elige la que se encuentra más cercana al resto de interpretaciones (en el esquema), pero como las interpretaciones de Paris como la ciudad de una oficina o de un conjunto de clientes se encuentran en la sucesión de operación join necesarias para unir al resto de palabras clave, la selección es aleatoria. En ocasiones Mragyati lo hará bien y en otras no tanto.

Afortunadamente para Mragyati no debería de haber un cambio muy grande en las tuplas otorgadas como salida; es probable que muchos de los clientes de la oficina de Paris también sean parisinos.

7. ***Gerard Martin*.-** En el análisis de la consulta número 4 se comentaba que existían empleados con nombre de pila "Gerard" y también con "Gerard" como apellido, por lo que no sólo existe una entrada con valor "Gerard" en el índice, sino dos. Esta situación delicada se incrementa al buscar el sentido correcto para "Martin" pues en el índice existen entradas que apuntan a tres tablas *{products, employees, customers}*. A pesar de que con los valores del tamaño de cada red candidata se pueden discriminar dos opciones(las que interpretan a *"Martin"* como producto o cliente), aún queda el problema de saber si se pregunta por *"Gerard"* como nombre o apellido. Como puede apreciarse en la tabla 4.3 Mragyati no puede solucionar tal dilema, pero *KweryDB*, nuestra propuesta; la cual presenta una estrategia para descubrir si dos valores del índice se refieren a misma tupla en la base de datos, sí. Encontramos que *"Gerard"* apellido se encuentra en la misma tupla que *"Martin"* nombre de pila, es decir son el mismo individuo y por lo tanto la respuesta más apropiada.

8. ***UK employees jobtitle*.-** Como era digno de sospecharse, no todo tipo de consultas podrán ser respondidas con éxito. Los cuatro criterios de calificación de las Interpretaciones Candidatas no son suficientes cuando dos interpretaciones de una palabra clave con el mismo valor de semejanza se encuentran a la misma distancia de el resto de palabras clave (sin que esto cambie la longitud de red candidata).

   Esta consulta presenta lo anterior, la tabla *employees* no tiene atributos que hagan referencia a ubicación, por lo que lo lógico sería utilizar la de la oficina a la que pertenecen. Pero al existir un registro *UK* en customers y en offices, que se encuentran directamente conectados (Vía FK-PK) con la tabla employyes, el sistema no puede decantarse por ninguno con los criterios de evaluación asignados. En ambos sistemas se realiza una selección entre estas dos IC de manera aleatoria.

9. ***The queen mary customers*.-** Incluso en este caso sencillo, donde la primera parte de la consulta hace referencia al producto de tipo barco llamado *"The Queen Mary"* y la parte final sólo puede interpretarse como la tabla clientes, Mragyati no puede responder correctamente; pues no contabiliza las ocurrencias de una entrada del índice para considerarse como la determinación del usuario para que éstas coincidiesen y expresaran mejor la intención de su consulta.

10. ***studio art models*.-** Mismo caso que en la consulta pasada, un conjunto de interpretaciones de distintas palabras clave como el mismo registro del índice invertido que son pasadas por alto.

4.4. Discusión

Es posible decir que la ejecución de un conjunto de consultas por palabras clave en ambos sistemas es insuficiete para contrastar el trabajo con los sistemas basados en redes candidadatas y, en un nivel superior, con los sistemas de búsqueda por palabras clave en general. Pero para evaluar el objetivo principal de esta investigación, *arrojar una sóla respuesta que refleje exactamente la intención del usuario* al introducir una consulta, es el primer paso. Y así demostrar tanto la flexibilidad y posibilidades, así como las deficiencias del sistema.

Del lado de ventajas es muy claro que Mragyati, y el resto de sistemas que utlizan un índice por columnas sin contemplar los criterios de coincidencia de tuplas de la base de datos o del índice invertido para seleccionar la mejor opción, no serán capaces de interpretar consultas como Gerard Martin, EMEA employees phone, etc.. Es claro que el sistema *KweryDB* ha superado con creces al sistema Mragyati, desarrollado por Sarda y Jain (2001); donde las limitaciones de coincidencia de valores y de nombres de atributos iguales, son solucionadas (parcialmente) con estrategias puntuales, ademas de una evaluación de dos fases que apoya a reducir el número de Redes Candidatas computadas.

Dirigiéndose a la tabla 4.3 se puede observar que Mragyati sólo acertó en las consultas donde no había muchas opciones; por ejemplo la consulta *"offices"* tiene solo una interpretación candidata, al igual que la cosulta *"in process on hold orders"*. La excepción es *"Gerard Bondur"* donde sólo el órden de aparición de la palabra *"Gerard"* ha ayudado a tomar la determinación adecuada.

