UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

POSGRADO EN CIENCIA E INGENIERIA
DE LA COMPUTACION

"TECNICAS PARA LA RECUPERACION DE INFORMACION Y BUSQUEDA DE TEXTO EN BASES DE DATOS RELACIONALES"

T E S I S

QUE PARA OBTENER EL GRADO DE

MAESTRA EN CIENCIAS

P R E S E N T A :

JUDITH JARAMILLO LOPEZ

DIRECTORA DE TESIS: DRA. AMPARO LOPEZ GAONA

MEXICO, D.F. JUNIO DEL 2001

Agradecimientos

Este trabajo no hubiera sido posible concluirlo sin un gran equipo de ángeles que estuvieron a mí alrededor, son los CONSTRAINTS y los TRIGGERS de mi base de datos personal. Con mi más profunda gratitud y sin definir un ORDER BY o una Prioridad:

A la Dra. Amparo López Gaona, por su apoyo y confianza.
Muchas gracias por su tiempo, consejos y amistad.

A mis padres, por 31 años de amor y
enseñarme todos los valores que me han formado.

A mis hermanos, por su cariño,
apoyo e insistencia en que llegara a la meta.

A *pachihuis*, por su ejemplo de lealtad y amor
en mis mejores y peores momentos.

A *"mis cómplices"* en el camino de este sueño hoy hecho realidad.
Por sus consejos, apoyo, ánimo, recopilación de información, y confianza en mí.
¡ Gracias Ismael, Armando, César, Oliver, *Memorandum*,
Ricardo, Dora, Lulú, Jesús, Carlos y Sergio ! ¡Gracias amigos!

A Jesús de Nazareth, por acompañarme en los retos
que me impongo y en los que como él, no se pueden decidir.

A la UNAM, por acogerme entre sus aulas y ser parte de su espíritu.

Al CONACYT, por apoyar el costo de mi anhelo.

JUDITH

Dedicatoria

A JJL ...

Por ser fiel a tus convicciones.

CONTENIDO

Capítulo III Técnicas para búsqueda de texto

Capítulo IV Máquinas de búsqueda en los principales RDBMS

PROLOGO

En el mundo actual, el acceso a la información es crucial; pero la información es más que datos estructurados en bases de datos relacionales; el 90 % de la información digital es texto, y en todas sus formas, representa un recurso enorme y valioso de información. Sin embargo, contar con ese recurso e incorporarlo con los datos estructurados ha sido en gran medida una meta no alcanzada y es preciso que se tengan soluciones para atender a todas las necesidades que se presentan, sobre todo con la popularidad del Web.

Mientras que robustas bases de datos relacionales basadas en el SQL fueron usadas para datos estructurados, máquinas de recuperación fueron usadas para el texto. Estas tecnologías se integran como ambientes costosos y complicados; el texto almacenado en las bases de datos relacionales no es buscado de forma nativa; se le ha almacenado, accesado y manipulado de manera *ad hoc*, desperdiciándose así las bondades de las bases de datos (seguridad, consistencia, concurrencia, integridad, etc.) Esto no sólo es inconveniente para los desarrolladores de una aplicación, sino también para los administradores de la base de datos que se encuentran ante la disyuntiva de administración de datos o de texto. Así, la difícil tarea de optimización de consultas y los beneficios de la transparencia de la base de datos son imposibles de lograr.

En este trabajo se presenta un estudio teórico de la búsqueda y recuperación de información que se ve aplicado al texto en bases de datos y cómo han presentado algunas soluciones los principales productos comerciales. El interés surge porque una vez que el modelo relacional de bases de datos ha llegado a su madurez y plenitud, dando pie a un nuevo modelo como lo es el orientado a objetos, aún prevalece en el mundo actual y conociendo las bondades que ofrecen algunos sistemas manejadores de bases de datos en cuanto a la manipulación, consulta y administración de datos, es impactante encontrarse con "las cosas que no se pueden hacer". Teniendo entre esto, el almacenamiento, indexación, búsqueda y recuperación de texto basada en su forma y contenido, que muchas veces depende del lenguaje natural. Se analizan algunos productos comerciales de tratamiento de texto disponibles para los principales ORDBMS como *DB2, Informix* y *Oracle*. Partiendo de que dichos productos son un complemento del manejador de bases de datos y trabajan con un SQL extendido sobre ellos, se presenta un análisis de las capacidades de cada herramienta en cuanto a sus tipos de consultas, recursos lingüísticos y modelos que al parecer, proponen una alternativa a los métodos clásicos como las consultas booleanas y los espacios vectoriales.

El mundo, necesita una solución unificada, basada en estándares y escalable para administrar información de cualquier tipo. Su almacenamiento, acceso, análisis y distribución, son algunos de los desafíos a los que nos enfrentamos los involucrados en las ciencias de la computación y que debemos marcar la pauta para la construcción de este software.

Capítulo I Limitantes de los RDBMS para la recuperación de texto

En este capítulo se presenta el estado actual de las bases de datos relacionales, su importancia y su manera de tratar el texto, visto como un tipo de dato no estructurado y los problemas que se presentan para el almacenamiento, la búsqueda y recuperación del mismo con el SQL. Se resaltan los problemas semánticos que se presentan así como la creación de índices sobre este tipo de datos que limitan a miles de necesidades en las aplicaciones actuales.

Capítulo II Teoría de la recuperación de información

En este capítulo se presenta la teoría que soporta la búsqueda y recuperación de información. Se hace un planteamiento de los modelos clásicos y sus tendencias; el procesamiento del texto y los distintos tipos de consultas. Por último se definen los parámetros de evaluación para un sistema de recuperación de información.

Capítulo III Técnicas para búsqueda de texto

En este capítulo se presenta un recorrido por las diferentes técnicas que se han hecho para llevar a cabo el tratamiento, la indexación y búsqueda de patrones de texto. Es así como se aplica la teoría planteada del capítulo II.

Capítulo IV Máquinas de búsqueda en los principales RDBMS

En este capítulo se presentan algunos productos comerciales con sus propuestas para solucionar las búsquedas de texto por contenido, lematización, sinonimia, búsquedas difusas, por formatos y otras consultas sofisticadas. Estos productos son extensiones sobre los manejadores de bases de datos; las herramientas analizadas en su funcionalidad son, por parte de *DB2 (IBM): **Text Extender;*** de *IDS (Informix): **Excalibur Text Search DataBlade Module*** y ***Verity Text Search DataBlade Module;*** y por parte de *Oracle8 (Oracle): **ConText Cartridge***. Se hace una reflexión sobre los límites de sus capacidades comparándolos entre sí.

Capítulo I

Limitantes de los RDBMS para la recuperación de texto

" Today, if you have a well-designed management system, you have the keys to the kingdom of data processing and decision support. That is why there now exists a prototype machine whose complete design is based on the relational model. In fact, the old term 'computer system' now seems like a misnomer "

Edgar F. Codd

1. 1 Antecedentes de los Sistemas Manejadores de Bases de Datos Relacionales

La complejidad de las organizaciones ha demandado eficiencia en la utilización de datos y generación de información. Gracias al uso de las computadoras ha sido posible manejar enormes cantidades de datos con mayor rapidez y precisión. A la par, se han dado cambios en las estructuras de datos empleadas, en las técnicas destinadas a la explotación de la información, así como en las interfaces de las aplicaciones.

La evolución en el manejo de grandes volúmenes de datos surge a principios de los años 60's con el comité **CODASYL** (*Conference Of DAta Systems Languages*), que representaba a fabricantes de computadoras, agencias gubernamentales, organizaciones de usuarios y universidades; este comité preparó el marco de trabajo de **COBOL** (*COmmon Business Oriented Language*), lenguaje que fue diseñado específicamente para el procesamiento de los datos de tipo comercial. La estructura de datos predominante fue el ***archivo***.
Los archivos estaban por lo general, diseñados para una aplicación determinada o para un grupo de aplicaciones muy similares. Era posible el acceso secuencial y el acceso directo a los registros.
Con este software se proporcionaban métodos de acceso, pero no una administración de datos.

Al surgir la necesidad de aplicaciones más complejas, se observó la necesidad de agregar al compilador de **COBOL** paquetes que facilitaran el ordenamiento y clasificación de datos así como la generación de reportes. Surgieron también las organizaciones lógicas de alto nivel para los datos, y las aplicaciones comenzaron a integrarse para ponerse a disposición de un mayor número de usuarios. En 1971, el **DBTG** (*Data Base Task Group*), un subgrupo de *CODASYL*, presentó un documento acerca de las bases de datos, en el cual quedaron asentados los principios para el desarrollo de lo que serían los **DBMS** (*Data Base Management Systems*).

Durante las últimas décadas, las bases de datos han jugado un papel importante en el manejo de grandes volúmenes de información; hardware y software (en este caso representado por los DBMS), han tenido que evolucionar para poder manipular las nuevas estructuras de datos y atender las cada vez más complejas demandas.
Es así como se han presentado cronológicamente los diversos modelos de bases de datos a través de los años: **El *Jerárquico*, el de *Red*, el *Relacional* y el *Orientado a Objetos***. En los dos primeros modelos, se tenía un código de la aplicación totalmente dependiente a la implementación de las estructuras de datos. El programador debía trabajar con ligas físicas tanto en la construcción como en la explotación de datos. Esto generó un lento desarrollo y un alto costo en el mantenimiento. Además, se tenían grandes limitantes en la modificación de las estructuras.

El modelo relacional prometía resolver estos problemas y proporcionar otros beneficios de negocios.

1.2 El Modelo Relacional

El Modelo Relacional tuvo origen en las investigaciones del Dr. Edgar F. Codd quien publicó en los años 1969 y 1970 sus artículos: "*Derivability, Redundancy, and Consistency of Relations stored in Large Data Banks*" [Codd69] y "*A Relational Model of Data for Large Shared Data Banks*" [Codd70], colocando un nuevo modelo para las bases de datos en una posición firme, robusta, sencilla y bien documentada. El modelo relacional tiene sus bases matemáticas en la lógica de predicados de primer orden y la teoría de las relaciones [Codd90]. De su idea, surgen *de jure*, los primeros RDBMS (Relational Data Base Management Systems) comerciales como el legendario *System R* y los actuales *DB2(IBM), Informix Dynamic Server (Informix), Ingres, Oracle8 (Oracle), Progress* y *Sybase* .

Un modelo de datos es la combinación de al menos, tres componentes:

(1) Una colección de tipos de estructuras de datos;
(2) Una colección de operadores o reglas de inferencia, los cuales pueden ser aplicados a cualquier instancia válida de los tipos de datos de (1) para recuperar, derivar o modificar datos de cualquier parte de la estructura sobre cualquier combinación deseada;
(3) Una colección de reglas generales de integridad, las cuales definen (explícita o implícitamente) el conjunto de estados consistentes, los cambios de estado de la base de datos o ambos. Estas reglas son generadas en el sentido de que se aplican a cualquier base de datos usando este modelo.

El modelo relacional es un modelo de datos en este sentido y fue el primero en ser definido como tal. En su parte estructural, consta de dominios, relaciones de un grado determinado (con tablas como su principal representación conceptual), tuplas, llaves candidatas y llaves primarias.
La parte de manipulación del modelo está dada por los operadores algebraicos que permiten transformar las relaciones en nuevas relaciones.
La integridad se define por medio de dos reglas de integridad: Integridad de entidades e integridad referencial. [Codd82]

A continuación, se presenta la terminología de la teoría relacional:

Una *Relación* sobre un conjunto de dominios D_1, D_2, ..., D_n (no necesariamente distintos), se compone por el subconjunto del producto cartesiano de los dominios: $R \subseteq D_1 X D_2 X \ldots X D_n$ y tendrá un *grado n*.
La relación se define como único elemento de estructuración y se le asocia un nombre.
Las relaciones están organizadas como conjuntos de *tuplas* de la forma $< d_1, d_2, \ldots, d_n >$.
Cada dominio tiene un tipo de dato básico asociado.

Tabla es la representación tabular de una relación, está formada por renglones y columnas, donde cada columna representa un *atributo* y los valores de los codominios aparecen como entradas (*instancias*) en la tabla.

El *grado* de una relación es el número de dominios que forman la tupla de dicha relación, incluyendo el caso especial de una relación unaria (donde el número de atributos = 1).
La *cardinalidad* es el número de tuplas en la relación.

El Modelo Relacional es una extensión de la teoría matemática de las relaciones para adaptarla a los objetivos del procesamiento de datos y se basa en colecciones de tablas con propiedades especiales, que permitan representar los distintos tipos de asociaciones (1:1, 1:M y M:N) por medio de las técnicas de mapeo.

En la tabla 1.1 se muestran las equivalencias de términos matemáticos y de bases de datos relacionales.

Término matemático	***Término de Bases de Datos***
Relación R de grado n	Tabla-R con n columnas
Atributo	Columna de la tabla-R
Dominio	Tipo de dato extendido
Tupla	Renglón de la tabla-R
Cardinalidad de la relación	Número de renglones en la tabla-R

Tabla 1.1 Equivalencias entre terminologías

El Modelo Relacional trata con las tuplas mediante su contenido de información, no por medio de identificadores, números, etiquetas o direcciones; se logra así, la independencia de las estructuras físicas de las lógicas que se presentaba en los modelos anteriores.

1.2 .1 El Álgebra Relacional y el Cálculo Relacional

El conjunto de operaciones definidas a continuación, fue introducido por E. F. Codd y se aplican específicamente a las relaciones [Codd72]. El ***álgebra relacional*** está formada por un conjunto de operadores de alto nivel que operan sobre relaciones. Cada uno de estos operadores toma una o dos relaciones como entrada y produce una nueva como salida manteniéndose así, la *propiedad de cerradura*. Codd definió un conjunto de ocho operadores de este tipo, divididos en dos grupos:

- **Las operaciones tradicionales de conjuntos.**

Unión $R_1 \cup R_2$

Este operador acepta como entradas dos relaciones con los mismos atributos en el mismo orden, y produce como resultado todos los atributos y todas las tuplas de ambas relaciones. Si existe alguna tupla con la misma información en ambas relaciones, en la relación resultado, sólo aparece una vez.

Intersección $R_1 \cap R_2$

Este operador selecciona de ambas relaciones de entrada, aquellas tuplas que tengan la misma información en todos los atributos, generando así la relación resultante.

Diferencia $R_1 - R_2$

Acepta como entrada dos relaciones que tengan al menos un atributo en común, en donde la relación resultante tendrá todas las tuplas de la primera relación que no aparezcan en la segunda relación.

Producto Cartesiano $R_1 \times R_2$

A partir de dos relaciones especificadas, se genera una tercera que contiene todas las combinaciones posibles de tuplas, una de cada una de las dos relaciones.

- **Las operaciones relacionales especiales.**

Proyección $\Pi(d_1 [, d_2, \ldots, d_n])R_1$

Este es un operador unario que tiene como entrada una relación y produce como resultado sólo aquellos dominios especificados, alterando así, el grado de la relación de entrada con respecto a la resultante. El orden en el cual aparecen los dominios, es el que se indica cuando se hace la proyección, permitiéndose así la permutación de los mismos.
El número máximo de dominios que se puede proyectar, es el grado que tiene la relación y como mínimo solo uno.

Selección o Restricción $\sigma_{(condición)} R_1$

Este es también un operador unario que tiene como entrada una relación y produce como resultado los mismos atributos que contiene la relación de entrada y las tuplas que sean especificadas, es decir, las que cumplan el criterio definido. Se altera así, la cardinalidad de la relación de entrada con respecto a la resultante.
Las condiciones de selección de tuplas pueden tener varios grados de complejidad y pueden incluir a los operadores booleanos AND, OR y NOT.

Join o Junta $R_1 \Theta R_2 [\Theta R_3 \Theta R_4 \ldots]$

El operador de join acepta como entrada dos o más relaciones, teniendo cada una al menos un atributo en común con las otras relaciones, y produce como resultado una nueva relación con todos los dominios de las relaciones originales, las tuplas se seleccionan con aquellas cuyas instancias en las relaciones de entrada cumplen la condición que se indica para hacer la junta.
Los operadores relacionales para indicar las condiciones de join pueden ser: >, <, =, !=, <= y >=. Los atributos en común, sólo se muestran una vez.

División $R_1 : R_2$

Toma dos relaciones, una binaria y una unaria, y construye una relación formada por todos los valores de un dominio de la relación binaria que concuerden (en el otro dominio) con todos los valores en la relación unaria.

El cálculo relacional fue propuesto por Codd en 1971 como alternativa al álgebra. La diferencia entre un lenguaje algebraico y un lenguaje predicativo (denominado así por el uso del cálculo de predicados para la formulación de consultas), es que en el primero hay que especificar qué operadores se tienen que aplicar a las relaciones para obtener el resultado, mientras que en el segundo, sólo es preciso indicar cuál es el resultado que se quiere obtener, expresándolo mediante el cálculo de predicados de primer orden.

Chris J. Date diferencia el álgebra del cálculo de la siguiente manera: "Los lenguajes basados en el cálculo relacional son *descriptivos*, mientras que los algebraicos son *prescriptivos*" [Date01].
Sin embargo, las distinciones anteriores son solo superficiales, ya que es un hecho que el álgebra y el cálculo son lógicamente equivalentes. Para cada expresión algebraica hay una equivalente del cálculo, y viceversa.

Codd mostró que el álgebra es al menos tan poderosa como el cálculo. Esto lo hizo dando un algoritmo conocido como "*El algoritmo de reducción de Codd*" [Codd72], mediante el cual, una expresión arbitraria del cálculo podía ser reducida a una expresión del álgebra semánticamente equivalente.

Es así como se presenta el modelo relacional, basado en una sencilla y a la vez firme teoría matemática que si para muchos representó solo una forma elegante de especificar un nuevo modelo de bases de datos, es sin duda el fundamento de la tecnología moderna de ese campo.

E. F. Codd recibió el *ACM Turing Award* en 1981 por su trabajo sobre el modelo relacional y con ese motivo presentó un artículo titulado "*Relational Database: A Practical Foundation for Productivity*" donde presenta evidencias para sugerir que la solución para atacar el problema de la alta demanda de aplicaciones de cómputo contra el atraso de los departamentos de sistemas para satisfacer esa demanda, la ofrece la tecnología relacional. [Codd82]

"... [el modelo] relacional es en verdad diferente. Es diferente porque no es ad hoc. Los sistemas anteriores fueron ad hoc; tal vez hayan ofrecido soluciones a ciertos problemas importantes del momento, pero no estaban apoyados por una base teórica sólida. En contraste, los sistemas relacionales sí están apoyados por esa base ... lo que significa que son sólidos como la roca.

Gracias a esta base sólida, los sistemas relacionales se comportan en formas bien definidas; y (tal vez sin darse cuenta de ese hecho) los usuarios tienen en mente un modelo de comportamiento sencillo, uno que les permite predecir de manera confiable lo que hará el sistema en una situación dada. No hay (o no debe haber) sorpresas. El que sean predecibles significa que las interfaces de usuario son fáciles de entender, documentar, enseñar, utilizar y recordar."

Chris J. Date [Date90]

1.2 .2 El SQL (Structured Query Language)

A partir de la teoría relacional del Dr. Codd, se tenía que construir un lenguaje que permitiera implementarla sobre las tablas de las bases de datos que siguieran el modelo.
Los lenguajes relacionales son aquellos que realizan bajo una sintaxis definida, algunas o todas las operaciones del modelo relacional. Tales lenguajes fueron creados a mediados de 1970.
A continuación se tiene una breve reseña histórica de SQL y su influencia en los RDBMS.

- Se define SEQUEL ("Structured English QUEry Language"), en los laboratorios de investigación de IBM en San José en 1974 por D. D. Chamberlin.
- IBM implementa el primer lenguaje de consulta prototipo llamado SEQUEL-XRM entre los años de 1974 - 1975.
- Se define una versión revisada de SEQUEL llamada SEQUEL/2 entre 1976-1977.
- Se reduce el nombre a SQL por razones legales.
- Se publica el SQL en 1976 de manera formal.
- En el ambicioso proyecto de investigación Sistema R, se implementa SQL, empezando a operar en 1977.
- Relational Software Inc. (Actualmente, Oracle Corporation), crea Oracle que es de los primeros RDBMS en emplear SQL en el año de 1979.
- En 1981, IBM anuncia el producto llamado SQL/DS, para el ambiente DOS/VSE y para VM/CMS en 1982.
- IBM crea un producto en 1983 para MVS, llamado DB2 compatible con SQL/DS.
- Data General Corporation introduce el producto DG/SQL en 1984.
- Sybase Inc. crea el RDBMS Sybase en 1986.
- Relational Technology Inc. entre 1981-1985 crea el RDBMS INGRES.
- Britton – Lee Inc. entre los años 1982-1985 crea IDM .
- SQL es adoptado como estándar por ANSI en octubre de 1986 para el manejo de las bases de datos relacionales.
- SQL es adoptado como estándar por ISO (International Standards Organization) en 1987.

Así, SQL se presenta como el lenguaje relacional por excelencia que con pocas instrucciones y reglas sintácticas muy apegadas al lenguaje natural, permite la consulta, manipulación, creación y administración de una base de datos relacional.

SQL es en realidad un híbrido del álgebra y cálculo relacionales, ya que permite implementar a ambos. Aunque la idea original de Codd era la consulta de la base de datos con operadores algebraicos, las necesidades de manipulación y creación de tablas era real, por ello, se han creado diversas instrucciones que hoy conforman el lenguaje SQL.

Se tiene así, que SQL se ha dividido en 3 grupos principales de instrucciones para realizar diferentes tareas sobre la base de datos relacional:

- El grupo llamado de Definición de Datos (*DDL, Data Definition Language*)
  Este grupo de instrucciones permite la definición o construcción de elementos de la base de datos, siendo éstos principalmente: usuarios, índices, tablas, vistas, etc. Las instrucciones que pertenecen a este grupo son **CREATE, ALTER** y **DROP,** sus funciones respectivas son construir, modificar y eliminar los elementos de la base de datos. Se tiene entonces que estas instrucciones están orientadas a trabajar con las estructuras. Con la instrucción CREATE se pueden establecer las reglas del negocio como parte de la definición de la

estructura de una tabla. Entre estas reglas de negocio o políticas de integridad se encuentran:

- No permitir valores nulos
- No permitir duplicidad de valores
- Definir los valores o rangos permitidos
- Establecer Llaves Primarias
- Establecer Llave Foráneas
- Permitir borrados en cascada en las relaciones Maestro-Detalle

- El grupo de Consulta (*Query*), permite efectuar las operaciones del álgebra relacional, este grupo está formado por la instrucción **SELECT**. Esta instrucción es muy potente pues permite la generación de nuevas relaciones bajo las necesidades de consulta del usuario, sin verse en la necesidad de alterar los datos o la estructura.

- El grupo llamado de Manipulación de Datos (*DML, Data Manipulation Language*)
  Este grupo permite la manipulación o procesamiento de los datos almacenados en las tablas de la base de datos. Las instrucciones son **INSERT, DELETE** y **UPDATE** que tienen como objetivo agregar nuevos renglones, borrar renglones de las tablas y actualizar los valores que guardan las tablas, respectivamente. Aquí, las instrucciones se orientan a la manipulación de los datos, no de las estructuras.

Algo importante de resaltar es que una vez reconocido el SQL como estándar para los RDBMS, cada compañía creadora de su manejador, creó su propio "dialecto" de SQL y le puso un nombre especial, como el caso del SQL de Oracle que se llama SQL*Plus y proporciona algunas bondades para generar reportes con un formato básico.

1.2 .3 La espera de un RDBMS completo

A los a 25 años del nacimiento del modelo relacional, pese a la firmeza de su fundamento y a la existencia de manejadores de bases de datos en el mercado construidos bajo su modelo, aún no se contaba con un producto 100 % relacional, es decir, que cumpliera con las famosas "*12 reglas de Codd*" para los RDBMS que E.F. Codd define formalmente [Codd85]; aunque las compañías se manifestaban con la justificación de "apego y solución a las necesidades reales", aún los llamados RDBMS presentaban muchas deficiencias formales. [McGo94]

Para 1990, Codd vuelve a su crítica contra los pseudo-RDBMS publicando en la versión 2 de su libro del Modelo Relacional, 333 reglas para la evaluación de esos productos comerciales. [Codd90]

Por su parte, Chris J. Date, también analiza al SQL y escribe al respecto algunas anomalías encontradas [Date89] con respecto al modelo formal, pero pese a todo lo anterior, el modelo relacional y el SQL, son usados por los productos comerciales actuales con gran éxito proporcionando verdaderas soluciones.

1.3 Estado del arte del almacenamiento de texto

Originalmente los registros pertenecientes a un archivo, con una estructura heterogénea, estaban formados por campos con tipos de datos básicos, éstos eran los tipos *CHAR*, *NUMBER* y *DATE* para almacenar caracteres alfanuméricos, números y fechas respectivamente; el tipo *CHAR* tenía un tamaño máximo definido para su longitud lo mismo que el *NUMBER*, que en su caso correspondía al rango.
El almacenamiento de texto estaba limitado al tipo *CHAR* que era el único que permitía almacenar caracteres de tipo alfanumérico; pero las necesidades de los usuarios, generaron nuevos tipos de datos.

Ya desde los años 80's, con los *Sistemas Manejadores de Archivos*, se tenía presente la necesidad de almacenamiento de grandes cadenas de texto, surgiendo por ejemplo el tipo de datos *MEMO* en dBase III plus, donde un campo de este tipo podía almacenar bloques de texto de hasta 4,000 caracteres por cada registro. Este tipo de campos, se guardaba de forma diferente a como lo hacia el resto de los datos.
Así, un problema latente no fue el almacenamiento, sino la búsqueda sobre este tipo de dato, es decir, el usuario no podía plantear búsquedas sobre estos campos pues quedaban inhabilitados por el sistema para plantear los criterios de búsqueda sobre ellos.

Con la aparición de los RDBMS, surgen los tipos de datos *CHAR* y *VARCHAR* para almacenar datos alfanuméricos sobre una columna de una tabla. Ambos tipos deben tener una longitud definida (número máximo de caracteres que pueden almacenar) en el momento de crear la estructura de la tabla que las contiene. Así, el RDBMS debe verificar que las cadenas de texto almacenadas no excedan la longitud asignada.
Para las columnas de tipo *CHAR*, la longitud definida es fija, si el texto almacenado tiene una longitud menor a la permitida, el RDBMS completa la longitud de la columna con el caracter blanco.

Por el contrario, si la columna tiene asignado el tipo de dato *VARCHAR*, ésta almacena el texto tal y como ha sido especificado, es decir, sin agregar blancos para completar la longitud; teniendo así la flexibilidad de un almacenamiento dinámico. La longitud máxima permitida, varía de un manejador a otro pero generalmente es superior a la del tipo *CHAR*.

Dependiendo del RDBMS usado para crear la base de datos los nombres de los tipos de datos mencionados anteriormente pueden variar, encontrándose entre ellos los tipos llamados: *ALPHANUMERIC, CHAR, CHARACTER, CHARACTER VARYING, CHAR VARYING* y *STRING*.
La longitud máxima permitida para cada tipo de dato, depende de cada RDBMS. En el Anexo A, se muestran los tipos de datos capaces de almacenar texto sobre columnas de tablas creadas en las bases de datos con alguno de los RDBMS más populares del mercado actual.

Una ventaja de almacenar texto (visto como cadena de caracteres alfanuméricos) sobre estos tipos es que se pueden declarar las "reglas del negocio" que los datos almacenados sobre esas columnas deben cumplir, y dejar al RDBMS esta tarea como parte de su tarea de *integridad*. Estas restricciones de integridad, se pueden definir por medio de lo que en bases de datos relacionales se conoce como *CONSTRAINT* asociado a una o varias columnas en una tabla. La cláusula para crear estas restricciones, aparece en la instrucción ***CREATE TABLE*** del grupo DDL del SQL.

Los tipos de restricciones o reglas de integridad que se pueden definir sobre texto son los siguientes:

- **Null.** Especifica que la columna puede contener valores nulos.
- **Not Null.** Especifica que la columna no puede contener valores nulos.
- **Unique.** Especifica que la columna no puede contener valores repetidos.
- **Check.** Especifica una condición que el valor dentro de la columna debe cumplir.
- **Primary Key.** Designa una columna o combinación de ellas como la llave primaria (PK) de la tabla.
- **Foreign Key** Designa una columna o combinación de ellas como la llave foránea (FK) de la tabla y por ende, una integridad referencial que se debe cumplir.

La creación de *índices* es algo importante de resaltar ya que este tipo de elementos de una base de datos relacional es frecuentemente construido para proporcionar un acceso directo y rápido a los datos si se plantea una consulta bajo un criterio de selección que utilice la(s) columna(s) indexada(s). Sobre las columnas de tipo *CHAR* y *VARCHAR* es posible la construcción de índices con el fin de ayudar al rendimiento del RDBMS.

Si el texto excede los 4000 caracteres de longitud, es necesario usar otro tipo de dato que no sea *CHAR* o *VARCHAR*, esto, dependiendo del manejador de la base de datos. Se necesita de tipos de datos, que permitan almacenar datos textuales no estructurados como artículos de revistas, páginas Web, faxes, contratos, documentación, etc.
Han surgido nuevos tipos de datos para las columnas de una tabla, teniendo así los llamados *LONG*, *LONG RAW*, *BFILE* y los *LOB's* (Large Objects) que incluyen a los *BLOB* (Binary Large Objects), *CLOB* (Character Large Object) y *DBCLOB* (Double Byte Character Large OBject).

Las columnas definidas como *LONG* pueden almacenar datos de tipo caracter (texto) de longitud variable con una capacidad de hasta 2 gigabytes.
Las columnas con tipo *LONG RAW* se usan para almacenar datos binarios o cadenas de bytes. Almacenan gráficas, sonido, documentos o arreglos binarios de datos, la interpretación depende de su uso. Generalmente, tienen una capacidad superior a la de las *LONG*.
Los tipos de dato *LOB* permiten almacenar grandes bloques de datos no estructurados como texto, gráficas, imágenes, video y sonido; proporcionando un eficiente acceso a los datos.

Existen diferencias entre los tipos *LOB* y los tipos *LONG* y *LONG RAW*, éstas dependiendo del manejador de base de datos, por ejemplo, en una base de datos creada con *Oracle8i* (última versión liberada hasta la fecha de este escrito):

- Una tabla puede contener múltiples columnas de tipo *LOB* pero solo una de tipo *LONG*.
- Una tabla que contenga una o más columnas *LOB* puede ser particionada, pero una tabla con una columna *LONG* no.
- El tamaño máximo de una columna *LOB* es de 4 gigabytes, pero el tamaño máximo de una *LONG* de dos gigabytes.
- Los tipos *LOB* soportan acceso aleatorio de datos, el tipo *LONG* soporta solo acceso secuencial.
- Los tipos *LOB* (excepto el tipo *NCLOB*) pueden ser atributos de un tipo de objeto definido por el usuario, pero un tipo *LONG* no.

- Las variables temporales de tipo *LOB* (*BLOB, CLOB,* y *NCLOB*) se crean en el tablespace temporal de la base de datos y son independientes de las tablas, para las variables de tipo *LONG* no hay estructuras temporales disponibles.

El tipo de dato *BFILE (LOB externo)* almacena datos binarios no estructurados en archivos del sistema operativo, es decir, fuera de la base de datos. Una columna *BFILE* almacena un apuntador localizador del archivo externo que almacena los datos, almacena hasta 4 gigabytes. Es un tipo de dato que sólo permite lectura aleatoria y no puede formar parte de una transacción.
Es el sistema operativo y no el RDBMS quien debe mantener su integridad y durabilidad, siendo así una responsabilidad extra del DBA verificar que los procesos del RDBMS tengan permisos de lectura sobre ese archivo. Los archivos pueden ser localizados en discos duros, discos compactos, etc.

La creación de índices sobre columnas de tipo *LOB* no es posible, al menos en el RDBMS Oracle8. Si se pretendiera crear un índice sobre una columna *LOB* de la tabla anterior, no sería posible su construcción y se obtendría un mensaje de error que lo reporta al usuario creador de ese elemento:

```
SQL> CREATE UNIQUE INDEX indice_lob ON lob_table(c_lob);

ERROR at line 1:
ORA-02327: cannot create index on column with datatype LOB
```

La existencia de los tipos de datos anteriormente mencionados, así como sus nombres y capacidades de almacenamiento dependerán del RDBMS y de su versión. (Ver Anexo A.)

En un momento dado, se pensó que se podía resolver en parte el problema de almacenamiento y manipulación de grandes cantidades de texto con el tipo de dato *LONG*. el RDBMS de *Oracle* lo trabaja para almacenar hasta 2Gb de texto pero este valor queda limitado por la memoria de la computadora en la que se tiene la base de datos. Permite hacer referencia a estas columnas en las siguientes instrucciones de SQL:

- **SELECT** [*lista de columnas*]
- cláusula **SET** del **UPDATE**
- cláusula **VALUES** del **INSERT**

Pero el uso de los valores *LONG* está sujeto a muchas restricciones:

- Una tabla no puede contener más de una columna tipo *LONG*.
- Las columnas *LONG* no pueden aparecer en las restricciones de integridad, excepto de tipo *NULL* y *NOT NULL*.
- Las columnas *LONG* no pueden formar parte de un índice.
- En una instrucción SQL, todas las columnas *LONG*, tablas actualizadas y tablas bloqueadas deben estar localizadas en la misma base de datos (para Base de Datos Distribuidas).
- Las columnas de tipo *LONG* no pueden aparecer en ciertas partes de las sentencias SQL:

- **WHERE, GROUP BY, ORDER BY** de la instrucción **SELECT**.
- En el operdor **DISTINCT** de la sentencia **SELECT**
- En la cláusula **UNIQUE** de la sentencia **SELECT**
- En algunas funciones SQL para la obtención de subcadenas (por ejemplo las funciones: **SUBSTR** o **INSTR**)
- En expresiones o condiciones de selección
- En subconsultas en la sentencia **INSERT**
- En subconsultas para la sentencia **CREATE SELECT AS**

- En subconsultas o consultas combinadas por operadores de conjuntos

- Una función almacenada no puede regresar un valor *LONG*

Los *Triggers* pueden usar el tipo de dato *LONG* de la siguiente manera:

- Una sentencia SQL en un trigger puede insertar datos en una columna de tipo *LONG*.
- Si el dato de una columna *LONG* puede ser convertido a un tipo de dato limitado (como *CHAR* y *VARCHAR2*), una columna *LONG* puede ser referenciada en una sentencia SQL dentro del *trigger*. Cumpliendo la restricción de la longitud máxima de estos tipos de datos.
- Las variables dentro de los *triggers* no pueden ser declaradas usando el tipo de dato *LONG*.
- Las variables :NEW y :OLD no pueden ser usados en las columnas de tipo *LONG*.
- Se debe usar el OCI (Oracle Call Interfaces) para obtener una parte de un valor *LONG* de una base de datos.

1.4 Límites en la definición de dominios para texto y otras estructuras

La definición de tipos especiales para almacenar texto en bases de datos relacionales se ha proporcionado en la mayoría de los productos comerciales, por desgracia, éstos tienen algunas limitantes comparadas con los otros tipos de datos, pues la creación de ellos en una tabla, en una vista o la asociación de índices, no se permite libremente.

A continuación se muestran una serie de pruebas sobre la versión 8.0.4.0 del RDBMS de Oracle en la creación de tablas, vistas e índices con columnas de tipo *LONG*.

Errores Oracle

Se presentan en la siguiente tabla, los mensajes de error Oracle [Orac97] asociados a las pruebas efectuadas. Se muestra el número de error y su texto; la causa que propicia el error y la acción necesaria para su corrección.

ORA-00997	*illegal use of LONG datatype*
Causa:	*Un valor del tipo de dato LONG fue usado en una función o en las cláusulas DISTINCT, WHERE, CONNECT BY, GROUP BY u ORDER BY. Un tipo LONG sólo puede ser usado en la cláusula SELECT.*
Acción:	*Remover el valor de tipo LONG de la función o cláusula.*
ORA-01754	*a table may contain only one column of type LONG*
Causa:	*Solo una columna por tabla puede ser definida con el tipo de dato LONG.*
Acción:	*Remover el tipo LONG de todas o alguna de las columnas y reintentar la operación.*
ORA-01732	*data manipulation operation not legal on this view*
Causa:	*Se intentó usar la sentencia UPDATE, INSERT, o DELETE sobre una vista que contiene expresiones o funciones o fue derivada de más de una tabla. Si se usó una operación de join para crear la vista o la vista contiene columnas virtuales derivadas de funciones o expresiones, entonces la vista solo puede ser consultada.*
Acción:	*Efectuar las operaciones de UPDATE, INSERT o DELETE en sus respectivas tablas, la vista sólo puede usarse para consultas.*

Tabla1.2 Mensajes de error Oracle

Pruebas

SQL*Plus: Release 8.0.4.0.0 - Production on Wed Jul 19 11:27:3 2000
(c) Copyright 1997 Oracle Corporation. All rights reserved.

Connected to:
Oracle8 Personal Edition Release 8.0.4.0.0 – Production

PL/SQL Release 8.0.4.0.0 – Production

Prueba 1. Creación de una tabla que contenga dentro de su estructura, dos columnas del tipo LONG y efectuar consultas sobre ella.

```
SQL>  CREATE TABLE Reglas( Clave NUMBER(4) PRIMARY KEY,
      2  Descripcion1 LONG, Descripcion2 LONG);

Descripcion1 LONG, Descripcion2 LONG)
             *
ERROR at line 2:
ORA-01754: a table may contain only one column of type LONG
```

Conclusión: En *Oracle* no se permite tener más de una columna de tipo Long en una tabla.

Prueba 2. Creación de un índice sobre una columna de tipo LONG de una tabla existente.

```
SQL> L
  1  CREATE TABLE Reglas( Clave NUMBER(4) PRIMARY KEY,
  2* Descripcion1 LONG,  Descripcion2 VARCHAR2(100))
SQL> /

Table created.


SQL> CREATE INDEX Indice_prueba ON Reglas(Descripcion1);
CREATE INDEX Indice_prueba ON Reglas(Descripcion1)
                              *
ERROR at line 1:
ORA-00997: illegal use of LONG datatype


SQL> CREATE INDEX Indice_prueba ON Reglas(Clave, Descripcion1);
CREATE INDEX Indice_prueba ON Reglas(Clave, Descripcion1)
                                    *
ERROR at line 1:
ORA-00997: illegal use of LONG datatype
```

Conclusión: No se permite crear un índice sobre una columna de tipo *Long*.

Prueba 3. Crear una Vista de columnas de tipo LONG existentes en diversas tablas.

```
SQL> DESC Reglas
 Name                      Null?    Type
 ------------------------- -------- ----
 CLAVE                     NOT NULL NUMBER(4)
 DESCRIPCION1                       LONG
 DESCRIPCION2                       VARCHAR2(100)


SQL> DESC Reglas2
 Name                      Null?    Type
 ------------------------- -------- ----
 CVE                       NOT NULL NUMBER
 DESC1                              LONG
 DESC2                              VARCHAR2(100)



SQL> 1  CREATE VIEW Reglas3 AS
  2   SELECT Descripcion1, Desc1
  3* FROM Reglas,Reglas2
SQL> /

View created.


SQL> DESCRIBE Reglas3
 Name                      Null?    Type
 ------------------------- -------- ----
 DESCRIPCION1                       LONG
 DESC1                              LONG
```

Conclusión: Sí se puede crear una vista con varias columnas de tipo Long.

Prueba 4. Efectuar instrucciones DML y SELECT sobre las vistas con columnas *LONG*.

```
SQL> DESC PRUEBAS
 Name                      Null?    Type
 ------------------------- -------- ----
 UNO                       NOT NULL NUMBER
 DOS                                LONG


SQL> DESC REGLAS
 Name                      Null?    Type
 ------------------------- -------- ----
 CLAVE                     NOT NULL NUMBER(4)
 DESCRIPCION1                       LONG
 DESCRIPCION2                       VARCHAR2(100)


SQL> DESC REGLAS2
 Name                      Null?    Type
 ------------------------- -------- ----
 CVE                       NOT NULL NUMBER
 DESC1                              LONG
 DESC2                              VARCHAR2(100)



SQL> CREATE VIEW TOTAL3 AS
2       SELECT *
3*      FROM  PRUEBAS, REGLAS,REGLAS2;

View created.


SQL> DESC TOTAL3
 Name                      Null?    Type
 ------------------------- -------- ----
 UNO                                NUMBER
 DOS                                LONG
 CLAVE                              NUMBER(4)
 DESCRIPCION1                       LONG
 DESCRIPCION2                       VARCHAR2(100)
 CVE                                NUMBER
 DESC1                              LONG
 DESC2                              VARCHAR2(100)
```

Al pretender efectuar alguna operación DML (INSERT, UPDATE, DELETE) sobre la vista, se produce un error :

ORA-01732: data manipulation operation not legal on this view

Debido a que sólo se puede CONSULTAR (SELECT) sobre las vistas creadas a partir de varias tablas, lo cual no es limitante por el uso de columnas de tipo *LONG*, sino una manera de mantener la integridad de los datos.

Conclusión: Se puede crear una vista con varias columnas de tipo *LONG* pero se mantiene la limitante de sólo consultar sobre este tipo de elemento relacional.

1.5 El alcance del SQL en la búsqueda de texto

Una vez creada la estructura de la tabla y capturados los datos sobre sus respectivas columnas, surge un nuevo reto para el manejador de la base de datos: Efectuar búsquedas eficientes y eficaces sobre las columnas con texto, que satisfagan los criterios establecidos por el usuario.