Continuando con el análisis de la tabla 4.3, es apreciable el buen hacer del sistema propuesto, ya que únicamente ha fallado en una consulta. Para el resto, los criterios implementados han sido suficientes para determinar cuál es la mejor interpretación candidata para un conjunto específico de palabras clave. A pesar de ello un consulta salta a la vista: *"uk employees jobtitle"* y aunque en sí misma la consulta es muy clara, al buscar la interpretación adecuada a cada palabra clave y la conexión más corta de éstas el sistema se topa con un dilema que no es capaz de solucionar: *¿Qué camino tomar cuando las calificaciones de*

dos Interpretaciones Candidatas son iguales después de las dos fases de evaluación?.En el presente trabajo tal pregunta queda sin responder, sin otra palabra clave que apoye a la desmbiguación respecto a la interpretación de *"UK"* como ubicación de clientes o de oficinas, es imposible tomar la determinación adecuada.

Como se indicó hace algunos párrafos, la valoración del sistema propuesto se ha realizado considerando la completitud de resultados que éste ofrece. Sin embargo es interesante observar a Mragyati y a *KweryDB* con otra perspectiva, por ejemplo un indicador importante es el tiempo; aunque Mragyati falló más veces que el sistema desarrollado en este trabajo, no tiene registros de tiempo de respuesta tan diferentes al de *KweryDB*). Observando la figura 4.2, en consultas con 1 o 2 palabras clave se aprecia que *KweryDB* ha superado el tiempo de respuesta de Mragyati en céntimas de segundo mientras que en las consultas de 3 a 5 términos la difrencia es de milésimas de segundo.

Evidentemente la valoración de si tal diferencia en tiempo es significativa dependerá del uso que se haga de este sistema de búsqueda, así como de una evaluación con bases de datos más grandes; pero es notorio que la evaluación de dos fases que se ha incluído en el sistema no ha causado grandes estragos en el desempeño. Y gracias al diseño de *KweryDB* no se vislumbra gran dificultad para el escalamiento, pues siendo el índice invertido la única estructura de datos que crecería si la BD es mayor, sólo se hacen $n+1$ accesos a la base de datos durante una consulta por palabras clave; n para obtener las interpretaciones del índice invertido de cada palabra clave y una última para obtener el ResultSet de la consulta SQL resultante.

Con respecto al tamaño de las redes candidatas resultantes en Mragyati y en el sistema desarrollado la expectativa se ha cumplido. Se esperaba que *KweryDB* tuviera un mayor número de respuestas correctas que Mragyati, y las tuvo, pero al analizar detenidamente la salida de ambos sistemas se puede ver que en ningún caso Mragyati ofreció una red candidata más pequeña que el sistema propuesto en este trabajo lo que podría interpretarse como una victoria ya que mientras menores sean las redes candidatas se presume se encontraron relaciones entre las palabras claves que descubrieron su significado (al menos parcialmente). La razón recae en las métricas de desambiguación; al momento de interpretar consultas donde se necesitaba considerar las coincidencias de tuplas o de registros del índice donde éstas no influyan en el tamaño de la red cadidaata Mragyati pasa por alto tales coincidencias y ve como sus redes candidatas se tornan más grandes pues sólo consideró el tamaño de la red candidata, como muchos de los sistemas de la literatura.

Para conseguir el resultado correcto de esta consulta es necesario poseer más información acerca del esquema de base de datos, del usuario y su entorno, etc. Existe una rama de la computación que pretende mejorar los servicios a través del procesamiento de datos extras provenientes del contexto, a

tal campo se le conoce como Cómputo Consciente del Contexto. La respuesta a muchos incovenientes puede encontrarse en el uso de información conntextual que ayude a la desambiguación de consultas, ya que es probablemente uno de los problemas más importantes en Informmation Retrieval y también la consulta de bases de datos por palabras clave; encontrar el significado que el usuario le dio a su consulta, lo cual depende inevitablemente del contexto en que esa consulta fue introducida, por quién, cuándo, etc.

Como trabajo futuro se propone realizar una validación experimental más fuerte, que evalúe principalmente la completitud de resultados en otros sistemas de búsqueda en bases de datos por palabras clave, sin importar la manera en que éstos sean implementados; basados en grafos o redes candidatas, con soporte de funciones de agregado (como promedio, máximo, etc.) o sin él, concepciones distintas del índice invertido, y demás diferencias.

Capítulo 5

Conclusiones

5.1. Resumen

Gracias a los avances tecnológicos y sociales, el número de usuarios de servicios de la red ha aumentado exponencialmente. La enorme cantidad de información que éstos generan se almacena generalmente en bases de datos, las cuales deben ser cuestionadas de una manera específica por las aplicaciones y usuarios que requieren de su información; sea a través de interfaces predefinidas que sólo permiten una comunicación limitada, o "libremente" introducir una consulta codificada en el lenguaje correcto, SQL en la mayoría de los casos. Si bien usuarios con conocimientos en informática pueden utilzar la segunda vía, aún se encuentran con un obstáculo: conocer el esquema de la base de datos. Como puede suponerse, en un entorno de consulta dinámico los usuarios finales no tendrían porqué recordar exactamente el nombre de cada tabla o columna de la base de datos. Sería muy cómodo contar con una alternativa de consulta a las interfaces predefinidas en las aplicaciones, o en su defecto, a ingresar consultas SQL al manejador de base de datos.

Respondiendo a la anterior necesidad, diversos científicos desarrollaron sistemas capaces de sostener una comunicación libre mediante palabras clave. Según Chaudhuri y Das (2009) existen dos clases de sistemas de búsqueda en bases de datos por palabras clave: *Sistemas de consulta basados en grafos* y *en redes candidatas*. Los primeros necesitan vaciar el contendido de la base de datos en un grafo, para que sobre éste se apliquen algoritmos de búsqueda que indiquen la respuesta correcta. En cambio, los sistemas basados en redes candidatas no requieren de un grafo de datos, pues ellos basan su procesamiento en el conocimiento del esquema de base de datos y las interpretaciones candidatas de la consulta para encontrar el mejor resultado.

Este nuevo paradigma de consulta surge inspirado en la rama de Information Retrieval (Manning et

al., 2008), la cual busca encontrar ciertos elementos en un conjunto de documentos haciendo uso de una estructura de datos llamada índice invertido, la cual indica en qué documento se encuentra cada término. El anterior concepto al ser trasladado al entorno de las bases de datos indicará la ubicación en la BD de cada palabra clave.

Ambos tipos de sistemas hacen uso de un índice invertido para localizar los elementos a los que hacen referencia las palabras clave de la consulta. En un sistema basado en grafos las entradas del índice señalarán nodos del grafo de datos. En sistemas basados en redes candidatas, un registro del índice apuntará a una tupla de la base de datos o, como se realiza en Mragyati (Sarda y Jain, 2001) y DBXplorer (Agrawal et al., 2002), a una columna de una tabla. El primer tipo de índice para sistemas basados en redes candidatas sugiere la opción más sencilla de implementar, con la consecuencia de tener índices mucho más grandes que en la otra opción. En cambio, los índices por columnas, donde una entrada del índice invertido asegura la presencia de un término en una columna específica en la base de datos (una o más veces), son más cortos pues ellos resumen a 1 el número de registros de la misma palabra clave en diferentes tuplas de una misma columna. Al proceso de obtener los metadatos de la BD, junto con la creación y llenado del índice invertido, se le conoce como *Publicación de la Base de Datos*.

Una vez publicada la base de datos, un sistema de consulta por palabras clave es capaz de brindar un conjunto de interpretaciones a cada término de la cosulta. Si bien los sistemas basados en grafos únicamente interpretan las palabras clave como registros del índice que apuntan a un nodo del grafo de datos, los sistemas de redes candidadatas también pueden realizar interpretaciones de términos como algún elemento de los metadatos (tablas y atributos).

Después de interpretar las palabras clave se tiene un conjunto de porciones del resultado final, extremos que necesitan ser unificadas en una sola respuesta. Esto se realiza mediante una cadena de operaciones join entre las tablas que contienen dichas interpretaciones. Para construir tal sucesión de operaciones de join es necesario computar el Árbol Steiner mínimo (o red candidata) que contenga todas las tablas donde se encuentran las palabras clave. Concluído lo anterior, el paso final es ensamblar la consulta SQL utilizando la información de las interpretaciones de cada término y de la red candidata o árbol Steiner, según sea el caso.

Sin importar la categoría a la que pertenzca cada sistema, la respuesta al insertar un conjunto de palabras clave en el sistema es un grupo con las supuestas k mejores respuestas, lista de donde se espera que el usuario encuentre la respuesta deseada. A pesar de las ventajas que otorgan este tipo de sistemas, la selección de la respuesta final es delegada al usuario. Sería mucho más natural obtener una única consulta SQL como respuesta, la cual refleje la intención del usuario.

En este contexto surge la motivación de este trabajo: *a partir de una consulta por palabras clave (en*

una base de datos), obtener la consulta SQL esperada por el usuario de manera exacta. Tal escenario plantea retos nuevos en comparación con los sistemas de la literatura, pues el procesamiento de consultas debe considerar más elementos para seleccionar la respuesta deseada que sólo el tamaño de la red candidata o árbol Steiner, como se realiza en la mayoría de estos trabajos.

En este trabajo se presenta *KweryDB*, un sistema que contempla nuevos criterios de evaluación que permiten escoger una interpretación entre todas las posiblidades. La puntuación de las Interpretaciones Candidatas se realiza en dos momentos. La primera calificación es influída por el número de tablas involucradas en la interpretación y las interpretaciones de palabras clave como registros del índice, considerando cuando dos o más términos aluden a la misma entrada del índice.

Este primer filtro sirve para determinar cuáles posibilidades se presumen como las más convenientes, pues los criterios antes mencionados son favorables para identificar coincidencias en las interpretaciones de términos y resultados más cercanos, donde las tablas que contienen a las palabras clave requieren menos operaciones join. Aplicando tal filtro, las interpretaciones Candidatas quedan organizadas y se pueden ignorar las que obtuvieron peores calficaciones para limitar la cantidad de redes candidatas a calcular. La segunda calficación incluye las mismas variables más el tamaño del árbol Steiner, resultando en una evaluación equilibrada entre tamaño de la red canditata y concomitancia de las interpretaciones, y de este modo tener más posibilidades para seleccionar la mejor opción.