1.5.1 Consultas sobre patrones definidos

En un principio, las consultas bajo algunos criterios eran cubiertas por las capacidades más simples del operador de selección implementado con la cláusula WHERE de la instrucción SELECT.
En el siguiente ejemplo, se tiene una consulta donde se busca un patrón exacto, es decir, una cadena específica con la que deben coincidir todos los caracteres de la cadena almacenada en la columna a la que se le aplica el criterio de selección en cada tupla:

SELECT *nombre, telefono*
FROM *institutos*
WHERE *incorporado* = **'U.N.A.M.'**

Una consulta más compleja podría ser ignorar las diferencias entre mayúsculas y minúsculas en un texto almacenado, para ello, se podía hacer uso de funciones de conversión de letras a minúsculas (LOWER), o a mayúsculas (UPPER) o bien, para cambiar sólo el primer caracter a mayúsculas (INITCAP). Algunos ejemplos podrían ser:

1) En la siguiente consulta se aplica la función UPPER sobre el dominio *especialidad* (considerado de tipo caracter) para convertir las cadenas a mayúsculas y compararlas contra un patrón de letras en mayúsculas.

SELECT *ap_paterno, nombre, telefono*
FROM *investigadores*
WHERE UPPER(*especialidad***) = 'BOTANICA'**

2) En la siguiente proyección de datos se aplica la función LOWER sobre el dominio *especialidad* para convertir las cadenas a minúsculas y compararlas contra un patrón en ese mismo tipo de letras.

SELECT *ap_paterno, nombre, telefono*
FROM *investigadores*
WHERE LOWER(*especialidad***) = 'botanica'**

3) En la siguiente proyección de datos se aplica la función INITCAP sobre el dominio *especialidad* para convertir la letra inicial del texto a mayúsculas y el resto en minúsculas para compararlas contra un patrón con ese mismo comportamiento.

SELECT *ap_paterno, nombre, telefono*
FROM *investigadores*
WHERE INITCAP(*especialidad***) = 'Botanica'**

Un factor no considerado en las consultas anteriores es la existencia de los acentos en los criterios de búsqueda, así, es necesario establecer una consulta más detallada:

4) **SELECT** *ap_paterno, nombre, telefono*
FROM *investigadores*
WHERE LOWER(*especialidad***) = 'botanica'**
OR LOWER(*especialidad***) = 'botánica'**;

Algo importante también, es considerar el tipo de dato de la columna sobre la que se efectuará la búsqueda debido a que no todos los tipos de datos trabajan igual. Así, tenemos algunos criterios de comparación.

Las cadenas de texto se comparan usando una de las siguientes reglas:

- **Comparación semántica completando con blancos**. Si las dos cadenas tienen diferentes longitudes, primero se agregan blancos al final de la cadena más corta hasta que ambas cadenas tengan la misma longitud. Entonces, se comparan los valores caracter por caracter hasta encontrar el primer caracter diferente. La cadena con el valor más grande en el caracter de la primer posición diferente se considera más grande. Si las dos cadenas no tienen caracteres diferentes, entonces se consideran iguales. Esta regla significa que dos valores son iguales si ellos difieren solo en el número de blancos agregados para igualar las longitudes. Este tipo de comparación es empleado en algunos RDBMS para comparaciones entre valores ambos de tipo *CHAR*.

- **Comparación semántica sin completar con blancos**. En este tipo de comparación, se comparan dos cadenas caracter por caracter hasta encontrar el primer caracter diferente entre ellas. La cadena con el valor más grande en el caracter de la primer posición diferente se considera más grande. Si dos cadenas de longitudes diferentes son idénticas hasta el final de la cadena más corta, la cadena más larga se considera más grande. Si dos cadenas de igual longitud no difieren en sus caracteres, se consideran iguales. Este tipo de comparación es empleado en algunos RDBMS para comparaciones con valores de tipo *VARCHAR2*.

La tabla 1.3 muestra los resultados de comparación entre cinco pares de cadenas de caracteres usando cada comparación semántica.

Rellenando con blancos	***Sin rellenar con blancos***
'ab' > 'aa'	'ab' > 'aa'
'ab' > 'a '	'ab' > 'a '
'ab' > 'a'	'ab' > 'a'
'ab' = 'ab'	'ab' = 'ab'
'a ' = 'a'	'a ' > 'a'

Tabla1.3 Resultados de comparaciones semánticas

La comparación entre caracteres se efectúa de acuerdo a sus valores dentro de la secuencia del conjunto de caracteres usado en la base de datos (Orden Lexicográfico). Un caracter es más grande que otro si éste tiene un valor más grande que el otro en la secuencia. Algunos de los conjuntos de caracteres o códigos más empleados son:

- El código ASCII de 7-bits
- El código EBCDIC
- El ISO 8859/1
- El JEUC (Japan Extended UNIX)
- El SJIS Japan Shift JIS

El tipo de código que utiliza cada base de datos se determina en el momento de la creación de la base, ya que es en ese momento donde se indica al manejador el código que deberá emplear.
Es muy importante el código, pues de éste depende que en la base de datos se puedan almacenar caracteres propios del idioma, como las letras ñ, ü, á, ó, etc. del español.

1.5.2 Consultas sobre patrones no-explícitos

Para consultas más generales, no es suficiente un símbolo igual (=) en la cláusula WHERE, ya que la igualdad con un patrón de búsqueda no es muchas veces del todo conocida o bien, no es un problema numerable. Así, aparece el operador LIKE aplicable sobre columnas de tipo texto como CHAR o VARCHAR. Se utiliza para comparar una expresión de cadena con un modelo en una expresión SQL. Su sintaxis dentro de la instrucción SELECT es la siguiente:

SELECT ...
FROM ...
WHERE expresión **LIKE** *modelo*

En donde *expresión* es una cadena o columna contra la que se compara un modelo (patrón definido). Se puede utilizar el operador LIKE para encontrar valores en las columnas que coincidan con el modelo especificado. Por *modelo* se puede especificar un valor completo, o se pueden utilizar caracteres comodín como los reconocidos por los sistemas operativos para encontrar un rango de valores: (*, _ , %).

El operador LIKE se puede utilizar en una expresión para comparar un valor dentro una columna con una expresión de cadena. Por ejemplo, si se introduce LIKE C*, la consulta devuelve todos los valores de la columna que comiencen por la letra C y sigan o no con una serie de caracteres. En una consulta con parámetros, es posible que el usuario escriba el modelo que se va a utilizar.

Los comodines dependen del RDBMS empleado, así por ejemplo, el efecto del asterisco (*) antes mencionado, en el SQL*Plus de Oracle es el símbolo %.

El siguiente ejemplo del operador LIKE sobre SQL devuelve los datos que comienzan con la letra P seguidos de cualquier letra entre A y F pero finalizando con tres dígitos:

SELECT nombre
FROM peliculas
WHERE titulo **LIKE 'P[A-F]###'**

En la siguiente consulta, se recuperan los renglones cuyo contenido sobre ciertas columnas, empiece con una letra de la A a la D seguidas de cualquier cadena.

SELECT nombre
FROM peliculas
WHERE titulo **LIKE '[A-D]*'**

En la tabla 1.4 se muestra cómo utilizar el operador LIKE para comprobar expresiones con diferentes modelos.

Tipo de coincidencia	*Modelo Planteado*	*Coincide*	*No coincide*
Varios caracteres	'a*a'	'aa', 'aBa', 'aBBBa'	'aBC'
Carácter especial	'a[*]a'	'a*a'	'aaa'
Varios caracteres	'ab*'	'abcdefg', 'abc'	'cab', 'aab'
Un solo carácter	'a?a'	'aaa', 'a3a', 'aBa'	'aBBBa'
Un solo dígito	'a#a'	'a0a', 'a1a', 'a2a'	'aaa', 'a10a'
Rango de caracteres	'[a-z]'	'f', 'p', 'j'	'2', '&'
Fuera de un rango	'[!a-z]'	'9', '&', '%'	'b', 'a'
Distinto de un dígito	'[!0-9]'	'A', 'a', '&', '~'	'0', '1', '9'
Combinada	'a[!b-m]#'	'an9', 'az0', 'a99'	'abc', 'aj0'

Tabla1.4 Comparaciones con expresiones

Así, algunas consultas con SQL*Plus aplicando algunos criterios de selección sobre columnas con texto serían las siguientes para poder tener una respuesta satisfactoria:

5) En la consulta siguiente, se puede apreciar que la cláusula WHERE tiene una estructura más compleja para poder ser más flexible en la búsqueda de diversos patrones dentro de la columna *descripcion*:

```
SELECT *
FROM Video
WHERE descripcion LIKE '%editor de texto y multimedia'
OR descripcion LIKE '%editor de texto y multi-media%'
OR descripcion LIKE '%editor de documento multimedia%'
OR descripcion LIKE '%editor de documento multi-media%';
```

Algunos ejemplos aplicando la notación de *SQL Server 7*, se tienen a continuación.

6)
```
SELECT *
FROM Gastos
WHERE adeudo LIKE '[C- N]oche'
```

Con la instrucción anterior, se proyectará todo renglón cuyo valor en la columna *adeudo* comience con cualquiera de las letras que están de la letra C a la N (lexicográficamente hablando), y finalice con los caracteres "oche".

7)

```
SELECT *
FROM Alumnos
WHERE nombre LIKE '[^I-Z]'
```

Con la instrucción anterior, cualquier valor de la columna *nombre* que comience con una letra comprendida entre la I y la Z, no será recuperado.

8)

```
SELECT *
FROM Productos
WHERE descripcion LIKE '_oche'
```

Con la instrucción anterior, se muestra un ejemplo del comodin de coincidencia de un caracter, que puede ser cualquier valor, seguido por el resto de caracteres de la cadena.. Así, se seleccionará en la tabla cualquier renglón que contenga en la columna *descripcion* un caracter, seguido por las letras "oche".

1.6 Consultas sin respuesta

Algunos RDBMS, empleando el SQL, se apoyan de los *"comodines"* para efectuar búsquedas de patrones de texto en las columnas que almacenan caracteres alfanuméricos, como se mostró en la sección anterior, pero existen algunas necesidades de recuperación difíciles de satisfacer con lo que por el momento se ofrece.

A continuación, se muestra un ejemplo con la necesidad de tener una tabla que almacene texto de grandes y diversas longitudes así como también, consultar dicha tabla bajo diversos criterios.

" Se tiene una tabla determinada de una base de datos en una institución gubernamental que se rige por un determinado reglamento, el cual está formado por una serie de articulos identificados por un número exclusivo. Cada artículo pertenece a una determinada categoria o clasificación según su objetivo; fue establecido en un año determinado y se requiere almacenar su texto de manera digital. "

La estructura correspondiente a tal problema, podría verse de la siguiente manera:

REGLAMENTO

Articulo_numero	Categoria	Fecha	Texto

Donde:

Articulo_numero..... Number(3)
Categoria..... Varchar2(10)
Fecha ..... Date
Texto ..... Varchar2(250)

Algunas posibles necesidades de consultas por parte de los usuarios (indicadas entre comillas " ") y su respectiva implementación por parte del programador (codificada en *SQL*Plus de Oracle*), serían las siguientes:

Consulta 1)

"Se desea conocer cuales son los artículos del reglamento que hablen de la **educacion.**"

```
SELECT Articulo_numero
FROM   Reglamento
WHERE  Texto LIKE '%educacion%';
```

Observación: Esta consulta regresará los artículos que contengan las siguientes palabras: **educacion, educacional, educacionista, pre-educacion, antieducacional,** entre otras, y es probable que esto exceda las necesidades del usuario y sea necesaria una segunda depuración no automatizada. También, se observa que la consulta no contempla la posibilidad de que los textos estén capturados en mayúsculas, minúsculas o una combinación de ambas.

Consulta 2)

" Se desea conocer la clave de los artículos del reglamento que contengan la palabra **trabajo** "

```
SELECT Articulo_numero
FROM   Reglamento
WHERE LOWER(Texto) LIKE '%trabajo%';
```

Observación: Esta consulta regresará los artículos que contengan las siguientes palabras: **trabajo, trabajosamente, trabajoso, trabajosa, TRABAJO, TRABAJOSAMENTE, TRABAJOSO, TRABAJOSA, Trabajo, Trabajosamente, Trabajoso, Trabajosa,** etc. que se encuentren dentro del reglamento. Esta consulta es más poderosa que la anterior debido al uso de la función *LOWER* que permite convertir el texto de la columna texto en minúsculas y compararla con el patrón definido; así, se tiene una mayor capacidad obtener una consulta más apegada a lo esperado.
Pero si en la consulta sólo se deseaba el término trabajo, tampoco se ve satisfecha.

Consulta 3)

" Se necesita saber el número de artículo del reglamento que contenga el patrón *trabajo* y el patrón *educacion* en su texto ".

(i)
```
SELECT  Articulo_numero
FROM    Reglamento
WHERE   texto = 'educacion' AND texto = 'trabajo';
```

(ii)
```
SELECT  Articulo_numero
FROM    Reglamento
WHERE   LOWER(texto) LIKE  '%educacion%'
AND     LOWER(texto) LIKE  '%trabajo%';
```

Observación: Estas consultas son una combinación de las anteriores, donde se observa que las limitantes continúan pese a la función *lower*. Además, se observa la necesidad de incluir el operador lógico de conjunción (AND) lo cual complica que la consulta solicitada sea satisfecha conforme lo espera el usuario, debido a que ambos predicados lógicos deberán cumplirse y esto ocasiona toda una serie de combinaciones tal vez, no deseadas.
Pero si la consulta sólo se deseaba el término trabajo, tampoco se ve satisfecha.

Otras posibles consultas apoyadas por el uso del comodín, serían las que usan el guión bajo (_) en el lugar del %. Mientras que el símbolo % sirve para buscar cualquier cadena con 0 o más caracteres, i.e. de cualquier longitud, el símbolo _ se emplea para coincidir con un solo caracter y se pueden emplear de manera combinada para efectuar búsquedas algo más complejas.

Un ejemplo se tiene a continuación:

```
SELECT  Articulo_numero
FROM    Reglamento
WHERE   tipo LIKE '_ _ S' OR tipo LIKE '_ _ %';
```

La sentencia anterior, permite proyectar el número de artículo de aquellos cuyo tipo inicie con 3 caracteres y termine con la letra S o de aquellos que empiezan al menos con 2 letras.

Estas consultas aunque poderosas, no solucionan algunas búsquedas más complejas como:

Obtener para la consulta 1 aquellos artículos que contengan la palabra **Educacion, EDUCACION, Educación, EDUCACIONAL, Educacional, Educacionales,** etc. ya que se observa que todas ellas no son iguales entre sí, pues al ser unas mayúsculas, otras minúsculas, otras con una mezcla de ambas y finalmente, otras con acento, son patrones de cadenas totalmente distintas aunque sea la misma palabra.
Otra posible búsqueda satisfactoria, sería que al buscar el patrón Educación, se pudieran asociar a éste, lo referente a las palabras: **EDUCATIVO, EDUCATIVA, ESCOLAR, PREESCOLAR, PEDAGOGIA, ENSEÑANZA**, etc.

Como éste, hay muchos ejemplos de instituciones o empresas que tienen la necesidad de poder emplear este tipo de búsquedas, con una tecnología que les brinde flexibilidad, eficacia, rapidez y sencillez para plantear la consulta; entre estas instituciones se encuentran: las bibliotecas, y las hemerotecas, sin mencionar a todos los usuarios de la World Wide Web.

1.7 Comentarios finales

En el mundo actual que gira alrededor de la información, el acceso a los datos es crucial, la información de negocios es más que datos estructurados en bases de datos relacionales. El 90 % de la información digital es texto [Barb98], éste en todas sus formas representa un recurso enorme y valioso de información. Sin embargo, contar con ese recurso e incorporarlo con la información estructurada ha sido en gran medida una meta no alcanzada y es preciso que se tengan soluciones para atender a todas las necesidades que se presenten con este tipo de datos.

En su momento, SQL basado en el álgebra relacional fue usado para datos estructurados, de manera adecuada y en alta escala esta ingeniería se usó para el texto.
Pese al uso flexible del LIKE, las consultas que lo requieren se vuelven lentas, dependiendo del tamaño de la tabla y de la cantidad de texto en la columna analizada, esto debido a la imposibilidad de crear índices y por ello, se debe hacer un acceso secuencial sobre la tabla completa.

Si se ha almacenado texto en los tipos de datos LOB, éstos no podrán ser accesibles vía SQL. En esencia, se encuentran almacenados en un formato binario no interpretado. El sistema aconsejado para ver los datos almacenados en un LOB pasa por la utilización de métodos. Pero los métodos son códigos de lenguajes de programación que permitan la comunicación con el RDBMS y no es suficiente el SQL, así sólo con estos lenguajes podrán ver y/o manipular los datos almacenados en una columna LOB.

Actualmente, las aplicaciones requieren del acceso a un almacén de datos donde se puedan recuperar contratos, correos electrónicos, artículos de revistas, memorándums, etc. todos ellos sobre un tema específico, lo cual, tampoco es posible solucionar con una consulta SQL. Además, otro factor importante a considerar es la diversidad de formatos: *ASCII, HTML, MSWord, PDF*, etc.

Resulta entonces, de gran interés y utilidad, desarrollar herramientas que permitan almacenar, administrar y recuperar la información no trivial del texto. El presente trabajo se enfoca a los textos que se pueden almacenar o controlar desde una base de datos relacional y que se pueden consultar por medio de un "dialecto" del SQL, entendiendo como consulta, responder a todas las necesidades de búsqueda planteadas anteriormente como casos sin respuesta.

Así, un cuestionamiento queda en el aire:

¿ Qué tan confiable es la edad de la información

si no se puede encontrar lo que se busca?

Capítulo II

Teoría de la recuperación de información

" Toda nuestra ciencia,
comparada con la realidad,
es primitiva e infantil ...
y sin embargo,
es lo más preciado que tenemos "

Albert Einstein

En este capítulo se presenta la teoría en que se apoyan las diversas soluciones comerciales con que se cuenta en el mercado actual para resolver la necesidad de búsqueda y recuperación sobre texto. Aunque pudiera parecer que la recuperación de información es tan antigua como la teoría de las bases de datos relacionales, podemos decir, que ambas áreas se han fusionado poco para dar una buena respuesta a las búsquedas sobre texto.

2.1 Sistemas de Recuperación de Información (SRI) (Information Retrieval System, IRS)

La Recuperación de Información (RI) se encarga del estudio de: la representación, el almacenamiento, la organización y el acceso a la información. Debe proporcionar al usuario una interfaz que le permita plantear sus necesidades de información; sin embargo, caracterizar tal información no es tan sencillo, ya que el usuario debe transformar sus necesidades a consultas. En su forma más común, esta traducción incluye un conjunto de palabras clave que resumen la descripción de la información requerida.
La consulta debe ser procesada por un SRI que recupera la información útil o relevante para el usuario de acuerdo a los datos de entrada.

Los Sistemas de Recuperación de Información involucran texto en lenguaje natural, el cual no está estructurado y puede ser semánticamente ambiguo. Cuando se realiza una búsqueda sobre texto, se debe llevar a cabo una "interpretación" del contenido de los documentos y una clasificación de acuerdo a su grado de relevancia para la consulta. Esta "*interpretación*" del contenido de un documento involucra la extracción de la información sintáctica y semántica del texto, la cual se usa para igualar o emparejar (*match*) con la consulta requerida por el usuario. La dificultad no es sólo conocer cómo extraer esta información, sino conocer cómo usarla para decidir su relevancia.
Así, la meta de un sistema de recuperación de información es recuperar todos los documentos relevantes para un usuario evitando como sea posible, recuperar aquellos no-relevantes. El propósito de un sistema de recuperación de textos es la obtención de documentos que son similares en contenido a una consulta dada, es decir, que tienen muchas palabras en común con ella.

En la figura 2.1 se muestra de manera general, el proceso de recuperación de la información por medio de un sistema: El administrador de la base de datos (Data Base Administrator, DBA) define la base de datos de texto, documentos a ser usados, operaciones posibles de ejecutar sobre el texto y el modelo. Los documentos entonces se transforman en vistas lógicas sobre las que se crean *índices*. Los documentos a los que se quiere acceder por contenido deben ser preprocesados y adecuadamente indexados por sus términos, para que después se puedan realizar búsquedas sobre ellos. Se suelen utilizar *índices invertidos*, que almacenan, para cada término en la colección de documentos, una cadena que identifica cada documento en el que aparece el término y su posición. De esta manera, ante una consulta de un usuario no es necesario buscar en todos los documentos. Basta examinar el índice correspondiente a las palabras presentes en la consulta y localizar así los documentos, y la posición dentro de éstos, en donde se encuentran los términos buscados.
Así, el usuario define su consulta, la cual se procesa y permite recuperar aquellos documentos que cumplen los criterios de la búsqueda, posteriormente, éstos pasan por un proceso de clasificación *(rank)* que les otorga un nivel de relevancia según cumplan con el criterio del usuario, es entonces cuando se entregan al usuario como respuesta.

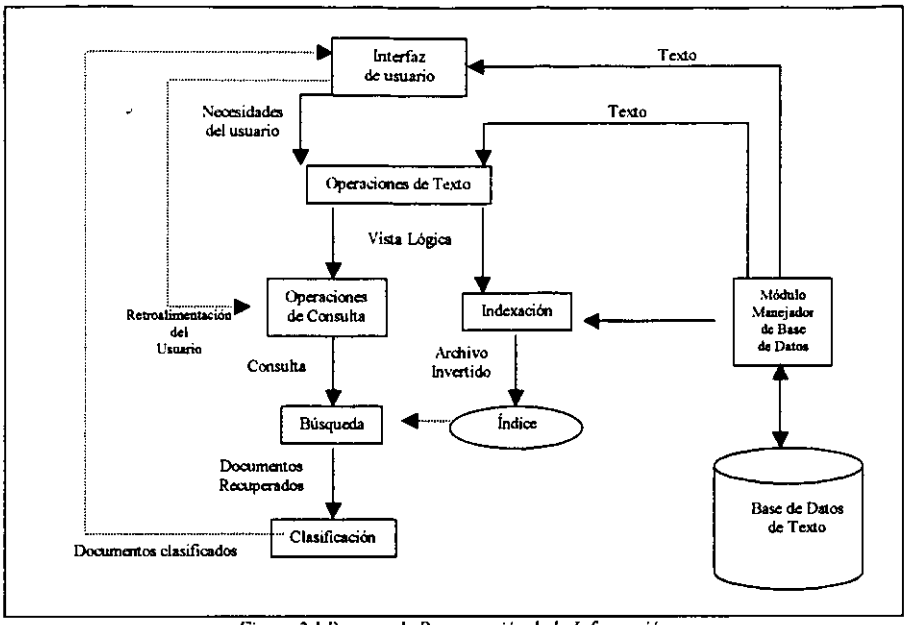


Figura 2.1 Proceso de Recuperación de la Información

Por su parte, los sistemas de recuperación de datos determinan los documentos que contienen las palabras clave que el usuario busca, los cuales, generalmente no satisfacen las necesidades de información, sino una expresión regular o una expresión de álgebra relacional.
Los sistemas de recuperación de datos tratan con datos que tienen una estructura bien definida así como una semántica. Pero pese a que proporcionan una solución al usuario de la base de datos, no resuelven problemas de recuperación de información sobre un tema o tópico.

Tanto en un Sistema de Recuperación de Datos como en uno de Recuperación de Información, el usuario debe ejecutar una consulta o tarea de recuperación que involucra una interacción, existiendo para ello, dos tipos de estas tareas:

- Recuperación (*Retrieval*). El usuario debe especificar un conjunto finito de palabras, las cuales expresan la semántica de las necesidades de información o bien proporcionar una expresión que determine las restricciones que deben satisfacer los documentos recuperados. Los SRI clásicos tienen este tipo de tarea de recuperación.

- Navegación (*Browsing*). Aquí el usuario no tiene un interés específico. Puede ser un interés general o bien, el objetivo puede cambiar durante la interacción con el sistema. Un ejemplo de este tipo de tarea, lo tenemos en los sistemas de hipertexto.

En ambos tipos de tareas, debe existir una interfaz que permita la recuperación interactiva.

2. 2 Modelos de Recuperación

El problema central en los SRI, es determinar cuales son los documentos relevantes y cuales no; tal decisión depende del algoritmo empleado para determinar el grado de importancia de cada documento, los cuales se listarán respecto a su relevancia. Así, éste tipo de algoritmos es el núcleo de los SRI.

Los Algoritmos de Recuperación de Información, operan de acuerdo a premisas básicas que asignan la importancia de los documentos. Dependiendo del conjunto de premisas, se tienen distintos modelos de recuperación de información.

Definición *Un modelo de recuperación de información es una cuádrupla [D, Q, F, $R(q_i, d_j)$]. donde:*

(1) **D** *es un conjunto de vistas lógicas (o representaciones) para los documentos en la colección.*
(2) **Q** *es un conjunto de vistas lógicas (o representaciones) para las necesidades de información del usuario.*
(3) **F** *es un marco para modelar la representación de los documentos, consultas y sus relaciones entre sí.*
(4) $\mathbf{R}(q_i, d_j)$ *es una función de rango que asocia un número real con una consulta* $q_i \in \mathbf{Q}$ *y una representación de documento* $d_j \in \mathbf{D}$. *Tal función define un orden entre los documentos considerados para la consulta* q_i.

Los modelos clásicos son: **Booleano, Vectorial** y **Probabilista**.
Estos modelos consideran que cada documento es descrito por un conjunto de palabras clave representativas llamadas *términos de índice*, estas palabras tienen una semántica que permite resaltar los principales temas de un documento.
Tomando un conjunto de términos índice para un documento, se tiene que no todos los términos son igualmente usados para describir su contenido. Existen términos que son más ambiguos que otros; decidir la importancia de un término para resumir el contenido de un documento no es un problema trivial. Una manera de resolverlo, es asignar un *peso* numérico a cada término índice de un documento.

Definición *Sea t el número de términos índice en el sistema y* k_i *un término índice genérico.* $K = \{ k_1, \ldots, k_t \}$ *es el conjunto de todos los términos índice. Un peso* $w_{ij} > 0$ *es asociado con cada término índice* k_i *de un documento* d_j. *Para un término índice, el cual no aparece en el texto del documento,* $w_{ij} = 0$. *Al documento* d_j *se asocia un vector término índice* $\vec{dj}$ *representado por:* $\vec{dj} = (w_{1j}, w_{2j}, \ldots, w_{tj})$. *Además, sea* g_i *una función que regresa el peso asociado con el término índice* k_i *en cualquier vector t-dimensional. Es decir,* $g_i(\vec{dj}) = w_{ij}$.

2. 2.1 Modelo Booleano

Este es un modelo de recuperación sencillo basado en el formalismo de la teoría de conjuntos y el álgebra booleana.
Las consultas son especificadas como expresiones booleanas, las cuales, tienen una buena precisión semántica; aunque no es fácil para cualquier usuario traducir una necesidad de información en una expresión booleana.
Su estrategia de recuperación se basa en un criterio de decisión binaria, esto es, un documento puede, o no, ser relevante. No existe la capacidad de tener una escala gradual que permita calificar a los documentos y determinar un buen desempeño de la recuperación.

Definición *Para el modelo Booleano, el peso del término índice es siempre binario, es decir, $w_{i,j} \in \{0,1\}$. Una consulta q es una expresión booleana convencional y se forma de términos índice ligados por tres conectivos: AND, OR y NOT. Sea $\vec{q}_{dfn}$ la forma normal disjunta para la consulta q. Además, sea $\vec{q}_{cc}$ cualquiera de los componentes conjuntivos de $\vec{q}_{dfn}$.*
La similitud de un documento d_j para una consulta q se define como:

$$sim(d_j, q) = \begin{cases} 1 \text{ si } \exists\ \vec{q}_{cc} \mid (\vec{q}_{cc} \in \vec{q}_{dfn}) \wedge (\forall k_i, g_i(\vec{d}_j) = g_i(\vec{q}_{cc})) \\ 0 \text{ de otro modo} \end{cases}$$

Si $sim(d_j, q) = 1$ entonces el modelo booleano predice que el documento d_j es relevante para la consulta q. En caso contrario, la predicción es que el documento no es relevante.

La aproximación booleana es la más común en la recuperación de información (buscadores en el Web y sistemas bibliográficos) y la más sencilla de implementar, pero tiene tres grandes problemas:

- La utilización de operadores lógicos es compleja y poco intuitiva. La mayoría de los usuarios interpretan el AND y el OR como el "*y*" y el "*o*" del lenguaje natural y no como operadores lógicos. Por ejemplo, la expresión en lenguaje cotidiano "agua o vino" suele interpretarse como una elección mutuamente exclusiva. Por si esto fuera poco, a muchos usuarios les resulta complicado el uso de paréntesis para evaluaciones anidadas, así como manejar las nociones de precedencia de los operadores.
- El operador AND restringe demasiado la consulta, mientras que el operador OR la restringe muy poco.
- En una consulta booleana pura, un documento o bien satisface la consulta o no la satisface, pero no permite ordenar los documentos recuperados según su grado de similitud con la consulta.

2.2.2 Modelo Vectorial

Este modelo reconoce que la asignación de pesos binarios a los documentos es muy limitado y propone un marco en el cual sean posibles emparejamientos parciales.
Esto se hace asignando pesos no binarios para términos índice en consultas y en documentos. Los pesos de los términos son usados para calcular el grado de similitud entre los documentos almacenados en el sistema y la consulta. Su principal ventaja es que la clasificación de documentos recuperados es un poco más precisa que la obtenida con el modelo booleano.

Definición *Para el modelo vectorial, el peso $w_{i,j}$ asociado con el par (k_i, d_j) es positivo y no binario. Además, a los términos índice en la consulta se les asignan pesos.*
Sea $w_{i,q}$ el peso asociado al par $[k_i, q]$, donde $w_{i,q} \geq 0$. Entonces, el vector consulta $\vec{q}$ es definido como $\vec{q} = (w_{1,q}, w_{2,q}, ..., w_{t,q})$ donde t es número total de términos índice en el sistema. De tal modo, que el vector para un documento d_j es representado por $\vec{d}_j = (w_{1,j}, w_{2,j}, ..., w_{t,j})$.

El modelo vectorial propone evaluar el grado de similitud del documento d_j con respecto a la consulta q como la correlación entre los vectores $\vec{d}_j$ y $\vec{q}$. Esta correlación puede ser cuantificada, por el coseno del ángulo entre estos dos vectores:

$$sim(d_j, q) = \frac{\vec{d}_j \cdot \vec{q}}{|\vec{d}||\vec{q}|}$$

$$= \frac{\sum_{i=1}^{t} w_{i,j} \cdot w_{i,q}}{\sqrt{\sum_{i=1}^{t} w^2_{i,j}} \cdot \sqrt{\sum_{j=1}^{t} w^2_{i,q}}}$$

Donde: $|\vec{d}_j|$ y $|\vec{q}|$ son los vectores de las normas del documento y la consulta.

Dado que $w_{i,j} \geq 0$ y $w_{i,q} \geq 0$, tendremos que $0 \leq sim(d_j, q) \leq 1$, teniendo así, que este modelo permite determinar un rango a los documentos de acuerdo a su grado de similitud con la consulta, por ello, un documento puede tener un *emparejamiento parcial* con la consulta.

Solución al problema de *Cluster* con el Modelo Vectorial

El problema de *Cluster*, consiste en la operación de agrupar documentos en clases similares o relacionadas. No es realmente una operación sobre el texto, sino una operación sobre una colección de documentos. Existen dos tareas importantes en este problema: La primera consiste en determinar las características que mejor describen a los objetos de un conjunto; a esto se le llama la *cuantificación de la similitud intra-cluster*. La segunda, consiste en determinar las características que mejor distinguen a los objetos en el conjunto determinado del resto del conjunto total, lo cual proporciona la *cuantificación de la distinción inter-cluster*.
Los métodos de cluster son frecuentemente empleados en la RI para transformar la consulta original en una tentativa por mejorar la representación de las necesidades del usuario. Desde esta perspectiva, la agrupación de documentos es una operación más relacionada con la transformación de la consulta que con la transformación del texto de los documentos.

En el modelo vectorial, la similitud de *intra-cluster* se cuantifica midiendo la frecuencia de un término k_i dentro de un documento d_j. Tal término de frecuencia es referido como el *factor tf* y proporciona la medida de qué tan bien el término describe el contenido del documento.

El *inter-cluster* se cuantifica calculando la inversa de la frecuencia de un término k_i entre los documentos de la colección. Este término es referido como la inversa de la frecuencia del documento o el *factor idf*. La relevancia de su uso radica en que los términos que aparecen en muchos documentos no se usan para distinguir entre un documento relevante de uno que no lo es.

Definición *Sea N el número total de documentos en el sistema y sea n_i el número de documentos en los cuales el término índice k_i aparece. Sea $freq_{i,j}$ la frecuencia del término k_i en el documento d_j (es decir, el número de veces que el término k_i se menciona en el texto del documento d_j). Entonces, la frecuencia normalizada $f_{i,j}$ del término k_i en el documento d_j está dada por:*

$$f_{i,j} = \frac{freq_{i,j}}{\max_l freq_{l,j}}$$

Donde el máximo se calcula sobre todos los términos que se mencionan en el texto del documento d_j. Si el término k_i no aparece en el documento d_j, entonces $f_{i,j} = 0$. Además, sea idf, la frecuencia inversa del documento para k_i, está dada por:

$$\text{idf}_i = \log\left(\frac{N}{n_i}\right)$$

El mejor esquema usa pesos dados por:

$$w_{i,j} = f_{i,j} \log\left(\frac{N}{n_i}\right)$$

Las principales ventajas del modelo son las siguientes:

- El esquema de pesos proporciona un buen rendimiento en la recuperación.
- La estrategia parcial de emparejamiento permite recuperar documentos que se aproximen a las condiciones de búsqueda.
- La clasificación de la fórmula del coseno ordena los documentos de acuerdo al grado de similitud con la consulta. Se calcula el coseno del ángulo entre los mismos, que será un valor entre 0 y 1. Cuando es 0 indica que los dos vectores no tienen términos en común, y cuando es 1 indica que el ángulo entre los vectores es 0, es decir, que el documento y la consulta tienen los mismos términos y con los mismos pesos.

Teóricamente, el modelo vectorial tiene la desventaja de que los términos índice se asumen mutuamente independientes. Pero en la práctica, la consideración de los términos dependientes puede ser una desventaja.

En este modelo no sólo se usa un vector como medio para representar adecuadamente la significación de cada término de la colección en un documento determinado sino que, además, se considera que ese vector es un objeto geométrico real en el espacio n-dimensional. El modelo del espacio vectorial puede usar diferentes medidas de similitud o de cálculo de la cercanía entre los vectores del documento y la consulta.

2.2.3 Modelo Probabilístico

Este modelo se introdujo en 1976 e intenta captar el problema de la recuperación de información desde un punto de vista probabilístico.

Dada una consulta q de usuario y un documento d_j de la colección, el modelo probabilístico trata de estimar la probabilidad de que el usuario encuentre el documento d_j relevante. El modelo asume que esta probabilidad de relevancia depende de la consulta y la representación del documento. Además, se asume que existe un subconjunto R, con un máximo de probabilidad para todos los documentos que el usuario requiere como respuesta a su consulta q. A los documentos fuera del subconjunto R se les llama *no-relevantes*.

Definición *Para el modelo probabilistico. los pesos para los términos índice son todos binarios, es decir, $w_{i,j} \in \{0,1\}$, y asi $w_{i,q} \in \{0,1\}$. Una consulta q es un subconjunto de los términos índice. Sea R el conjunto de documentos esperado como relevante. Sea $\bar{R}$ el complemento de R (i.e., el conjunto no-relevante.) Sea $P(R \mid \vec{d}_j)$ la probabilidad de que el documento d_j sea relevante para la consulta q y $P(\bar{R} \mid \vec{d}_j)$ es la probabilidad de que d_j no sea relevante para q. La similitud del documento d_j con la consulta q se define como la razón*

$$sim(d_j,q) = \frac{P(R \mid \vec{d}_j)}{P(\bar{R} \mid \vec{d}_j)}$$

Usando la regla de Bayes, tenemos

$$sim(d_j,q) = \frac{P(\vec{d}_j \mid R)\ P(R)}{P(\vec{d}_j \mid \bar{R}) P(\bar{R})}$$

$P(\vec{d}_j \mid R)$ establece la probabilidad de seleccionar aleatoriamente el documento d_j del conjunto R de documentos relevantes. P(R) establece la probabilidad de que un documento aleatoriamente seleccionado de la colección completa sea relevante. Los complementos se representan con $P(\vec{d}_j \mid \bar{R})$ y $P(\bar{R})$ respectivamente.

La principal ventaja del modelo es que los documentos tienen una clasificación que permite ordenarlos de acuerdo a su probabilidad de relevancia.

Sus desventajas son:

1) La necesidad de proporcionar una separación inicial de documentos en dos conjuntos: Los relevantes y los no-relevantes.
2) El modelo no toma en cuenta la frecuencia con la que ocurre cada término índice en un documento (i.e., los pesos son binarios)
3) Parte de la idea de independencia entre los términos índice.

La importancia de las medidas de similitud definidas en todos los modelos es que se puede establecer un límite para el conjunto de los documentos recuperados. Asi, se puede establecer que el conjunto de documentos recuperados sea aquél cuyas representaciones mostraron un valor de similitud mayor a un determinado valor con respecto a la petición. Otra alternativa, es presentar los primeros N documentos cuyas representaciones mostraron un valor de similitud con respecto a la de la petición.

También es conveniente que los documentos recuperados sean presentados al usuario en orden descendiente con respecto a los valores de similitud que mostraron sus representaciones con la petición de búsqueda. Esto es de gran importancia en una situación de recuperación interactiva, pues nuevas y mejores formulaciones de peticiones de búsqueda pueden ser construidas a partir de la información obtenida de los documentos previamente recuperados.

A través de los años, en todos los modelos se han presentado nuevos paradigmas alternativos, la figura 2.2 muestra esta evolución. Estos nuevos modelos parten de una idea base con algunas modificaciones que la mejoran.

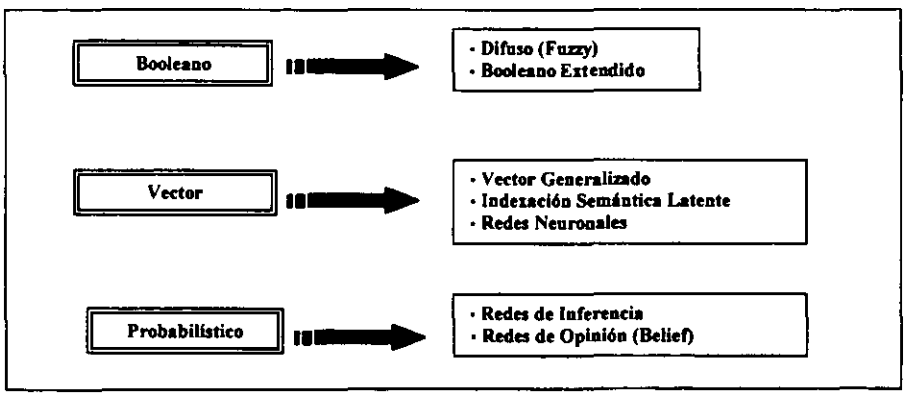


Figura 2.2 Modelos de Recuperación de Información

Para los modelos básicos de recuperación de información, la recuperación de palabras clave es la principal tarea para satisfacer una consulta.

2. 3 Lenguajes de consulta

Una *consulta* es la formulación o planteamiento de una necesidad de información de un usuario. Las consultas más elementales que se pueden formular en un sistema de recuperación de texto son de una palabra, pero éstas también puede ser en general una combinación más compleja de operaciones que involucran varias palabras.

2.3.1 Las consultas booleanas

La forma más antigua de combinar palabras en una consulta es por medio de los operadores booleanos. Así, una consulta booleana tiene una sintaxis compuesta de átomos (consultas básicas) que recuperan documentos, y de operadores booleanos que trabajan sobre sus operandos (conjuntos de documentos).Se dice que este esquema de consulta es *compuesto*, debido a que los operadores pueden estar compuestos de los resultados de otros operadores; se usan para ello los *árboles de sintaxis de la consulta*, cuyos niveles corresponden a las consultas básicas y los nodos internos a los operadores. El árbol opera sobre los conjuntos por medio de una álgebra de los documentos obteniendo como respuesta a la consulta, también un conjunto de documentos. La figura 2.3 muestra un ejemplo de un árbol que permite la recuperación de documentos que contengan la palabra "análisis" así como la palabra "léxico" o la palabra "sintáctico".

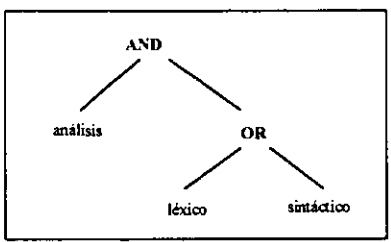


Figura 2.3 Árbol de sintáxis de una consulta.

Los operadores booleanos requieren de al menos dos consultas básicas o subexpresiones booleanas e_1 y e_2:

- **OR**. La consulta (e_1 OR e_2) selecciona todos los documentos que satisfacen e_1 o e_2. Duplicados se eliminan.
- **AND**. La consulta (e_1 AND e_2) selecciona todos los documentos que satisfacen tanto e_1 como e_2.
- **BUT**. La consulta (e_1 BUT e_2) selecciona todos los documentos que satisfacen e_1 pero no e_2. La lógica booleana clásica usa la operación NOT, donde la sentencia (NOT e_2) significa siempre que no ocurra e_2. El operador BUT restringe el universo de los elementos de recuperación al resultado e_1.

2.3.2 Búsqueda de Patrones

El planteamiento de consultas específicas, basadas en la búsqueda de patrones, permite la recuperación de piezas de texto con ciertas propiedades.
Estos tipos de consultas se emplean por los lingüistas, en estadísticas de texto, y extracción de texto. Sus resultados se obtienen por mecanismos de composición que forman frases y por consultas de proximidad.

Un *patrón* es un conjunto de características sintácticas que deben ocurrir en un segmento de texto. Tales segmentos, que coinciden con un determinado patrón, se dice que se emparejan. El interés del sistema de recuperación, es obtener aquellos documentos que contienen los segmentos que coinciden con el patrón.

Aunque existen muchos modelos para encontrar palabras similares, el más aceptado es el llamado "***Distancia de Levenshtein***" o "***Distancia de edición***". Se define este método como: el número mínimo de caracteres insertados, borrados o reemplazados entre dos cadenas, tales que éstas sean iguales entre sí.
Así, la consulta especifica el número máximo de errores permitidos para una palabra que se emparejará a un patrón; es decir, la consulta especifica la distancia de edición máxima. Este modelo se puede extender a la búsqueda de subcadenas, no solo de palabras; recuperando cualquier segmento de texto que se apegue a una distancia de edición del patrón de búsqueda determinado.

Cada sistema permite la especificación de algunos tipos de patrones, desde su forma más simple como lo pueden ser las palabras; hasta las más complejas como lo son las expresiones regulares.

Los tipos de patrones más usados son: Palabras, Prefijos, Sufijos, Subcadenas, Rangos, Expresiones Regulares y Errores Permitidos.

- La ***palabra*** es una secuencia finita de caracteres pertenecientes a un alfabeto.

- El ***prefijo*** es una cadena con la cual debe empezar una palabra. Por ejemplo, dado el prefijo "*univer*", todos los documentos que contengan palabras como: "*universal*", "*univer*sidad", "*univer*sitario", "*univer*so", etc., se recuperarán.

- El ***sufijo*** es una cadena, la cual, debe formar la terminación de la palabra. Así, por ejemplo, se tiene que dado el sufijo "*ción*", se recuperarán los documentos que contengan las palabras: "interac*ción*", "computa*ción*", "extrac*ción*", "can*ción*", etc.

- Las ***subcadenas*** son cadenas que pueden formar parte de una palabra. Dada la subcadena "*rica*", se recuperarían los documentos que contengan las palabras: "fab*rica*nte", "ame*rica*no", "a*frica*na", "colé*ricas*", etc.

- Los ***rangos*** son un par de cadenas, que representan un límite superior e inferior, donde se considera que una cadena empareja si se encuentra entre el intervalo lexicográfico marcado. Este orden se tiene para todos los alfabetos e induce a que las cadenas sigan un orden. Se tiene entonces, por ejemplo, que dado el rango entre las palabras "*caoba*" y "*cuna*", se recuperarían los documentos que contengan palabras como: "*casa*", "*censo*", "*ciudad*", "*convento*", etc.

- Los ***errores permitidos*** son aquellas palabras que se encuentran dentro de un umbral de error. Estos patrones de búsqueda recuperan todas las palabras "similares" a una palabra dada. El concepto general permite que el patrón tenga errores (tales como de captura, ortografía, entre otros) y la consulta trate de recuperar la palabra dada y aquellas otras parecidas con sus variantes de errores.
  Aplicando este criterio, si se aplica a la palabra "*so nido*" considerando que existe un error, se pueden recuperar los documentos que contengan la palabra "*sonido*".

- Las ***Expresiones Regulares*** son la construcción de un patrón en general a partir de cadenas simples basadas en los siguientes operadores:
  - Unión: si e_1 y e_2 son expresiones regulares, entonces (e_1 + e_2) significa que el emparejamiento se presenta por la existencia de e_1 o de e_2.
  - Concatenación: si e_1 y e_2 son expresiones regulares, las ocurrencias de (e_1 e_2) se forman por la ocurrencia de e_1 inmediatamente seguida de e_2.

- Repetición: si e es una expresión regular, entonces (e^*) empareja con las secuencias de cero o más ocurrencias continuas de e.

Se tiene la siguiente consulta: "pro (blema + teína) (s + ε) (0 +1+2) * " , donde ε denota la cadena vacía (algunas veces denotada por λ). Entonces, este patrón empareja con palabras tales como: *"problema231"*, *"proteína"*, *"problemas"*, *"problema"*, *"proteína2"*, etc.

- Los ***Patrones Extendidos*** se emplean para proporcionar un lenguaje de consulta más amigable al usuario que le permita representar algunos casos de las expresiones regulares. La conversión de patrones extendidos a expresiones regulares se efectúa a través del sistema de recuperación o se aplican algoritmos específicos para su búsqueda. Cada sistema soporta su propio conjunto de patrones extendidos, por ello, no existe una definición sintáctica. Algunos ejemplos son:
  - Clases de caracteres, i.e. una o más posiciones en un patrón empareja con cualquier carácter de un conjunto predefinido. Esto involucra características tales como casos insensibles, uso de rangos de caracteres (especificar por ejemplo que algunos caracteres pueden ser un dígito), complementos (indicar por ejemplo que algunos caracteres no pueden ser una letra), numeración (definir que un caracter debe ser una vocal), comodines (i.e. una posición en un patrón empareja con cualquier cosa), etc.
  - Expresiones condicionales, i.e. donde una parte del patrón puede o no aparecer.
  - Caracteres comodín, los cuales emparejan cualquier secuencia en el texto, por ejemplo, cualquier palabra que empiece con "ilu" y termine con "ión", puede emparejar con "iluminación" así como con "ilusión".
  - Combinaciones que permitan que algunas partes del patrón emparejen exactamente y otras partes con error.

2.3.3 Consultas por Proximidad

El resultado de una consulta por palabras, es el conjunto de documentos que contienen cuando menos una de las palabras planteadas, dependiendo de su rango de similitud.
La división del texto en palabras no es arbitraria, debido a que las palabras implican muchos significados en el lenguaje natural.

Algunos sistemas complementan las consultas de una palabra con la posibilidad de buscar palabras en un determinado contexto, lo cual se da por la *proximidad* de unas palabras cerca de otras o bien por la formación de frases.

Se tiene *proximidad* cuando no se obliga necesariamente a tener una frase. En este caso, se tiene una secuencia de palabras o frases con una distancia máxima permitida entre ellas. Esta distancia se puede medir en caracteres o palabras dependiendo del sistema.
Otro factor a considerar es que el orden de aparición de las palabras puede no ser el mismo que el definido por la consulta.

El modelo de conjunto difuso *(Fuzzy Set Model)* propone una representación de clases cuyos límites no están bien definidos. La idea principal es asociar a cada valor de la función con los elementos de las clases. Esta función toma valores dentro del rango [0,1] correspondiendo un 0 a los elementos que no pertenecen a la clase y un 1 a los pertenecientes a la clase. Aquellos miembros cuyos valores de la función estén entre 0 y 1 se llaman *elementos marginales*.
Así, pertenecer a un conjunto difuso es una noción *gradual* a diferencia de una decisión brusca como ocurre con el modelo Booleano original.

Los conjuntos difusos permiten representar imprecisiones aplicadas a varios dominios.
El uso de adoptar un *Thesaurus* es una manera de modelar el proceso de recuperación de información con términos relacionados. La idea es expandir el conjunto de términos índice de la consulta con los términos encontrados en el *thesaurus*, los cuales agregarán documentos relevantes.
El thesaurus puede ser construido definiendo término a término una matriz de correlación c, también llamada *Matriz de Conexión de Palabras Clave*, cuyos renglones y columnas son asociados a los términos índice de la colección de documentos. En esta matriz, el factor normalizado de correlación $c_{i,l}$ entre dos términos k_i y k_l puede definirse por

$$c_{i,l} = \frac{n_{i,l}}{n_i + n_l - n_{i,l}}$$

donde n_i es el número de documentos que contienen el término k_i, n_l es el número de documentos que contienen el término k_l y $n_{i,l}$ es el número de documentos que contienen ambos términos.

Se puede definir el término de matriz de correlación $\vec{c}$ para definir un conjunto difuso asociado a cada término índice k_i. En este conjunto difuso, un documento d_j tiene un grado de pertenencia

$$\mu_{i,j} = \prod_{k_l \in d_j} (1 - c_{i,l})$$

2.3.4 Thesaurus

En su forma más simple, un *thesaurus* consta de una lista de palabras importantes en un campo del conocimiento y cada una de ellas es asociada a un conjunto de palabras relacionadas las cuales son el resultado de una relación de sinonimia. Algunos *thesaurus* no solo se apegan a la asociación de sinónimos, sino de frases cuyo significado es un concepto más complejo.

La ventaja de tener un *thesaurus* es el empleo de un vocabulario controlado (también llamado *Sistema de Referencia*) para la búsqueda e indexación. Con esta normalización de conceptos se pueden tener grandes logros como:

- La asistencia al usuario en la localización de términos para una consulta apropiada.
- La reducción del "ruido" en la recuperación.
- La identificación de términos índice con un claro significado semántico y por ende, una recuperación basada en conceptos más que en palabras.

Frecuentemente, en la creación del *thesaurus* es necesario proporcionar una breve explicación o definición para cada término, pues esta especificación precisa el significado de un término en el contexto del *thesaurus* utilizado.
En un campo del conocimiento específico es mucho más sencillo establecer el *thesaurus*, mientras que en un ámbito general es difícil pues no existe un límite bien definido. Otros factores que dificultan el mantenimiento del *thesaurus* son: La constante incorporación de documentos, los documentos muy extensos o los documentos dinámicos en los que se presentan cambios frecuentes. En el Web son justamente estos factores los que complican su uso.

2.3.5 Consultas sobre texto estructurado

En los tipos de consultas anteriores se partía de un conjunto de documentos que podían ser recuperados considerando su contenido. Pero este modelo es incapaz de aprovechar las características de estructura que tienen algunos textos, donde se debe considerar este factor en las consultas de los usuarios, las que serán más costosas.

Algo importante en el manejo del texto estructurado es la diferencia que existe entre la estructura que se tiene y las posibles consultas que se pueden efectuar ya que esto cambia dependiendo del modelo empleado.
Por ejemplo, si en un artículo se necesita una estructura de secciones y subsecciones, pero el modelo de la consulta no permita estructuras recursivas. En este caso, no se podrán consultar secciones incluidas en otras.

A continuación, se describen las principales estructuras que presentan los documentos y en la figura 2.4 se aprecian de manera gráfica dichas estructuras.

2.3.5.1 Texto con estructura fija

La estructura del texto estuvo en un principio, muy restringida. Los documentos tenían campos fijos, similares a las formas de llenado para solicitudes; en los campos se almacenaba el texto. Las consultas permitían la búsqueda de un patrón determinado sobre un campo.

Las limitantes que se presentaban en esta estructura: No todos los documentos tenían la misma estructura (los mismos campos), los campos podían aparecer en orden distinto o repetirse en algunos documentos, un texto no podía tener texto no clasificado bajo un campo y no se permitían campos anidados. Actualmente, con el uso de Internet, existen muchos archivos en HTML, donde este modelo no es adecuado para representar la estructura jerárquica que se tiene en este tipo de archivos.

Este modelo es razonable cuando la colección de texto posee una estructura definida y constante como ocurre en un *archivo de correos electrónicos*, donde cada correo tendrá los siguientes campos: un remitente, un receptor, una fecha, un tema y un cuerpo. Así, una búsqueda puede ser la recuperación de todos los correos electrónicos enviados con el tema: "*Diplomado LANIA* ".

Si la división de los campos puede ser determinada y el contenido de algunos campos puede ser interpretado como un valor numérico o fecha, en lugar de texto, se aprecia entonces la facilidad de almacenar estos datos en una ***base de datos relacional***, donde cada campo (y su tipo de dato respectivo), tendrá un lugar dentro de la columna de una tabla. La combinación de estos tipos de datos (columnas tipo texto y columnas de otros tipos de datos simples), ha sido el trabajo de los últimos años, ya que el rendimiento óptimo del sistema, se ve afectado al querer administrar ambos dominios de datos.

2.3.5.2 El Hipertexto

El hipertexto podría ser la máxima libertad con respecto a la estructura. Es una gráfica dirigida donde cada nodo almacena texto y las ligas representan conexiones entre nodos o las posiciones entre ellos. El hipertexto ha recibido mucha atención desde la explosión del Web, el cual es en realidad, una enorme base de datos de texto al rededor del mundo.

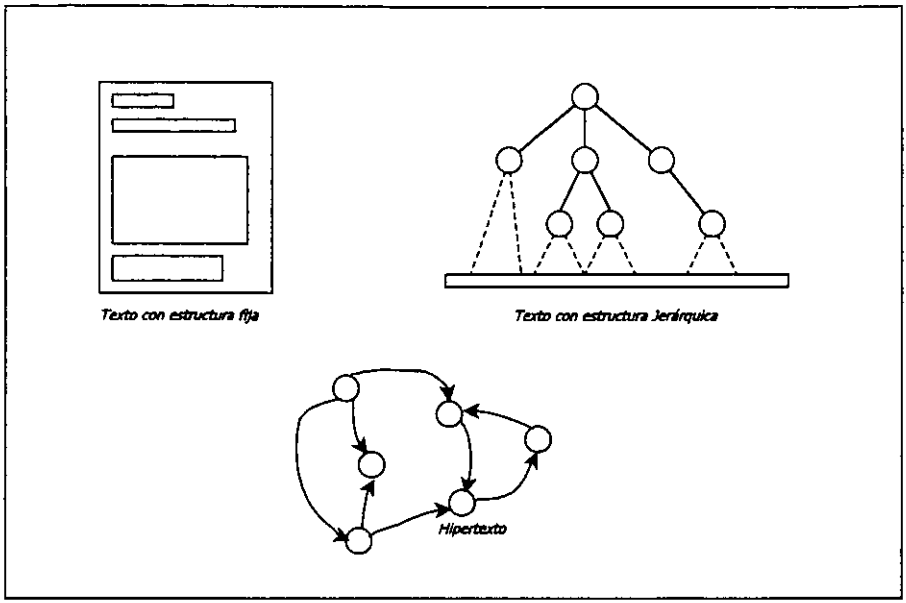


Figura 2.4 Estructuras presentes en un texto

Sin embargo, la recuperación del texto inició como una actividad navegacional. Siguiendo las ligas, el usuario puede recorrer los nodos para buscar lo que necesita. Aunque en el Web se puede buscar por el contenido del texto de los nodos, no se puede hacer por la estructura de conectividad de los nodos vecinos.
Un problema interesante es entonces, la posibilidad de combinar la búsqueda y la navegación. Actualmente, algunas herramientas de consulta tienen el objetivo de cubrir ambas necesidades.

2.3.5.3 Estructura Jerárquica

Un modelo intermedio entre el estructurado y el hipertexto es el jerárquico. Representa una descomposición del texto y es un modelo natural para muchas colecciones de texto como: libros, artículos, documentos legales, programas estructurados, etc.
Un ejemplo de esto, lo tenemos en la figura 2.5, que muestra la página de un libro, su esquema y un árbol de análisis sintáctico para una consulta determinada.
Se aprecia, la estructura de un libro: contiene capítulos, los capítulos están formados por secciones las cuales tienen títulos y dentro de ellas existen figuras. Así, si se desea buscar el patrón "*características*" dentro del título de una sección, se puede obtener por el recorrido del árbol que refleja la jerarquía de los objetos.

El hecho de poder asignar una jerarquía al texto, proporciona también la posibilidad de crear algoritmos más rápidos para responder a las consultas.

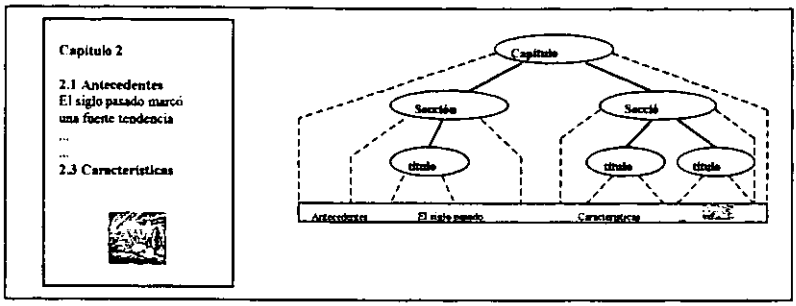


Figura 2.5 Ejemplo de estructura jerárquica

2.4 SFQL como lenguaje de consulta (*Structured Full-text Query Language*)

Las consultas sobre texto almacenado dentro de una tabla de una base de datos relacional se plantea con el lenguaje SFQL, el cual está basado en SQL y también tiene una arquitectura cliente-servidor. SFQL ha sido adoptado como un estándar por la comunidad aeroespacial (*The Air Transport Association*) Los documentos son renglones en una tabla relacional y pueden ser etiquetados usando SGML. El lenguaje no define un formato específico o marca. Por ejemplo, una consulta en SFQL es la siguiente:

SELECT resumen
FROM periodico
WHERE encabezado **CONTAINS** "búsqueda de texto"

El lenguaje soporta operadores lógicos y booleanos, thesaurus, operaciones de proximidad y algunos caracteres como comodines y de repetición. Ejemplo:

SELECT resumen
FROM periodico
WHERE encabezado **CONTAINS** "base de datos textual" **OR LIKE** "info%"
AND fecha > 1/1/99

SFQL se basa en los llamados protocolos de consulta que son los lenguajes de consulta utilizados automáticamente por las aplicaciones de software para consultar bases de datos de texto. Debido a que estos lenguajes no son para uso humano, se les llaman protocolos. Los más importantes son:

- ***Z39.50*** Es un protocolo aprobado como estándar por ANSI en 1995. Este protocolo permite consultar información bibliográfica usando una interfaz entre el cliente y el servidor donde se encuentra el manejador de la base de datos, es independiente tanto de la interfaz del cliente como del lenguaje de consulta en el servidor. Se asume que la base de datos es una colección de texto con algunos campos fijos. Este protocolo no solo especifica el lenguaje de consulta y su semántica sino también la manera en la cual el cliente y el servidor establecen una sesión,

comunicación, intercambio de información, etc. Aunque este protocolo originalmente se utilizó para información bibliográfica, se ha extendido para consultar otros tipos de información.

- *WAIS* (*Wide Area Information Service*) Es una serie de protocolos muy populares a inicios de 1990 antes de la explosión del Web. La meta fue tener un protocolo público de red disponible para consultar bases de datos a través de Internet.

2.5 Vista lógica de un documento: Los términos índice

Un documento es una unidad de información que generalmente se presenta como texto en su forma digital. Puede ser una unidad lógica completa como un artículo, un reporte técnico, un libro, un manual, etc. o quizá parte de un texto extenso como un párrafo o una secuencia de párrafos. Con respecto a su representación física, un documento puede ser una unidad física como lo es un archivo.

Un documento tiene una sintaxis y una estructura determinada por cada aplicación particular. Además de una presentación de estilo asociado, también posee una semántica especificada por el autor del documento,

Los SRIs efectúan búsquedas sobre documentos, dependiendo de cómo realizan estas búsquedas, se pueden dividir en dos categorías:
a) Los que emplean índices para la búsqueda de la información. Estos sistemas no utilizan los textos de los documentos para decidir qué documentos están más relacionados con una petición de búsqueda dada. En su lugar, se construye por cada documento una representación, la cual es la que realmente se emplea al momento de la búsqueda. A esta tarea de representación de documentos se le llama ***proceso de indexación***, que es realizado una sola vez con anterioridad a cualquier consulta.
b) *Los que no utilizan índices.* Estos sistemas se basan en algoritmos rápidos de búsqueda de cadenas de caracteres. Este tipo de algoritmos se emplea comúnmente para el desarrollo de rutinas de búsqueda de editores de texto y manuales en línea.

El *proceso de indexación* puede ser realizado de manera manual o automática, cuyo objetivo en ambos casos, es obtener un documento representativo formado por los términos del lenguaje de indización. La indización manual, como su nombre lo indica, se realiza de forma manual y la lleva a cabo personal capacitado por medio de una serie de herramientas como listas de terminología, manuales de instrucción y hojas de trabajo estructuradas para registrar los productos de indización.
La indización automática se realiza por medio de algoritmos, algunos muy simples que se limitan a indexar un documento con un término dado si este término (o su prefijo) aparece tal cual en el documento u otros muy sofisticados que son capaces de "comprender" la morfología del idioma en que están escritos los documentos. Cualquier clase de estos algoritmos es capaz de optimizar la lista de términos de la representación de un documento.

La selección de **términos índice** implica escoger aquellas palabras que pueden describir al documento de manera semántica. Una posible opción es considerar que todas las palabras son significativas, pero es lógico no tener en cuenta palabras sin significado propio, tales como artículos, preposiciones, conjunciones, etc.

En los lenguajes escritos, algunas palabras poseen un mayor significado que otras. Generalmente, los *sustantivos* son las palabras más representativas del contenido de un documento; por lo tanto, los

sustantivos son considerados para preprocesar el texto de los documentos para determinar los términos que serán usados como ***términos índice***.

Cuando un grupo de sustantivos se adopta como términos índice, se obtiene una **vista lógica conceptual** del documento. Al ser común combinar dos o tres sustantivos en un solo componente (por ejemplo en *"Ciencias Computacionales"*), se ha optado por formar grupos de sustantivos vecinos o cercanos en un texto como un solo término índice; así, estos grupos son un conjunto de sustantivos cuya distancia sintáctica en el texto (medida del número de palabras entre dos sustantivos) no excede un valor determinado.

Se suelen utilizar **índices invertidos**, que almacenan, para cada término en la colección de documentos, una cadena que identifica cada documento en el que aparece el término y su posición. De esta manera ante una consulta de un usuario no es necesario buscar en todos los documentos. Bastaría examinar el índice correspondiente a las palabras presentes en la consulta y localizar así los documentos, y la posición dentro de éstos, en donde se encuentran los términos buscados.

2.6 El preprocesamiento del texto

Usar el conjunto de todas las palabras de los documentos para indexarlos genera mucho "ruido" para la tarea de recuperación. Una manera de disminuir este "ruido" es reducir el conjunto de palabras que pueden ser referenciadas en los índices. Así, el **preprocesamiento** de los documentos en la colección debe ser visto simplemente como un proceso para controlar el tamaño del vocabulario (i.e. el número de palabras distintas usadas como un término índice.) Lo anterior, introduce un paso adicional en el proceso de indexación que no percibe el usuario.

Preprocesar un documento puede dividirse principalmente en cinco operaciones de texto también llamadas transformaciones:

(1) **Selección de los términos índice**

(2) **Construcción de estructuras de categorías de términos**

(3) **Análisis léxico del texto**

(4) **Eliminación de *stopwords***

(5) **Obtención de la raíz de las palabras**

2.6.1 Análisis Léxico

El análisis léxico es el proceso de convertir un grupo de caracteres (el texto de un documento) en un grupo de palabras, las cuales serán candidatas a formar parte de los términos índice. Así, uno de los objetivos de este análisis es la identificación de las palabras del texto. Aunque todo parece indicar que el reconocimiento de palabras está dado por los espacios en blanco al ser usados como delimitadores o separadores, se deben considerar factores como: la reducción de varios espacios a uno solo, la aparición de dígitos, guiones, signos de puntuación y la diferencia entre letras mayúsculas y minúsculas.

Los números no se consideran buenos términos índice debido a que sin un contexto que los acompañe, son ambiguos. Un tratamiento preliminar para el tratamiento de los dígitos en el texto, puede ser la eliminación de todas las palabras que contengan una secuencia de dígitos a menos que se especifique de otra manera (a través de expresiones regulares). Sin embargo, se les debe

considerar en muchos casos cuando van acompañados de alguna o varias palabras, como por ejemplo: 500 D.C., 450 A.C., 1052-A, 3B, 3° A, 42°C, 10 °F, etc.
Un procedimiento avanzado de análisis léxico debe efectuar una normalización para unificar algunos formatos.

Los guiones son otra difícil decisión para el analizador léxico. Algunas palabras se separan por medio de guiones debido a un uso inconsistente; por ello, en algunas palabras los guiones deben formar parte integral de las mismas. Por ejemplo en algunas palabras como: anti-derrapante, pre-inscripciones, pos-operatorio, etc. Así, el procedimiento más fácil es adoptar una regla general y especificar las excepciones a partir de un caso base.

Los signos de puntuación se podrían eliminar por completo del análisis léxico, pero el problema es que muchos signos de puntuación son parte integral de una palabra como los casos 500 D.C. y 450 A.C., anteriormente considerados.
Eliminar los puntos parece no tener un impacto en la recuperación porque el riesgo de perder la interpretación es mínimo. Pero en algunos escenarios, será conveniente la utilización de una lista de excepciones.

El uso de mayúsculas y/o minúsculas, no afecta en mucho al análisis léxico pues generalmente aplica funciones de conversión al texto. Escenarios particulares, pueden no verse satisfechos con esta conversión de caracteres y es entonces conveniente que el usuario lo determine. Algunas palabras ejemplo de estos casos lo son: Banco y banco, cuyas diferencias son considerables ya que el primero se puede referir a una institución bancaria y el segundo, a un asiento de madera.

Aunque las tareas del análisis léxico no tienen mayor problema en su implementación, se deben cuidar, pues representan un impacto en el tiempo de recuperación. Es en ellas, donde muchas veces el usuario no es conciente de lo que hace la estrategia de indexación. Algunas máquinas de búsqueda del Web han optado por evitar estas operaciones sobre el texto, simplificando así la interpretación de lo solicitado por el usuario, pero recuperando así una enorme lista de información no deseada.

2.6.2 "Palabras de parada " o No Significativas (*Stopwords*)

Las palabras que aparecen de manera frecuente en los documentos (por ejemplo en un 80%), no son generalmente empleadas o se consideran irrelevantes en la tarea de recuperación, estas palabras no significativas reciben el nombre de "*palabras de parada*" o "*diccionario negativo*" (o *stop wordlist*) y no se consideran como parte de los términos índice con el objetivo de que no sean procesadas y obtener así una mayor eficiencia en las búsquedas al disminuir de manera considerable el tamaño de la estructura del índice.

La lista de palabras de parada incluye a los artículos, preposiciones y conjunciones; esta lista puede obtenerse manualmente o con base a un análisis estadístico de los documentos. Pueden agregarse a ella algunos verbos, adverbios y adjetivos. También se recomienda que si aparecen términos fuera del campo semántico o son muy frecuentes y pudieran representar un término poco relevante, se incluyan en la lista de parada.
Un ejemplo de términos frecuentes en una base de datos de temas de computación, no sería pertinente incluir en el índice aquellas palabras muy frecuente en su campo semántico como: "computadora", "compilador", "sistemas", "programa", "lenguaje", etc.

De manera inversa, si se trata por ejemplo, de una base de datos de temas de astronomía, sería adecuado no considerar a aquellas palabras poco frecuentes en el texto, tales como: "virus", "guerra", "política", "arteria", etc.

Por esta razón, las máquinas de búsqueda del Web adoptan un índice completo del texto, es decir, se consideran todas las palabras que forman a un documento como parte del índice.
En el Anexo B se proporciona un listado de las palabras de parada más frecuentes en inglés y en el Anexo C se proporciona una lista de parada de términos en español.

2.6.3 Raíz de las palabras (*Stemming*)

Frecuentemente, el usuario especifica una palabra en la consulta y solo el patrón exacto de esa palabra será recuperado; sus diferentes variaciones morfológicas (conjugación, género, número, diminutivos, sufijos, etc.) se consideran irrelevantes siendo que éstas variantes deberían también ser recuperadas y consideradas como parte de lo que al usuario le interesa obtener. Este problema puede verse resuelto con la sustitución de las palabras por su raíz sintáctica. La raíz es la porción que se encuentra de lado izquierdo de una palabra, previa a sus afijos (prefijos y sufijos). Un ejemplo de este concepto se muestra en la figura 2.6, donde se aprecian las palabras variantes que tienen como raíz la palabra *Niñ*.

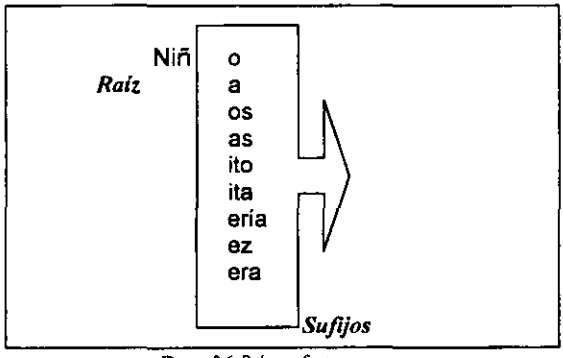


Figura 2.6 Raíz y sufijos

Se puede apreciar que la obtención de la raíz de las palabras permite obtener un concepto en común y como segundo gran beneficio está la reducción de la estructura del índice.

Todas las palabras de una misma raíz se consideran idénticas a efectos de una búsqueda. La tarea de asignación de las raíces de un conjunto de palabras es realizada por el llamado ***stemmer***. Su tarea se realiza, en la mayoría de los casos, eliminando el sufijo de la palabra y sustituyendo los caracteres finales por otros. Esto se hace con base a las reglas gramaticales del idioma en que estén escritos los documentos. Sin embargo, este tipo de obtención de la raíz no es del todo válido, ya que existen variaciones de una misma palabra que no deberían pertenecer a la misma raíz, por ejemplo entre las palabras: "*hábito*"y "*habitable*", que tienen raíces distintas. También existen palabras que no tienen la misma semántica pero deberían pertenecer a la misma raíz como ocurre con las palabras: "*voy*" y "*fui*" que deben pertenecer al lema "*ir*".

Otro aspecto a considerar es que ciertas combinaciones de palabras deben representarse mediante un único término. Por ejemplo, "*sistema operativo*" tiene un significado diferente a las palabras "*sistema*" y "*operativo*" por separado.

Querer encontrar la raíz de las palabras ha sido un tema polémico en cuanto a su valor significativo en la tarea de recuperación. William Frakes [FrBa92], muestra las comparaciones de ocho estudios experimentales en los que se aprecia un verdadero beneficio de su uso, sin embargo, los resultados de esos estudios no permiten obtener una conclusión satisfactoria. Como resultado de esta polémica, muchas máquinas de búsqueda en el Web no adoptan ningún "A*lgoritmo de stemming*".
Existen cuatro estrategias para determinar la raíz de las palabras.

1. *Afijos removibles.* Es una técnica intuitiva, simple y puede ser implementada de manera eficiente. La parte más importante es remover el sufijo porque las variantes de una palabra se generan por la introducción de sufijos.
2. *Tabla de búsqueda.* Consiste en buscar la raíz de una palabra dentro de una tabla definida (similar a un catálogo) a partir de un idioma específico.
3. *Sucesor variante.* Está basado en la determinación de los límites del morfema, emplea el conocimiento de las estructuras lingüísticas y es más complejo que los algoritmos para la estrategia de afijos removibles.
4. *n-Gramas.* Se basa en la identificación de digramas y trigramas para poder descomponer una palabra; es más que un procedimiento para clasificar términos bajo una raíz.

Después de eliminar las palabras no significativas y su raíz, cada documento queda representado por la lista de términos que contiene. El sistema de recuperación puede asignar ***pesos*** de forma automática a los términos, para así poder caracterizarlos con diferentes ***grados de significación***, en función del número de veces que aparece en el documento y la petición de búsqueda.
Cada documento puede representarse entonces por un vector *n-dimensional*, como el que se muestra en la figura 2.7.

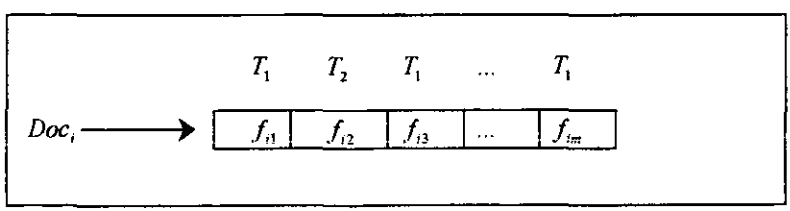


Figura 2.7 Construcción de la representación del i-ésimo documento de una colección, donde m es la cardinalidad del lenguaje de indexación y $f_{i,j}$ es la ponderación del término T_j dentro del i-ésimo documento doc_i

Una vez que cada documento queda identificado por su vector, la colección puede procesarse automáticamente para:

- Extracción de palabras clave (términos con mayores pesos de cada documento)
- Clusterización o categorización (agrupación de documentos con vectores parecidos entre sí más de un cierto umbral)
- Recuperación de textos mediante consultas en lenguaje natural. La frase que expresa la consulta se procesa como un documento más, recuperándose aquellos documentos cuyo vector se parezca al de la consulta más que un determinado umbral.

La recuperación de información sobre grandes cantidades de texto, no es simple y muchas veces no es satisfactoria, la forma en que se define la búsqueda es algo importante así como los métodos de los que se está haciendo uso. La manera de llevar acabo estas técnicas, queda del lado del estudio de algoritmos, pues son realmente éstos los que se perfeccionan cada día.

2.7 Evaluación de un Sistema de Recuperación de Información

"¡No basta con hacer algo rápido, hay que hacerlo BIEN!"

Los SRIs también conocidos como máquinas de búsqueda, recuperan una lista de documentos relevantes (***hitlist***) a partir de una consulta planteada por el usuario. De esta lista, existen documentos que verdaderamente satisfacen la consulta pero muchos otros no.
La calidad de una máquina de búsqueda se mide en términos de la proporción de éxitos en la lista, es decir, el número de las buenas recuperaciones contra las malas y la proporción de documentos que satisfacen la búsqueda pero que no aparecen en la lista recuperada.

Lo ideal es que una máquina de búsqueda deba recuperar todos y sólo aquellos documentos que satisfacen las condiciones de la búsqueda planteada. Así, una máquina de búsqueda puede tener muy buena calidad al poseer una alta *precisión*, *utilidad* y *evocación* (*recall*), sin embargo, las máquinas de búsqueda actuales son capaces de recuperar sólo algunos de los documentos deseados.

2.7.1 Correctez y relevancia

Toda consulta estructurada sobre una base de datos relacional determina un conjunto de tuplas que deben ser recuperadas, de tal modo que las tuplas recuperadas siempre serán correctas y no será posible que se recuperen tuplas que no satisfagan el criterio de búsqueda definido, es decir, que sean incorrectas. Pero lo anterior, no ocurre al tratarse del texto, en este caso no se define de manera exacta la lista de documentos relevantes, salvo en los casos triviales, y no se proporciona una especificación completa de las intenciones del usuario. Todas las consultas tienen una intención implícita que no se puede describir completamente en la consulta y por ello, la máquina de recuperación la desconoce.
Un ejemplo de lo anterior, se plantea a continuación: Si un usuario está buscando aquellos libros cuyo autor sea *Carl Sagan*, una consulta estructurada puede ser:

```
SELECT titulo
FROM libros
WHERE autor= 'Carl Sagan';
```

Esta consulta no regresará tuplas irrelevantes para el usuario, a menos que existan más autores con el mismo nombre y sólo se quieran conocer los libros del ganador del premio Pulitzer.
Una consulta sobre texto, es decir sobre un tipo no estructurado, se define así:

```
SELECT titulo
FROM revisiones_libros
WHERE CONTAINS(notas_revision, 'Carl Sagan') > 0;
```

Es probable que se recuperen libros no revisados por Carl Sagan, sino que pueden haberlo mencionado como una referencia, pero no se garantiza que la revisión haya sido de él. Así, desde el punto de vista del usuario, tal recuperación es incorrecta pero para la máquina de búsqueda, la recuperación y búsqueda son correctas pues se cumplió el emparejamiento con el patrón establecido. En general, no hay manera de definir una mejor consulta que permita recuperar sólo los documentos relevantes. Agregar una descripción de la intención del usuario y efectuando una **consulta de texto libre**, solo puede recuperar documentos menos deseables en la mayoría de los casos.
La relevancia, es algo subjetivo, pero no significa que no se deba considerar como un factor en la calidad de los documentos recuperados. Los juicios de relevancia de documentos dados por los usuarios, pueden usarse para comparar diferentes máquinas y también pueden emplearse para medir la calidad de una máquina particular contra los valores obtenidos por técnicas estadísticas.

Rango de utilidad y relevancia

Las medidas de calidad se basan en la idea fundamental del costo y el beneficio de recuperar un documento. La obtención de un documento relevante tiene un beneficio para el usuario y obtener un documento no-relevante implica un costo: tiempo perdido en su lectura, edición y traducción, etc.
El costo o el beneficio también depende de *la posición del documento en la lista de recuperación*, ya que los primeros documentos de la lista generalmente aportan un mayor beneficio que los últimos de la lista; así, un documento no-relevante al inicio genera un alto costo.
Si se llegan a encontrar muchos documentos no-relevantes al inicio de la lista, pueden desalentar al usuario en descender y revisar el resto de los documentos hasta encontrar los que le interesan.
Por lo anterior, es que muchas máquinas de búsqueda intentan ordenar la lista de documentos recuperados en orden decreciente de relevancia para la consulta (como fue determinado por el sistema). Existe un costo asociado con los documentos relevantes pero que no fueron recuperados ya que el usuario puede perder información importante contenida en esos documentos.
La utilidad de una lista de documentos relevantes es una medida sobre los beneficios o costos de la lista para el usuario.

Una definición general de *utilidad* es:

$$\textbf{Utilidad} = C1*N1 - (C2*N2+C3*N3)$$

Donde:

C1 es el beneficio de recuperar un documento relevante.
N1 es el número de documentos relevantes en la lista recuperada.
C2 es el costo de un documento no-relevante en la lista recuperada.
N2 es el número de documentos no-relevantes en la lista.
C3 es el costo de no recuperar un documento relevante.
N3 es el número de documentos relevantes en la colección total de documentos que no son recuperados.

Algunas deficiencias en esta métrica son:

- No existe un rango limitado, incluye valores negativos que dependen del tamaño de los documentos en la colección y en la lista.
- Se asume que los costos y los beneficios son los mismos para todas las posiciones en la lista. Es decir, el costo de un documento no-relevante al inicio de la lista, es el mismo que para el de la posición última.
- Es fácil alcanzar una utilidad máxima recuperando menos documentos. Por ejemplo, una máquina que no recupera para una consulta todos los documentos relevantes, puede tener un alto valor en su utilidad.

2.7.2 Precisión y Evocación (*Recall*)

Estas medidas son más usadas que la utilidad, debido a que tienen un rango fijo [0%-100%] y son más fáciles de comparar contra las consultas y las máquinas de búsqueda entre sí.
La precisión es una medida de la utilidad de una lista de documentos relevantes.
La evocación o *recall* es una medida de la *completez* de la lista.

Estas medidas se definen formalmente en los siguientes términos: Se tiene el requerimiento de información I y el conjunto R de documentos relevantes en la colección. $|R|$ es la cardinalidad del conjunto R. Se asume que dada una estrategia de recuperación, ésta procesa el requerimiento I y generará un conjunto de documentos respuesta A. $|A|$ es su cardinalidad. Así, se define $|RA|$ como el número de documentos en la intersección de los conjuntos R y A.
Finalmente,

$$\textbf{Precisión} = \frac{|RA|}{|A|} \qquad \textbf{Evocación} = \frac{|RA|}{|R|}$$

Dicho de otras maneras:
Precisión es la cantidad de documentos recuperados que son relevantes, es una medida de qué tan bien la máquina no recupera documentos no-relevantes.
Evocación es la cantidad de documentos relevantes que han sido recuperados, es una medida de qué tan bien la máquina encuentra los documentos relevantes.

El valor de la *evocación* será del 100% cuando todos los documentos relevantes se recuperen. En teoría, es fácil alcanzar un buen valor de *recall*: ¡Simplemente, recuperando todos los documentos en la colección para toda consulta!, por esto, no se le considera como una buena métrica de calidad.
El valor de la *precisión* será del 100% cuando todos los documentos recuperados sean verdaderamente relevantes para la consulta.

La *precisión* y la *evocación* son métricas sobre la lista de documentos relevantes recuperados y no permiten medir la calidad de los documentos verdaderamente relevantes aunque se pueden obtener rangos de métricas no sobre toda la lista, sino por ejemplo, los n primeros de la lista y los m-n restantes obteniendo así altos porcentajes de precisión sobre un determinado subconjunto de la lista.

La *precisión promedio* es una nueva medida que combina el rango de relevancia y la precisión. Sean:

n	el número de documentos recuperados como relevantes,
h[i]	el i-ésimo documento recuperado como relevante,
rel[i]	1 si *h[i]* es relevante ó 0 en otro caso.
R	es el número total de documentos relevantes en la colección para la consulta.

Así, la precisión del j-ésimo documento recuperado es:

$$\textbf{precision [j]} = \sum_{k=1}^{j} \frac{rel[k]}{j}$$

$$\textbf{precisión promedio} = \sum_{j=1}^{n} \frac{(precision[j] * \text{rel[j]})}{R}$$

La precisión promedio es una medida ideal de la calidad de las máquinas de recuperación. Para obtener un valor de 1 en esta métrica, se deben recuperar todos los documentos relevantes (1) en un rango perfecto (1). Con esta fórmula, se puede apreciar que los documentos no-relevantes recuperados no alteran al numerador.

Las máquinas de recuperación deben proporcionar un alto grado de *evocación* para ser admisibles en la mayoría de las aplicaciones, pero la clave es construir mejores máquinas que incrementen la *precisión* sin sacrificar la *evocación*. Por mencionar un ejemplo, las máquinas de búsqueda en el Web tienen una razonable métrica de *evocación* pero un bajo valor en la *precisión*.

Otros factores que también pueden ser evaluados en un SRI son los siguientes criterios:

- **La eficiencia de la ejecución** se mide con respecto al tiempo que toma al sistema en llevar a cabo un cálculo computacional, este tipo de evaluación es muy importante en los SRIs pero sobre todo, en aquellos que son interactivos.

- **La eficiencia en el almacenamiento** se mide por el número de bytes necesarios para almacenar datos. Esta medida se determina por la razón del tamaño de los archivos de índices más el tamaño de los archivos de los documentos, sobre el tamaño de los archivos de los documentos. Un valor de 1.5 a 3 en este parámetro es típico en un SRI basado en archivos invertidos.

La importancia de todos los criterios anteriores, debe decidirlos el diseñador del sistema, así como las estructuras de datos y algoritmos para la implementación.

2.8 Comentarios finales

Pese a que en el presente capítulo se ha dado una visión teórica de la recuperación de información, específicamente de texto, es importante señalar que sí existe una aplicación real de toda esta teoría y sobre todo, que existe un **Benchmark de calidad: TREC** (*Text Retrieval Conference*).
El TREC es una conferencia anual con fines académicos y comerciales sobre la industria de los sistemas de recuperación de texto, es coordinado por el *National Institute of Standards and Technology (NIST. Instituto Nacional de Estándadres y Tecnología)*. Dentro del TREC existen áreas de interés particular y aplicaciones, a éstas les llaman *tracks* y existe uno principal llamado: *ad hoc retrieval track*.