Recapitulando, para este trabajo se tomaron elementos de los sistemas revisados, como la estructura base de Mragyati (Sarda y Jain, 2001), un índice invertido por columnas similar al usado en DBXplorer (Agrawal et al., 2002) y algunas otras características originales que se implementaron para mejorar el desempeño, como las métricas de calificación descritas en el párrafo anterior.

Finalmente el sistema propuesto fue sometido a prueba, comparandolo con el sistema más semejante revisado en la literatura: Mragyati (Sarda y Jain, 2001). Los resultados arrojaron resultados satisfactorios, que demuestran una mejor evaluación de las Interpretaciones candidatas en *KweryDB*, al precio de céntimas de segundo extras en el tiempo de procesamiento. A pesar de estos resultados, no es posible afirmar que el sistema responde exactamante a la intención del usuario, existen diferentes patrones en las consultas que no pueden ser procesados correctamente, pues las métricas utilizadas son insuficientes y es necesario poseer información extra que apoye la desambiguación, como el contexto del usuario o la semántica de los datos.

5.2. Contribuciones

El trabajo ha presentado el desarrollo y validación experimental de un sistema de consulta a bases de datos por palabras clave. Este se ha adjudicado la misión de emitir un resultados exactos que reflejen la intención del usuario, en vez del $top-k$ de supuestos mejores resultados planteado en diversos sistemas en la literatura. Con este enfoque, *KweryDB* se inspira en algunos sistemas de la literatura; como Mragyati (Sarda y Jain, 2001) y DBXplorer (Agrawal et al., 2002), de los que se incluyen técnicas destacadas como el índice invertido por columnas o la manera de defininir la interpretación de una palabra clave. Aún después de la inclusión de tales elementos continuaba el problema de cómo escoger la opción más próxima a la intención original de la consulta, por lo que los esfuerzos se concentraron en desarrollar un módulo que suprimiera la salida top-k por la interpretación adecuada de las palabras clave.

En este sentido la mayor contribución del trabajo es el mecanismo que se implementó para desambiguar la consulta. Se plantea el problema siguiente: a partir de un conjunto de Interpretaciones Candidatas, *¿Qué criterios son propicios para seleccionar la opción que responda mejor a la intención del usuario?* y *¿Qué tanto debe influir cada uno de estos criterios?*. Para responder a estas preguntas fue necesario considerar qué tipo de respuestas son más relevantes para distintas combinaciones de palabras clave. A manera de ensayo y error se determinó realizar una evaluación de dos fases con cuatro criterios: en la primera parte se contemplan el número de tablas, de interpretaciones de términos como registros del índice, de coincidencias de interpretaciones (como entradas del índice) o interpretaciones que hagan referencia a la misma tabla, y finalmente para la segunda evaluación se suma el tamaño de la Red Candidata.

Cabe destacar que en ningún trabajo revisado se implementó una función de calificación que tuviera todas estas variables, ni que estuviera repartida en dos momentos diferentes. Lo anterior se debe en gran medida al tipo de salida que estos ofrecen: una lista de la que el usuario escogerá la opción que mejor le parezca. Por lo que hacer una evaluación tan selectiva no serviría de mucho, pues de cualquier manera se deben de mostrar muchos resultados (el número que denote k en cada sistema), el tamaño de la red candidata es suficiente para el paradigma $top-k$.

Además de las aportaciones en el módulo de interpretación y desambiguación, el trabajo comparte los algoritmos de la secuencia del sistema en general, la construcción de consultas SQL a partir de una Interpretación candidata y también el correspondiente a la interpretación de palabras clave y desambiguación de interpretaciones (diríjase a la página 87 para observar los algorimos antes mencionados).

Los resultados obtenidos en la comparación que se realizó con Mragyati (Sarda y Jain, 2001) sugieren que la interpretación implementada en el prototipo de este trabajo es considerablemente mejor que la que se realiza en Mragyati, además de no salir mal librado en las pruebas de tiempos de respuesta. Estos

prometedores indicadores permiten cuestionarse si *KweryDB* será igual de efectiva al evaluarse contra sistemas más sofisticados y con bases de datos más grandes.

5.3. Trabajos Futuros

En primer lugar es necesario completar la validación del prototipo con otros trabajos de búsqueda en bases de datos por palabras clave; sean estos basados en grafos, en redes candidatas (como en este trabajo) o de otra clasificación, ya que las pruebas únicamente se realizaron comparando *KweryDB* con Mragyati, el sistema más sencillo de los revisados de la literatura. Para poder decir que el trabajo ha mejorado al menos parcialmente a los trabajos anteriores es indispensable que esta evaluación se realice. También sería conveniente examinar el desempeño del sistema con una base de datos más grande.

Por otro lado, el sistema está sumamente recortado en torno a flexibilidad, por ej. no existe soporte de funciones de agregación como promedio, suma, mayor, etc. Incluir en la propuesta funciones de agregación permitiría la construcción de consultas más complejas, como propone SQAK (Tata y Lohman, 2008). Además, al hacer referencia a los metadatos es necerario expresar el nombre de la tabla o atributo tal como se encuentra en la BD. Lo anterior es negativo ya que el sistema debería ser capaz de recibir sinónimos o conceptos relacionados al metadato objetivo, por ejemplo usar *clients* en vez de *customers* para la base de datos que se utilizó en la descripción del sistema. Una posibilidad es consultar un tesáuro o algún otra estructura que brinde semántica a la consulta antes de ser interpretada.

Concluyendo este documento es importante recalcar que el objetivo no se cuplió totalmente, la evaluación de dos fases planteada en el sistema no fue capaz de encontrar la mejor respuesta en algunas consultas donde las interpretaciones cantidatas obtuvieron el mismo puntaje, se obligó al sistema a seleccionar una de éstas aleatoriamente. Esto se debe a que la información de la consulta en sí misma, del esquema de base de datos y del contenido de la misma no son suficientes en algunos casos; se necesita de información extra que permita asegurar responder satisfactoriamente a la inquietud del usario.

Bajo esta óptica, es interesante contemplar técnicas de otras ramas de la computación para agregar elementos que describan mejor la intención del usuario. Por la naturaleza del objetivo, la consciencia del contexto se presenta como una posible solución a encontrar la verdadera intención del usuario, comenzado por evaluar la situación actual del usuario o usuarios, las actividades que éste realiza o tiene planeado realizar, su ubicación, etc. Existen diversos trabajos relacionados que podrían apoyar a incluir consciencia contextual en *KweryDB* como un criterio más de evaluación para la desambiguación.

Parte I

Apéndices

Apéndice A

Consultas y procedimientos SQL

A.1. Consultas para obtener metadatos

- **Tablas:**

```
SELECT table_name FROM INFORMATION_SCHEMA.TABLES WHERE
   table_type = 'BASE TABLE' AND table_schema = 'base
   de datos';
```

- **Atributos:**

```
SELECT column_name, data_type,character_maximum_length
   FROM INFORMATION_SCHEMA.COLUMNS WHERE table_name = '
   tabla' AND table_schema='base de datos';
```

- **Restricciones de Llave Primaria:**

```
SELECT table_name,column_name FROM INFORMATION_SCHEMA.
   KEY_COLUMN_USAGE WHERE constraint_name='PRIMARY'
   CONSTRAINT_SCHEMA='base de datos';
```

- **Restricciones de Llave Foránea:**

```
SELECT table_name,column_name,constraint_name,
   referenced_table_name,referenced_column_name FROM
   INFORMATION_SCHEMA.KEY_COLUMN_USAGE WHERE
   referenced_table_name = 'tabla' AND
   constraint_schema='base de datos';
```

- **Número de tuplas en la base de datos:**

```
SELECT SUM(TABLE_ROWS) FROM INFORMATION_SCHEMA.TABLES
   WHERE TABLE_SCHEMA = 'base de datos';
```

A.2. Índice invertido

- **Crear índice**

```
CREATE TABLE indice (id INT UNSIGNED AUTO_INCREMENT NOT NULL
   PRIMARY KEY, keyword VARCHAR(80),tabla VARCHAR(40),atrib
   VARCHAR(40),FULLTEXT (keyword));
```

- **Procedimiento almacenado agregar**

```
DROP PROCEDURE IF EXISTS agregar;

CREATE PROCEDURE agregar(IN palabra varchar(80),IN tabla
   varchar(40),IN atributo varchar(40))
BEGIN
        DECLARE finished INTEGER DEFAULT 0;
        DECLARE id INTEGER DEFAULT 0;
        DECLARE kw varchar(80) DEFAULT '';
        DECLARE tab varchar(40) DEFAULT '';
        DECLARE atrib varchar(40) DEFAULT '';

        DECLARE un_cursor CURSOR FOR SELECT * FROM indice WHERE
            indice.keyword=palabra;
        DECLARE CONTINUE handler FOR NOT FOUND SET finished =
           1;

        OPEN un_cursor;
        get_kw: LOOP
                FETCH un_cursor INTO id,kw,tab,atrib;
                IF finished = 1 THEN
                        LEAVE get_kw;
                END IF;

                IF palabra=kw and tabla=tab and atributo=atrib
                   THEN
                        LEAVE get_kw;
                END IF;
        END LOOP get_kw;

        IF finished=1 THEN
                INSERT INTO indice (keyword,tab,atrib) values (
                   palabra,tabla,atributo);
    END IF;

    CLOSE un_cursor;
END
```