Los participantes del TREC reciben una colección de 2GB de documentos con aproximadamente medio millón de documentos (incluye artículos de "LA Times", "Financial Times", documentos de emisiones extranjeras del servicio de información y del registro federal norteamericano), posteriormente, los participantes reciben 50 temas, donde cada uno de ellos describe una consulta en inglés. Los participantes tienen dos meses para dar resultados, proporcionando para cada tema una lista de 1000 documentos relevantes por tema en orden descendente de relevancia.
El *NITS* evalúa los resultados y publica las métricas de *precisión* y *evocación* para cada participante, promediando los valores de todos los temas. También se publica la *precisión promedio* para cada tema. Los valores de las métricas del TREC no son absolutos debido a que la colección de documentos es muy grande y no es posible la lectura de la colección completa.
Los resultados del TREC se pueden emplear de acuerdo con los criterios del NIST y no se pueden usar en anuncios o propaganda.

Es así como se concluye este capítulo, dando a conocer el soporte teórico de la recuperación de la información en texto y cómo se utilizará para poder crear técnicas y algoritmos que permitan construir mejores máquinas de búsqueda. Hoy en día, con grandes cantidades de datos almacenados, su transformación en información demanda nuevas y mejores técnicas para hacerlo, sobre todo con un buen tiempo de respuesta, con una recuperación satisfactoria y adaptable a lo que el usuario busca realmente.

Capítulo III

Técnicas para búsqueda de texto

" El universo es un gran libro
que no puede leerse
hasta que uno aprende a comprender
el idioma y se familiariza con el
alfabeto de que está compuesto "

Galileo Galilei

En el capítulo anterior, se contempló la teoría que apoya la recuperación de la información, en el presente capítulo se presentan las principales técnicas con que se cuenta para llevar a cabo cada una de las tareas que se requieren para efectuar dicha recuperación. Se podrá apreciar que muchas contribuciones teóricas de la ciencia de la computación (tales como la teoría de autómatas y lenguajes formales) se ponen en práctica y son de uso general en los SRI's.

Las técnicas se dividirán de acuerdo a las tareas básicas de los SRI:

- Técnicas de preprocesamiento (o Filtrado)
- Técnicas de indexación
- Técnicas de búsqueda secuencial de cadenas

Para las técnicas de indexación, se tienen algoritmos que sirven para construir una estructura de datos que permita una búsqueda rápida del texto, son los más usados y recientes. Los algoritmos de preprocesamiento se usan para efectuar un proceso de filtrado sobre el texto que reciben como entrada. Es una transformación típica en la tarea de la recuperación de información para reducir el tamaño de un texto o normalizarlo para simplificar la búsqueda. En una búsqueda secuencial, se hace una lectura sobre el texto completo y se buscan los patrones de las cadenas.

3.1 Técnicas de preprocesamiento

3.1.1 Análisis léxico

El análisis léxico es el proceso para convertir una cadena de caracteres de entrada en un conjunto de palabras o *tokens* apoyado por el reconocimiento de los delimitadores (caracteres como el blanco, que permiten diferenciar una cadena de otra). Así, un *token* es un grupo de caracteres con significado colectivo, en el cual, ya se interpretó el significado de los elementos del texto. (Figura 3.1)
El análisis léxico es el primer estado de una indexación automática ya que produce términos candidatos a ser índices, los cuales pueden ser procesados y considerados como tal; y también el analizador léxico es el primer estado de un procesamiento de búsqueda, ya que analiza una consulta y la compara con los índices para encontrar los términos relevantes y producir *tokens* que serán usados en una representación interna para la comparación con los índices.

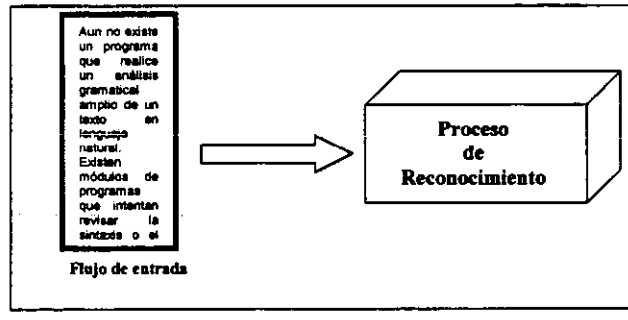


Figura 3.1. Análisis léxico del texto para la obtención de tokens

Un buen principio para elaborar un analizador léxico, para un sistema de indexación automático debe ser la definición de aquellas cadenas que deben ser consideradas como tales, como se presentó

en el capitulo II, un procedimiento avanzado de análisis léxico debe efectuar una normalización para unificar algunos formatos como el manejo de los guiones, los signos de puntuación y el uso de mayúsculas o minúsculas. Esto no se debe olvidar cuando se establezcan las políticas del analizador.

La implementación del analizador léxico, puede ser generalmente, de dos maneras:

- Construyendo el analizador léxico como una máquina de estado finito
- Usando un generador automático que construya el analizador léxico

La teoría y algunas definiciones formales de autómatas y expresiones regulares, no se discuten a detalle en este trabajo, pero se parte de un conocimiento previo para ponerlo en práctica en la creación de un analizador léxico. Trabajos clásicos sobre estos temas son los desarrollados por Glenn Brookshear [Broo93], Daniel Cohen [Cohe86] y por Hopcroft y Ullman [HoUl79]

• Autómatas finitos como analizadores léxicos

La manera más sencilla de iniciar la implementación de una máquina de estado finito, es dibujar un diagrama de transición que modele su comportamiento. Este diagrama debe reconocer los *tokens* a partir del examen de los caracteres de entrada. En la figura 3.2 se muestra el diagrama de un **autómata** que representa el reconocimiento del *token* de un nombre de identificador en un lenguaje de programación. Se observa que para llegar al estado de aceptación, es preciso iniciar con un caracter de tipo letra y después pueden llegar letras o números, si se inicia con un número, se sabe que se puede seguir leyendo cualquier caracter (letra o digito), pero no será un nombre de identificador aceptado. Así, con esta representación se tiene sin *ambigüedad* la estructura de un nombre aceptable.

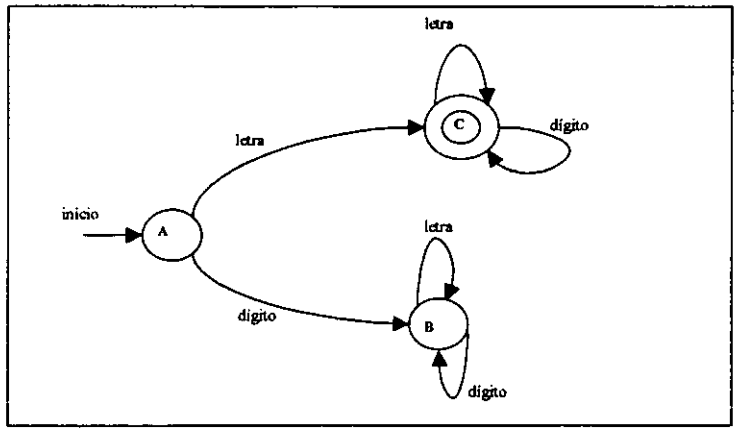


Figura 3.2 Diagrama de transición que reconoce nombres válidos de identificadores. corresponde a un Autómata Finito Determinista (AFD)

Una vez que se tiene el diseño del autómata, se puede crear una **tabla de transición**, la cual es un arreglo (o matriz) bidimensional cuyos elementos proporcionan el resumen del diagrama de transiciones correspondiente. Para elaborar una tabla de este tipo, se construye un arreglo, con una fila para cada estado del diagrama de transiciones y una columna para cada símbolo o categoría de símbolos que podrían ocurrir en la cadena de entrada. El elemento que se encuentra en la fila *m* y la

columna *n* es el estado que se alcanzaría en el diagrama de transiciones al dejar el estado *m* a través de un arco con etiqueta *n*. Si no existe arco *n* alguno que salga del estado *m*, entonces la casilla correspondiente de la tabla se marca como un estado de *error* y en todo caso, no se aceptará como un término adecuado para ser usado como índice.
Para completar la tabla de transiciones, se agrega una columna EOS (fin de cadena), la cual contendrá en la casilla de la fila correspondiente, el valor ACEPTAR si en esa fila, el estado representado corresponde a uno de aceptación en el diagrama. El valor de error, se presentará en caso contrario. A continuación, se muestra la tabla de transición correspondiente al autómata de la figura anterior.

ESTADOS \ ENTRADA	letra	digito	EOS
A	C	B	Error
B	Error	Error	Error
C	C	C	Aceptar

Tabla 3.1 Tabla de Transición

A partir de esta tabla, se puede diseñar de manera muy simple, el analizador léxico. Lo único que se tiene que hacer es asignar a una variable un valor inicial, correspondiente al estado inicial, y luego actualizar repetidamente esta variable con base en la tabla, conforme se leen los símbolos de la cadena de entrada, hasta llegar al fin de la cadena. A continuación se presenta el pseudocódigo del analizador léxico.

```
Estado:= A;
REPEAT
        Leer siguiente caracter de la cadena de entrada;
        CASE simbolo OF
                letra:   Entrada:="letra";
                digito:  Entrada:="digito";
                marca de fin de cadena:    Entrada:= "EOS";
                ninguno de los anteriores: Salir a rutina de error;
        Estado:= Tabla[Estado, Entrada];
        IF Estado= "Error" THEN Salir a rutina de error;
UNTIL Estado = Aceptar
```

Una vez reconocidos los *tokens* factibles a ser términos índice, se procede a generar la lista de parada o también llamada *stoplist*.

- LEX: Un generador de analizadores léxicos

Las herramientas para construir analizadores léxicos parten de notaciones de propósito especial basadas en *Expresiones Regulares*. LEX genera un autómata finito determinista para las expresiones regulares indicadas en la especificación de entrada. El autómata es interpretado en lugar de compilado, con la finalidad de reducir su consumo de memoria, el resultado es un analizador lo suficientemente rápido. En particular, el tiempo requerido por un programa generado por LEX para reconocer y partir la entrada es proporcional al tamaño de la misma. El número de reglas de LEX o

la complejidad de las mismas, no son aspectos que influyan de manera importante en la velocidad a menos que las reglas impliquen contexto adelantado que requiera una cantidad significativa de análisis repetitivo de la entrada. Lo que crece con la cantidad y complejidad de las reglas es el tamaño del autómata finito, y en consecuencia, del programa creado por LEX.

LEX es una herramienta de UNIX y trabaja por medio de expresiones regulares que permiten combinar la especificación de patrones con acciones. En la figura 3.3 se observa cómo la tarea de construcción del analizador se ve reducida a escribir un programa **lex.l**, el cual proporciona las expresiones regulares que generan los *tokens*, será compilado por LEX, obteniendo asi, un archivo **lex.yy.c** que a su vez se compilará por lenguaje C (generalmente) siendo este programa el responsable de leer un archivo de entrada y hacer su análisis léxico.

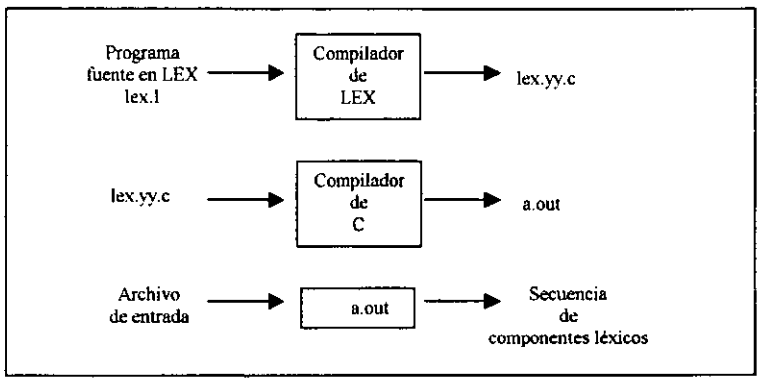


Figura 3.3 La creación de un Analizador Léxico con LEX.

En las figuras, 3.4 y 3.5, se presenta respectivamente:

- El autómata reconocedor de los operadores relacionales sobre el que se apoya LEX.
- El código de un programa en LEX, el cual permite reconocer aquellas cadenas de caracteres que representan *tokens* de operadores relacionales. El programa se presenta comentariado para su documentación.

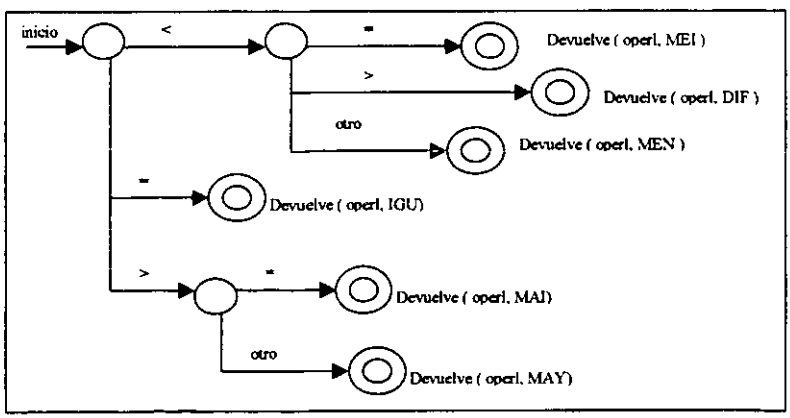


Figura 3.4 Autómata Finito para reconocer operadores relacionales.

```
%{
/*      Zona de declaración de variables, constantes y expresiones regulares.
        Definición de constantes:
                        MEN ... Menor, MEI ... Menor Igual, IGU ... Igual,
                        DIF ... Diferente, MAY ... Mayor, MAI ... Mayor igual,
                        ID ... Identificador, OPREL... Operador relacional,
                        IF, THEN, ELSE y NUMERO */
%}

/* Definiciones con Expresiones Regulares */

delim           [ \t\n]         /*Delimitador*/
eb              {delim}+        /*Espacio en blanco*/
letra           [A-Za-z]        /*Letras mayúsculas y minuúsculas */
digito          [0-9]
id              {letra}({letra}|{digito})*
numero          {digito}+(\.{digito}+)?(E[+\-]?{digito}+)?

%%
/* Reglas de traducción */

{eb}            {/*No hay acción ni se devuelve nada*/}

/* En las siguientes sentencias, se observa la diferencia entre:
   un Componente LÉXICO y un TOKEN */

if              {return (IF);}
then            {return (THEN);}
else            {return (ELSE);}
{id}            {yylval= instala_id( );return (ID);}
{numero}        {yylval=instala_num; return(NUMERO);}
"<"             {yylval=MEN; return(OPREL);}
"<="            {yylval=MEI; return(OPREL);}
"="             {yylval=IGU; return(OPREL);}
"<>"            {yylval=DIF; return(OPREL);}
">"             {yylval=MAY; return(OPREL);}
">="            {yylval=MAI; return(OPREL);}

%%

/* Procedimientos Auxiliares */

instala_id( ) {
/*  procedimiento para instalar el lexema, cuyo primer caracter está apuntando por yytexto
    y cuya longitud es yylong, dentro de la tabla de símbolos y devuelve un apuntador a él */
}

instala_num( ) {
/* procedimiento similar para instalar un lexema que es un número */
}
```

Figura 3.5 Un sencillo programa en LEX

Actualmente, la creación de un analizador léxico es rápida y fácil. Por ejemplo, en una máquina de estado finito generada para una lista de parada de 425 palabras, con 318 estados y 555 arcos, el tiempo empleado en realizar el análisis léxico de un texto de 28 páginas, fue menor a un segundo en una Sun SparcStation 1 [FrBa92]

- Recuperación de errores léxicos

Los errores de tipo lexicográfico u ortográfico suelen ser muy frecuentes. Los trabajos de Morgan encaminados en este sentido clasificaron estos errores en cuatro grupos:

1. Se ha incluido un símbolo o caracter absolutamente extraño para vocabulario terminal de la gramática del lenguaje de programación.
2. Se ha omitido un caracter.
3. Se ha incluido un nuevo caracter que viene a sumarse a los que realmente componen el nombre.
4. Han sido permutados dos caracteres del token analizado.

Puede ocurrir que, involuntariamente, se cambie un caracter en el texto. Un procedimiento corrector para tal situación consiste en formar los subconjuntos de los mismos *n* caracteres del término no encontrado y comparar. Esta recuperación requiere de sumo cuidado pues puede suceder que una vez "interpretado" el error mediante este procedimiento no sea el correspondiente con la idea original capturada.

Para dichos casos, se suministran al subsistema de recuperación de errores unas tablas de ocurrencias de anomalías:

O (letra) se confunde con O (número)

I (letra) se confunde con I o l (ele) o el número 1

S (letra) se confunde con 5 o con $

A continuación, basado en la definición formal de gramática, se presentan tres posibles errores léxicos y la manera de recuperarse de ellos.

Errores por sustitución de un caracter por otro

Se define C como operador de cambio o sustitución para cada carácter de la gramática del modo:

$C(a) = T - \{a\}$ Donde T = {terminales}

Se afirma entonces:

$C(x) = \{y \mid y \in T^*\}$ Es decir, se obtenga la cadena *x* cambiando un solo caracter.

Generalmente:

$$C^n(x) = \begin{cases} 0 & \text{Si } |x| < n \\ y \mid y \in T^* & \text{Se obtenga de la cadena } x \text{ cambiando } n \text{ caracteres} \end{cases}$$

Esta definición representa la aparición de *n* errores.
n = número de T caracteres cambiados

Nota: $A = \alpha_1 + \alpha_2$ Si $\begin{cases} A \rightarrow \alpha_1 \\ A \rightarrow \alpha_2 \end{cases}$ O bien: $A ::= \alpha_1 \mid \alpha_2$

El lenguaje generado es:

$$L(G) = \{ y \mid y \in T^* \wedge y \in C^n(x) \wedge x \in L(G); n \geq 0 \}$$

Ejemplo:

Sea la gramática formal de tipo 2

$G = \{ \{S,B\}, \{a,b\}, P, S \}$

$P = \{ \; S \rightarrow aB$
$\quad S \rightarrow aSa$
$\quad S \rightarrow b \; \}$

Cambiando a la notación convenida:

$S = aB + aSa$
$B = b$

Generación de conjuntos de cambio:

$C^1(aB) = C(a)B = bB$
$C^1(aSa) = C(a)Sa + aSC(a) = bSa + aSb$
$C^2(aSa) = C(a)SC(a) = bSb$
$C^1(b) = a$

Que resultan ser todos los posibles errores por sustitución en las reglas gramaticales.
La "nueva gramática" generadora de cadenas perfectas e imperfectas es:

$G_N = \{ \{S,B\}, \{a,b\}, P_N, S \}$

$P_N = \{ \; S \rightarrow aB \mid aSa \mid bB \mid bSa \mid aSb \mid bSb$
$\quad B \rightarrow a \mid b \; \}$

Con lo anterior, se obtiene un esquema de traducción capaz de reconvertir cadenas imperfectas en sus lógicamente válidas.

Las reglas de traducción del ejemplo anterior, se muestran:

$S \rightarrow aB, aB$
$S \rightarrow aSa, aSa$ } Sin error
$B \rightarrow b, b$

$S \rightarrow bB, aB$
$S \rightarrow bSa, aSa$
$S \rightarrow aSb, aSa$ } Con un error de cambio
$B \rightarrow a, b$

$S \rightarrow bSb, aSa$ } Con dos errores de cambios

Errores por borrado o eliminación de caracteres

Se define un nuevo operador B de borrado:

$$B(a) = \lambda \ \forall a \in T$$

$\therefore$ Si $x \in T^*$ se tiene que:

$$B^n(x) = \begin{cases} 0 & \text{Si } |x| < n \\ y \mid y \in T^* & \text{Se produce por omisión de } n \text{ terminales} \end{cases}$$

Ejemplo:

Para la gramática anterior se tiene:

$B^1(aB) = B(a)B = B$
$B^1(aSa) = B(a)Sa = Sa$
$B^1(aSa) = aSB(a) = Sa$ $\Longrightarrow$ $Sa + aS$
$B^1(b) = \lambda$
$B^2(aSa) = B(a)SB(a) = S$

El esquema de traducción por borrado se construye:

$S = aB + B + aSa + Sa + aS + S$
$B = b + \lambda$

Las reglas de traducción:

$S \rightarrow aB, aB$
$S \rightarrow aSa, aSa$
$B \rightarrow b, b$

Sin error

$S \rightarrow B, aB$
$S \rightarrow Sa, aSa$
$S \rightarrow aS, aSa$
$B \rightarrow \lambda, b$

Con un error de borrado

$S \rightarrow S, aSa$

Con dos errores de borrado

Inclusión de caracteres

Se define el operador I de inclusión:

$$I(a) = Ta + aT \quad \forall a \in T$$

La inclusión es a la izquierda o a la derecha.

Si $T = \{ a_1, a_2, \ldots, a_k \}$ se deduce que:

$$I(a) = a_1 a + a_2 a + \ldots a_k a + a a_1 + a a_2 + \ldots + a a_k$$

Las reglas de inclusión son para la gramática anterior:

$S \rightarrow aaB, aB$
$S \rightarrow aBa, aB$
$S \rightarrow baB, aB$
$S \rightarrow aBb, aB$
$B \rightarrow ab, b$
$B \rightarrow ba, b$
$B \rightarrow bb, b$
$B \rightarrow aa, a$
$B \rightarrow ab, a$
$B \rightarrow ba, a$

.
.
.

Otras posibles recuperaciones de errores son:

- Borrar un caracter extraño
- Intercambiar dos caracteres adyacentes

3.1.2 Raíz de las palabras (*Stemming*)

Una técnica para mejorar la ejecución de la IR es proporcionar al usuario que efectúa la búsqueda, la manera de encontrar las *variantes morfológicas* de los términos buscados.

Así, un beneficio al obtener la raíz de una palabra es almacenar en una tabla los términos índice y sus raíces. Por ejemplo, como se muestra en la figura 3.6, en la tabla del índice no sólo se almacena el término, sino su raíz lo cual permite ampliar las búsquedas referentes a un tema.

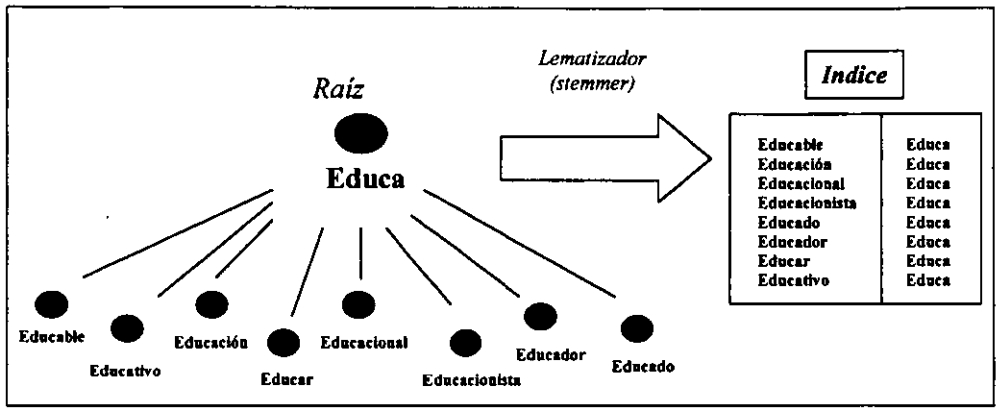


Figura 3.6 El índice se forma por el término y su raíz

Así, para efectos de búsqueda, todos los términos que tienen la misma raíz serán considerados como equivalentes y recuperarán los documentos en los que aparece cada uno de ellos, brindando así al usuario una búsqueda flexible. Los términos en la consulta y en los índices se buscan por medio de un *árbol-B* o una *tabla hash*.

Un problema de estas técnicas es que muchos términos encontrados en la base de datos pueden no estar representados debido a que su dominio no es un estándar en el idioma en que se ha obtenido el término raíz. Como se ejemplificó en el capítulo anterior, existen muchas variantes que dependen del idioma y de las palabras mismas, que al obtener su raíz ésta no corresponde a su semántica.
Otra desventaja puede ser el almacenamiento excesivo para la tabla, aunque puede justificarse este factor a cambio del tiempo de recuperación. El almacenamiento de datos "precalculados" es empleado cuando los cálculos son frecuentes o muy costosos.

A continuación se presentan a detalle las principales técnicas para obtener la raíz de una palabra.

- Raíces por sucesor variante

Se basa en trabajos de lingüística estructural, los cuales son un intento para determinar el morfema límite de una palabra en la distribución de fonemas en una palabra extensa.
Este método usa letras en lugar de fonemas y un cuerpo de texto en lugar de la trascripción fonética de las palabras. La definición formal describe a la técnica así:

Sea α *una palabra de longitud* n; α_i *es una longitud i prefijo de* α. *Sea D el grupo de palabras.* D_{α_i} *se define como el subconjunto de D que contiene aquellos términos cuyas primeras i letras emparejan exactamente* α_i. *El sucesor variante de* α_i, denotado como S_{α_i}, *se define como el número de letras distintas que ocupan la (i+1)-ésima posición de palabras en* D_{α_i}.
Una palabra de prueba de longitud n *tiene* n *sucesores variantes* $S_{\alpha 0}, S_{\alpha 2}, \ldots, S_{\alpha n}$.

En términos menos formales, el sucesor variante de una cadena es el número de caracteres distintos que le siguen en palabras en algún cuerpo del texto. Por ejemplo, el sucesor variante de la cadena "*texto*", dentro de el siguiente cuerpo de palabras, sería:

textual, término, análisis, tiempo, relacional,
dato, transformación, sintaxis, semántico,
álgebra, cadena, compilador

Prefijos	*Sucesor variante*	*Letras*
α_1 =*t*	*4*	*e, é, i, r*
α_2 =*te*	*2*	*x,r*
α_3 =*tex*	*0*	
α_4 =*text*	*1*	*u*
α_5 = *texto*	*0*	

Una vez que se han obtenido los sucesores variantes para una palabra dada, esta información es usada para segmentar la palabra. Existen cuatro métodos para hacer esto:

- *Método de Corte* (*Cutoff Method*). Permite elegir un valor de corte. El problema con este método es cómo elegir el valor de corte ya que si es muy corto, los cortes pueden ser incorrectos, si es muy grande, los cortes correctos pueden perderse.

- *Método de la Cima y la Meseta* (*Peak and Plateau Method*). Un segmento de corte se hace después de que un caracter cuyo sucesor variante excede a aquél caracter que lo precede y el caracter que le sigue inmediatamente.

- *Método de la Palabra Completa* (*Complete Word Method*). Se hace un corte después de un segmento, si el segmento es una palabra completa en el cuerpo del texto.

- *Método de Entropía (Entropy Method)*. Aprovecha la distribución del sucesor variante de las letras.
  Después de que una palabra ha sido segmentada, el segmento que será la raíz, debe ser seleccionado. Algunos autores proponen la siguiente regla:

  if (primer segmento ocurre <= 12 palabras en el cuerpo del texto) then
  primer segmento es raíz
  else (segundo segmento es raíz)

  Por supuesto, el valor del número de ocurrencia, puede variar dependiendo de las observaciones hechas al texto y dependiendo del idioma en el que se encuentra, pues de ello dependerá este valor de ocurrencias. Por ejemplo, en inglés, se tienen muchos prefijos poco frecuentes.

  El texto, posee una semántica que comunica información. La entropía es un concepto que permite conocer la cantidad de información contenida en un texto por medio de la distribución de los símbolos. Si el alfabeto tiene σ símbolos, y cada uno tiene una probabilidad p_i (definida como la frecuencia del símbolo sobre el número total de símbolos) en un texto, la entropía de este texto se define como: $E = -\sum_{i=1}^{\sigma} p_i \log_2 p_i$. En esta fórmula, los símbolos σ están codificados en binario, de tal modo que la entropía es una medida en número de bits. Así, la cantidad de información en un texto, puede ser cuantificada por su entropía.

En resumen, el proceso de encontrar una raíz por el método de sucesor variante, se divide en tres partes:

1) Determinar el sucesor variante de cada palabra del texto.
2) Usar la información del sucesor para segmentar la palabra.
3) Seleccionar uno de los segmentos como la raíz.

- Raíces por n-gramas

Este método se reportó en 1974 y se llamó *método de digrama compartido*. Es polémico debido a que se le ha clasificado como un método para la obtención de raíces, aunque no las produce.
Un digrama es un par de letras consecutivas, debido al uso de trigramas o n-gramas, se ha decidido generalizar su nombre.
Por ejemplo, las palabras *síntesis* y *sintaxis*, se pueden dividir en digramas de la siguiente manera:

síntesis	⇒	sí	ín	nt	te	es	si	is
Digramas únicos:		es	ín	**is**	**nt**	sí	**si**	te
sintaxis	⇒	si	in	nt	ta	ax	xi	is
Digramas únicos:		ax	in	**is**	**nt**	**si**	ta	xi

Así, las palabras *síntesis* y *sintaxis* tienen 7 digramas respectivamente, de los cuales todos son únicos, si hubiesen repetidos, se tendrían que eliminar. Las dos palabras comparten solo tres digramas: is, nt, si.

Una vez que los digramas para cada palabra se han identificado y contado, se puede plantear una medida de similitud. La medida de similitud usada es el *Coeficiente de Dice*, el cual se define como:

$$S = \frac{2C}{A+B}$$

Donde:

A es el número de digramas únicos en la primera palabra.
B es el número de digramas únicos en la segunda palabra.
C es el número de digramas únicos compartidos por A y B.

Las medidas de similitud se determinan para todos los pares de términos en la base de datos formando una matriz de similitud, debido a que el coeficiente es simétrico ($S_{ij} = S_{ji}$), una matriz triangular inferior puede ser empleada:

	término$_1$	*término*$_2$	*término*$_3$	. . .	*término*$_{n-1}$
término$_1$					
término$_2$	S_{21}				
término$_3$	S_{31}	S_{32}			
.					
término$_{n-1}$	S_{n1}	S_{n2}	S_{n3}		$S_{n(n-1)}$

Una vez que la matriz de similitud está disponible, los términos se agrupan usando algún método (*Link Cluster Method*). Algunos ejemplos donde se obtiene la medida de similitud, son los siguientes:

IIMAS	Digramas:	II	IM	MA	AS	A = 4
UNAM	Digramas:	UN	NA	AM		B = 3

C = 0, así entonces, $S = \frac{2(0)}{4+3} = 0$ Obteniendo una similitud nula

Comparando otras cadenas:

IIMAS	Digramas:	II	IM	MA	AS	A = 4
IIMAS	Digramas:	II	IM	MA	AS	B = 4

C = 4, así entonces, $S = \frac{2(4)}{4+4} = 1$ Obteniendo un máximo de similitud

Un último par de cadenas:

IIMAS	Digramas:	II	**IM**	**MA**	AS	A = 4
IMAN	Digramas:	**IM**	**MA**	AN		B = 3

C = 2, así se tiene $S = \frac{2(2)}{4+3} = 0.57$ Obteniendo una similitud media

- Raíces removiendo afijos

Las técnicas para remover afijos, remueven sufijos y/o prefijos de los términos, dejando así, una raíz. Algunos de ellos, también transforman la raíz resultante.
Un ejemplo de remover afijos, consiste en eliminar plurales, pero depende de la palabra, pues no es tan simple como eliminar la 's', pues no todas las palabras siguen ese patrón de comportamiento. Por ejemplo, en español las siguientes palabras: *raíces* (*raíz*), *lápices* (*lápiz*), *árboles (árbol)*, *barnices* (*barniz*), *avestruces* (*avestruz*), *codornices* (*codorniz*), etc. donde no es suficiente la eliminación del sufijo 's'; a diferencia de las palabras *libros* (*libro*), *cartas* (*carta*), *discos* (*disco*), etc. que basta con eliminar el prefijo 's' para obtener el singular. Otra variante es para las palabras: *comedores* (*comedor*), *televisores* (*televisor*), *plumones* (*plumón*), etc. donde se debe eliminar más de un caracter.
Hasta el momento, no se han definido algoritmos para obtener los singulares de las palabras en español, en cambio, se ha establecido un conjunto de reglas para el idioma inglés por medio del *algorimo de Porter*, el cual consta de un conjunto de reglas de condiciones/acciones para la obtención del término raíz.
Las condiciones son de tres clases: 1) Condiciones en la raíz, 2) Condiciones en el sufijo y 3) Condiciones en las reglas.
Las reglas del *algoritmo de Porter* se separan en cinco distintas fases, las cuales son aplicadas a las palabras del texto iniciando desde la fase uno hasta la cinco. Se aplican secuencialmente una después de otra como las instrucciones de un programa.
Dado que los productos comerciales del capítulo IV, tienen la capacidad de obtener las raíces de textos en inglés, se presenta a continuación el algoritmo de Porter.

- *Algoritmo de Porter*

Notación empleada:

- Una variable consonante representada por el símbolo C es cualquier letra distinta de las vocales (*a,e,i,o,u*) y de la letra *y* precedida por una consonante.
- Una variable vocal representada por el símbolo V es usada para referenciar a cualquier letra distinta de una consonante.
- Una letra genérica (vocal o consonante) se denota por el símbolo L.
- El símbolo ∅ es usado para referir a la cadena vacía.
- Las combinaciones de C, V y L son usadas para definir patrones.
- El símbolo * se usa para referir a cero o más repeticiones de un patrón dado (cerradura de Kleene)
- El símbolo + es usado para referir a una o más repeticiones de un patrón dado (cerradura positiva)
- El uso de paréntesis es para afectar a una serie de variables por las cerraduras.
- Un patrón genérico es una combinación de símbolos, paréntesis y operadores de cerraduras.
- Las reglas de sustitución son tratadas como comandos, para ello se utiliza la marca del punto y coma (;) como separador.
- Las reglas de sustitución se aplican a los sufijos de la palabra analizada.
- Toda línea que inicia con % se considera como comentario.

Procedimiento:

% Fase 1: Plurales y participios pasados

Seleccionar la regla con el sufijo más largo {

sses → ss;
ies → i;
ss → ss;
s → ∅; }

Seleccionar la regla con el sufijo más largo {

if ((C)*((V)+(C)+)*(V)*eed) *then* eed → ee;
if (*V)*ed or *V* ing) *then* {

Seleccionar la regla con el sufijo más largo {

ed →∅;

ing →∅; }

Seleccionar la regla con el sufijo más largo {

at → ate;
bl → ble;
iz → ize;
if ((*C_1C_2) and (C_1= C_2) and
(C_1 ∉ {l, s, z})) *then* C_1C_2 → C_1;
if (((C)*((V) +(C)+) C_1 V_1 C_2) and
(C_2 ∉ {w, x, y})) *then* C_1 V_1 C_2 → C_1 V_1 C_2e; }

}

}
if ((*V*y) *then* y → i;
if ((C)*((V)+(C)+)*(V)*) *then*
Seleccionar la regla con el sufijo más largo {

ational → ate;
tional → tion;
enci → ence;
izer → ize;
abli → able;
alli → al;
entli → ent;
eli → e;
ousli → ous;
ization → ize;
ation → ate;
ator → ate;
alism → al;
iveness → ive;
fulness → ful;
ousness → ous;
aliti → al;
iviti → ive;
biliti → ble; }

if ((C)*((V)+(C)+)+(V)*) *then*
Seleccionar la regla con el sufijo más largo {

icate	→ ic;	ical	→ ic;
ative	→ ∅;	ful	→ ∅;
alize	→ al;	ness	→ ∅; }
iciti	→ ic;		

if ((C)*((V)+(C)+)((V)+ (C)+)+ (V)*) *then*
Seleccionar la regla con el sufijo más largo {

al	→ ∅;	ment	→ ∅;
ance	→ ∅;	ent	→ ∅;
ence	→ ∅;	ou	→ ∅;
er	→ ∅;	ism	→ ∅;
ic	→ ∅;	ate	→ ∅;
able	→ ∅;	iti	→ ∅;
ible	→ ∅;	ous	→ ∅;
ant	→ ∅;	ive	→ ∅;
ement	→ ∅;	ize	→ ∅;

if (*s or *t) *then* ion → ∅; }

Seleccionar la regla con el sufijo más largo {
if ((C)*((V)+(C)+)((V)+ (C)+)+ (V)*) *then* e → ∅;
if (((C)*((V)+(C)+)((V)*) and not ((*C_1 V_1 C_2) and ($C_2 \notin$ {w, x, y})))
then e → *nil*;}

if ((C)*((V)+(C)+)((V)+ (C)+)+ V* ll) *then* ll → l;

Otras técnicas usadas por algunos "*stemmers*", consisten simplemente en reducir la longitud de los términos, pero estas técnicas pueden no ser exitosas pues al no considerar reglas, se obtienen reducciones incorrectas. Existen dos maneras de solucionar esto:

1) ***Recodificando***.- Es una transformación dependiente del contexto de la forma AxC → AyC, donde A y C especifican el contexto de la transformación, *x* es la cadena de entrada y *y* es la cadena transformada. Por ejemplo, se puede especificar que si una raíz termina en una "*i*" seguida de "*k*", entonces, i → y.
2) ***Emparejamiento parcial***.- Sólo los *n* caracteres iniciales de las raíces se usan para compararlas. Así, dos cadenas son equivalentes si ellas coinciden, salvo en sus últimos caracteres.

Como se mencionó previamente, el español posee una gran riqueza que difícilmente se puede definir por medio de unas cuantas reglas gramaticales, simplemente al querer encontrar las raíces de algunas palabras, tenemos grandes diferencias a las del idioma inglés. Por ejemplo, la palabra *constipad-* es raíz las palabras *constipado, constipada, constipados, constipadas;* la raíz *clar-* de las palabras *clara, claro, claras, claros, clarísimo, clarísima, clarísimas, clarísimo, claramente;* la raíz *construc-* tiene cerca de 700 palabras tales como *construir, construyo, construido, constructivamente, constructivo, constrúyanmelo,* etc.
Trabajos importantes sobre el español se han hecho en México, muchos de ellos, se han presentado por los investigadores del Laboratorio de Lenguaje Natural del Centro de Investigación en Computación (CIC) del Instituto Politécnico Nacional. [GaBG99], [BoGe99], [MaAS99], [Gelb99]

Uno de los productos más importantes es ***CLASITEX+*** que es una herramienta para el análisis inteligente de textos. Este sistema analiza un texto en español o inglés e indica cuál es el tema o los temas que se abordan en dicho texto, fue desarrollado por el Dr. Adolfo Guzmán Arenas y es propiedad de *SoftwarePro International.* Este trabajo usa **árboles de conceptos** (grafo acíclico en el cual cada nodo es un término que representa a un concepto y las aristas representan relaciones entre los conceptos) y desmembradores de cada palabra en raíz, prefijo y sufijo. Una aportación importante de este trabajo es la cantidad de conceptos tanto en español como en inglés que maneja el sistema, además de la velocidad en el análisis de los documentos y que se buscan los conceptos, no las palabras. Por ejemplo, un artículo que habla de *defensa delantero, portero, gol, tiro de esquina,* será correctamente clasificado como en "*futbol soccer*", aún cuando este par de palabras no aparezca en el artículo. [Guzm98], [Guzm97]

3.2 Lista de Parada

La lista de parada o "diccionario negativo", contiene las palabras que no formarán parte de los términos índice; éstas se encuentran después de un análisis léxico como los descritos anteriormente.
Un problema latente de la obtención de estas palabras es su determinación, pues depende del lenguaje en el que se tengan los textos a analizar. Tradicionalmente, se consideran los artículos, los pronombres y los adverbios; pero la decisión final debe definirla el usuario final de la aplicación ya que no es un problema trivial. Se propone que se dé una lista con las palabras que se desean ignorar como parte de los índices, dicha lista se puede tomar de los *tokens* de salida del analizador léxico.
Buscar las palabras de parada dentro de los *tokens*, se convierte en un simple problema de búsqueda cuyas soluciones se pueden dar con búsquedas sobre árboles binarios, búsquedas binarias sobre arreglos o por funciones de dispersión.
Pero para ello, es preciso determinar las palabras que serán eliminadas, algunas posibles palabras de parada para el español, serían las siguientes: *a, ante, aún, bajo, cabe, con, contra, de, desde, para, por, según, sin, sobre, tras, un, uno, una, unas, él, el, la, los, las, hay, yo, tu, él, ella, nosotros, ellas, mí, ti, nos, les, conmigo, contigo, mio, tuyo, suyo, mis, tus, nuestro, sus, su, éste, aquél, que, quien, quien,* etc.

La implementación de la lista de parada consistiría prácticamente de los siguientes pasos:

- Leer la lista de parada definida por el usuario.
- Construir el autómata finito determinista mínimo para la lista de parada.
- Abrir el archivo o documento de entrada y eliminar las palabras de parada.
- Obtener los *tokens* candidatos para ser indexados.

Así, se tendría un autómata como el de la figura 3.7 para encontrar las palabras de parada de un texto en español.