Apéndice B

Algoritmos

Algoritmo 5: Generación de Consulta SQL

Entrada: Interpretación Candidata con la mejor puntuación *ie*

Salida: Consulta SQL

```
1  SQL ← "SELECT" ;                                   // cadena donde se pegarán los de SQL
2  FK ← " WHERE ";                                    // segmento predicados FK-PK
3  TB ← "" ;                                          // seg. tablas
4  VAL ← "" ;                                         // seg. predicados valores
5  COL ← "" ;                                         // seg. proyección columnas
6  si número de tablas involucradas |TI(ie)| = 1 entonces
7  |  agregar t ∈ TI(ie) a TJ;
8  |  TB ← i.tabla + i.tabla.num;
9  en otro caso
10 |  TJ ← {} ;                                       // tablas de la cadena de joins
11 |  para cada pareja de tablas conectadas {t_i, t_j} ∈ J(Red Candidata(ie)) hacer
12 |  |  buscar restricción FK-PK(t_i, t_j);
13 |  |  agregar t_i a TJ;
14 |  |  predicado ← "t" + t_i.num + "." + t_i.nombre_restricción ;
15 |  |  predicado ← predicado + "= t" + t_j.num + "." + t_j.nombre_restricción ;
   |  |  /* Esto crea cadenas de predicados, por ejemplo:
   |  |     "t1.salesRepEmployeeNumber=t2.employeeNumber"                          */
16 |  |  si FK ≠ " WHERE" entonces
   |  |  |  /* ya se incluyeron predicados para rest. FK-PK                        */
17 |  |  |  FK ← FK + " AND ";
18 |  |  fin
19 |  |  FK ← FK + predicado;
20 |  fin
   |  /* agregar tablas a la cadena TB                                            */
21 |  para cada t ∈ TJ hacer
22 |  |  TB ← TB + t.nombre + t.num;
23 |  |  si |TJ| > 1 y no es la última tabla del conjunto entonces
24 |  |  |  TB ← TB + ",";
25 |  |  fin
26 |  fin
27 fin
   /* Por cuestiones de espacio se omite del algoritmo la sección que se encarga de mejorar la
      proyección de columnas                                                       */
28 PV ← {} ;                                          // Conjunto que almacena los predicados de valores
29 para cada interpretación {i ∈ ie : tipo(i) = valor del índice} hacer
30 |  predicado ← "t" + i.tabla.num + "." + i.tabla.nombre + "." + i.columna.nombre = i.valor;
31 |  agregar {{i.tabla.nombre, i.columna.nombre}, predicado} a PV;
32 fin
33 revisados ← {};
34 para cada {{x,y}, predicado} ∈ PV hacer
35 |  VAL ← VAL + predicado;
36 |  si {x,y} ∈ revisados entonces                  /* si hay más de un predicado para la misma tabla y columna */
37 |  |  VAL ← VAL + "OR";
38 |  en otro caso
39 |  |  VAL ← VAL + "AND";
40 |  |  agregar {x,y} a revisados;
41 |  fin
42 fin
43 SQL ← SQL + COL + FK + VAL ;                       // Juntar los segmentos para tener una consulta SQL completa
44 devolver SQL;
```

Apéndice C

Desarrollo de ejemplo *“1969 camaro customes usa”* desglozado

C.1. Primera Evaluación

	Interpretación de Palabras Clave				Calificaciones			
	1969	camaro	customers	usa	tablas	índice	tuplas BD	total
1	"1969 Corvair Monza" *0.66*	"1969 Chevrolet Camaro Z28" *0.75*	customers	"USA" *0.001 c*	0.1875	0.1618	0.2427	0.592
2	"1969 Corvair Monza" *0.66*	"1969 Chevrolet Camaro Z28" *0.75*	customers	"USA" *0.001 o*	0.0937	0.1618	0.2427	0.4982
3	"1969 Corvair Monza" *0.66*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001 c*	0.1875	0.1674	0.2512	0.6061
4	"1969 Corvair Monza" *0.66*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001 o*	0.0937	0.1674	0.2512	0.5123
5	"1969 Ford Falcon" *0.66*	"1969 Chevrolet Camaro Z28" *0.75*	customers	"USA" *0.001 c*	0.1875	0.1618	0.2427	0.592
6	"1969 Ford Falcon" *0.66*	"1969 Chevrolet Camaro Z28" *0.75*	customers	"USA" *0.001 o*	0.0937	0.1618	0.2427	0.4982
7	"1969 Ford Falcon" *0.66*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001 c*	0.1875	0.1674	0.2512	0.6061
8	"1969 Ford Falcon" *0.66*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001 o*	0.0937	0.1674	0.2512	0.5123
9	"1969 Dodge Charger" *0.66*	"1969 Chevrolet Camaro Z28" *0.75*	customers	"USA" *0.001 c*	0.1875	0.1618	0.2427	0.592
10	"1969 Dodge Charger" *0.66*	"1969 Chevrolet Camaro Z28" *0.75*	customers	"USA" *0.001 o*	0.0937	0.1618	0.2427	0.4982
11	"1969 Dodge Charger" *0.66*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001 c*	0.1875	0.1674	0.2512	0.6061
12	"1969 Dodge Charger" *0.66*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001 o*	0.0937	0.1674	0.2512	0.5123
13	"1969 Dodge Super Bee" *0.75*	"1969 Chevrolet Camaro Z28" *0.75*	customers	"USA" *0.001 c*	0.1875	0.1562	0.2343	0.578
14	"1969 Dodge Super Bee" *0.75*	"1969 Chevrolet Camaro Z28" *0.75*	customers	"USA" *0.001 o*	0.0937	0.1562	0.2343	0.4842
15	"1969 Dodge Super Bee" *0.75*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001 c*	0.1875	0.1618	0.2427	0.592
16	"1969 Dodge Super Bee" *0.75*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001 o*	0.0937	0.1618	0.2427	0.4982
17	*"1969 Chevrolet Camaro Z28" 0.75*	*"1969 Chevrolet Camaro Z28" 0.75*	*customers*	*"USA" 0.001 c*	*0.1875*	*0.1562*	*0.3046*	**0.6483**
18	"1969 Chevrolet Camaro Z28" *0.75*	"1969 Chevrolet Camaro Z28" *0.75*	customers	"USA" *0.001 o*	0.0937	0.1562	0.3046	0.5545
19	"1969 Chevrolet Camaro Z28" *0.75*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001 c*	0.1875	0.1618	0.2427	0.592
20	"1969 Chevrolet Camaro Z28" *0.75*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001 o*	0.0937	0.1618	0.2427	0.4982

Tabla C.1: Primera Evaluación de interpretaciones para la consulta *"1969 camaro customers usa"*

C.2. Generación de Redes Candiatas

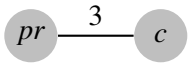
1. $TI = \{customers, products\}$

   a) **Caminos más Cortos**

t1	t2	Acotación	Ruta	Distancia
customers	products	c-pr	c-or-od-pr	3

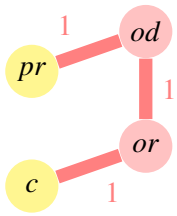
Tabla C.2: Caminos más cortos de $TI = \{customers, products\}$

   b) **Grafo Resumido**



   c) **Árbol de Spanning Mínimo**

   d) **Sustitución por caminos más cortos**



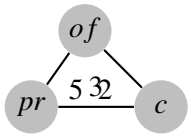
2. $TI = \{customers, products, offices\}$

a) **Caminos más Cortos**

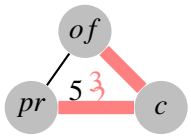
t1	t2	Acotación	Ruta	Distancia
customers	products	c-pr	c-or-od-pr	3
customers	offices	c-of	c-e-of	2
products	offices	pr-of	pr-od-or-c-e-o	5

Tabla C.3: Caminos más cortos de $TI = \{customers, products, offices\}$

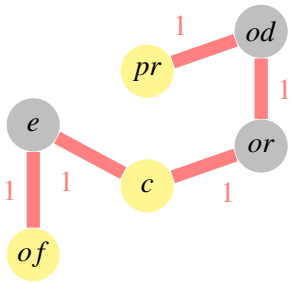
b) **Grafo Resumido**



c) **Árbol de Spanning Mínimo**



d) **Sustitución por caminos más cortos**



C.3. Segunda Evaluación y Selección de Interpretación

	Interpretación de Palabras Clave				Calificaciones				
	1969	camaro	customers	usa	tablas	índice	tuplas BD	long(rc)	total
1	"1969 Corvair Monza" *0.66*	"1969 Chevrolet Camaro Z28" *0.75*	customers	"USA" *0.001* **c**	0.12	0.103	0.155	0.08	0.458
2	"1969 Corvair Monza" *0.66*	"1969 Chevrolet Camaro Z28" *0.75*	customers	"USA" *0.001* **o**	0.06	0.103	0.155	0.04	0.358
3	"1969 Corvair Monza" *0.66*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001* **c**	0.12	0.107	0.1608	0.08	0.468
4	"1969 Corvair Monza" *0.66*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001* **o**	0.06	0.107	0.160	0.04	0.368
5	"1969 Ford Falcon" *0.66*	"1969 Chevrolet Camaro Z28" *0.75*	customers	"USA" *0.001* **c**	0.12	0.103	0.155	0.08	0.458
6	"1969 Ford Falcon" *0.66*	"1969 Chevrolet Camaro Z28" *0.75*	customers	"USA" *0.001* **o**	0.06	0.103	0.155	0.04	0.358
7	"1969 Ford Falcon" *0.66*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001* **c**	0.12	0.107	0.160	0.08	0.468
8	"1969 Ford Falcon" *0.66*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001* **o**	0.06	0.107	0.160	0.04	0.368
9	"1969 Dodge Charger" *0.66*	"1969 Chevrolet Camaro Z28" *0.75*	customers	"USA" *0.001* **c**	0.12	0.103	0.155	0.08	0.458
10	"1969 Dodge Charger" *0.66*	"1969 Chevrolet Camaro Z28" *0.75*	customers	"USA" *0.001* **o**	0.06	0.103	0.155	0.04	0.358
11	"1969 Dodge Charger" *0.66*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001* **c**	0.12	0.1072	0.1608	0.0800	0.468
12	"1969 Dodge Charger" *0.66*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001* **o**	0.06	0.107	0.160	0.04	0.368
13	"1969 Dodge Super Bee" *0.75*	"1969 Chevrolet Camaro Z28" *0.75*	customers	"USA" *0.001* **c**	0.12	0.099	0.2259	0.08	0.525
14	"1969 Dodge Super Bee" *0.75*	"1969 Chevrolet Camaro Z28" *0.75*	customers	"USA" *0.001* **o**	0.06	0.099	0.2259	0.04	0.425
15	"1969 Dodge Super Bee" *0.75*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001* **c**	0.12	0.103	0.225	0.08	0.528
16	"1969 Dodge Super Bee" *0.75*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001* **o**	0.06	0.103	0.225	0.04	0.428