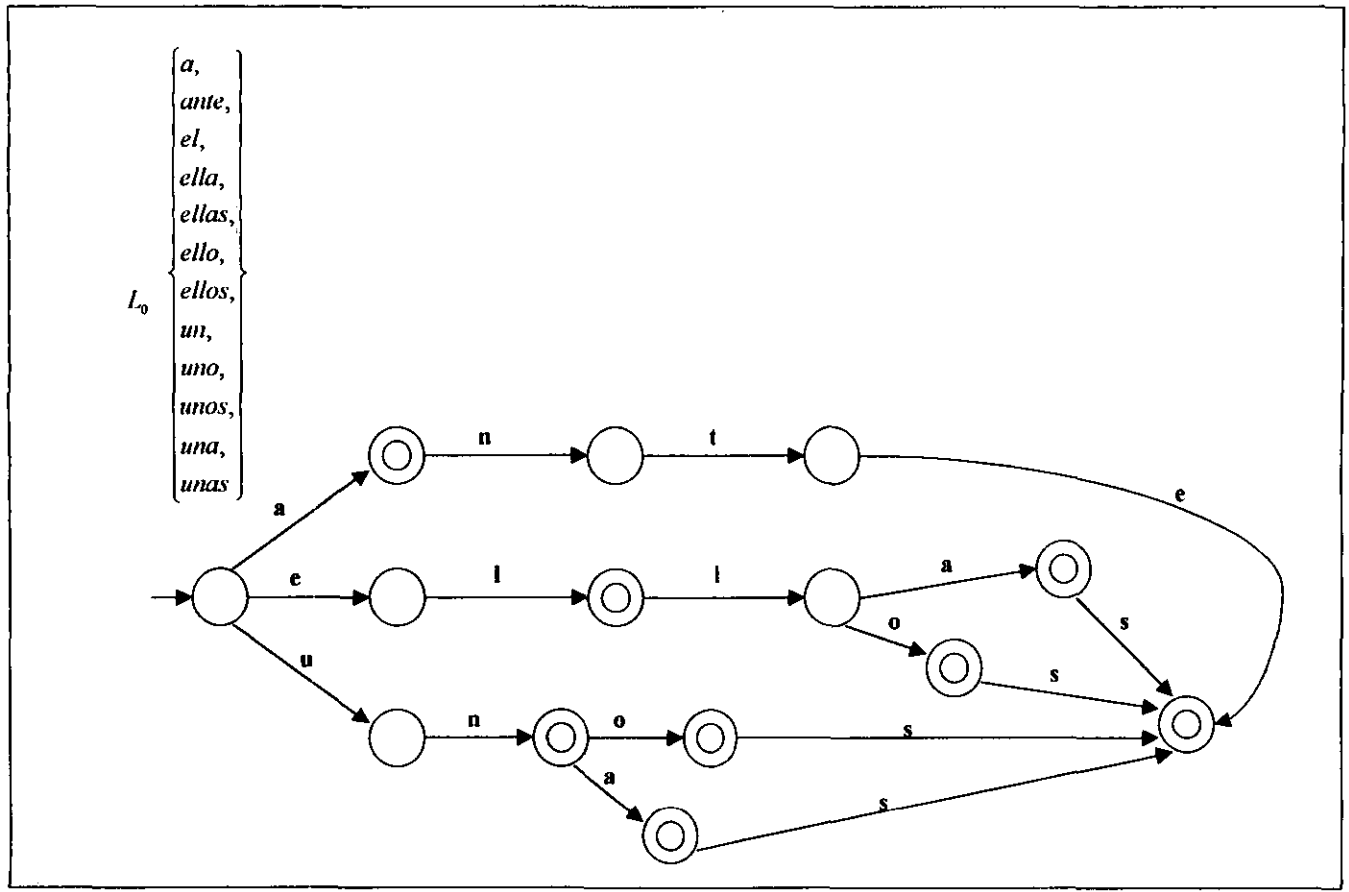


Figura 3.7 Autómata reconocedor de algunas "palabras de parada" en español

3.3 Técnicas de búsqueda secuencial de cadenas

3.3.1 Definición de la búsqueda de cadenas

Las cadenas son evidentemente el centro de los sistemas de tratamiento de texto ya que proporcionan un gran número de posibilidades para la manipulación y búsqueda sobre textos.
Una operación fundamental sobre las cadenas es la búsqueda de patrones; es una tarea común con cadenas, a la que también se le conoce como *pattern matching* (también llamado *reconocimiento de patrones*), consiste en lo siguiente: dada una cadena alfanumérica de longitud N y un patrón de longitud M, encontrar una ocurrencia del patrón dentro del texto.

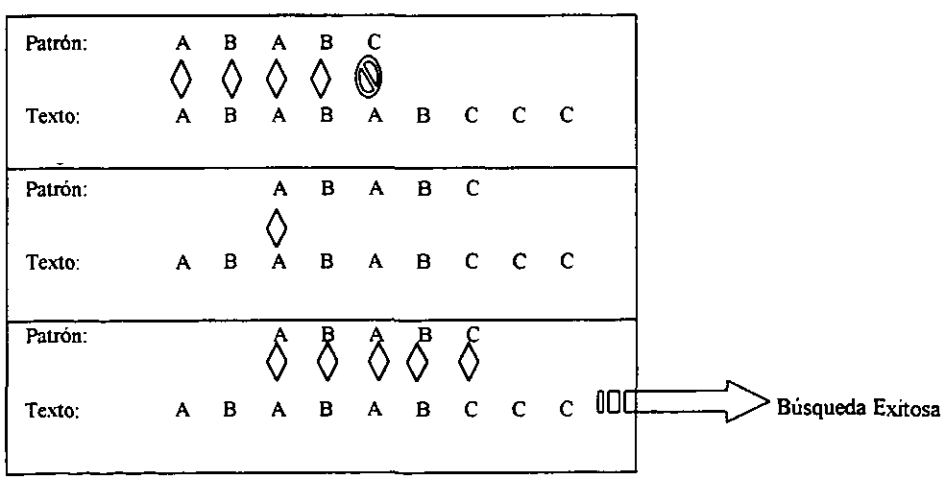
El problema del *reconocimiento de patrones* puede caracterizarse como un problema de búsqueda del patrón como llave; aunque teniendo en cuenta que éste puede ser grande y además no se sabe como se encuentran alineados los caracteres en el texto. Los algoritmos actuales no sólo proveen un espectro de sus métodos de uso práctico, sino también ilustran algunas técnicas fundamentales de diseño. Muchos algoritmos para este problema fácilmente pueden ser extendidos a encontrar todas las ocurrencias del patrón en el texto, ya que estos emplean la búsqueda secuencial y pueden ser reiniciados en un punto posterior al de inicio. Información detallada sobre la implementación, análisis y complejidad de estos algoritmos se puede encontrar en las referencias bibliográficas [Sedg95] y [Cama98].

3.3.2 Algoritmos para búsqueda de patrones

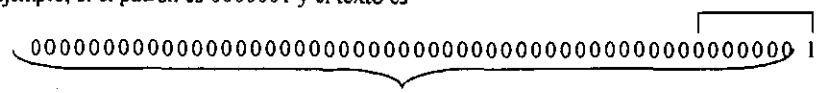
Para resolver el problema de búsqueda de patrones, se han desarrollado varios y diferentes algoritmos que son usados en la práctica, a continuación se presenta una muy breve descripción de los más importantes.

- Algorimo de fuerza bruta (*brute force*)

Consiste simplemente en verificar, para cada posible posición en el texto en la cual el patrón puede ser encontrado. Tenemos así:



Este algoritmo puede ser muy lento para algunos patrones.
Por ejemplo, si el patrón es 0000001 y el texto es



En este caso, dada una implementación del algoritmo, se puede corroborar que se analiza 315 (7*45) veces antes de que el patrón sea encontrado.

En general, la búsqueda de cadenas por fuerza bruta puede necesitar alrededor de NM comparaciones de caracteres.

• Algorimo Knuth – Morris – Pratt

La idea básica de este algoritmo es que cuando se detecte un error en el acoplamiento del patrón en el texto (llamado emparejamiento o *match*), se observen todos los caracteres que se conocieron durante su desarrollo (ya que están en el patrón) y de algún modo se tome ventaja de esta información, y en lugar de retroceder al apuntador sobre todos los caracteres conocidos, se recorra hasta donde sea conveniente.

Si se tiene el patrón 10100111 en el texto 1010100111, el primer error que es detectado está en el quinto caracter, por lo tanto apuntador debe retroceder, pero sólo hasta el tercer caracter y continuar la búsqueda, puesto que no sucede otro error se encuentra la cadena. No se puede decir con exactitud el tiempo que consume este proceso, ya que, depende sólo del patrón, como se muestra a continuación:

j	*p[1...j-1]*	*next[j]*
2	1	1
3	10	1
4	10 1	2
5	10 10	3
6	10 100	1
7	10 100 1	2
8	10 100 11	2

Donde el arreglo next[1..M] puede ser empleado para determinar qué tan lejos se debe o puede regresar cuando un error es detectado.

El algoritmo *Knuth-Morris-Pratt* probablemente no es significativamente más rápido que el de *Fuerza Bruta* en muchas de las aplicaciones reales porque son pocas las que involucran búsqueda de patrón altamente autorepetitivos en texto altamente repetitivo. Sin embargo, el método tiene una virtud principal desde un punto de vista práctico, esto es, el método es muy conveniente para usar un archivo grande leído desde algún dispositivo externo.

En general, la búsqueda de cadenas por el algoritmo de Knuth-Morris-Pratt nunca efectúa más de N+M comparaciones de caracteres.

• Algorimo Boyer-Moore

Boyer y *Moore* sugirieron combinar los dos métodos vistos anteriormente para realizar la búsqueda de derecha a izquierda del patrón, escogiendo el más grande de los saltos. La idea es decidir lo que se debe hacer en función del caracter que provocó la discordancia tanto en el texto como en el patrón. La etapa del preprocesamiento consiste en decidir, para cada caracter que podría figurar en el texto, lo que se haría si dicho caracter hubiese provocado la no-concordancia. La realización más simple de esto conduce inmediatamente a un programa bastante útil por algunos llamado "búsqueda de cadenas de *Boyer-Moore* utilizando la heurística de la no-concordancia".

En general, la búsqueda de cadenas de Boyer-Moore nunca utiliza más de M+N comparaciones de caracteres, y necesita alrededor de N/M pasos si el alfabeto no es pequeño y el patrón no es largo.

• Algorimo Rabin-Karp

Rabin y *Karp* encontraron una manera sencilla de resolver el problema de la búsqueda de patrones por medio de la función de dispersión. Al no ser necesario mantener una tabla de dispersión completa debido a que el problema se plantea de forma que sólo se busque una clave; todo lo que se necesita hacer es evaluar la función de dispersión para cada una de las posibles series de M caracteres del texto y verificar si es igual a la función de dispersión del patrón. El problema con este método es que parece tan difícil calcular la función de dispersión para M caracteres del texto como verificar si éstos son iguales al patrón.
La función propuesta por *Rabin* y *Karp* parece ser lineal y es la siguiente: $H(k)=k \bmod q$, donde: q es el tamaño de la tabla y es un número primo grande.
En el caso de que se encuentre una posición en el texto que tenga el mismo valor que el patrón, se puede realizar una comparación directa del texto con el patrón.

En cuanto al número de pasos que realiza este algoritmo se dice que teóricamente puede realizarse en MN pasos en el peor caso, pero en la práctica se ha observado que realiza un poco más de N+M pasos.

3.4 Técnicas de indexación

El tiempo de procesamiento necesario para construir un índice es armonizado por el uso del mismo en las búsquedas.

3.4.1 Indexación por archivos invertidos

Este tipo de índices corresponde a índices lexicográficos. Su concepto es el siguiente: Dado un conjunto de documentos, a cada documento se le asigna una lista de palabras clave o atributos, con un peso asociado. El archivo invertido es entonces, la lista ordenada (o índice) de atributos, los cuales tienen ligas a los documentos que contienen ese atributo. En la figura 3.8 se muestra gráficamente este proceso.

Permite búsquedas eficientes sobre peticiones de gran magnitud, lo cual es una necesidad para archivos de texto muy grandes.
El pago por esta eficiencia, es el almacenamiento de la estructura de datos cuyo tamaño oscila del 10 al 100% o más del tamaño del archivo de texto mismo, y una necesidad de actualización del índice cuando los datos cambian.
La implementación de archivos invertidos puede ser por medio de varias estructuras como lo son: arreglos ordenados, árboles-B, función de dispersión o la combinación de estas estructuras.

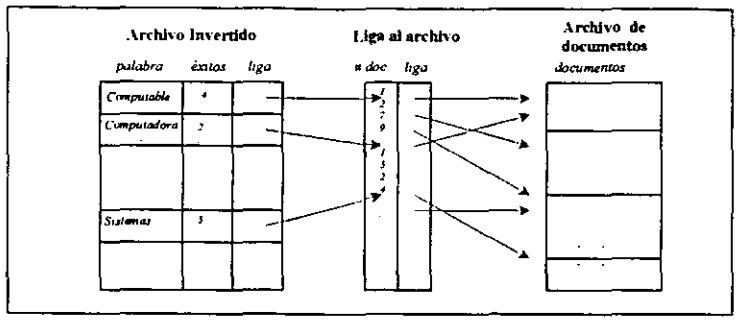


Figura 3.8 Uso del archivo invertido y la liga a los documentos

Los índices invertidos asumen que el texto puede verse como una secuencia de palabras. Esto restringe algunas clases de consultas que pueden ser contestadas, algunas otras como búsquedas por frases resultan muy costosas de resolver. Además, el concepto de palabra no existe en algunas aplicaciones tales como las bases de datos genéticos.

- Índices invertidos por arreglos ordenados

Implementar con arreglos ordenados, implica almacenar la lista de palabras clave sobre un arreglo, incluyendo el número de documentos asociados con cada palabra y una liga a los documentos. Este arreglo es analizado por una búsqueda binaria, aunque los sistemas basados en gran almacenamiento secundario adaptan el arreglo y su búsqueda a las características del almacenamiento secundario.
Su desventaja es la actualización del índice, pues es una operación cara debido a que toda las operaciones de manipulación deben verse reflejadas en el índice. La ventaja es la facilidad de implementación y rapidez razonable.

El proceso por arreglos ordenados puede apreciarse en la figura 3.9, el texto de entrada es analizado sintácticamente en una lista de palabras con su localización en el texto. Esto consume la mayoría del tiempo de indexación. Esta lista es invertida de una lista de términos en orden de localidad a una lista de términos ordenados para usar en la búsqueda (orden alfabético). Un tercer paso opcional es el post-procesamiento de estos archivos invertidos tales como agregar a los términos sus pesos, o la compresión de archivos.
Crear la lista inicial, requiere de algunas operaciones:

1°. Las palabras que deben reconocerse del texto.
2°. Se verifica que cada palabra no pertenezca a una lista de parada y si no lo es, pasa por un algoritmo de obtención del término raíz. Esta raíz se almacena en la lista de palabras junto con su localidad.

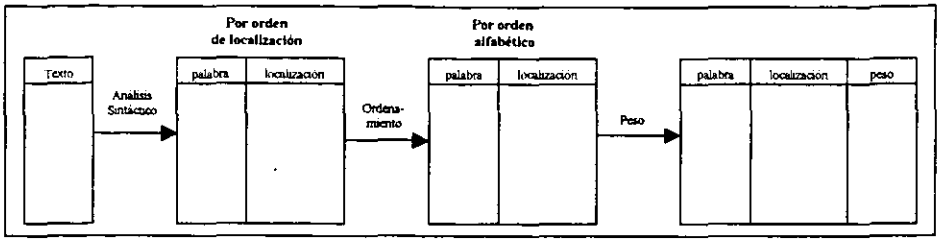


Figura 3.9 Esquema del arreglo ordenado en la creación de un archivo invertido

- Índices invertidos por árboles-B

Un archivo invertido puede implementarse con un árbol-B. Un caso especial de esta estructura es el llamado árbol-B de prefijos, como el que se observa en la figura 3.10; el cual usa prefijos de palabras como llaves primarias en un índice del árbol, lo cual, lo hace conveniente para almacenar índices de texto. Cada nodo interno tiene un número variable de llaves y cada llave es la palabra más corta que distingue a las palabras almacenadas en el siguiente nivel. La llave no necesita ser un prefijo de un término actual en el índice, es hasta el último nivel del árbol donde se almacenan las llaves con sus datos asociados.
Como las llaves de los nodos internos y sus longitudes son variables dependientes del conjunto de datos, el tamaño de cada nodo del árbol-B es también variable. Las actualizaciones mantienen el árbol balanceado, los métodos para los árboles de prefijo balanceados pueden romperse si hay muchas palabras con el mismo prefijo. Así, prefijos comunes deben ser adicionados para disminuir el espacio ocupado.

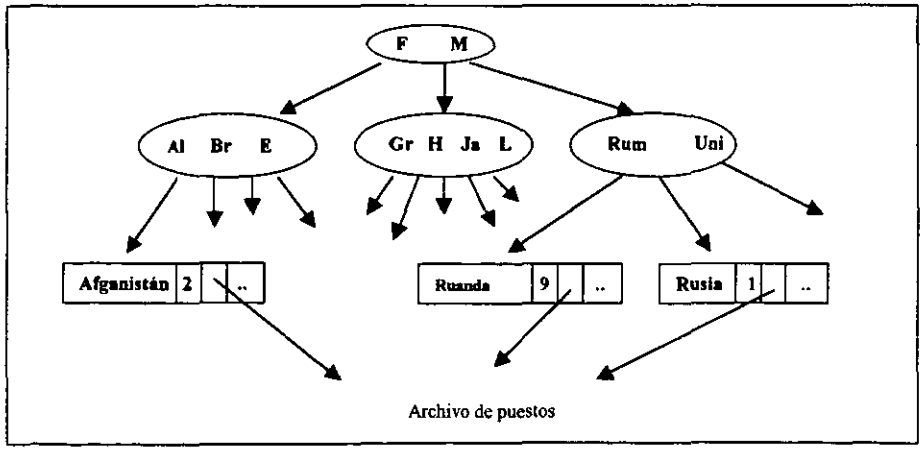


Figura 3.10 Árbol- B de prefijos

Comparando las dos técnicas anteriores, se puede decir que los árboles-B ocupan más espacio y son más complejos que la implementación por arreglos, sin embargo las actualizaciones son más fáciles y el tiempo de búsqueda se reduce considerablemente.

3.4.2 Archivos de firma (*signature files*)

Los archivos de firma son una aplicación de las funciones de dispersión para la búsqueda secuencial de texto. Permiten construir una versión compacta de el texto (entre 10% y 20% del tamaño original). Para esta tarea, el texto se divide en palabras y bloques que palabras, transformando el texto en una secuencia de bits ("la firma"). Para buscar en este índice, la consulta se transforma a una cadena de bits, y esa cadena es buscada secuencialmente en el índice (o archivo de firmas). Debido a que más de una palabra puede ser mapeada a una misma cadena de bits, se pueden obtener respuestas no deseadas (colisiones), por ello, toda respuesta potencial, deberá ser revisada. Con una selección adecuada de parámetros, se puede tener un mejor desempeño al disminuir el tamaño del índice y con ello, un número menor de respuestas no deseadas.
Este método no se recomienda para grandes archivos o aplicaciones donde no es posible dividir en texto en palabras, como en las bases de datos de secuencias de proteínas.
Un archivo de firma usa una función de dispersión o firma que mapea las palabras a cadenas de B bits. Así se divide el texto en bloques de b palabras cada uno. En la figura siguiente (3.11) se aprecia un archivo de firmas para un ejemplo de texto dividido en bloques.

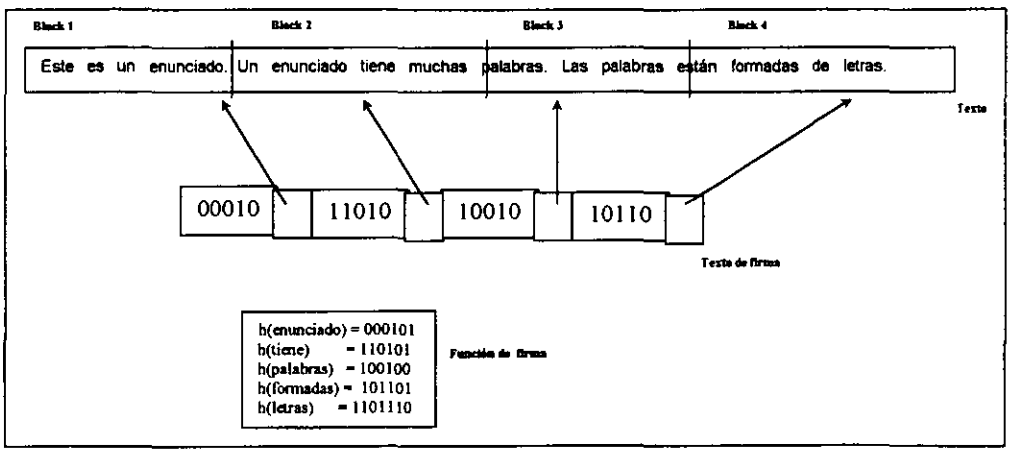


Figura 3.11 Archivo de firmas para un texto

El archivo de firmas es un método mucho más rápido que la revisión completa del texto. Las inserciones de nuevos datos son más sencillas que en los archivos invertidos, debido a que sólo se necesitan operaciones de apertura sin necesidad de hacer una reorganización o reescritura de alguna porción de firmas. Los métodos para la inserción de datos tienen las siguientes ventajas:

a) Incrementar la concurrencia durante las inserciones debido a que los lectores pueden continuar leyendo las viejas porciones de la estructura del índice mientras se efectúa la inserción.
b) Este método es el mejor para trabajar con discos ópticos WORM (*Write-Once-Read-Many*) los cuales son un medio de almacenamiento excelente.

Sin embargo, los archivos de firma pueden ser muy lentos para grandes bases de datos debido a que su tiempo de respuesta es lineal respecto al número N de artículos en la base de datos. Los archivos de firma han sido usados en los siguientes ambientes: Máquinas paralelas y bases de datos de texto distribuidas.
Para permitir la búsqueda de partes de las palabras, se ha sugerido un método que consiste en dividir una palabra en caracteres sucesivos, formando tripletas (por ejemplo, "da", "ato", "tos" para la palabra "datos"). Así a cada tripleta se le aplica una función de dispersión.

3.4.3 Árbol de sufijos

Permite contestar algunas consultas complejas de manera eficiente que pueden resultar difíciles para un índice invertido. Su principal desventaja es su costoso proceso de construcción, que el texto debe estar disponible para su lectura al momento de la consulta y que los resultados no se muestran de acuerdo al orden de aparición en el texto. Esta estructura puede ser usad para indexar palabras (sin lista de parada) como el índice invertido así como para indexar cualquier caracter individual; lo cual, lo hace conveniente para una gama de aplicaciones. Sin embargo, para búsquedas basadas en palabras, los archivos invertidos realizan mejor la búsqueda a menos que la búsqueda compleja sea un asunto primordial.
Estos índices ven el texto como una cadena muy larga y cada posición en el texto es considerada como un sufijo de texto (es decir, una cadena que va desde esa posición de texto, hasta el final). Así, si se parte de la idea de ver al texto como un arreglo de caracteres, una cadena semi-infinita es una secuencia de caracteres de este arreglo, partiendo de un punto inicial hacia la derecha. El nombre de la cadena semi-infinita se define por la posición donde empieza.
Por ejemplo:

TEXTO *Erase que se era, en una tierra muy lejana ...*

Cadena semi-infinita 1	*Erase que se era, en una tierra muy lejana ...*
Cadena semi-infinita 2	*rase que se era, en una tierra muy lejana ...*
Cadena semi-infinita 7	*que se era, en una tierra muy lejana ...*
Cadena semi-infinita 11	*se era, en una tierra muy lejana ...*

Dos sufijos que inician en diferentes posiciones, son lexicográficamente diferentes, así, cada sufijo es identificado de manera única por su posición. No todas las posiciones de texto necesitan ser indexadas. Los puntos de índice se seleccionan del texto, iniciando desde la posición del texto que será recuperada. En la figura 3.12 se puede apreciar dicha estructura.

En esencia, un árbol de sufijos es una estructura de datos llamada *trie* construida sobre todos los sufijos del texto. Los apuntadores a los sufijos se almacenan en los nodos hoja. Para mejorar el espacio utilizado, este *trie* se compacta en un *árbol Patricia*. Esto involucra caminos unarios comprimidos (caminos donde cada nodo tiene un hijo). Una indicación de la siguiente posición del caracter a considerar se almacena en los nodos raíz.

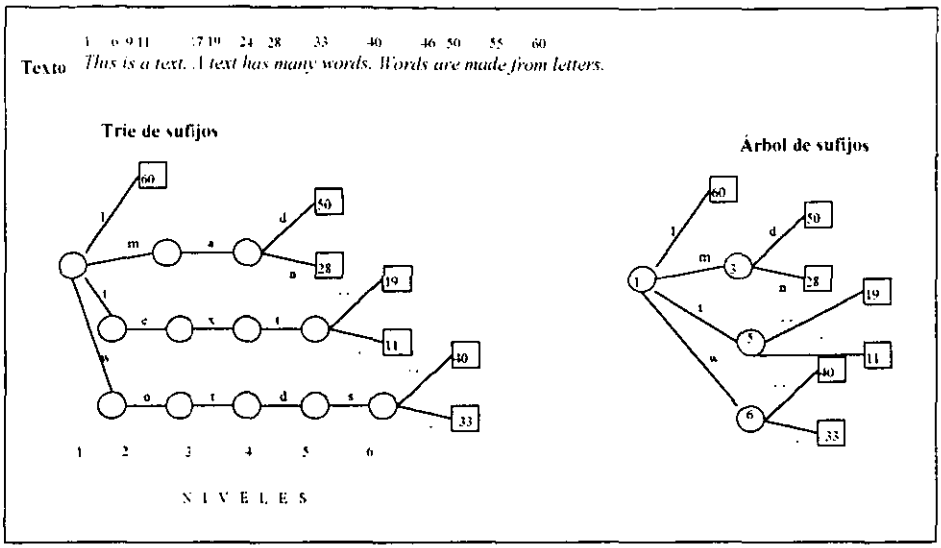


Figura 3.12 Comparación de tamaños del árbol de sufijos y el trie de sufijos para un texto en inglés

El problema con esta estructura es su espacio. Dependiendo de la implementación, cada nodo en el *trie* ocupa de 12 a 24 bytes, y por lo tanto aunque solo una palabra sea indexada, se produce un espacio del 120% al 240% sobre el tamaño del texto.
Si se implementa un árbol de sufijos, muchos patrones básicos tales como palabras, prefijos y frases se pueden buscar, aunque no son prácticos para textos largos.

3.4.4 Tries y árboles Patricia

Un archivo invertido puede implementarse usando una estructura llamada *trie*, de hecho, es una de las estructuras tradicionales para la creación de índices. Su búsqueda está basada en la descomposición digital de las cadenas semi-infinitas que lo forman. Si el alfabeto está ordenado, se tiene un árbol ordenado lexicográficamente. La raíz del *trie* usa el primer caracter, el hijo de la raiz usa el segundo caracter y así sucesivamente.

Un *árbol Patricia (PAT tree)* es un *trie* con características adicionales que permite que nodos únicos descendientes sean eliminados. Su nombre es acrónimo de "*Practical Algorithm To Retrieve Information Coded In Alphanumerical*". El objetivo de esta estructura es construir un índice para el texto con un tamaño menor o igual al mismo. Es un árbol digital donde los bits individuales de las llaves se usan para decidir la rama. Un bit cero genera una subrama a la izquierda, un uno, genera subrama a la derecha. Así que los árboles *Patricia* son árboles binarios. Estos árboles tienen en cada nodo interno un indicador de cual bit de la consulta será usado por la rama. Esto es dado por una posición absoluta del bit, o por un contador del número de bits que se tienen que "saltar" o desplazarse. Esto permite que los nodos internos con un descendiente sean eliminados, y de este modo todos los nodos internos del árbol producen una rama útil, es decir, ambos subárboles son no-nulos. Los árboles *PAT* son similares a los árboles compactos de sufijos.

Los árboles *PAT* almacenan valores llave en los nodos externos; los nodos internos no tienen información clave, sólo el contador de saltos y los apuntadores a los subárboles. Los nodos externos en un árbol *PAT* son cadenas semi-infinitas.
Para un texto de tamaño *n*, hay *n* nodos externos en el árbol *PAT* y *n-1* nodos internos. La siguiente figura (3.13) muestra un ejemplo de un árbol *PAT* sobre una secuencia de bits (normalmente, esto puede ser sobre una secuencia de caracteres). Se muestra un árbol PAT para el texto "01100100010111 ... " después de que las primeras 8 cadenas semi-infinitas (*sistrings*) hayan sido insertadas. Los nodos externos contienen una referencia a la cadena semi-infinita (indicados por recuadros). Los nodos internos contienen un desplazamiento (se indican por círculos). En este caso, se ha usado en cada nodo interno, el total de desplazamientos en bits que serán inspeccionados.

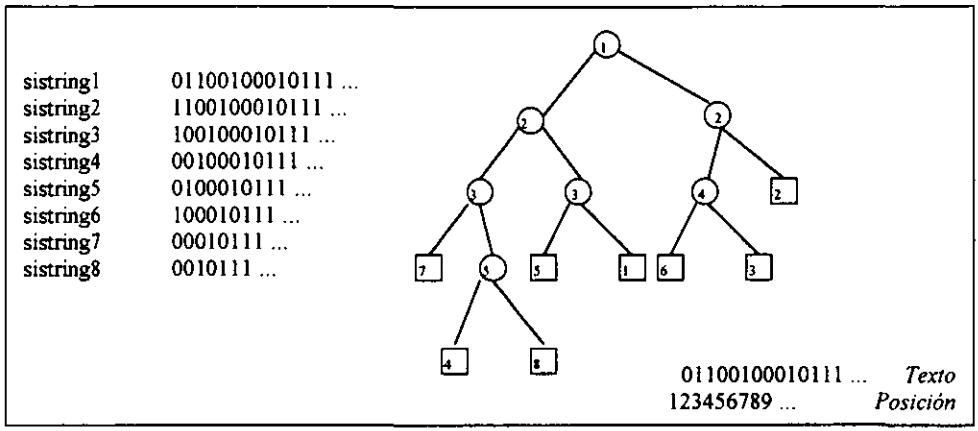


Figura 3.13 Árbol PAT con las primeras 8 sistrings insertadas

Para buscar el nodo externo para la consulta 00101, se inspecciona el primer bit (como es un 0, se va a la izquierda), el segundo bit (también 0, se va a la izquierda), entonces el tercer bit (que es un 1, se va por la derecha), y entonces el bit cinco (es un 1, se va a la derecha). Debido a que se tienen que "saltar" o desplazar algunos bits (en este caso el bit 4), una vez que se alcanza el nodo deseado, se debe hacer una comparación final con una de las cadenas semi-infinitas almacenada en un nodo externo del subárbol actual, para asegurar que todas las posiciones de los bits coinciden. Si no, entonces la clave no está en el árbol.
Para algunos tipos de búsquedas es deseable indexar todas las posiciones del texto, pero para muchos otros no son necesarios. Por ejemplo, si solo se tiene interés de buscar palabras o frases, entonces solo aquellas cadenas semi-infinitas con las que empiezan las palabras serán necesarias. La decisión de cuantas cadenas semi-infinitas se incluyen en el árbol, depende de la aplicación y puede decidirse dependiendo del tamaño del índice y los requerimientos de búsqueda.

Esta estructura soporta datos de prefijos y búsquedas por rangos, así como búsquedas por proximidad y de expresiones regulares. Un árbol *PAT* necesita un espacio O(n), cuya constante es importante actualmente en las aplicaciones prácticas. Se necesitan cuando menos dos apuntadores para todo nodo interno y uno para todo nodo externo. Una manera de implementarlo es por medio de un arreglo *PAT* que contiene los apuntadores al texto (los nodos externos del *trie*) ordenados lexicográficamente por las cadenas semi-infinitas. A continuación, se muestra la figura 3.14 que muestra el arreglo PAT.

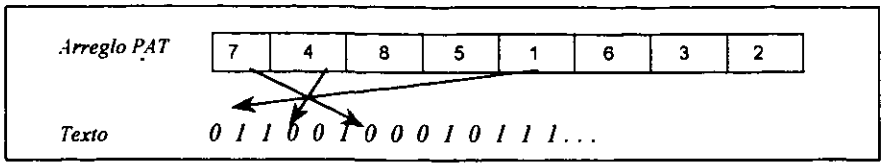


Figura 3.14 Un arreglo PAT asociado a su texto

Se puede concluir que comparando los arreglos PAT contra los índices invertidos y los archivos de firma ocurre lo siguiente:
Los archivos de firma emplean técnicas *hash* para producir índices iniciales con un tamaño entre el 10 y 20% del tamaño del texto; además de que su almacenamiento es pequeño. Sus desventajas: El tiempo de búsqueda es lineal lo cual es un grave inconveniente para grandes textos; y segundo, se pueden encontrar resultados que no satisfagan a la consulta que aunque no es algo frecuente, puede ocurrir.

Por otro lado, los archivos invertidos requieren de un almacenamiento que varía del 30 al 100% (dependiendo de la lista de parada y la estructura de datos) y su tiempo de búsqueda es logarítmico. Una ejecución similar se puede hacer con los arreglos *Patricia*. La gran ventaja de los arreglos *Patricia* es su uso potencial en otra clase de búsquedas que no se puedan resolver con los archivos invertidos o sea ineficiente, tal es el caso de las búsquedas por frase, búsquedas de expresiones regulares, búsquedas por aproximación, repeticiones frecuentes, etc.

3.4.5 Indexación por tríadas

Este método propone la indexación no de palabras, sino de tríadas de letras; se basa en la investigación y vivencias del desarrollo de un sistema de información.
Una tríada válida para una palabra dada es una subcadena formada por la concatenación de tres caracteres contiguos que conforman la palabra. Por ejemplo, el conjunto de tríadas que puede ser construido con base en la palabra **CIENCIA** es: **CIE, IEN, ENC, NCI,** y **CIA**.
La indexación por tríadas consiste en descomponer cada uno de los *términos índice* de un documento en tríadas de letras y utilizar la frecuencia de ocurrencia de éstas para construir un patrón de tríadas (histograma) que conformará la indexación del documento.

Esta técnica cuenta con algunas características que la hacen muy atractiva:

- La facilidad de su aplicación.
- El proceso de indexación deja de depender casi en su totalidad de la lengua alfabética en que están escritos los documentos.
- Rapidez en las búsquedas.
- Archivos de indización no demasiado grandes.
- Sistemas de consulta simples y naturales en su empleo para un usuario final, ya que no es necesario emplear operadores booleanos para formular las búsquedas.

La razón por la que el proceso de indexación por tríadas sigue dependiendo del idioma es debido a que no todas las posibles combinaciones de tríadas son igualmente válidas en todas las lenguas alfabéticas. Por ello, se debe realizar un análisis estadístico de las tríadas válidas del idioma alfabético en que está escrita la colección de documentos. Este análisis no sólo permitirá determinar

las tríadas válidas con las que deberán indexarse todos los documentos, sino también determinar aquellas triadas poco o muy frecuentes.

Algunas observaciones negativas de este sistema de indexación son:

- Las tríadas son poco frecuentes por su misma rareza, y por ello, pueden ser eliminadas sin mayor perjuicio al rendimiento de un SRI.
- Las tríadas muy frecuentes aparecerán con toda seguridad en todos los documentos de la colección, lo cual hace a estas tríadas malos candidatos para caracterizar y diferenciar los documentos de la colección.
- La tercer observación es una consecuencia de las dos anteriores, pues al eliminar aquellas tríadas poco frecuentes, así como las muy frecuentes, el conjunto de tríadas válidas que conformarán el lenguaje de indexación, puede reducirse considerablemente.

Es preciso definir el conjunto de tríadas que se contemplarán para cada idioma, pues éstas varían de un lenguaje a otro. Una vez establecido el conjunto de tríadas que conformará el lenguaje de indización, se pueden construir las indizaciones de los documentos. En esta construcción dado que no se basa en palabras sino en tríadas, cada representación estará formada por uno o más vectores, en donde el número de vectores dependerá de la longitud promedio de los documentos en la colección. Lo anterior se debe a que no es conveniente usar un solo vector para la representación de documentos muy extensos, pues, en este caso, se corre el riesgo de que la representación reporte la presencia de la mayoría de las tríadas válidas; una mejor opción es segmentar los documentos muy extensos y construir para cada uno de ellos un vector de ponderación de tríadas.
La ponderación por tríadas se acotan de la siguiente manera:

f^k_{ij}
- 0 Si la tríada T_j no aparece dentro del k-ésimo segmento del i-ésimo documento.
- 1 Si la tríada T_j aparece una vez dentro del k-ésimo segmento del i-ésimo documento.
- 2 Si la tríada T_j aparece más de una vez dentro del k-ésimo segmento del i-ésimo documento.

Así, es posible entonces construir la matriz de indización, construida por la concatenación sucesiva de las representaciones de los documentos en la colección.
Dado que la consulta está formada por palabras no nulas, se puede definir también como el documento pero ahora con un contador para las tríadas que se forman a partir de sus palabras.

La manera de medir la similitud entre el texto y la consulta, está dada por la siguiente expresión:

$$\text{Sim}^{*}_{s} = \sum_{j=1}^{m} p_j f^{*}_{sj} + 5\sum_{j=1}^{m} P_j F^{*}_{sj}$$

Donde:

$P_j = 0$ Si $p_j = 0$

$P_j = 1$ Si $p_j <> 0$, análogamente ocurre con F^{*}_{sj}

Como el objetivo es cuantificar la similitud máxima entre un documento y una petición dada, se puede establecer esta cuantificación por medio del máximo de los valores de similitud de los segmentos de un documento con la siguiente expresión:

$$Sim_i = \text{Máx}(\sum_{j=1}^{m} p_j f_{ij}^k + 5\sum_{j=1}^{m} P_j F_{ij}^k)_{k=1}^{r}$$

Donde de nuevo, F_{ij}^k se define análogamente a P_j.

Al ser este método de reciente investigación en la UNAM, se pueden consultar a detalle las pruebas efectuadas y sus resultados en [Vega94].

3.5 Thesaurus

3.5.1 Estado del arte del thesaurus

El término *thesaurus*, como se usa en el área de la recuperación de información, se emplea para denotar una lista de términos junto con alguna información sobre su uso y mutua relación.
Muchos *thesaurus* tienen un propósito especial en el sentido de que sus términos han sido seleccionados del vocabulario de una disciplina particular o base de datos. Por cada término listado, algunas relaciones pueden ser incluidas para indicar: Otros términos que tienen significado idéntico (*sinónimos*); términos que son reducidos o generalizados en significado, notas para indicar aquellos términos que pueden tener diferentes significados (*homónimos*), etc. Es importante destacar que el *thesaurus* no tiene la definición de los términos como ocurre en un *diccionario* o un *glosario*.

Cuando se usa con una base de datos particular, su propósito es dar a conocer al usuario el vocabulario relevante y las reglas que gobiernan el descriptor seleccionado para el vocabulario de la base de datos.

A continuación, se presenta en las tablas 3.2 y 3.3 una muy breve parte del *thesaurus de U.S. Bureau of Ships* (1965) y del *Thesaurus of Engineering Terms* (*TEST*, 1967) respectivamente.

COAXIAL CABLES
BROADER TERMS:
ELECTRICAL CABLES
NARROWER TERMS:
LIQUID FILLED COAXIAL CABLES
RELATED TERMS:
PULSE CABLES
RADIOFREQUENCY CABLES
COAXIAL FILTERS
BROADER TERMS:
ELECTRIC FILTERS
FILTERS (ELECTROMAGNETIC WAVE)
RELATED TERMS:
RADIOFREQUENCY FILTERS
COBALT
BROADER TERMS:
GROUP VIII ELEMENTS
METALS
TRANSITION ELEMENTS
COBALT ALLOYS
BROADER TERMS:
ALLOYS
COBALT COMPOUNDS
COBALT ISOTOPES (RADIOACTIVE)
USE:
COBALT
RADIOACTIVE ISOTOPES

Tabla 3.2 *Thesaurus del U.S. Bureau of Ships*

El thesaurus original reporta 4,600 términos basados en 170,000 reportes.
Lista los términos en orden alfabético con un indicador de ampliación o limitación (BROADER o NARROWER, respectivamente), términos relacionados (RELATED), sinónimos (INCLUDES) y preferencias (USE).

Coaxial cables 0901
UF Coaxial lines
Lineal filled coaxial cables
BT Transmission lines
RT-Power lines
Submarine cables
Telegraph cables
Telephone cables
Coaxial filters 0901
BT Electric filters
RT Microwave filters
Radiofrequency filters
Coaxial lines
USE Coaxial cables
Cobalt 0702
BT Metals
Transition metals
RT - Cobalt isotopes
Cobalt 60 1802
BT Cobalt isotopes
Isotopes
Nuclides
Radioactive isotopes

Tabla 3.3 *Thesaurus of Engineering Terms (TEST)*

Este *thesaurus* es usado por la *Enginners Joint Council*.
Para 1967 contenía 23,364 entradas asociadas a las extensiones (BT), limitantes (NT), términos relacionados (RT), sinónimos o términos de uso (UF) y preferencias de términos (USE).
Los números que le siguen a los términos principales se refieren a un conjunto particular de subclases. Por ejemplo, 0901 denota el tema: "Ingeniería eléctrica y electrónica: Componentes"; y el 1802 denota el tema: "Ciencia nuclear y tecnología: isótopos".

Lo anterior, presenta al *thesaurus* como un proveedor y controlador de un vocabulario apropiado que permite ayudar al usuario a refinar sus consultas, pero actualmente, el *thesaurus* tiene otros objetivos más importantes desde el punto de vista de la máquina de búsqueda: Permite la indexación de los documentos y por ende, su recuperación. Así que un *thesaurus* bien diseñado puede ser de gran valor. El índice se puede construir por las entradas más representativas del *thesaurus* y en la búsqueda, si no se recuperan documentos, el *thesaurus* puede ampliar la consulta siguiendo las ligas entre términos; ya que un *thesaurus* puede ser visualizado como una red semántica donde cada término es representado por un nodo. Si dos términos están relacionados entre sí, sus nodos se conectan por un arco etiquetado con el nombre de la relación. Todos los términos que están directamente relacionados a un término dado pueden ser alcanzados de modo iterativo siguiendo todas las conexiones. Entonces es recomendable proporcionar el *thesaurus* en línea para que simplifique el proceso de la reformulación de consultas.

Existe una basta literatura sobre los principios, metodologías y problemas que se involucran en la construcción de un *thesaurus*. Sin embargo, una pequeña parte de dicha bibliografía trata la construcción automática. Esto es importante de resaltar, debido a que refleja un estado del arte muy marcado de la construcción manual sobre la automática.
Existe mucho escepticismo sobre la posibilidad de un proceso automático ya que se involucran una amplia variedad de relaciones entre términos: jerárquicas, no-jerárquicas, equivalencias y asociaciones.