17	"1969 Chevrolet Camaro Z28" *0.75*	"1969 Chevrolet Camaro Z28" *0.75*	customers	"USA" *0.001* **c**	0.12	0.099	0.194	0.08	0.494
18	"1969 Chevrolet Camaro Z28" *0.75*	"1969 Chevrolet Camaro Z28" *0.75*	customers	"USA" *0.001* **o**	0.06	0.099	0.194	0.04	0.394
19	"1969 Chevrolet Camaro Z28" *0.75*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001* **c**	0.12	0.103	0.155	0.08	0.458
20	"1969 Chevrolet Camaro Z28" *0.75*	"1982 Camaro Z28" *0.66*	customers	"USA" *0.001* **o**	0.06	0.103	0.155	0.04	0.358

Tabla C.4: Primera Evaluación de interpretaciones para la consulta *"1969 camaro customers usa"*

Bibliografía

AGRAWAL, S., CHAUDHURI, S. y DAS, G. Dbxplorer: A system for keyword-based search over relational databases. En *Data Engineering, 2002. Proceedings. 18th International Conference on*, páginas 5–16. IEEE, 2002.

BHALOTIA, G., HULGERI, A., NAKHE, C., CHAKRABARTI, S. y SUDARSHAN, S. Keyword searching and browsing in databases using banks. En *Data Engineering, 2002. Proceedings. 18th International Conference on*, páginas 431–440. IEEE, 2002.

CHAUDHURI, S. y DAS, G. Keyword querying and ranking in databases. *Proceedings of the VLDB Endowment*, vol. 2(2), páginas 1658–1659, 2009.

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., STEIN, C. ET AL. *Introduction to algorithms*, vol. 2. MIT press Cambridge, 2001.

DOE, R. Mysql sample database. 2015.

ELMASRI, R. *Fundamentals of database systems*. Pearson Education India, 2008.

HRISTIDIS, V. y PAPAKONSTANTINOU, Y. Discover: Keyword search in relational databases. En *Proceedings of the 28th international conference on Very Large Data Bases*, páginas 670–681. VLDB Endowment, 2002.

HUHTALA, Y., KÄRKKÄINEN, J., PORKKA, P. y TOIVONEN, H. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal*, vol. 42(2), páginas 100–111, 1999.

KOU, L., MARKOWSKY, G. y BERMAN, L. A fast algorithm for steiner trees. *Acta informatica*, vol. 15(2), páginas 141–145, 1981.

LI, G., FENG, J., ZHOU, X. y WANG, J. Providing built-in keyword search capabilities in rdbms. *The VLDB Journal*, vol. 20(1), páginas 1–19, 2011.

LI, G., OOI, B. C., FENG, J., WANG, J. y ZHOU, L. Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. En *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, páginas 903–914. ACM, 2008.

LI, Y., YANG, H. y JAGADISH, H. Nalix: A generic natural language search environment for xml data. *ACM Transactions on Database Systems (TODS)*, vol. 32(4), página 30, 2007.

MANNING, C. D., RAGHAVAN, P. y SCHÜTZE, H. *Introduction to information retrieval*, vol. 1. Cambridge university press Cambridge, 2008.

ORSI, G., TANCA, L. y ZIMEO, E. Keyword-based, context-aware selection of natural language query patterns. En *Proceedings of the 14th International Conference on Extending Database Technology*, páginas 189–200. ACM, 2011.

SARDA, N. L. y JAIN, A. Mragyati: A system for keyword-based searching in databases. *arXiv preprint cs/0110052*, 2001.

SAYYADIAN, M., LEKHAC, H., DOAN, A. y GRAVANO, L. Efficient keyword search across heterogeneous relational databases. En *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, páginas 346–355. IEEE, 2007.

TATA, S. y LOHMAN, G. M. Sqak: doing more with keywords. En *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, páginas 889–902. ACM, 2008.