Algunas metodologías automáticas han surgido en las últimas décadas y como muchas otras ramas de la recuperación de la información, estos métodos son fuertemente apoyados por la estadística como una alternativa a los métodos lingüísticos.

Una característica importante en un *thesaurus* automático es la coordinación, que se refiere a la construcción de frases a partir de los términos individuales. Existen dos opciones de coordinación, pero ambas pueden ser implementadas (nivel intermedio de coordinación).

La pre-coordinación.- Es aquella que contiene frases, las cuales estarán disponibles para su indexación y recuperación. Su ventaja que el vocabulario es muy preciso y en consecuencia, se reduce la ambigüedad en la indexación y búsqueda. Su desventaja es que el usuario que plantea la consulta, no es consciente de las reglas empleadas para la construcción de frases.

La post-coordinación.- No permite frases, por lo que éstas se construirán mientras se efectúa la búsqueda. Su ventaja es que el usuario no tiene que preocuparse sobre la ordenación exacta de las palabras en la frase, ya que las combinaciones para generar las frases se pueden hacer durante el proceso de búsqueda

La construcción automática de frases aún es muy difícil y los *thesaurus* automáticos generalmente emplean la post-coordinación.

La relación que debe existir entre los términos es otro aspecto importante debido a la conexión que existe entre ellos. En 1972 se especificaron tres categorías para las relaciones:

Relaciones de equivalencia.- Incluyen sinonimia y casi-sinonimia (por ejemplo: genética y herencia)

Relaciones jerárquicas.- Se refiere a las relaciones de género y especie (por ejemplo: perro y french poodle)

Relaciones no-jerárquicas.- Identifican términos relacionados conceptualmente (por ejemplo: autobús y asiento)

En 1985, se proporcionó una alternativa a la clasificación anterior:

Relación de taxonomía y sinonimia.- Similares a la categoría anterior.

Relación total y parcial.- Incluye ejemplos como conjunto y elemento.

Relación de colocación.- Hace referencia a las palabras que co-ocurren frecuentemente en la misma frase o sentencia.

Relación paradigmática.- Se refiere a las palabras que tienen la misma semántica. Por ejemplo los términos: sinónimo y sinonimia.

Relación de antónimos.- Se refiere a las palabras que tienen un significado opuesto

3.5.2 Etapas de construcción

La construcción automática parte de la idea de usar la colección de documentos como el origen del *thesaurus*. Se aplican procedimientos estadísticos para identificar los términos importantes así como sus relaciones significativas a partir del conocimiento semántico. Todo el proceso se divide en tres etapas:

1) ***Construcción del vocabulario.*** Involucra normalización y selección de términos dependiendo del tipo de coordinación seleccionada. Implica la identificación de los términos más informativos para el *thesaurus*, se apoya en las técnicas para la obtención de la lista de parada y para la obtención de raíces.

2) ***Cálculo de similitud entre términos.*** Identifica las asociaciones estadísticas significativas entre términos.

3) ***Organización del vocabulario.*** El vocabulario seleccionado se organiza generalmente por jerarquías a partir de las bases de la etapa 2.

A continuación, se presentan a detalle los métodos para cada una de estas etapas.

Construcción del vocabulario

Para la evaluación y selección existen métodos de evaluación estadística que asignan un valor para cada término, se presentan aquí tres de ellos:

- Selección por ocurrencia o frecuencia

  Es uno de los métodos más antiguos y discutido. La idea básica consiste en que cada término debe ser colocado en una de tres categorías con respecto a la colección de documentos: Frecuencia alta, mediana o baja. Los términos de frecuencia media son los mejores candidatos para la indexación y búsqueda; los términos del rango de baja frecuencia son de bajo impacto en la recuperación y los de alta frecuencia son muy generales y de impacto negativo en las tareas de recuperación con precisión.

- Selección por el valor de discriminación (VD)

  Mide el grado asignado a un término para determinar su capacidad de discriminación o distinción entre los documentos de la colección; para ello, se usa una función de similitud apropiada. Después de esto, el k-ésimo término evaluado se remueve del vocabulario de indexación y el mismo promedio de similitud vuelve a calcularse.
  El valor de discriminación (VD) se calcula como:

  $$VD(k) = (\text{promedio de similitud sin } k) - (\text{promedio de similitud con } k)$$

  Buenos discriminantes son aquellos que decrecen en su similitud promedio por su presencia, es decir, aquellos cuyo *VD* es positivo; estos términos son los que deben incluirse en el vocabulario y el resto debe desecharse. Por el contrario, aquellos términos que no deben ser discriminados tienen un *VD* negativo.

- Selección basada en el modelo Poisson

  Es uno de los métodos con las investigaciones más recientes (1974, 1975 y 1990). Se sabe que la distribución Poisson es una distribución aleatoria discreta que puede usarse para modelar muchos fenómenos aleatorios incluyendo los errores tipográficos en una página. En todas las investigaciones de este modelo, se ha encontrado que las palabras triviales siguen una distribución de tipo Poisson, mientras que la distribución de las no-triviales se desvía de dicho modelo. Estos resultados se usan para seleccionar las palabras no-triviales e incorporarlas como parte del *thesaurus*.

  Para detallar mayor detalle de algunos algoritmos para la distribución Poisson se recomienda [NBBC91]. Para estudio de la función de distribución Poisson y pruebas de hipótesis se recomienda la siguiente bibliografía: [Meye86] y [MeSW86]

La fase de construcción se usa para construir frases si se desea (como ya se mencionó, depende del nivel de coordinación). Dos métodos se consideran a continuación:

- Procedimiento de Salton y McGill

  Es un estándar propuesto en 1983 y se adaptó en 1986. Es una alternativa estadística para los métodos sintácticos y semánticos de identificación y construcción de frases. Las palabras componentes de una frase deben ocurrir frecuentemente en un contexto común, como por ejemplo la misma sentencia. Un criterio contextual más riguroso puede ser que se defina una distancia específica entre ellas. El segundo requerimiento en general es que las palabras componentes representen algún componente con significado más amplio y su frecuencia de ocurrencia sea suficientemente alta. El algoritmo que aplica estos criterios y su implementación en lenguaje C puede verse en [FrBa92]

- Procedimiento de Choueka

  Se planteó en 1988 y hace referencia a las frases cuyo significado no puede derivarse a partir de las palabras componentes. La propuesta del algoritmo es estadística y combinatoria y requiere de una colección de cuando menos un millón de términos para ser efectiva. El autor de este procedimiento ha tenido buenos resultados en la identificación de frases con significado y está efectuando adecuaciones al algoritmo [FrBa92].

Cálculo de similitud entre términos

Una vez que se tiene definido el vocabulario, el siguiente paso es determinar la similitud estadística entre pares de términos. Como ya se definieron previamente, existen algunas medidas de similitud que pueden ser empleadas: la del coseno, la de Dice, etc.

Organización del vocabulario

Este último paso puede efectuarse con cualquier programa de Cluster apropiado.
Es conveniente precisar que hay aún otras tareas por efectuarse sobre un *thesaurus* como lo son: su evaluación, su mantenimiento y la manera de automatizar su uso.

3.6 Asignación de puntos *(Keeping Score)*

– Algoritmo de Salton[1] –

Como se mencionó en el capítulo II, al recuperan una lista de documentos relevantes (*hitlist*) a partir de una consulta planteada, se asigna a cada uno de ellos un determinado valor o puntuación que se encuentra en el rango de 0 a 100. Un documento recibe una puntuación (también llamada *score*) de 0 si éste no empareja del todo con el criterio de búsqueda; es por ello que como se verá a detalle en el siguiente capítulo, la función **CONTAINS** regresa un número que indica tal puntuación. Así una búsqueda SQL con el criterio **WHERE (descripcion, 'distribuido') > 0** puede encontrar los documentos que emparejan con el criterio.

El presente algoritmo se utiliza para determinar la puntuación de cada renglón en la tabla:

$$\textbf{Puntuación} = 3f(1+\log(N/n))$$

Donde:
f = número de veces que el término aparece en el documento (frecuencia)
N = número total de renglones en la tabla
n = número de renglones que contienen cuando menos una ocurrencia del término de búsqueda

Después de que la puntuación es calculada, la máquina de búsqueda deberá convertirla en un entero entre 0 y 100.

[1] *Gerald Salton fue un pionero en el tema de la recuperación de la información, en el presente trabajo se utilizó bibliografía de su autoría [SaMc83] así como mucha otra en la que se le considera como una importante referencia [BaRi99], [FrBa92], [Vega94].*

Ejemplo:

Suponiendo que se desea plantear una búsqueda para la palabra "sonido" y suponiendo que hay 32,000 renglones, de los cuales sólo 3,200 contienen cuando menos una vez la palabra buscada, se tiene la sustitución:

N = 32,000; n = 3200; f = 1
puntuación = 3*1*(1+log(32000/3200))
= 3*(1+log(10))
= 3*(1+1) = 3*2
= 6

Si ahora se requiere que la palabra aparezca exactamente 5 veces, su puntuación cambia:

N = 32,000; n = 3200; f = 5
puntuación = 3*5*(1+log(32000/3200))
= 15*(1+log(10))
= 15*(1+1) = 15*2
= 30

Se puede apreciar que una frecuencia 0, siempre obtiene una puntuación 0, independientemente de los valores de los parámetros N y n.
Así, la frecuencia de un término de búsqueda en cada documento afecta su puntuación, pero éste puede incrementarse de manera proporcional para cada documento que contenga el término de búsqueda si éste aparece en un pequeño porcentaje de los renglones.

La puntuación puede ser alta cuando el término es relativamente inusual para el conjunto de documentos. En la tabla siguiente se pueden apreciar las variaciones de valores.

f	*N*	*n*	*P*	*Observaciones*
1	1,000	3	10.5	*0.0093% de los renglones contienen el término.*
3	1,000	3	31.7	
10	1,000	3	105.6	
1	32,000	3,200	6.0	*10% de los renglones contienen el término.*
5	32,000	3,200	30.0	
1	10,000	8,500	3.2	*31.25% de los renglones contienen el término.*
5	10,000	8,500	16.0	
10	10,000	8,500	32.1	
1	32,000	16,000	3.9	*50% de los renglones contienen el término.*
2	32,000	16,000	7.8	
10	32,000	16,000	39.0	
				A mayor frecuencia, aumenta la puntuación sin importar el resto de los parámetros.

Tabla 3.4 Cálculos de la fórmula de Salton con diferentes parámetros

3.7 Comentarios finales

Por muchos problemas que existan en la búsqueda de texto, no existe un algoritmo que sea el mejor para todas las aplicaciones. La elección de un algoritmo dependerá de la colección de documentos de entrada, tales como su tipo o texto usado (por ejemplo, si el texto está en lenguaje natural, es un conjunto de bits o son caracteres del DNA para bases de datos biológicas), el tamaño del texto, los patrones, etc. Además, recientes resultados muestran que próximas investigaciones emprenderán la exploración de algoritmos basados en la no-comparación (por ejemplo, basados en operaciones aritméticas y lógicas), algoritmos híbridos y adaptables, más casos prácticos, uso de nuevos conceptos (por ejemplo, n-gramas), reducciones para problemas más simples y nuevos índices para texto en casos especiales.

Tradicionalmente, los índices estaban basados en palabras, campos y documentos. Aunque aún se usan para explotar la estructura del texto, recientes resultados sugieren que los índices basados en cadenas son mucho más flexibles y poderosos. En particular, las principales diferencias son que la estructura dada (documentos, campos, etc.) no es necesaria para el texto; el concepto de palabra no se usa, obtención de un índice donde prefijos o frases puedan ser recuperados; y un particular énfasis en que se pueden recuperar posiciones de texto e indexarse.

En este capítulo no se presentaron los algoritmos, ni sus complejidades pues son parte de otro amplio tema de estudio ya que serán los trabajos de análisis de algoritmos los que se den a esa tarea.
Otras cosa importantes de resaltar que no se contemplaron, son los métodos de compresión de datos que se deben unir a las técnicas vistas, pues no puede ignorarse que los volúmenes de información son cada vez mayores y que los dispositivos de almacenamiento no marcan un límite en esto. Por el contrario, los nuevos equipos de cómputo poseen *n* procesadores, lo que permite pensar y de hecho, ya hay tratados del tratamiento en paralelo que se puede dar en las búsquedas.
Un caso especial sería también hablar de las bases de datos distribuidas, las cuales ya están trabajando sobre todo en la Web, donde ya se tienen accesos remotos y fragmentación.

Capítulo IV

Máquinas de búsqueda en los principales RDBMS

" We must try to trust one another. Stay and cooperate."

Jomo Kenyatta

4.1 Justificación

Retomando lo planteado en el capítulo I, donde se asegura que la aplicación de la tecnología de bases de datos ha sido esencial para la operación de muchas empresas. Los sistemas de bases de datos permiten a las organizaciones administrar datos estructurados, pero un alto porcentaje de los datos digitales actuales son texto no estructurado y documentos, contenidos en nuevas aplicaciones de bases de datos. Mucho de este texto, contiene información de misión crítica, almacenada en documentos del corporativo, herramientas de ventas, correos electrónicos, memorandums, páginas Web y más. Pese a ello, no existe la misma fuerza comercial de las bases de datos relacionales o de las objeto-relacionales en el manejo eficiente de texto que la que existe para (los ya no suficientes) datos estructurados.

Las organizaciones se agobian con la información textual, la cual no es fácil de ligar a la infraestructura existente o es inaccesible para la obtención de información que apoye una toma de decisiones. Por ejemplo, muchas compañías de seguros almacenan descripciones muy puntuales y descriptivas de sus clientes. Si se pudieran analizar estos textos, la información resultante podría ayudar a definir nuevas políticas o campañas de mercado que favorecerían indudablemente al desarrollo de los corporativos y de las naciones. Es una ironía que las empresas que han invertido grandes cantidades de dinero en los últimos años para construir aplicaciones que les permitan extraer rápidamente información a partir de sus datos, tengan un 90% de los mismos sin considerarse o simplemente, sin poder consultarse bajo algunos criterios tan sencillos de definir con el SQL. Mientras que robustas bases de datos relacionales basadas en el SQL fueron usadas para datos estructurados, máquinas de recuperación fueron usadas para el texto. El texto almacenado en las bases de datos relacionales no es buscado de forma nativa; estas tecnologías se integran como ambientes costosos y complicados. Sólo se le ha almacenado, accesado y manipulado de manera *ad hoc*, desperdiciándose así la disciplina de las bases de datos (seguridad, consistencia, concurrencia, integridad, etc.) No sólo es inconveniente para los programadores de la aplicación sino también la existencia de la disyuntiva de administración de datos que no permite forzar y asegurar la consistencia. De tal modo que la difícil tarea de optimización de consultas en el límite de las aplicaciones del programador y los beneficios de la transparencia de la base de datos son imposibles de realizar.

Los negocios, necesitan una solución unificada, basada en estándares y escalable para administrar información de cualquier tipo. Su almacenamiento, acceso, análisis y distribución, son algunos de los desafíos a los que se enfrentan.

Las compañías constructoras o "creadoras" de los RDBMS líderes en el mundo, ya se han dado a la tarea de crear productos que les permitan almacenar y resolver el manejo no trivial de texto, esto, desarrollándolo como parte del mismo RDBMS o bien, con productos extras dependiendo de la complejidad de la búsqueda o manipulación. Actualmente, los RDBMS han sido complementados con otros productos para poder reforzar las búsquedas no triviales o inteligentes sobre texto que les impone el uso del SQL. La idea central es tener el RDBMS como motor de creación, consulta, administración y manipulación de datos básicos y para los tipos complejos, poder incorporar una capa de software que satisfaga las necesidades del usuario. Por ejemplo para manejo de imágenes, texto, video, etc.

En el presente capítulo se presentan las capacidades de algunos de los productos comerciales que apoyan a los RDBMS en la búsqueda y el manejo de texto. Su eficiencia no es un factor posible de comparar debido a que no se cuenta con el acceso al software y sobre todo, bajo las mismas condiciones de plataforma y misma cantidad de datos, sin embargo, se presentan los alcances de cada uno de ellos y algunas diferencias sintácticas.

Las herramientas analizadas en su funcionalidad son, por parte de *DB2:* ***Text Extender;*** de *Informix:* ***Excalibur Text Search DataBlade Module*** y ***Verity Text Search DataBlade Module;*** y por parte de *Oracle:* ***ConText Cartridge.*** La justificación del análisis de estos productos es principalmente, que son los tres RDBMS más importantes a nivel mundial y sobre los que se desarrolla un gran porcentaje de las aplicaciones.

De manera muy general, se puede decir que todas estas herramientas trabajan de manera similar e incorporan las mismas capacidades de búsqueda y recuperación. Esto es debido a que se apegan a los estándares y a una teoría preestablecida. Aplican algunas políticas basadas en los conceptos teóricos de los capítulos anteriores, tales políticas se pueden apreciar en la figura 4.1, donde dependiendo del tipo de búsqueda se tendrá una técnica de recuperación diferente.

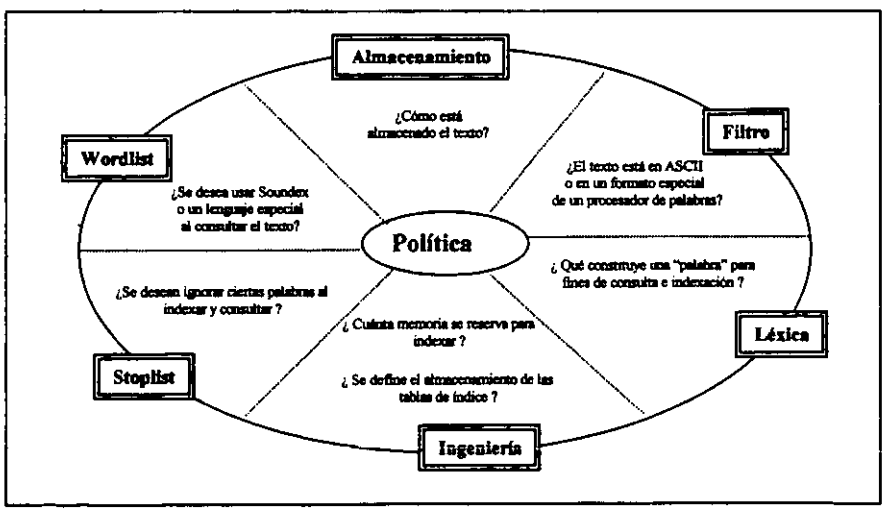


Figura 4.1 Políticas para la recuperación de texto con un RDBMS

DB2 Text Extender

4.2 DB2 Text Extender (IBM)

4.2.1 Generalidades del producto

El *DB2 TEXT EXTENDER* es un miembro de la familia *IBM DB2 Extender*. Los otros componentes son: *DB2 Image Extender, DB2 Audio Extender* y *DB2 Video Extender*. Cada uno de estos componentes trata con tipos de datos no estructurados como el texto, las imágenes, sonido y video. En este caso, sólo se aborda la extensión para el tratamiento de texto.

En el núcleo del *DB2 Text Extender* está la poderosa búsqueda lingüística de *IBM* y la tecnología de la minería de textos, lo que permite ser una herramienta de minería inteligente y versátil. Es una manera poderosa de buscar y extraer la información importante de documentos y archivos no sólo que estén guardados en la base de datos *DB2*, sino también en archivos almacenados fuera de la misma. La *minería de datos de DB2* proporciona fundamentos tecnológicos que permiten un análisis más sofisticado de texto.

Text Extender, al incorporarse al manejador de bases de datos *DB2*, adquiere todas las ventajas de su uso.

4.2.2 Capacidades de búsqueda del producto

Estos tipos de búsquedas pueden integrar el texto a búsquedas con datos estructurados del negocio de manera natural con el lenguaje *SQL*, que se ha visto extendido (*SFQL*).

- Búsqueda borrosa o difusa de palabras con ortografía similar a la del término de búsqueda.
- Búsqueda de texto libre, es decir, donde el patrón de búsqueda se define en lenguaje natural.
- Búsquedas por patrones (que contengan texto específico)
- Búsquedas por sinónimos de una palabra o frase.
- Búsqueda para documentos que contienen palabras en cualquier secuencia.
- Búsqueda para documentos por variación de palabra (raíces).
- Construcción de un *thesaurus* propio.
- Búsqueda de palabras por proximidad en la misma sentencia o párrafo.
- Búsquedas con comodines, usando máscaras para caracteres iniciales, intermedios o finales en una palabra.
- Búsqueda de documentos en varios lenguajes en varios formatos de documentos.
- Búsqueda de palabras con un sonido similar al término de búsqueda.

4.2.3 Idiomas, formatos y códigos

DB2 Text Extender se puede usar sobre los siguientes idiomas:

Franco Canadiense	Catalán	Danés	Árabe
Holandés	Finlandés	Francés	Hebreo
Alemán	Islandés	Italiano	Islandés
Noruego	Portugués	Sueco	Ruso
Suizo Alemán	Español	Inglés Británico	
Inglés Norteamericano	Japonés	Chino (tradicional y simplificado)	

Text Extender necesita conocer el idioma en el que se encuentran los documentos para usar el diccionario correcto en el proceso lingüístico. También necesita conocer el formato (o tipo) de los documentos sobre los que se efectuará la búsqueda. Los formatos aceptados son:

ASCII	*HTML*	*WordPerfect5*
TDS	*AMI*	*FFT*
MSWORD	*RFT*	

Para tipos de documentos no soportados, se especifica un identificador numérico. Los valores válidos están entre 1 y 100. Este valor se pasa como un formato de origen al ***user exit*** que convierte el formato original a *TDS*.

Cada base de datos *DB2* usa un código particular para el almacenamiento de los datos, *Text Extender*, como una aplicación del *DB2*, también emplea el mismo código que el de la base de datos. Los códigos soportados son: *EBCDIC, ASCII, CCSID* (se extendió para documentos en Unicode), *DBCS* (doble byte character), *UNICODE* y *UCS2* para soportar lingüística arábiga, hebrea y rusa.
Toda esta información es necesaria al momento de indexar el texto de los documentos.

4.2.4 El proceso de indexación

El proceso de indexación con el *Text Extender* de *DB2*, se lleva a cabo de la siguiente manera: cuando se indexan y recuperan los documentos, se efectúa un análisis del texto. El proceso lingüístico depende del tipo de índice; cuando se indexa por N-gramas, no se aplica un proceso lingüístico. El proceso lingüístico usado para indexar los documentos consiste en:

- Análisis básico de texto
  - Reconocimiento de términos (reconocimiento de tokens)
  - Normalización de términos a su forma estándar
  - Reconocimiento de sentencias
- Reducción de los términos a su forma base (obtención de la raíz)
- Filtrado de término contra la Lista de Parada
- Descomposición (división de términos compuestos)

En la tabla siguiente (4.1) se ejemplifica cómo se realiza el proceso lingüístico de un texto en inglés con el uso del *Text Extender*. Se observa el término del documento, el del índice y el que se obtiene una vez realizado el proceso lingüístico.

Texto del documento	*Término en el índice*	*Proceso Lingüístico*
Mouse Käfer	**mouse** **kaefer**	Análisis básico de texto (normalización)
mice swum	**mouse** **swim**	Reducción a su forma base
system-based Wetterbericht	**system-based,** **system** **base** **wetterbericht,** **wetter** **bericht**	Descomposición
a report on animals	**report** **animal**	Filtrado contra lista de parada. Las palabras de parada son: a, on

Tabla 4.1 Proceso lingüístico de indexación con Text Extender

Existen diversos tipos de indexación con el *Text Extender*:

- Indexación Precisa

En este tipo de indexación, *Text Extender* busca los términos tal cual han sido capturados por el usuario que plantea la búsqueda. Un ejemplo con SQL de una consulta posible dado el índice preciso, sería la siguiente:

```
SELECT fecha, tema
FROM DB2TX.muestra
WHERE DB2TX.CONTAINS (comentario, 'PRECISE FORM OF "educa"') = 1
```

También es posible efectuar búsquedas sobre expresiones regulares usando los comodines de las cerradura positiva y de Kleene.
La ventaja de este tipo de índice es que la búsqueda es más precisa y por ende, la indexación y la recuperación son más rápidas. Debido a que cada forma diferente y ortográfica de los términos son indexadas, es necesario más espacio de disco que para un índice lingüístico.

- Indexación Lingüística

Para este tipo de indexación, la búsqueda permite variación de términos, tales como plurales de un sustantivo o variantes de tiempo de un verbo. Un ejemplo con SQL de una consulta posible dado el índice lingüístico sería la siguiente:

```
SELECT fecha, tema
FROM DB2TX.muestra
WHERE DB2TX.CONTAINS (comentario,, 'STEMMED FORM OF "educa"') = 1
```

Los índices de este tipo requieren de menos espacio de disco. Sin embargo, la indexación y búsqueda pueden llevar más tiempo que para un índice preciso.

- Indexación Dual

Con este tipo de indexación, se puede buscar una palabra con variaciones o no.
Este tipo de índice es el que requiere de más espacio en disco; la búsqueda y recuperación son más lentas que para el índice lingüístico y no se recomienda para un gran número e documentos.

- Indexación por N-gramas

Un índice de este tipo analiza el texto efectuando un análisis sintáctico sobre un conjunto de caracteres; no se basa en algún diccionario. Si el texto analizado contiene caracteres DBCS, se debe crear un índice de este tipo pues es el único que lo soporta. La búsqueda puede ser difusa, pero con sus limitantes:

- Los primeros tres caracteres deben coincidir (trigramas).
- No se hace diferencia entre mayúsculas y minúsculas.

Si se desea diferenciar a las letras mayúsculas de las minúsculas, se debe crear un índice por n-gramas especial pues al crearlo se define la opción CASE_ENABLED y al momento de realizar la consulta, se especifica la sentencia PRECISE FORM OF.
Cuando se emplea la opción CASE_ENABLED, el índice necesita más espacio y la búsquedas pueden tardar más tiempo.

4.2.5 Las consultas con SQL

- Booleanas

Se puede tener más de un término en el argumento de una búsqueda. En términos lógicos, los términos de búsqueda se conectan por el operador OR. Ejemplo:

```
SELECT fecha, tema
FROM DB2TX.muestra
WHERE DB2TX.CONTAINS (comentario, '("compilador", "disco", "zip", "compact")') = 1
```

Los términos de búsqueda se pueden combinar con otras cadenas de búsqueda usando los operadores booleanos "&" (AND) y "|" (OR). Por ejemplo:

```
SELECT fecha, tema
FROM DB2TX.muestra
WHERE DB2TX.CONTAINS (comentario, '"interprete" & "compilador"') = 1
```

Se pueden combinar varios términos usando los operadores booleanos:

```
SELECT fecha, tema
FROM DB2TX.muestra
WHERE DB2TX.CONTAINS (comentario,'"interprete" | "compilador" & "DB2"') = 1
```

Si se usa más de un operador booleano, Text Extender evalúa de izquierda a derecha, teniendo mayor prioridad el operador AND que el OR, por ejemplo:

"DB2" & "compilador" | "soporte" & "tecnico" se evalúa como:
("DB2" & " compilador ") | ("soporte" & "tecnico ")

El operador NOT se utiliza para excluir texto: ("interprete", "compilador") & NOT "DB2"

- Expresiones Regulares

En una búsqueda por Expresiones Regulares, *Text Extender* utiliza: (_) y (%)

- % representa cualquier número arbitrario de caracteres (*Cerradura de Kleene*):

```
SELECT fecha, tema
FROM DB2TX.muestra
WHERE DB2TX.CONTAINS (comentario,'"%cion"') = 1
```

- _ representa un caracter en un término de búsqueda. El siguiente ejemplo puede encontrar las palabras: "CLOB" y "BLOB".

```
SELECT fecha, tema
FROM DB2TX.muestra
WHERE DB2TX.CONTAINS (comentario, '"_LOB"') = 1
```

- Búsqueda de frases

En una búsqueda de frase, basta con colocar la frase tal cual se desea localizar, por ejemplo:

```
SELECT fecha, tema
FROM DB2TX.muestra
WHERE DB2TX.CONTAINS (comentario, '"sistema operativo"') = 1
```

En el siguiente ejemplo, se encuentra un argumento de búsqueda que encuentra términos en una misma sentencia:

```
SELECT fecha, tema
FROM DB2TX.muestra
WHERE DB2TX.CONTAINS (comentario,
' "sistemas" IN SAME SENTENCE AS "computacionales" ') = 1
```

La búsqueda de términos en el mismo párrafo, también es posible:

```
SELECT fecha, tema
FROM DB2TX.muestra
WHERE DB2TX.CONTAINS (comentario,
'"sistemas" IN SAME PARAGRAPH AS "computacionales" AND "distribuidos"') = 1
```

- Búsqueda de términos en secciones de documentos estructurados

A continuación se muestra un ejemplo de una búsqueda con argumento que permite encontrar el término *Sor Juana* en la subsección *Author* de la sección *Obra* de un texto estructurado:
La estructura del documento se especifica por el modelo *Obra* el cual se describe en un archivo del modelo del documento.

```
SELECT fecha, tema
FROM DB2TX.muestra
WHERE DB2TX.CONTAINS (comentario,
'MODEL obra SECTIONS (obra/autor) "Sor Juana"') = 1
```

- Búsqueda de términos similares

Para un índice dual, se pueden hacer búsquedas más flexibles para buscar no solo los términos que se especifican, sino palabras con un significado similar, por ejemplo la palabra "*libro*" puede emplearse para buscar también sus sinónimos:

```
SELECT fecha, tema
FROM DB2TX.muestra
WHERE DB2TX.CONTAINS (comentario, 'SYNONYM FORM OF "libro" ') = 1
```

Donde el uso de SYNONYM FORM OF, asume que los sinónimos del término, se conectan por medio del operador OR y los argumentos se pueden interpretar del siguiente modo:
"libro" | "artículo" | "volumen" | "manual". Los sinónimos se encuentran en el diccionario que proporciona *Text Extender*, por omisión emplea el US_ENGLISH pero puede cambiarse especificando un idioma diferente. El siguiente es un ejemplo donde se cambia al inglés británico:

```
SELECT DATE, SUBJECT
FROM DB2TX.SAMPLE
WHERE DB2TX.CONTAINS (COMMENTHANDLE,
'SYNONYM FORM OF UK_ENGLISH "programme"') = 1
```

- Búsqueda difusa *(fuzzy search)*

Una búsqueda difusa busca palabras que se deletrean de modo similar al término de búsqueda. Este tipo de búsqueda sólo es posible por índices de n-gramas. Por ejemplo:

```
SELECT DATE, SUBJECT
FROM DB2TX.SAMPLE
WHERE DB2TX.CONTAINS (COMMENTHANDLE,
'FUZZY FORM OF 2 "compress"') = 1
```

Esta búsqueda puede encontrar una ocurrencia de la palabra con falta ortográfica como: *conpress*.
El nivel de emparejamiento, para el ejemplo "2", especifica el grado de precisión.
Se soportan cinco niveles, donde el nivel 1 proporciona el nivel más amplio de holgura con un emparejamiento de un 20% aproximadamente; el nivel 5 es el más cerrado, buscando un emparejamiento de 90%. Este tipo de búsquedas son muy empleadas cuando el documento o texto se creó usando un dispositivo OCR o una entrada fonética como los reconocedores de voz.

- Búsqueda de palabras por sonidos similares

Las búsquedas por sonidos encuentran palabras cuya fonética es similar a la del argumento. Esta se usa cuando los documentos pueden contener palabras que se asemejan pero son diferentes en su ortografía. El nombre alemán que se pronuncia como *my-er*, por ejemplo, tiene muchas maneras ortográficas:

```
SELECT fecha, tema
FROM DB2TX.muestra
WHERE DB2TX.CONTAINS (comentario,'SOUNDS LIKE "Meyer"') = 1
```

Esta búsqueda puede encontrar las ocurrencias: *"Meyer"*, *"Mayer"*, y *"Maier"*.

- Búsqueda por Thesaurus

El thesaurus es una poderosa expansión de las funciones de búsqueda que proporciona *Text Extender*. Los términos adicionales buscados se toman de un thesaurus proporcionado por el usuario y sobre el cual, él mismo tiene el control absoluto.
Si se desea buscar por ejemplo, la frase "*base de datos*", se pueden encontrar también los términos: "*repositorio*", "*database*", y "*DB*".

El siguiente ejemplo, toma el término "*object relational database management system*" y se expande agregando todas las instancias de este término encontradas en el thesaurus "*myterms*".

```
SELECT fecha, tema
FROM DB2TX.muestra
WHERE DB2TX.CONTAINS (comentario,
' THESAURUS "myterms"
  EXPAND "INST" TERM OF "object relational database management system" ') = 1
```

El siguiente ejemplo, toma el término "*sistema administrador de documentos*" y se expande a todos sus sinónimos:

```
SELECT fecha, tema
FROM DB2TX.muestra
WHERE DB2TX.CONTAINS (comentario,
' THESAURUS "myterms"
  EXPAND "SYN"
  TERM OF "sistema administrador de documentos" ') = 1
```

- Búsqueda de "texto Libre"

Es una búsqueda en la cual el término buscado se expresa en una forma libre. Una frase o sentencia describe en lenguaje natural el tema buscado. La secuencia de palabras en este tipo de consulta no es relevante. Además, las llamadas afinidades léxicas se soportan; son ciertos pares de palabras que ocurren en el texto del documento con una cierta frecuencia mínima y una cierta distancia mínima. La distancia para documentos en inglés es por ejemplo, de cinco palabras.

```
SELECT fecha, tema
FROM DB2TX.muestra
WHERE DB2TX.CONTAINS (comentario,
'IS ABOUT "todo lo referente a la instalacion en UNIX"') = 1
```

Las **búsquedas híbridas** son una combinación de búsquedas booleanas y texto libre:

```
SELECT fecha, tema
FROM DB2TX.muestra
WHERE DB2TX.CONTAINS (comentario,
'"DB2" & IS ABOUT "todo lo referente a la instalacion en UNIX"') = 1
```

Otro ejemplo, donde se utilizan varios conceptos:

```
SELECT * FROM MyTextTable
WHERE version = ©2©
AND DB2TX.CONTAINS (
DB2BOOKS_HANDLE, "authorization"
IN SAME PARAGRAPH AS "table" AND SYNONYM FORM OF "delete") = 1
```

- Búsqueda por límites de frase

Estos tipos de búsquedas se han desarrollado para el koreano, permite que se respeten los límites de las palabras durante su búsqueda.

```
SELECT fecha, tema
FROM DB2TX.muestra
WHERE DB2TX.CONTAINS (comentario,
'BOUND "expresion-koreana"') = 1
```

4.2.6 El proceso de recuperación

La consulta permite la búsqueda de términos relevantes para incrementar el número de documentos relevantes recuperados. *Text Extender* lo hace por medio de dos operaciones básicas sobre los términos consultados: Expansión y Reducción.

- **La expansión** toma una palabra o frase buscada y la asocia con un conjunto de términos alternativos. Así, ambas expresiones forman una expresión booleana OR en el lenguaje de consulta. Las operaciones de expansión son así: La *sinonimia* y el uso del *thesaurus*.
  Los *sinónimos* se obtienen de diferentes archivos dependiendo del idioma y se buscan a partir de su forma base.
  Text Extender permite expandir un término agregando términos de un *thesaurus* proporcionado por el usuario como un archivo externo compilado. Los componentes básicos de un thesaurus son: "*términos*" y "*relaciones*".
  Los términos son palabras que denotan un concepto dentro del dominio del *thesaurus*. Se clasifican a su vez en *descriptores* (aquellos empleados para la búsqueda y la indexación) y los *no-descriptores*. Un *thesaurus* por ***N-gramas***, no distingue a estos tipos de términos.
  Las relaciones son expresiones de una asociación entre dos términos, no existen relaciones predefinidas, deben ser definidas y nombradas por el usuario. Sus propiedades son la *profundidad* que es el número de niveles sobre los que la relación se extiende y la direccionalidad que especifica si la relación es cierta en modo unidireccional o bidireccional.
  Existen varios *tipos de relaciones* proporcionadas por el *thesaurus* de *Text Extender*:

  - Associativas: Es una relación bidireccional entre descriptores extendiéndose a profundidad. La relación predefinida RELATED_TO se basa en este tipo; por ejemplo: *tenis* **RELATED_TO** *raqueta.*
  - Sinónimas: Text Extender hace diferencia entre los descriptores y los no-descriptores, por ello, la relación de sinonimia es bidireccional entre términos sinónimos y uno de ellos se determina como el descriptor. La relación predefinida **SYNONYM_OF** se basa en este tipo; por ejemplo: *USA* **SYNONYM_OF** *EUA.*
  - Jerárquicas: Es una relación unidireccional entre descriptores, donde uno es más general y el otro más particular permitiendo así, la existencia de una jerarquía.
  - Otras: Es la relación más general y difícil de cumplir pues puede ser bidireccional o unidireccional, no existen restricciones de profundidad y existen relaciones entre descriptores y no-descriptores. Se recomienda emplearse para nuevos términos en el *thesaurus* hasta que se pueda determinar una relación definitiva.

- **La reducción** cambia el término de búsqueda a una forma más general que la proporcionada por el usuario. La reducción depende del tipo de índice empleado para asegurar el emparejamiento de los términos con el cambio. Las reducciones empleadas son: Lematización, Normalización y Lista de parada.

- **Algunas operaciones cambian y expanden los términos.** La expansión por sonido (se expande un término a un conjunto de palabras con un sonido similar) y la máscara (expresiones regulares) sobre caracteres y palabras son las operaciones que cumplen con estos criterios. Por ejemplo si se busca ***pas%*** no se pueden encontrar las palabras ***pasión*** y ***paseo*** debido a que tienen distinto término raíz, sólo buscando por medio de la expresión regular, se tendrá una recuperación satisfactoria.

4.2.7 Carga de datos textuales

Agregar o actualizar los registros con texto en la base de datos, se puede hacer a través de las utilerías de administración llamadas: *Commerce Suite Administrator* y la utilería *Mass Import*. Ésta última herramienta permite importar datos en formato ASCII o XML. El archivo de importación contiene la información necesaria para llenar las tablas. En el diagrama de la figura 4.2, se aprecia el uso del *Mass Import* usando archivos XML. En la figura 4.3 se tiene para archivos ASCII.

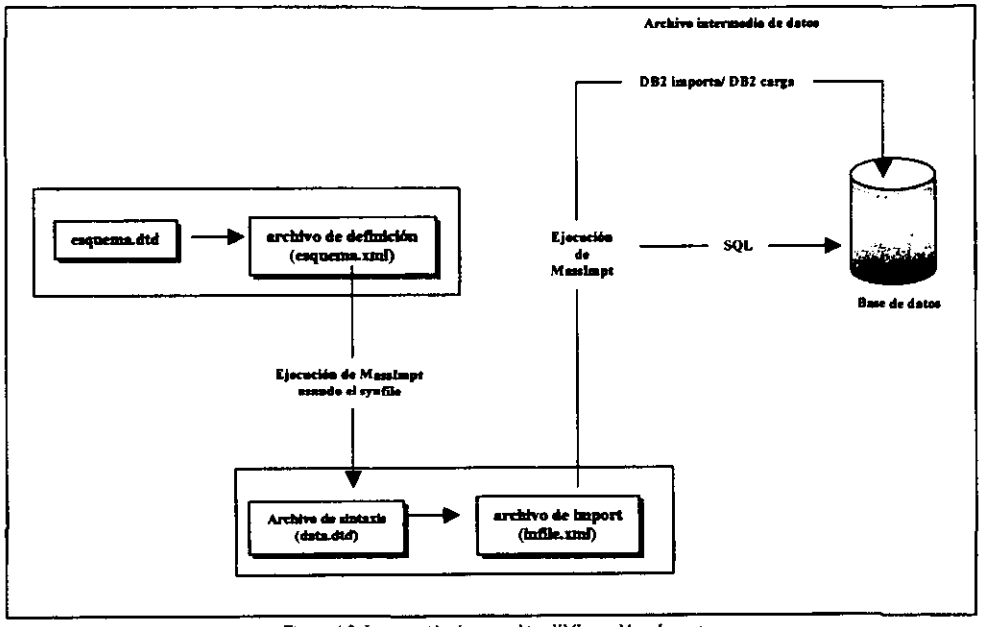


Figura 4.2 Importación de un archivo XML con Mass Import

El archivo esquema.dtd describe la estructura y las reglas del archivo de definición. No debe ser modificado.
El archivo esquema.xml describe el esquema por omisión. Si se requiere, se debe modificar para incluir extensiones.
El archivo data.dtd describe la sintaxis del archivo de importación (Import)
El archivo infile.xml es un archivo simple que importa datos a tablas.

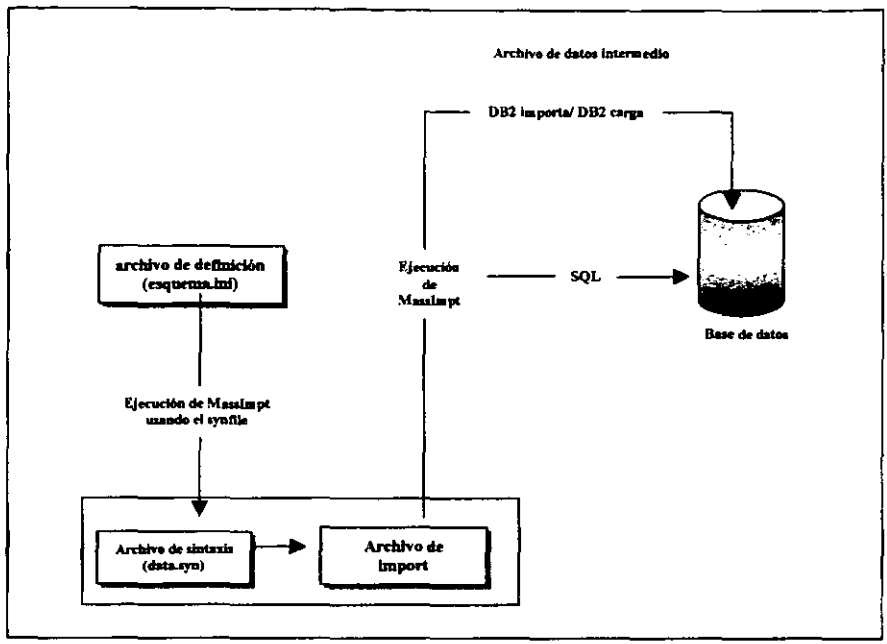


Figura 4.3 Importación de un archivo ASCII con Mass Import

El archivo esquema.ini describe el esquema por omisión. Se puede crear el propio para representar el esquema de una base de datos propia, incluyendo cualquier extensión.
El archivo data.syn describe la sintaxis del archivo de importación (import)
El archivo infile.txtl es un archivo que importa datos a tablas.

4.2.8 Los Clientes

IBM con su *DB2*, cuenta con varios clientes en el ámbito internacional, algunos de ellos son:

- La biblioteca vaticana digitalizó 150,000 manuscritos y 1.5 millones de libros incluyendo 8,000 publicaciones que se realizaron durante los primeros 50 años de impresión.

- La biblioteca del Congreso de los Estados Unidos de Norteamérica que cuenta con la historia de la fundación de aquél país.

- El archivo general de las Indias cuenta con el depósito más importante de documentos sobre el descubrimiento, exploración y administración de España sobre el Nuevo Mundo, digitalizó sus archivos de 90 millones de páginas de material histórico.

- *Yomiuri Shimbun* (Japón) el periódico más grande del mundo, se apoya en el DB2 para apoyar y organizar sus funciones de redacción, escritura y publicación.

- La Nueva Red de Negocios (NDN) dedicada a los proyectos de arte y publicación del nuevo periodo de negocio. La base de datos trabaja sobre 3,000 artículos incluyendo fotografías, caricaturas políticas, discursos y rotulación.

4.3 Las propuestas de INFORMIX

Con *Informix* se tiene algo similar que con *IBM*, pero la diferencia es que *Informix* hizo dos convenios con dos corporativos independientes e hicieron una fusión que dió origen a dos productos diferentes que se ofrecen sobre el ***ORDBMS*** de *Informix* llamado *Informix Dynamic Server (IDS)*, los nombres de tales productos son: *Excalibur Text Search DataBlade* y *Verity Text Search DataBlade*.

4.3.1 Excalibur Text Search DataBlade

Generalidades

El nombre comercial de este módulo es: *Excalibur Text Search DataBlade Module* de aquí en adelante, llamado simplemente *Excalibur*. El módulo de búsqueda de texto, llamado *Excalibur* es la propuesta de *Informix Corporation* para permitir las búsquedas de información sobre texto de manera más rápida y sofisticada que las permitidas por el SQL estándar. Es una colección de tipos de datos y rutinas que extienden al **ORDBMS** *Informix Dynamic Server*.

Usando el *IDS*, se pueden resolver algunas de las debilidades de las bases de datos tradicionales; con el *Excalibur* se puede proporcionar una búsqueda flexible de texto directamente sobre la base de datos en aquellos campos de tipo caracter de longitudes hasta de 2 Kb (de un documento ASCII). En la figura 4.4 se aprecia este proceso: A partir de una consulta, ésta se procesa por el *IDS* y la parte donde se requiere de búsqueda de texto se procesa por la máquina de búsqueda (en este caso, el *Excalibur*), generando así una búsqueda sobre el índice que recuperará la lista de éxitos (*hitlist*) o documentos que satisfacen exitosamente la consulta planteada. Una vez recuperados los documentos, se regresan al *IDS* que los incorpora a los datos estructurados y se presentan al usuario en su interfaz utilizada. Las principales características de *Excalibur* son las siguientes:

- Indexación de texto completo, incluyendo un extenso soporte para búsquedas de lógica difusa, especialmente importante cuando se efectúan búsquedas sobre texto.
- Métodos de indexación para resultados más fáciles, con un reconocimiento completo de los índices por parte del optimizador del IDS.
- Soporta múltiples listas de parada, búsquedas por proximidad y listas de sinónimos.
- Indexación parcial en memoria caché complementando el buen funcionamiento del IDS.
- Búsquedas de texto en el contexto de sentencias regulares de SQL; lo que significa que una instrucción DML o DDL debe ser permitida, extendiéndose esta funcionalidad sobre los tipos de datos de texto. Además la indexación será automática y las transacciones cumplirán la integridad y sólo tendrán acceso los usuarios con permiso asignados.
- Para sitios Web, las búsquedas de texto pueden efectuarse a través el sitio entero en una consulta, sin importar si las páginas se almacenan fuera o dentro de la base de datos. Para sitios complejos, esta característica es esencial en la navegación por el sitio o en su administración.

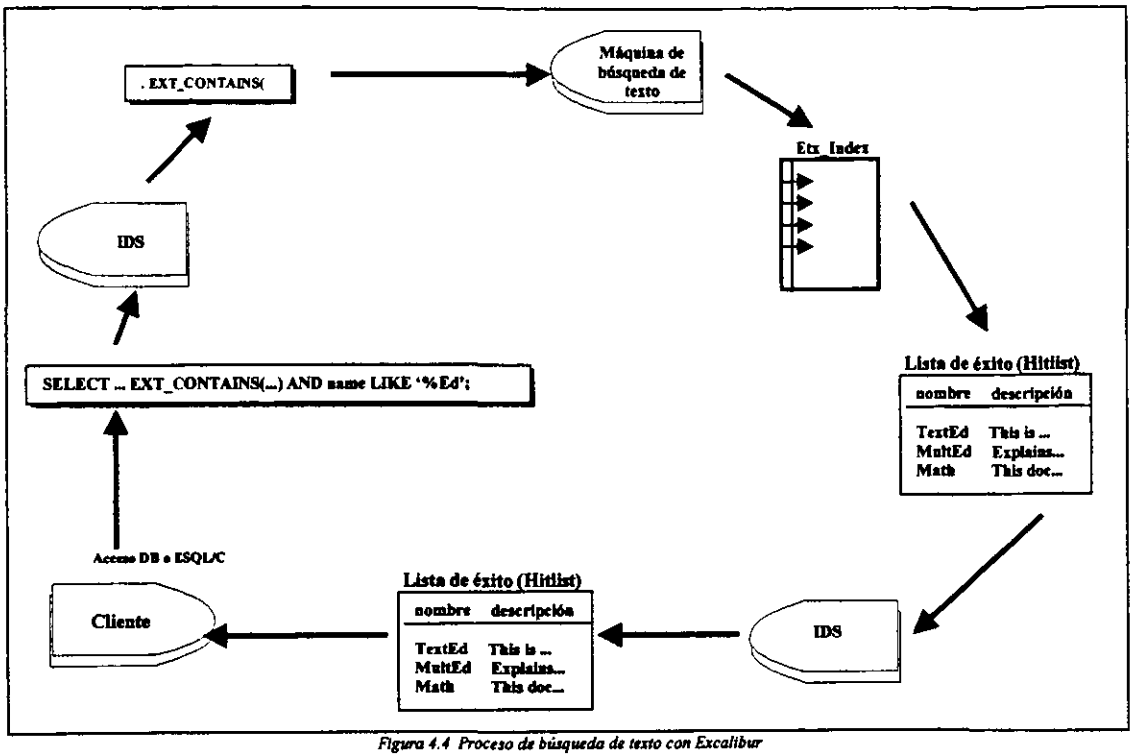


Figura 4.4 Proceso de búsqueda de texto con Excalibur

4.3.1.2 Formatos y códigos

Excalibur da soporte para documentos en formato:

ASCII *Word* *Excel* *PowerPoint* *HTML*

PDF *Word Perfect.*

También da soporte a lenguajes internacionales y conjuntos de caracteres definidos por el usuario para definirse con 8-bits o con un byte como parte de una palabra. Permite el uso de los conjuntos de caracteres: *ASCII, ISO y OVERLAP_ISO.* Algo importante, es que el usuario puede crear su propio conjunto de caracteres y después emplearlo en el proceso de indexación:

1.-
```
EXECUTE PROCEDURE etx_CreateCharSet
('Mi_Codigo', '/local3/excal/mi_archivo_codigo');
```

2.-
```
CREATE INDEX desc_idx ON libros (resumen etx_clob_ops)
USING etx (WORD_SUPPORT= 'PATTERN',
CHAR_SET = 'Mi_Codigo') IN space1;
```

Técnicas empleadas

Aunque este es un tema confidencial, propio de cada compañía de software, algunas veces se publica cuales son las técnicas que emplean los productos comerciales, *Informix* ha reconocido que: Las búsquedas estándar de SQL que incluyen las cláusulas LIKE y la comparación contra patrones fijos, usa métodos de acceso sobre *árboles-B*.

4.3.1.3 Componentes de Excalibur

Este módulo de búsqueda, consta de tres componentes principales:

(1) El método de acceso **etx**
(2) El operador **etx_contains()**
(3) Las rutinas de soporte definidas por el módulo

(1) Este método permite la **creación de índices** que soportan las búsquedas sofisticadas sobre columnas que contienen texto de una tabla. A los índices creados por el uso de este método, se llaman ***índices etx***. Para aprovechar las ventajas de estos índices, conviene almacenar el texto sobre una columna de tipo que soporta el *Informix Dynamic Server*: BLOB, CLOB, LVARCHAR, CHAR, VARCHAR, IfxDocDesc o IfxMRData.
El tipo de dato **IfxDocDesc** permite habilitar tipos de datos para almacenar documentos ya sea en la base de datos o como un archivo del sistema operativo.
El tipo de dato **IfxMRData** es un tipo de dato dinámico, es decir, el tipo de dato decide por si mismo, dependiendo del tamaño del documento, si lo almacena como un tipo LVARCHAR o como un objeto largo.

Cuando se crea un *índice etx*, se debe especificar el operador de clase dependiendo del tipo de la columna que será indexada. Este operador de clase es un conjunto de funciones que el ORDBMS asocia con el *método de acceso etx* para optimizar las consultas y construir índices.
En la siguiente tabla, se muestra cada tipo de dato con su operador de clase correspondiente.

Tipo de dato	*Operador de Clase*
BLOB	etx_blob_ops
CLOB	etx_clob_ops
LVARCHAR	etx_lvarc_ops
CHAR	etx_char_ops
VARCHAR	etx_varc_ops
IfxDocDesc	etx_doc_ops
IfxMRData	etx_mrd_ops

Tabla 4.2 Operadores de clase

Así, para crear un índice se tiene la siguiente secuencia de pasos a seguir:

1) Construir la tabla con columnas para almacenar texto.

```
CREATE TABLE peliculas
( clave     INTEGER,
  nombre VARCHAR(30),
  sinopsis CLOB);
```

2) Crear el índice sobre una la columna que almacena texto.

```
CREATE INDEX sinopsis_indx
ON peliculas (sinopsis etx_clob_ops)
USING etx in sub_space1;
```

Se construye así, un índice sobre la columna sinopsis y se utiliza un espacio lógico de almacenamiento llamado: sub_space1, el cual, se recomienda que sea distinto al espacio lógico donde se almacenan las tablas para efectos de baja contención.

La restricción que existe en la creación del índice es que no se pueden utilizar las opciones: UNIQUE, ASC, DISTINCT, DESC.

- La creación de un índice sobre caracteres internacionales y ASCII es posible, para ello, se debe especificar el parámetro CHAR_SET

```
CREATE INDEX indice_internacional
ON peliculas (sinopsis etx_clob_ops)
USING etx (CHAR_SET='ISO' ) in sub_space2;
```

4.3.1.4 Proceso de Filtrado

Este proceso es necesario para evitar la indexación de datos binarios, se filtran los documentos antes de ser indexados. Consiste en limpiar de todas las características de formateado que pudiera tener un documento y depurar sólo el texto en formato ASCII.
Así, un documento en MSWord que puede contener formatos, información de tipos de letras, colores, estilo de párrafos, etc. puede indexarse, pero como una consulta no emplea esas características para su planteamiento, no es conveniente emplearse en la indexación.
Para crear un índice sobre texto filtrado, se debe especificar:

```
CREATE INDEX indice_filtrado
ON peliculas (sinopsis etx_clob_ops)
USING etx (FILTER= 'STOP_ON_ERROR');
```

(2) El operador **etx_contains( )** se usa para **efectuar búsquedas** sobre los *índices etx*. Permite definir, calificar y refinar búsquedas sobre texto. Se deben definir tres argumentos siendo el último de ellos, opcional. El primero define la columna y el segundo sirve para especificar la búsqueda. El tercer argumento es una variable local (*SLV, statement local variable*) que sirve para recuperar el valor de la puntuación (*score*) de la búsqueda.
Se tienen a continuación, algunos ejemplos:

```
SELECT   clave, resumen
FROM     libros
WHERE    ext_contains (resumen, 'biografia');
```

4.3.1.5 Consultas

- Consultas booleanas

Estas consultas se tienen cuando hay un planteamiento con expresiones booleanas. Por ejemplo, se utilizan los símbolos: & para el operador AND, | para el OR y el ! ó ^ para el operador lógico NOT. Una consulta sería:

```
SELECT   clave, resumen
FROM     libros
WHERE    ext_contains (resumen,
Row ('compiladores & interpretes & !JAVA',
'SEARCH_TYPE = BOOLEAN_SEARCH' ));
```

Usando el tercer parámetro, para la puntuación:
Este parámetro tiene un valor entre 0 y 100:
0 indica que no hay semejanza alguna
100 es la puntuación para una búsqueda exacta.

```
SELECT   rc.score, clave, resumen
FROM     libros
WHERE    ext_contains (resumen,
Row ('Sistemas Operativos' ' Análisis y Diseño & Sistemas'),
rc # etx_ReturnType) AND rc.score > 85;
```

- Consultas por frases exactas

```
SELECT   *
FROM     libros
WHERE    ext_contains (resumen,
Row ('prologo editor', ' SEARCH_TYPE=PHRASE_EXACT'));
```

- Consultas por aproximación de frases

```
SELECT   *
FROM     libros
WHERE    ext_contains (resumen,
Row ('prologo editor', ' SEARCH_TYPE=PHRASE_APPROX'));
```

- Búsquedas por proximidad

El siguiente ejemplo, hace una búsqueda que permite recuperar documentos que contengan una de las palabras de la frase definida. Así, por ejemplo, se recuperarán documentos con la palabra *de* o *editor* o *prologo*.

```
SELECT   *
FROM     libros
WHERE    ext_contains (resumen,
Row ('prologo del editor', ' SEARCH_TYPE=WORD'));
```

Otra posibilidad, es solicitar proximidad de palabras con una distancia máxima entre ellas. Esta distancia se determina por un número entero que el número de caracteres entre las palabras. Como en el ejemplo siguiente donde las palabras "*encuentro*", "*cercano*" y "*tipo*"no se encuentren separadas por más de ocho palabras.

```
SELECT   clave, resumen
FROM     libros
WHERE    ext_contains (resumen,
Row ('encuentro cercano tipo',
'SEARCH_TYPE = PROX_SEARCH(8)' ));
```

- Búsquedas Difusas

Dos problemas se resuelven por el *Excalibur*: ***Errores ortográficos*** y ***errores por transposición*** de caracteres adjuntos. Un ejemplo de cada uno se tiene respectivamente: *editor* como *editpr* y para el segundo caso, *ensamblador* y *ensabmldor*.

Búsquedas por patrones pueden ser difusas si se consideran transposiciones, sustituciones, subcadenas o supercadenas. Se emplea para esto, el parámetro: PATTERN_TRANS, PATTERS_SUBS o el PATTERN_ALL. En el segundo ejemplo, se determina el número máximo de recuperaciones.

1)
```
SELECT   clave, resumen
FROM     libros
WHERE    ext_contains (resumen,
Row ('filosofía',
'PATTERN_ALL' ));
```

2)
```
SELECT   clave, resumen
FROM     libros
WHERE    ext_contains (resumen,
Row ('filosofía del siglo XVII',
'PATTERN_TRANS & PATTERN_SUBS' & MAX_MATCHES=3' ));
```

(3) Existen diversas rutinas de soporte que permiten crear o borrar sinónimos y listas de parada. Un ejemplo de ellas es la rutina **etx_CreateSynWlst( )** para la lista de términos empleados como sinónimos. Para poder usar este procedimiento, se debe crear primero la lista.

- Lista de sinónimos

```
EXECUTE PROCEDURE etx_CreateSynWlst
('list_sinon', '/local2/excal/sinonimos','space1');
```

En el ejemplo anterior, se creó la lista de sinónimos llamada *list_sinon* en un archivo del sistema operativo llamado */local2/excal/sinonimos* en un espacio llamado *space1*.

La lista de sinónimos debe contener una palabra raíz con uno o más sinónimos, todos en una línea separados por caracteres en blanco y una línea en blanco entre cada nueva definición, como a continuación se observa en un breve ejemplo:

```
rápido veloz apresurado aprisa

monitor terminal CRT pantalla
```

Es importante resaltar que la búsqueda de los sinónimos es a partir de la raíz, mas no, de modo inverso.

Finalmente, la búsqueda que emplea esta lista de sinónimos tiene la siguiente sintaxis:

```
SELECT clave, resumen
FROM   libros
WHERE  ext_contains (resumen,
Row ('documento', 'MATCH_SYNONYM= list_sinon'));
```

- Uso del thesaurus

Debido a que pueden existir diversas listas de sinónimos, se indica el parámetro MATCH_SYNONYM para indicar el uso de la lista deseada. Si no se especifica, El operador etx_contains( ) hace referencia a la dista por omisión llamada **etx_thesaurus**. Esta lista puede ser creada por el usuario. Se proporciona una basada en el idioma inglés llamada etx_thesaurus.txt

- Lista de parada

Para la *lista de parada*, ocurre algo similar:
Primero, la creación de la lista llamada *ListadeParada*

```
EXECUTE PROCEDURE etx_CreateStopWlst
('ListadeParada', '/local3/excal/lista_parada');
```

La búsqueda sería de la siguiente manera, si se desea buscar la frase "*To be or not to be*" y se quieren incluir aún los términos que se tienen en la lista de parada:

```
SELECT clave, resumen
FROM   libros
WHERE  ext_contains (resumen,
Row ('to be or not to be',
'SEARCH_TYPE=PHRASE_EXACT & CONSIDER_STOPWORDS'));
```

4.3.1.6 Almacenando datos tipo texto

Puede hacerse con el comando SQL INSERT, en el ejemplo siguiente se define la ubicación física del archivo de texto que debe ser referenciado:

```
INSERT INTO evaluaciones (numero, autor, titulo, resumen)
VALUES ( 350, 'Chris J. Date', 'Introduction to Database Systems',
FileToCLOB( '/local6/excal/dbms.txt', 'cliente') );
```

4.3.2 Verity Text Search DataBlade Module

Verity Text Search DataBlade Module, es una interfaz entre la máquina de búsqueda de *Verity* y la base de datos de *Informix*. Proporciona búsquedas sobre texto junto al buscador de la base de datos. La base de datos puede servir como un punto de integración para administrar texto de gran tamaño, audio, video y datos simples ofreciendo una flexibilidad poco común en las bases de datos de texto. Como uno de los líderes en búsquedas avanzadas de texto y recuperación, *Verity* proporciona capacidades de búsquedas avanzadas no disponibles en otros productos.
Incluye agrupación de documentos y resúmenes, QBE y búsquedas dinámicas sobre información no-indexada, ingeniería de proceso adaptada al reconocimiento de patrones (*APRP*, *Adaptive Pattern Recognition Process engine*) para maximizar la precisión y relevancia.
En suma, *Verity* proporciona conocidas herramientas que permiten la creación de diccionarios personalizados y soportan *thesaurus* para las necesidades específicas de las organizaciones.

La combinación entre el *Informix Web DataBlade Module* y el *Verity Text DataBlade Module* proporcionan una rica herramienta de búsqueda, ideal para los complejos sitios Web y administración de documentación de sistemas en línea. De aquí en adelante, al *Verity Text DataBlade Module* se le llamará simplemente *Verity*.

QBE permite que documentos en lenguaje natural sean usados en el proceso de consulta y permite que los usuarios puedan afinar sus consultas usando ejemplos de documentos relevantes.
Los resúmenes automáticos de documentos usan las sentencias más significativas de un documento. Esto permite al usuario determinar la relevancia de un documento mucho más rápido que lo que proporcionan los productos tradicionales, los cuales despliegan solo el título o las primeras líneas.

La agrupación automática organiza documentos recuperados por una búsqueda en grupos de documentos (*clusters*), basados en temas comunes. Esto permite al usuario identificar rápidamente grupos relevantes de documentos sin tener que examinarlos de manera individual.

Permite la búsqueda completa de texto en la base de datos, especialmente, usando aplicaciones que necesitan búsquedas borrosas que permiten a los usuarios encontrar lo que de otra manera no sería anunciado en los buscadores de texto estándar:

- Indexa texto, incluyendo soporte extensivo para búsquedas de lógica borrosa.
- Aplicaciones que requieren tanto recuperación de texto o consultas *ad hoc*.

- Múltiples listas de palabras, búsquedas de proximidad, listas de sinónimos.
- Para los sitios Web, los clientes pueden realizar una ingeniería avanzada que optimice las búsquedas y la indexación de texto usando búsquedas aproximadas.

4.3.2.1 Formatos, idiomas y códigos

Verity indexa y recupera documentos en la mayoría de los formatos conocidos, incluyendo entre ellos a:

HTML	*SGML*	*Adobe Acrobat PDF*	*Aplix Words 4.2*
Word	*Excel*	*PowerPoint*	*Corel Word Perfect para Mac*
WordPerfect	*AMI Pro*	*Lotus 1-2-3*	*Lotus AMI Pro*
ASCII	*XML*	*Corel WordPerfect*	*Lotus Word Pro*

Soporta caracteres ISO-8859-1, que es el estándar de 8 bits. Se pueden trabajar los siguientes idiomas:

Holandés	Inglés	Francés	Alemán
Español	Sueco	Italiano	

Verity permite que documentos escritos en cualquiera de los lenguajes soportados puedan estar en una sola tabla. Se puede crear una columna para cada lenguaje e insertarlos en su columna respectiva.

4.3.2.2 Componentes de Verity

En la figura 4.5 se pueden apreciar los componentes que forman al *Verity Text Search DataBlade*, el **contexto** determina las características de un **índice vts** cuando éste se crea y el comportamiento de las búsquedas que lo utilizan. Un contexto usa **archivos de estilos** para búsquedas por parámetros. Cuando se asocia un contexto con un renglón o tipo de dato distinto, el contexto se asocia con todos los índices construidos para las columnas de ese tipo de dato. Así si se desea construir un nuevo tipo de dato, se debe construir también su contexto y sus archivos de estilos (sustituirlos por los que proporciona el módulo por omisión).

Algunos archivos de estilos no deben ser modificados, como por ejemplo el **style.ngm** que define el índice de N-grama que habilita búsquedas con comodines y difusas. Otros archivos sí pueden ser modificados: **style.dft** que indica que se debe usar el filtro universal que soporta HTML, PDF y archivos Microsoft office y el **style.plc** que especifica el modo de indexación.

Los siguientes archivos de estilos, deben ser creados por el usuario:

style.prm	Permite características tales como: Subrayado, resúmenes, agrupación.
style.go	Permite indexar vocabularios especiales; se especifica de modo similar a *LEX*.
style.lex	Sirve para especificar caracteres no-alfanuméricos que serán indexados.
style.stp	Para especificar la lista de parada.

Para mayores detalles, se recomienda revisar la guía del usuario de *Verity* [VERI00]

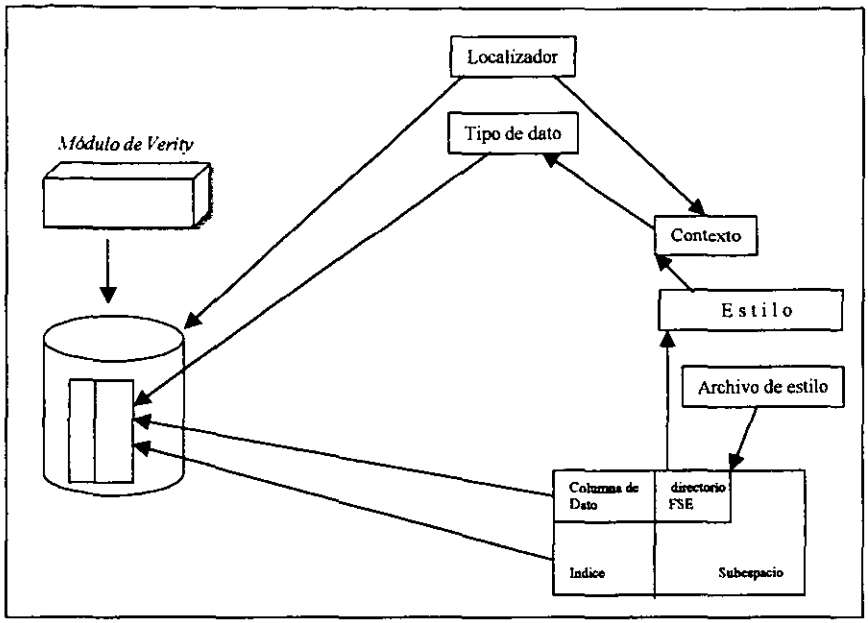


Figura 4.5 Componentes de Verity Text Search DataBlade Module

4.3.2.3 Creación de elementos de la base de datos

Con *Informix* se pueden crear distintos tipos de datos con la capacidad de heredar todas las propiedades de los tipos de datos básicos del manejador de bases de datos. Por ejemplo, con el siguiente DDL:

```
CREATE DISTINCT TYPE tipo_C AS CLOB;
```

Se puede crear un tipo renglón:

```
CREATE ROW TYPE pelicula
( director  varchar(30),
  titulo    varchar(60),
  sinopsis  tipo_C);
```

Para crear un **contexto** dentro de la base de datos, se debe ejecutar un procedimiento que usa la rutina **Vts_CreateContext( )**.

```
EXECUTE PROCEDURE Vts_CreateContext
( 'ContextoPelicula', '/infx/usuario/fse_dir/style01');
```

Para este ejemplo, el nombre del contexto es: ContextoPelicula
La información se buscará en el directorio: /informix/usuario/fse_dir/style01

Así, una vez creado el contexto, éste se asocia al tipo de dato con un procedimiento similar a este:

```
EXECUTE PROCEDURE Vts_AssociaContext
'ContextoPelicula', 'tipo_C');
```

Con esta instrucción se asegura que cuando se construya un índice sobre una columna de tipo 'tipo_C', su comportamiento será controlado por las características del contexto 'ContextoPelicula'. Los tipos de datos básicos para almacenamiento de texto, se controlan por el contexto definido por omisión. Siguiendo con una secuencia de pasos, se construirá una tabla que contengan columnas de los tipos predefinidos:

```
CREATE TABLE Peliculas_2000
( clave          INTEGER PRIMARY KEY,
  detalles       pelicula
  comentario     tipo_C )
  PUT item IN (space01), comentario IN (space02);
```

Para este ejemplo, los valores de la columna 'detalles' se almacenarán en el subespacio 'space01' y la columna 'comentario' en el subespacio 'space02'. Lo cual es importante de resaltar desde el punto de vista del administrador, que reparte los datos para un óptimo acceso.

- El proceso de indexación

La creación de **índices** sobre las columnas que serán consultadas, puede ser de dos tipos: **índices fragmentados** (permiten obtener una mejor administración y tiempo de respuesta) e **índices no-fragmentados**. Dependiendo del tipo de dato de la columna a indexar, se debe asociar la clase de operador adecuado:

Índice no-fragmentado sobre columna CLOB

```
CREATE INDEX peliculas_index01
ON peliculas_2000 ( comentario Vts_clob_ops)
USING vts
IN sub_space05;
```

Índice no-fragmentado sobre una columna de tipo ROW

```
CREATE INDEX peliculas_index02
ON peliculas_2000 ( detalles Vts_row_ops)
USING vts
IN sub_space06;
```

Índice fragmentado

```
CREATE INDEX peliculas_index03
ON peliculas_2000 ( comentario Vts_clob_ops)
USING vts
FRAGMENT BY EXPRESSION
clave  < 1000 IN subspace01,
clave  >= 1000 IN subspace02;
```

Para este ejemplo, dependiendo de los valores en la columna clave (que es llave primaria), los valores del dominio 'comentario' se almacenarán en diferentes subespacios, lo que al igual que en el caso de la tabla, permite al administrador repartir cargas de datos para un óptimo acceso y tiempo de respuesta.

El límite de indexación es que no se puede construir un índice multi-columna; sólo sobre columnas de tipo 'row'. Esto es importante considerar si se tienen llaves primarias compuestas.

• El proceso de filtrado

El filtro usado por el *Verity* reconoce una serie de formatos (previamente definidos) y automáticamente los convierte a formato ASCII cuando se indexan los documentos. A continuación, se muestra el proceso para almacenar, indexar y filtrar un documento en *Power Point*:

(1) Creación de la tabla

```
CREATE TABLE escritos
(autor  VARCHAR(50), titulo  VARCHAR(50), texto  BLOB);
```

(2) Creación del índice sobre la columna

```
CREATE INDEX escrito_ind ON escritos (texto Vts_blob_ops)
USING vts IN sbspace04;
```

(3) Habilitar el proceso del filtro (por omisión, está deshabilitada esta función)

```
BEGIN WORK;
        EXECUTE PROCEDURE VtsUTPSet("FilterEnable", "1");
COMMIT WORK;
```

(4) Almacenar el archivo convertido a formato ASCII

```
INSERT INTO escrito VALUES ('Juan Prawda',
'Investigacion de Operaciones',
FileToBLOB (' /user/presenta/ppt/red31.ppt', 'server'));
```

4.3.2.5 Tipos de Consultas

Continuando con el esquema de la tabla Peliculas_2000, se presentan los posibles planteamientos de consultas.

- Consultas simples y booleanas

La siguiente consulta permite recuperar los títulos de las películas donde en su comentario contengan el término *excelente*.

```
SELECT detalles.titulo FROM peliculas_2000
WHERE Vts_contains ( comentario, 'excelente');
```

La siguiente consulta permite recuperar los títulos de las películas del director *Steven Spielberg* que tengan el término *recomendado* en alguno de los subdominios del tipo renglón de la columna: *detalles*. El **índice vts** sólo se ocupa por la función **Vts_contains( )**, más no por la condición del nombre del director, debido a que no pertenece a la función.

```
SELECT detalles.titulo FROM peliculas_2000
WHERE Vts_contains (detalles, 'recomendado')
AND detalles.director = 'Steven Spielberg';
```

Si sólo se requiere que un subdominio del tipo renglón sea comparado, se puede hacer de las siguientes dos maneras:

```
SELECT detalles.titulo FROM peliculas_2000
WHERE Vts_contains (detalles.sinopsis, 'recomendada');

SELECT detalles.titulo FROM peliculas_2000
WHERE Vts_contains (detalles, '(recomendado) <IN> sinopsis');
```

Las siguientes consultas aplican el operador OR con distinta sintaxis, la primera buscará variaciones de raiz de la palabra *selección* o *selección nacional*. La segunda consulta recupera todos los documentos que contengan la palabra *integridad* o *seguridad*.

```
SELECT titulo FROM libros
WHERE Vts_contains (texto, 'selección OR selección nacional');

SELECT titulo FROM libros
WHERE Vts_contains (texto, 'OR ("integridad", "seguridad")');
```

La siguiente consulta permite recuperar los títulos de las películas **ordenando por puntuación** al aplicar el tercer parámetro sobre **vts_contains( )**.

```
SELECT detalles.titulo, ret.score
FROM peliculas_2000
WHERE Vts_contains (comentario, 'excelente',
ret # IfxVtsReturnType)
ORDER BY 2 DESC;
```

Donde **IfxVtsReturnType** es de tipo renglón. Su estructura se muestra en la figura 4.6
Aprovechando este tipo de dato, es posible agrupar y obtener resúmenes de los documentos, como se verá a continuación.

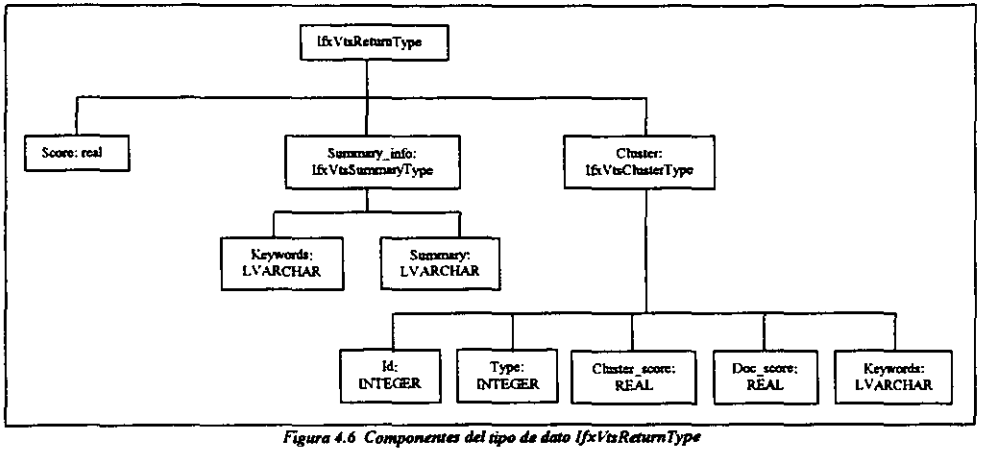


Figura 4.6 Componentes del tipo de dato IfxVtsReturnType

- Agrupación y resumen

El resumen es un conjunto de palabras clave o frases que sintetizan un documento. El tamaño máximo de un resumen para *Verity* es de 512 bytes. Un grupo (o *cluster*) es una lista de documentos con un contenido similar. La agrupación automática determina los subtemas en un grupo de documentos. Su tamaño se encuentra entre uno y dos Kb.
Para poder ver la información del resumen y agrupación, se debe indicar explícitamente su activación pues por omisión, están apagadas. En los siguientes ejemplos se aprecia:

```
SELECT titulo,ret  FROM libros_UNAM
WHERE Vts_contains (texto,
ROW('invierno', 'summary=on'),
ret #IfxVtsReturnType);
```

```
SELECT titulo,ret  FROM libros_UNAM
WHERE Vts_contains (texto,
ROW('invierno', 'cluster=on'),
ret #IfxVtsReturnType);
```

Número de renglones recuperados

Para especificar el número máximo de renglones que se desean recuperar de una búsqueda basada en su puntuación, se utiliza el parámetro TOPN. Si el índice está fragmentado, TOPN representa el número de renglones recuperados por fragmento.
Para el siguiente ejemplo, el operador MANY determina el rango para los registros recuperados de la consulta. El parámetro TOPN determina que los primeros 10 registros serán recuperados.

```
SELECT detalles.titulo FROM peliculas_2000
WHERE Vts_contains (detalles,
ROW(' (<MANY>  Snoopy <IN> detalles',
'TOPN = 10' ));
```

Distinguir entre mayúsculas y minúsculas

Debido a que las búsquedas sobre un patrón se hacen ignorando si el planteamiento de la consulta fue con letras en mayúscula o minúscula (i.e. es insensible), puede existir la necesidad de efectuar una búsqueda exacta con respecto a una palabra con minúsculas o mayúsculas o mixta. Para ello, se tiene en *Verity* el uso de la opción CASE. Al emplear esta opción, no se aplica la raíz del término. Dos ejemplos, podrían ser:

```
SELECT titulo FROM libros
WHERE Vts_contains (texto, '<CASE> sociedad <IN> resumen');

SELECT titulo, precio FROM libros
WHERE Vts_contains (texto, '<CASE> Jesús de Nazareth <IN> titulo');
```

- Uso del thesaurus

El uso del *thesaurus* es por medio de un operador, en la siguiente instrucción se tiene la sintaxis:

```
SELECT detalles.titulo FROM peliculas_2000
WHERE Vts_contains (detalles.sinopsis, '<THESAURUS> terror');
```

- Operador de frase

Para tratar a un grupo de palabras como una sola, se tiene el uso del operador PHRASE:

```
SELECT detalles.titulo FROM peliculas_2000
WHERE Vts_contains (comentario, '<PHRASE> (Encuentros Cercanos)
<IN> detalles.sinopsis');
```

- Búsqueda por proximidad

Para indicar qué tan cerca deben estar un par de palabras, para satisfacer un criterio de búsqueda, se cuenta con el operador NEAR/n. Donde *n* es un número entero que indica el número máximo de palabras entre un par de términos definidos.

```
SELECT detalles.titulo
FROM peliculas_2000
WHERE Vts_contains (detalles,
'(Encuentros <NEAR/2> Cercanos) <IN> detalles.sinopsis');
```

- Búsqueda por sonidos similares

```
SELECT detalles.titulo
FROM peliculas_2000
WHERE Vts_contains (detalles,
' <SOUNDEX>Maier <IN> detalles.director');
```

- Búsqueda por expresiones regulares

Se emplean los símbolos:

*	Para aplicar la cerradura de Kleene
?	Para solicitar la existencia de un caracter.
[ ]	Para definir un conjunto de caracteres aceptables
-	Para determinar rangos: 'c[a-r]t'
{ }	Para listas de patrones alternativos separados por comas
^	Para excluir elementos de una lista

Ejemplos:

```
SELECT titulo
FROM libros_UNAM
WHERE Vts_contains (texto, '<WILDCARD> (?an)' );
```

Recupera coincidencias con los términos: Pan, can, ...

```
SELECT autor, titulo
FROM libros_UNAM
WHERE Vts_contains (texto, '<WILDCARD> (farmac*)' );
```

Recupera coincidencias con los términos: farmaco, farmacia, farmacología, ...

```
SELECT autor
FROM libros_UNAM
WHERE Vts_contains (texto, '<WILDCARD> ' 'st[^oa]ck ' ' );
```

Recupera coincidencias con los términos: stick, stuck pero no:stock o stack.

- Resaltar resultados

Otra de las bondades de *Verity*, es su capacidad para resaltar los términos que se emplearon en el planteamiento de la condición de la consulta. Este proceso se hace por medio de una serie de pasos:

1. Dar la instrucción SELECT con el operador **vts_contains**
2. Ejecutar la rutina **Vts_GetHighlight ()**
3. Desplegar los documentos recuperados en su ambiente específico (*viewer*), por ejemplo, para documentos HTML se puede usar *Netscape Navigator* o *Microsoft Internet Explorer*. *Verity* proporciona un *viewer* que se puede usar para ver diferentes formatos. Se debe comprar directamente con *Verity, Inc.* y no es soportado por *Informix*.

El siguiente ejemplo, la columna *texto* contiene texto HTML. Dado que se manejan prefijos y sufijos en HTML, éstos se deben especificar para indicar el tipo de formato que se le dará a los textos recuperados, y resaltar la palabra definida (en este caso, con subrayado: Taj Mahal):

```
SELECT Vts_GetHighlight (texto, "Taj Mahal", "<U>", "</U>")
FROM libros_UNAM
WHERE Vts_contains (texto, "Taj Mahal");
```

- Uso de múltiples lenguajes

Para poder emplear documentos escritos en diversos lenguajes, se debe especificar un lugar (locale) por cada lenguaje que se vaya a ocupar en la base de datos. Además, de los archivos de estilos que ocuparán para las consultas. En los siguientes ejemplos, se crean lugares para alemán y francés así como sus archivos de estilos (*aleman1* y *frances1*, respectivamente)

```
1) EXECUTE PROCEDURE Vts_CreateLocale
   ( "aleman1", "$INFORMIXDIR/extend/VTS.1.20.UC1/vdkhome/common/",
    "$INFORMIXDIR/extend/VTS.1.20.UC1/vdkhome/common/style");

2) EXECUTE PROCEDURE Vts_CreateLocale
   ( "frances1", "$INFORMIXDIR/extend/VTS.1.20.UC1/vdkhome/common/",
    "$INFORMIXDIR/extend/VTS.1.20.UC1/vdkhome/common/style");
```

Lo siguiente es crear un tipo de dato específico para los documentos en los lenguajes definidos, el primero para los documentos en alemán y el siguiente, para documentos en francés; ambos tipos son *clob* y sus nombres son *german_lob* y *french_clob*:

```
3) EXECUTE PROCEDURE Vts_CreateType ("clob", "german_clob", "aleman1");

4) EXECUTE PROCEDURE Vts_CreateType ("clob", "french_clob", "frances1");
```

Una vez definidos los tipos, se puede construir la tabla que almacenará los textos en los idiomas establecidos previamente para después poderlos indexar:

5) CREATE TABLE documentos_internacionales
(doc_num int, doc_ingles clob,
doc_aleman german_clob, doc_frances french_clob);

6) CREATE INDEX Idx_anglo ON documentos_internacionales(
doc_ingles Vts_clob_ops) USING vts IN subspace1;

7) CREATE INDEX Idx_aleman ON documentos_internacionales(
doc_aleman Vts_clob_ops) USING vts IN subspace2;

8) CREATE INDEX Idx_frances ON documentos_internacionales(
doc_frances Vts_clob_ops) USING vts IN subspace3;

Así, se puede empezar a llenar la tabla con datos y consultarse:

9) INSERT INTO documentos_internacionales
VALUES(1,FILETOCLOB('/local/informix/extend/doc/reportes/Larry_Ellison.txt',
'server'),
FILETOCLOB('/local/informix/extend/doc/reportes/Beniers_Wampula.txt',
'server'): :german_clob, NULL: :french_clob);

10) SELECT doc_num FROM documentos_internacionales
WHERE Vts_contains (doc_frances , 'renard');

- Soporte a XML

Este tipo de documentos, al estar estructurados en secciones, se le definen a *Verity* como **zonas**. Así, se permite la indexación de los documentos por filtros de secciones. La serie de pasos para poder llevar a cabo este proceso, es la siguiente:

1. Agregar un archivo de estilo. **style.dft**
2. Establecer filtro con **VtsUPTSet( )**, definir **FilterStopOnError** con 0 y **FilterRawDocOnError** a 1.

```
begin work;
        execute procedure VtsUTPSet("FilterEnable", "1");
        execute procedure VtsUTPSet("FilterStopOnError", "0");
        execute procedure VtsUTPSet("FilterRawDocOnError","1");
commit work;
```

3. Construir la estructura y almacenar datos.

```
CREATE TABLE documentos( titulo varchar(50), texto blob);

INSERT INTO documentos
VALUES ('Contacto', FileToBlob('Sagan.xml','cliente'));
```

4. Consultar.

```
SELECT * FROM documentos
WHERE Vts_contains (texto, 'Jupiter');

SELECT * FROM documentos
WHERE  Vts_contains(text, ' * <in> entry <in>  entries <in> invoice);

SELECT * FROM documentos
WHERE Vts_contains(texto, '* <in> product  <when> (maker <contains> "BSA")');
```

- Búsquedas difusas

Permite que las búsquedas sobre los documentos contengan términos similares a los especificados. Se emplea el operador TYPO/n para buscar patrones aproximados. La variable *n* expresa el número máximo de caracteres diferentes entre el patrón de búsqueda y el término encontrado(al valor numérico *n* se le llama **error de distancia**); estas diferencias pueden darse debido errores ortográficos, de desconocimiento o en documentos que fueron originados por el uso de un *scanner* (OCR). El siguiente ejemplo, muestra una consulta que permite hasta 3 transformaciones sobre el patrón *Chris*.

```
SELECT titulo FROM analisis_tesis
WHERE Vts_contains (texto, '<TYPO/3> Chris)' );
```

El error de distancia entre dos palabras se basa en el cálculo de errores, donde un error se define como la inserción, cambio, borrado o transposición de un caracter. En el capítulo II se habló de esta parte teórica conocida también como **Distancia de Hamming** o **Distancia de Levenshtein**.
Para el *Verity*, se acepta un valor máximo de 32 para la variable *n*.

- Búsqueda por párrafo

Para efectuar búsquedas de términos por párrafos con variaciones de raíces, es posible a través de la siguiente sintaxis:

```
SELECT titulo FROM analisis_tesis
WHERE Vts_contains (texto, 'búsqueda <PARAGRAPH> análisis de algoritmos ');
```

La siguiente búsqueda se efectúa sobre los términos exactos a buscar, sin aplicar variaciones de ellos.

```
SELECT titulo FROM analisis_tesis
WHERE Vts_contains (texto, '<PARAGRAPH>("análisis de algoritmos", "búsqueda",
"recuperación") ');
```

Los documentos recuperados con las reglas sintácticas anteriores, no se les aplica un rango de relevancia, para ello, se necesita aplicar el modificador MANY como en el siguiente ejemplo:

```
SELECT titulo FROM analisis_tesis
WHERE Vts_contains (texto, '<MANY><PARAGRAPH>("DBA", "administrador")' );
```

- Búsqueda de frases

Si la búsqueda sobre documentos es por frases, es necesario precisarlo como en los siguientes ejemplos usando comillas sencillas o dobles:

```
SELECT titulo FROM peliculas
WHERE Vts_contains (comentario, ' Misión Imposible II ' );

SELECT titulo FROM peliculas
WHERE Vts_contains (comentario, ' " Misión, Imposible, II" ' );
```

Una manera explicita de realizar la consulta puede ser:

```
SELECT titulo FROM peliculas
WHERE Vts_contains (comentario, ' <PHRASE> (Misión Imposible II) ' );
```

Los documentos recuperados con las reglas sintácticas anteriores, no se les aplica un rango de relevancia, para ello, también se necesita aplicar el modificador MANY:

```
SELECT titulo FROM peliculas
WHERE Vts_contains (texto, '<MANY> <PHRASE> (CONTACTO)' );
```

- Búsqueda por oración

Si la búsqueda sobre documentos es de palabras que aparezcan en un mismo enunciado u oración, se debe especificar explícitamente:

```
SELECT titulo FROM peliculas
WHERE Vts_contains (texto, 'compilador <SENTENCE> Perl' );

SELECT titulo FROM peliculas
WHERE Vts_contains (texto, <SENTENCE> ("compilador","Perl")' );
```

- Otros operadores en consultas

El operador NEAR/n se puede usar junto con el modificador **ORDER**, cuyo efecto es que los términos aparezcan en el mismo orden en que se definieron en la consulta. Ejemplo:

```
SELECT titulo FROM analisis_tesis
WHERE Vts_contains (texto, 'analisis <ORDER> <NEAR/5> algoritmos
<ORDER> <NEAR/5> numerico');

SELECT titulo FROM analisis_tesis
WHERE Vts_contains (texto, ' <ORDER> <NEAR/1> ("world", "wide", "web")' );
```

El operador **STEM** por omisión está activo, es por ello que no es necesario especificarlo, así, las siguientes instrucciones hacen lo mismo:

```
SELECT titulo FROM libros  WHERE Vts_contains (texto, '<STEM> base' );

SELECT titulo FROM libros  WHERE Vts_contains (texto, 'base' );
```

- Operadores de lenguaje natural

Dentro de estos operadores está el **FREETEXT**, para texto libre que utiliza el análisis y puntuación de documentos.

```
SELECT titulo FROM analisis_tesis
WHERE Vts_contains (texto, '<FREETEXT> ("Negociación de paz en el Medio Oriente") ');
```

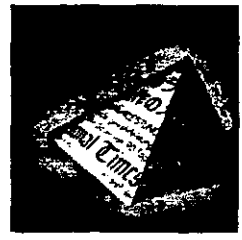
4.3.3 Los Clientes de Informix

Los *clientes en México que usan los productos de Informix son:*

Informix Dynamic Server 9.2 Server (Usado como DBMS)

- Secretaría de Hacienda (***Excalibur***)
- Palacio de Hierro (***Verity***)
- Librerías Gandhi (***Excalibur***)
- IMSS (***Excalibur***)
- CFE (***Excalibur***)
- TELMEX (***Verity***)

4.4 Oracle8 ConText Cartridge



4.4.1 Evolución

Oracle inicia en 1975 con una investigación lingüística que se prolonga hasta 1990 cuando se libera el producto llamado *SQL*TextRetrieval*. *ConText Option* se libera para 1996. *ConText Cartridge* aparece en junio de 1997. Es una extensión del RDBMS para la construcción de aplicaciones de misión crítica que integran datos relacionales y texto; esta solución de administración de texto permite a las organizaciones obtener información a partir del texto de manera rápida y fácil como se hace sobre cualquier dato estructurado. Combina el poder del RDBMS y el lenguaje *SQL* apoyados de la tecnología de recuperación de texto. Al integrarse a una base de datos *Oracle*, implica que cualquier tarea de administración para construir aplicaciones de texto se hará a través de *SQL* y *PL/SQL*. *Oracle ConText Cartridge* se integra al *Oracle Web Server*, de tal modo que los desarrolladores pueden extraer texto de la base de datos directamente y crear de forma dinámica páginas Web no siendo necesaria la construcción de *CGIs*.

4.4.2 *NCA (Network Computing Architecture)*

Oracle8 parte de la NCA (*Network Computing Architecture*), la estrategia multiplataforma de Oracle para desarrollar aplicaciones basadas en estándares abiertos que integren el entorno de desarrollo de la computación.
Esta propuesta consiste en tener cartuchos (*cartridges*) o software independiente del RDBMS que le permita comunicarse con este último y resolver las tareas limitadas. Son productos que se instalan sobre el manejador y que tienen un costo extra. En figura 4.7 se puede observar la presencia del *ConText Cartridge*, encargado de resolver la recuperación de texto.

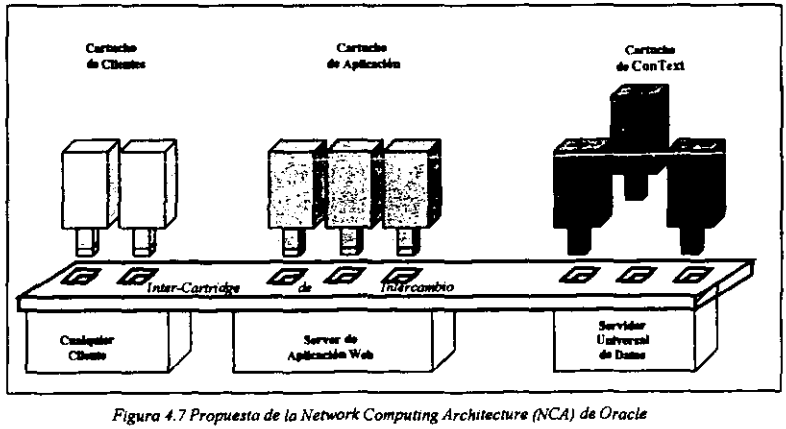


Figura 4.7 Propuesta de la Network Computing Architecture (NCA) de Oracle

Del diagrama anterior, se observa que el *ConText Cartridge* es una capa de software que se instala sobre el RDBMS, lo cual no altera al comportamiento del manejador de bases de datos, sino que es un agregado que lo apoya en sus limitantes de manejo de texto. *ConText* es una opción de servidor que habilita consultas de texto a través de *SQL* y *PL/SQL* desde las interfaces de herramientas cliente *Oracle* tales como *SQL*Plus*, *Designer/2000*, *Power Objects*, *Oracle Web Server*, *Precompiladores* y *Oracle Forms*. Muchas herramientas inclusive podrán hacer llamadas a *procedimientos almacenados* (*store procedures*) que ejecuten consultas y operaciones sobre el texto.

Una vez instalado el *ConText*, el proceso de conversión del texto a un tipo de dato nativo de la base de datos relacional es transparente, este proceso se observa gráficamente en la figura 4.8; los datos de tipo texto serán manipulados como cualquier otro tipo de dato en la base de datos. Cuando el texto es insertado, actualizado o borrado, *ConText* automáticamente administra el cambio. Se provee indexación avanzada, análisis, recuperación y funcionalidad de proyección que puede ser integrada en cualquier aplicación de texto que use el servidor *Oracle8*. Además, la creación de más de una columna tipo LOB, es posible así como la indexación. Esto es importante pues las restricciones vistas en el capítulo I quedan resueltas satisfactoriamente.

Dentro de los conceptos importantes para el *ConText*, se encuentran la definición de **preferencias, políticas e índices.**

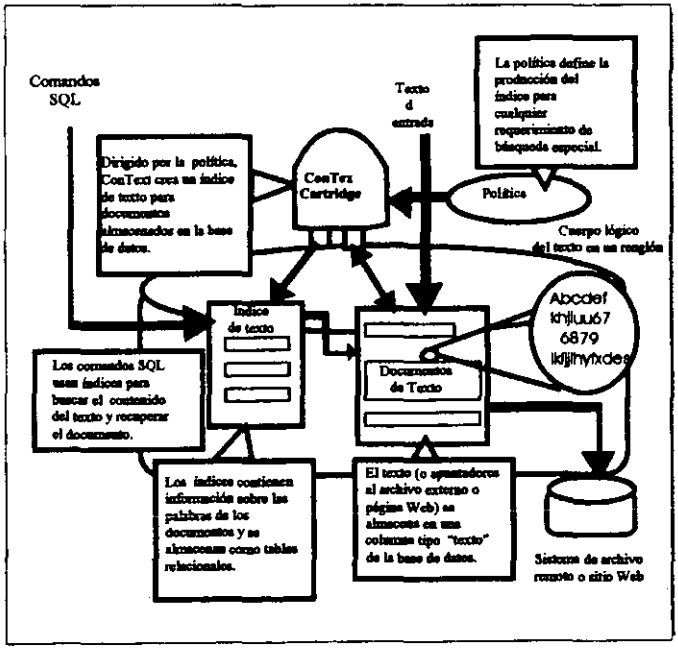


Figura 4.8 Tratamiento de texto como tipo de dato nativo en la base de datos

4.4.3 La arquitectura del *Context Cartridge*

A la arquitectura del ORDBMS *Oracle*, se agregan uno o más procesos servidores de *ConText* y una cola para manipular las operaciones sobre el texto. En la figura 4.9 se puede observar esta nueva arquitectura de integración entre los procesos servidores del manejador de la base de datos y los del servidor de textos en el área de memoria llamada *SGA* (*System Global Area*)

Context puede usarse tanto en modo de servidor dedicado (un proceso servidor por proceso de usuario) como en multi-hilo (servidor dedicado, despachador y procesos base o de *background*). En el modelo estándar del servidor *Oracle*, cuando un proceso usuario se conecta a la base de datos, el proceso servidor atiende los requerimientos solicitados. Con el *Context*, si se solicita un requerimiento, éste se envía a la cola de requerimientos de texto para ser procesada por el siguiente proceso disponible de *Context*. Los resultados de la consulta de texto se combinan con los datos estructurados y se regresan al proceso de usuario como un conjunto de datos resultantes.

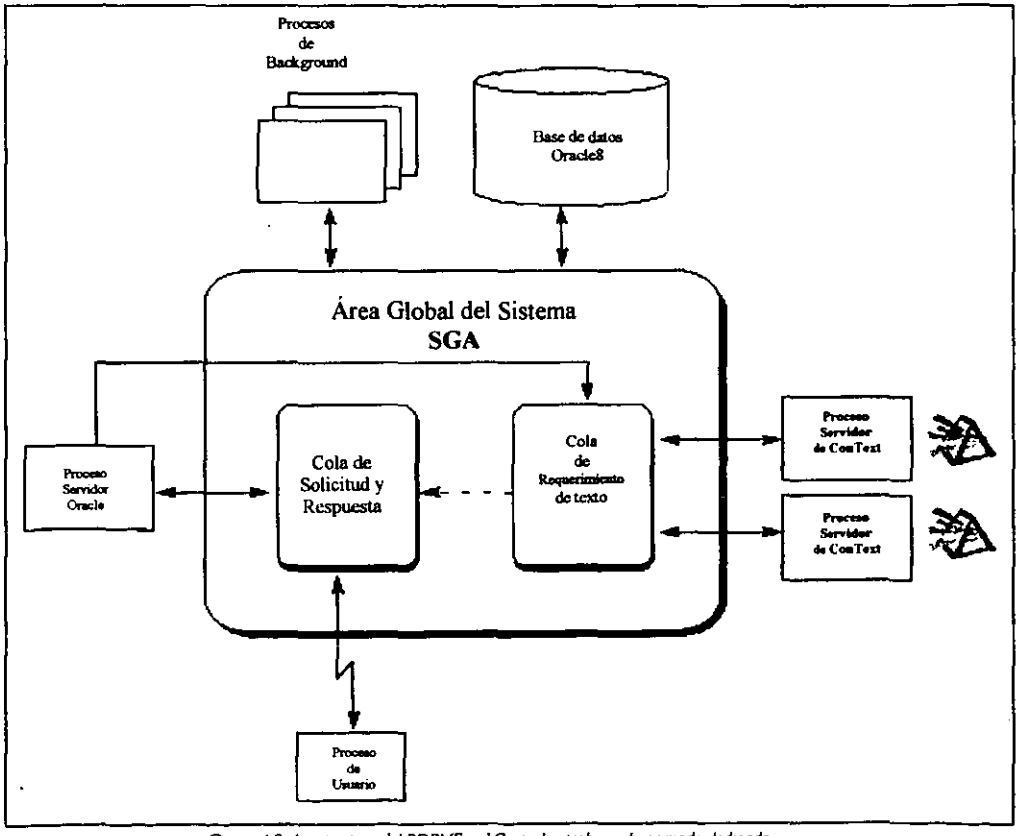


Figura 4.9 Arquitectura del RDBMS y el Cartridge trabajando en modo dedicado.

La administración se divide en tres áreas principales, las cuales implican algunas tareas primarias:

- Administración de *ConText*
  - Administración de usuarios de *ConText*
  - Administración y monitoreo de servidores *ConText*
  - Administración y monitoreo de colas *ConText*
- Servicios Lingüistas del sistema
  - Creación de políticas y preferencias
  - Definición de configuraciones
- Administración del texto
  - Definición de almacenamiento y opciones de indexación
  - Identificación de columnas texto a través de políticas
  - Creación de índices de texto y por tema
  - Creación y administración de *thesaurus*

ConText utiliza un diccionario de datos, diferente del diccionario de datos de *Oracle* para cada base de datos; lo emplea para almacenar **opciones de indexación, preferencias** y **políticas**. En la figura 4.11 se aprecia la relación gráfica entre estos elementos.

4.4.4 Idiomas y formatos

Inglés	Alemán	Francés	Italiano	Español
Japonés	Koreano	Chino	Holandés	

Los formatos soportados son:

ASCII	*HTML*	*MS Word*
WordPerfect	*Adobe Acrobat/PDF*	*AmiPro*

4.4.5 Proceso de creación de índices

Las opciones de indexación que deben ser especificadas para *Context* se dividen en siete categorías funcionales (clases), definiendo para cada una de ellas un grupo de *tiles*, que son objetos que especifican a los servidores de *ConText* cómo manejar el texto así como las instrucciones de indexación.

1. **Almacenamiento de datos (*data_store*).-** Se usan para crear preferencias que especifican cómo se almacenarán los datos de tipo texto en la base de datos (interno o directo, externo y maestro/detalle). Los *tiles* que se emplean para almacenamiento son: **DIRECT, OSFILE, URL**, y **MASTER DETAIL**.

2. **Filtro (filter).-** Se usan para crear preferencias que determinen cómo se debe filtrar el texto para indexarse y resaltarse. Los tiles que se emplean para el filtro son:

   - **FILTER NOP** (texto plano, sin formato)
   - **HTML FILTER** (texto plano con etiquetas de HTML, sin formato)
   - **BLASTER FILTER** (texto con formato, texto plano y con etiquetas de HTML)
   - **USER FILTER** (texto procesado a través de un filtro externo)

3. **Analizador léxico (lexer).-** Para textos en inglés, se proporciona un analizador léxico para crear índices por tema. Como su nombre lo dice, divide el texto en *tokens*. Contiene los siguientes *tiles*:
   - **JAPANESE V-GRAM LEXER**
   - **KOREAN LEXER**
   - **CHINESE V-GRAM LEXER**
   - **BASIC LEXER (analizador léxico para otros lenguajes)**
   - **THEME LEXER (analizador léxico para crear índices por tema)**
4. **Motor (*engine*).-** Sirve para crear preferencias que sirven para especificar cómo se crean los índices y dónde se deben almacenar. Solo contienen el *tile* **GENERIC ENGINE**.
5. **Lista de palabras (*wordlist*).-** Se usan para crear preferencias que habilitan tres métodos de expansión de consulta: Raíz (*stemm*), difuso y sonido. Solo contienen el *tile* **GENERIC WORD LIST**.
6. **Lista de parada (stoplist)** .- Permite que se definan las palabras de parada o no usadas en una búsqueda; se ignoran en el proceso de indexación. Solo contienen el *tile* **GENERIC STOP LIST**.
7. **Compresión.-** No definida para Oracle8.

Antes de crear los índices sobre las columnas de una tabla, es preciso definir las preferencias y las políticas. Una política es una agrupación lógica de seis preferencias, asignada a una columna en la base de datos. Una política especifica las opciones usadas por *ConText* para crear un índice en la columna tipo texto. La asignación de una política es previa a la indexación (figura 4.10).

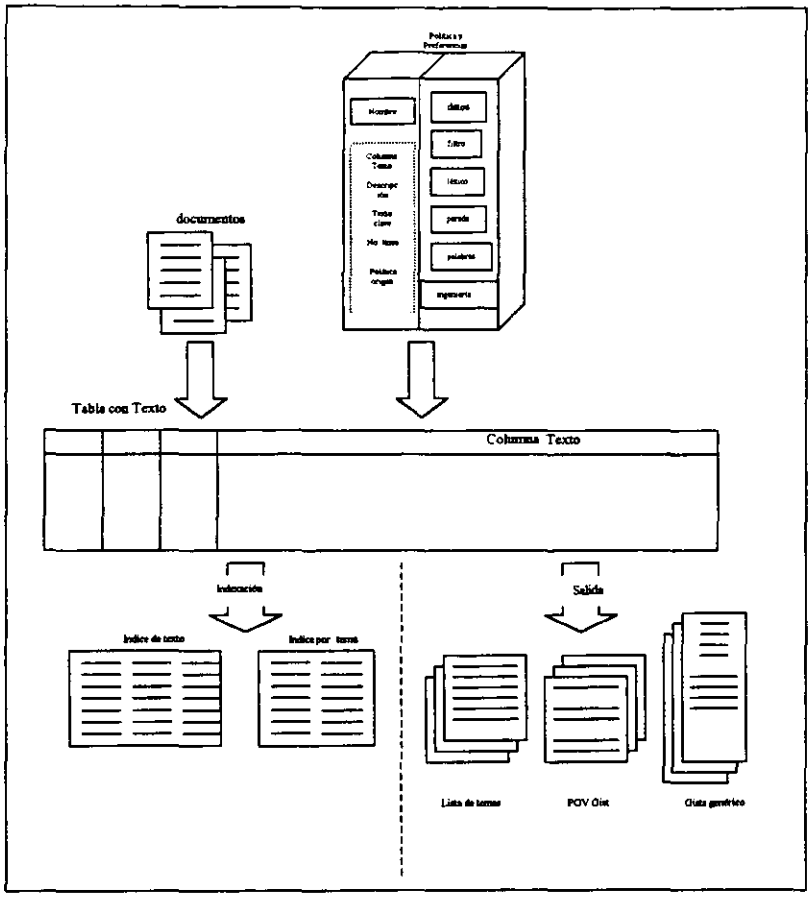


Figura 4.10 Las preferencias y políticas

Para crear una **preferencia** en el diccionario de datos, se emplean los procedimientos SET_ATTRIBUTE y CREATE_PREFERENCE:

Ejemplos:

```
begin
  ctx_ddl.set_attribute ('PATH', '/public/doc1:/public/doc2');
  ctx_ddl.create_preference ('PUB_DOCS', 'Docs stored in files', 'OSFILE');
end;
```

- Se crea un analizador:

```
begin
 ctx_ddl.create_preference ('MY_THEME_PREF', 'Pref para indice tema', 'THEME LEXER');
end;
```

- Crea una preferencia de filtro para una columna que contiene documentos en MS Word:

```
begin
 ctx_ddl.set_attribute('FORMAT','11')
 ctx_ddl.create_preference('WP6_FILT', 'WP6 filter', 'BLASTER FILTER');
end;
```

- Creación de una preferencia de filtro para una columna que contiene documentos en formatos *AmiPro. PDF (Adobe Acrobat)*, y *WordPerfect* 6.0

```
begin
 ctx_ddl.set_attribute('EXECUTABLE', 19,'amipro.sh', 1)
 ctx_ddl.set_attribute('EXECUTABLE', 57,'acrobat.sh', 2)
 ctx_ddl.create_preference('MULT_FILT', 'multiple ext/int filters', 'BLASTER FILTER');
end;
```

Una vez definidas las preferencias, se pueden asignar en la creación de las políticas. Para crear una política de indexación de texto para una columna, se llama al procedimiento CTX_DDL.CREATE_POLICY y se especifican los siguientes parámetros requeridos:

- nombre de la política
- calificador completo del nombre de la columna para la política

Por ejemplo:

```
exec ctx_ddl.create_policy('ctx_docs','ctxdev.docs.text')
```

Es decir, una política de indexación de texto llamada *ctx_docs* es creada para la columna *text* en la tabla *docs* propiedad de *ctxdev*. Se tienen las siguientes opciones por omisión:

- Almacenamiento directo de datos (texto almacenado en una columna de tipo texto; cada renglón en la tabla representa un documento completo por separado)
- No hay filtro (plan, texto sin formato)
- Léxico Básico (puntuación estándar, continuación, y caracteres de junta para texto indexado)
- Lista de parada (*stoplist*) por omisión (lista de palabras que no son incluidas en el texto indexado)
- No hay *wordlist* (Información de sonido no es generada durante la indexación de texto)
- Ingeniería de texto genérico indexado

Si se desean diferentes opciones para la política, éstas se pueden especificar así como otras preferencias al llamar el procedimiento CREATE_POLICY.

El siguiente código PL/SQL crea una política en una columna para indexar el texto.

```
begin
 ctx_ddl.create_policy (policy_name    => 'DOC_POL',
                        colspec        => 'DOCS.ARTICLES',
                        textkey        => 'PK',
                        dstore_pref    => 'PUB_DOCS',
                        engine_pref    => 'DOC_ENGINE',
                        filter_pref    => 'WORD6',
                        lexer_pref     => 'DOC_LINK',
                        wordlist_pref => 'CTXSYS.SOUNDEX',
                        stoplist_pref => 'MINI_STOP_LIST');
end;
```

Para crear una política de indexación por tema, se usa la preferencia de tema (THEME_LEXER).

```
begin
 ctx_ddl.create_policy (policy_name    => 'DOC_POL',
                        colspec        => 'DOCS.ARTICLE',
                        textkey        => 'PK',
                        dstore_pref    => 'PUB_DOCS',
                        engine_pref    => 'DOC_ENGINE',
                        filter_pref    => 'WORD6',
                        lexer_pref     => 'MY_THEME_PREF',
                        wordlist_pref => 'CTXSYS.SOUNDEX',
                        stoplist_pref => 'MINI_STOP_LIST');
end;
```

Oracle Context Cartridge requiere de cuatro fases para su indexación, cada fase tiene sus características que pueden ser especificadas a través de la cláusula PARAMETERS de la sentencia CREATE INDEX (ver figura 4.11)
El almacenamiento de datos especifica al proceso indexador de texto cómo serán almacenados los datos indexados, esto puede ser el nombre de la columna donde se almacenarán los datos o indicar que en la columna se almacenará un URL de una página Web. Así, este dato es procesado por un filtro que se encarga de trasladar los datos a su equivalente (*XML, HTML* o texto simple).
Las columnas *XML/HTML* se dividen en grupos, el seccionador se encarga de definir cómo se efectúa esta tarea. Finalmente, el analizador léxico, especifica cómo se deben formar las palabras.

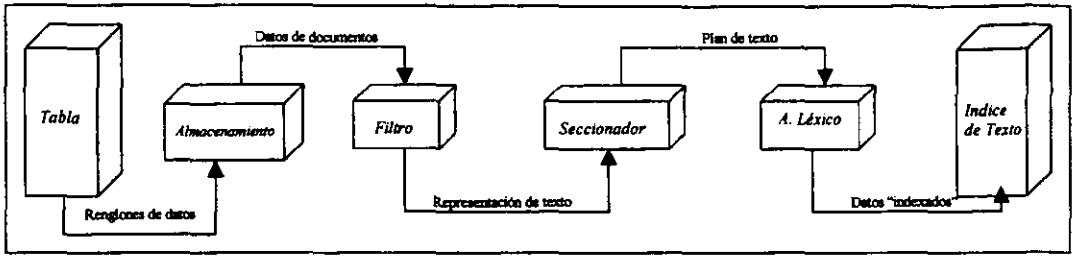


Figura 4.11 Las cuatro fases de indexación de texto con Oracle

Creación de índices para las columnas de tipo texto

Se llama el procedimiento almacenado CREATE_INDEX en el paquete CTX_DDL el cual está en el lenguaje *PL/SQL* y especifica la política del texto indexado para la columna. Se tiene así:

```
exec ctx_ddl.create_index('ctx_docs')
```

Esta instrucción crea un índice de texto para la columna *ctxdev.docs.text* en la política *ctx_docs*. Después de que se ha creado el índice para la columna, *ConText* puede procesar la consulta sobre la columna.

Opcionalmente, se puede incluir un valor numérico en los argumentos de CTX_DDL.CREATE_INDEX para especificar el número de servidores de *ConText* usados para una indexación en paralelo. Por ejemplo, aquí el CREATE_INDEX usa los primeros cuatro servidores disponibles de *ConText*.

```
execute ctx_ddl.create_index('DOC_POL', 4)
```

4.4.6 Opciones de búsqueda

Como se aprecia en la figura 4.12, antes de ejecutar una consulta sobre una columna, es preciso que ésta se encuentre indexada. Una consulta de texto implica codificar un criterio de búsqueda de tal modo que el texto pueda ser buscado de modo eficiente y los documentos relevantes puedan ser recuperados. *ConText* soporta expansión de consultas, en la forma de obtener sus raíces, sonido y emparejamiento difuso para el inglés, francés, español, italiano, alemán y holandés.
Para documentos en inglés, las consultas pueden ser por tema o por sus conceptos principales.
Para lenguajes multi-byte, *ConText* proporciona los siguientes analizadores léxicos: japonés, koreano y chino. Para el japonés, se tienen los siguientes sistemas de escritura: Kanji, Hiragana y Katakana.

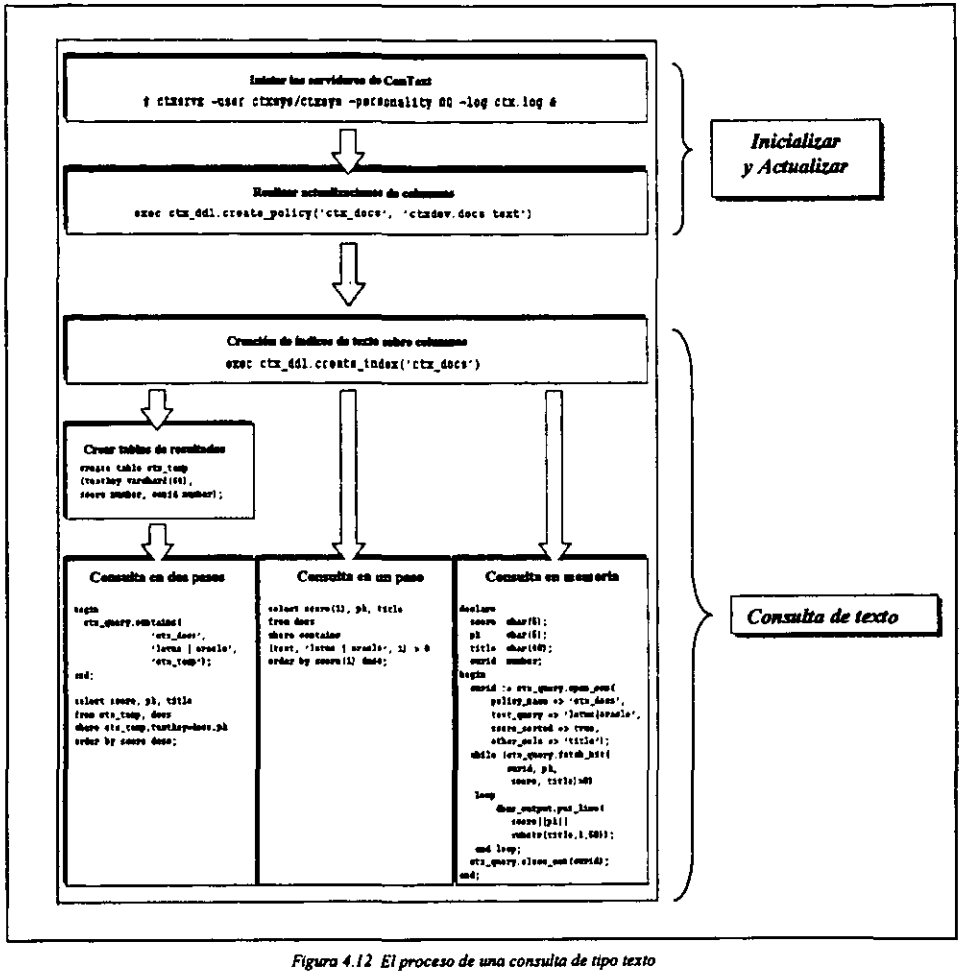


Figura 4.12 El proceso de una consulta de tipo texto

4.4.7 Métodos de Consulta

ConText soporta tres métodos para ejecutar las consultas:

- Un-paso (*One-step*)
- Dos-pasos (Two-step)
- En memoria (In-memory)

Consultas en un-paso

En una consulta de un paso, se crea una sentencia SQL que usa las funciones de consulta del *Context* para buscar los documentos relevantes y regresar un conjunto de registros seleccionados y columnas de la tabla de texto directamente para el usuario.

La lista de documentos exitosos *(hitlist)* es procesada por *Context* usando tablas internas de resultados. No se deben crear tablas de resultado antes de estar corriendo una consulta de un paso; sin embargo, la tabla interna de resultados no está disponible para el programa de aplicación. Una consulta de un paso usa la función *ConText* SQL llamada *CONTAINS*, la cual es llamada directamente en la cláusula *WHERE* de la sentencia *SELECT*.
Consulta booleana (OR) de un paso:

```
SELECT score(1), pk, titulo FROM doctos
WHERE CONTAINS(texto, 'lotus | oracle', 1) > 0
ORDER BY score(1) desc;
```

Consulta de un paso por frase:

```
SELECT articulo FROM tabla_texto
WHERE CONTAINS(texto, '{sociedad civil mexicana} ') > 0 ;
```

Consultas en dos-pasos

Las consultas de este tipo usan un procedimiento *PL/SQL* en el primer paso para crear una lista de documentos exitosos (*hitlist*) y almacenar los resultados en una lista específica llamada *tabla de resultados*. El segundo paso usa una sentencia SELECT para proyectar los resultados de la tabla. Además, ésta tabla puede ser usada en una operación de junta (*join*) con la tabla original para recuperar información más detallada. En el método de dos pasos, la tabla de resultados está disponible para el programa de aplicación.

```
CREATE TABLE CTX_TEMP(textkey  varchar2(64),
                      score    number,
                      conid    number);

EXECUTE CTX_QUERY.CONTAINS ('ARTICLE_POLICY', 'petroleo',
'CTX_TEMP');

SELECT score, title
FROM CTX_TEMP, TEXTTAB
WHERE texttab.PK= CTX_TEMP.TextKey
ORDER BY score DESC;
```

Si se desean ejecutar consultas en dos-pasos, primero se debe crear una tabla de resultados: la columna *textkey* almacena la llave primaria para los documentos y la columna *score* almacena las puntaciones generadas por la consulta. La tercer columna, *conid*, almacena un número, el cual identifica los resultados para cada consulta, esta columna es usada solo cuando la tabla de resultados es empleada para almacenar los resultados de múltiples consultas.
La tabla de resultados puede tener cualquier nombre; sin embargo, ésta debe tener una estructura fija (columnas con nombre y tipos).
En el segundo paso de una consulta de dos-pasos, se debe llamar al procedimiento almacenado *CONTAINS* del paquete *CTX_QUERY* que está en *PL/SQL* para llenar una tabla de resultados

existente. En el tercer paso, se debe consultar la tabla para recuperar una lista de documentos exitosos de los documentos. En el siguiente ejemplo, se tiene una consulta de dos-pasos:

Consultas en memoria

En las consultas en memoria, se usa un buffer y un cursor; el buffer regresará los resultados de la consulta, así como las tablas de resultados de las consultas de uno y dos pasos. Así este tipo de consulta es generalmente mucho más rápido para las listas cortas de documentos exitosos más cortas. *ConText* escribe los resultados de la consulta en el buffer, se efectúa el *fetch* de los resultados, y se cierra el cursor. Los resultados pueden ser recuperados en orden respecto a sus llaves de texto (*textkeys)* y ordenados por su puntuación.
Las consultas en memoria pueden ser ejecutadas usando procedimientos de *PL/SQL*: *OPEN_CON* abre un cursor *CONTAINS* para una buffer de consulta y ejecuta la consulta. Los resultados de la consulta se almacenan en el buffer. El *FETCH_HITS* recupera los resultados, una tupla a la vez, y el *CLOSE_CON* libera el cursor *CONTAINS*.
Las consultas en memoria son generalmente, más rápidas que las consultas de uno o dos pasos que regresan grandes listas de éxitos. Estas consultas no requieren de las tablas de resultados en la base de datos. Sin embargo, en las consultas de memoria, no soportan consultas por estructuras de datos. Si se desea consultar por estructura de datos, la estructura de datos debe ser ejecutada de manera separada de la consulta de memoria y los resultados de ambas consultas deben juntarse.
Ejemplo:

```
declare
score char(5);
pk    char(5);
title char(40);
curid number;
begin
curid := ctx_query.open_con(policy_name => 'ctx_docs',
                            text_query => 'lotus|oracle',
                            score_sorted => true,
                            other_cols => 'title');
while (ctx_query.fetch_hit(curid, pk, score, title)>0)
loop
dbms_output.put_line(score||pk||substr(title,1,50));
end loop;
ctx_query.close_con(curid);
end;
```

En este ejemplo, *score*, *pk*, *title*, y *curid* son declaradas como variables que son usadas por CTX_QUERY.OPEN_CON y CTX_QUERY.FETCH_HIT.
El argumento SCORE_SORTED para OPEN_CON especifica que los resultados de la consulta son almacenados en el buffer de manera descendente por el valor de su puntuación. El argumento OTHER_COLS especifica que la columna *title* de la tabla consultada es regresada junto con los resultados *score* y *pk* de la consulta. FETCH_HITS recupera la puntuación, llave primaria, y el título para cada éxito hasta que el buffer esté vacío.

La limitante de las consultas en memoria es que no pueden consultar datos estructurados.

Capacidades de consultas

- Operadores lógicos para consultas booleanas

Estos operadores combinan los términos en una expresión de consulta. Se aplican a palabras y frases. Dependiendo del operador lógico, *ConText* aplica criterios de ponderación de la puntuación de los documentos.

Operador	*Sintaxis*	*Ejemplo*
AND	term1 & term2 term1 AND term2	'batman & robin' 'batman and robin'
OR	term1 | term2 term1 OR term2	'peras | manzanas' 'peras or manzanas'
NOT	term1 ~ term2 term1 not term2	'animal ~ perro' 'animal not (perro or gato)'
EQUIVALENCE	term1 = term2 term1 EQUIV term2	'ave fenix= Sor Juana'

Tabla 4.3 Operadores lógicos. Sintaxis

- Consultas estructuradas

Es una consulta basada en una columna de tipo texto y datos estructurados. El procedimiento CTX_CONTAINS proporciona un parámetro adicional: STRUCT_QUERY para especificar la condición de WHERE en una consulta sobre un tipo de dato estructurado.

Por ejemplo, para proyectar todos los artículos que contengan la palabra *Oracle* que fueron escritos desde el 1° de octubre de 1996, se plantea:

```
EXEC CTX_QUERY.CONTAINS ('textos_nuevos', 'Oracle','res_tab',
        STRUCT_QUERY => 'pub_fecha >= (' '1-OCT-1996' ' )');
```

- Consultas por tema

Además de las consultas por palabra o frase (consulta de texto), se pueden efectuar consultas por tema o por conceptos principales de los documentos. En estas consultas, al igual que en las de texto, se debe crear un índice (que en este caso será por tema de los documentos) antes de efectuar la consulta. La ventaja en este tipo de consultas radica en que el usuario no necesita proporcionar la palabra patrón para la búsqueda. *ConText* interpreta la consulta conceptualmente de acuerdo con su visión del mundo y regresa una lista de documentos exitosos basada en el tema, dependiendo de la relevancia de cada documento con la consulta.

Se pueden emplear los métodos estándar para ejecutar consultas por tema en uno o dos pasos y en memoria. En una consulta por tema, se pueden usar la mayoría de los operadores empleados regularmente en las consultas de texto. Para crear un índice por tema, una política por indexación de tema debe definirse. Múltiples políticas pueden ser asignadas a una columna, así, una columna puede tener más de un índice. Cuando una consulta se ejecuta, se puede especificar el nombre de una política para indicar el índice que se usará en el proceso de consulta.

```
EXECUTE CTX_DDL.CREATE_POLICY('TEMA_POLITICA', 'tabla1.texto',
                              lexer_pref =>'CTXSYS.THEME_LEXER');
```

Al crear un índice de tema, *ConText* crea un archivo de firma (vector) para cada documento; cuando se hace una consulta, ésta también genera su vector por tema que es comparado con los archivos de firma; el número máximo de temas por documento que puede manejar *Context* es de 16. Cada

vector está formado por términos y pesos. Así, se puede implementar las consultas de uno y dos pasos por tema:

```
SELECT * FROM TEXTO_TABLA WHERE CONTAINS(texto, 'ingeniería de software') > 0;

EXECUTE CTX_QUERY.CONTAINS('TEMA_POLITICA', 'ingeniería de software','CTX_TEMP');
```

- Consulta por proximidad

El operador **NEAR** permite buscar dos términos que estén separados por no más de un número específico de palabras. Su sintaxis es: ' term1 ; term2 ' o bien ' term1 ACCUM term2 '

- Consulta por puntuaciones (*score*)

La puntuación está basada en un análisis numérico de las ocurrencias de la expresión de la consulta. Por ejemplo, un documento que contiene la expresión de búsqueda 10 veces es considerado más relevante que uno que contenga la expresión 5 veces. En las consultas básicas, la puntuación es calculada como el número de veces que una palabra seleccionada en una consulta aparece en el documento, este valor puede ser usado para ordenar la lista recuperada que tal manera que los documentos con el mayor valor que aparezca primero.

En consultas más complejas, la puntuación es afectada por varias relaciones entre palabras y frases; este valor es generado por la ingeniería de propósito general durante las consultas; los operadores ACCUMULATE, MINUS, NEAR y WEIGHT aplican criterios para calcular la puntuación final de la consulta. En la tabla 4. 4 se aprecia la sintaxis de estos operadores. Los valores asignados están en el rango [1- 100]. La función **SCORE** puede ser usada en la lista del SELECT, la cláusula **ORDER BY** o en la **GROUP BY**.

```
SELECT SCORE(10), SCORE(20), titulo FROM documentos
WHERE CONTAINS (texto, 'Sagan', 10) OR CONTAINS (texto,'Astronomia', 20)
OR CONTAINS (texto, 'Copernico',30)
ORDER BY SCORE(30)
GROUP BY SCORE (20);
```

Operador	*Sintaxis*	*Ejemplo*
ACCUMULATE	term1 , term2 term1 accum term2	batman , robin batman accum robin
MINUS	term1 - term2 term1 minus term2	batman - robin batman minus robin
NEAR	term1 ; term2 term1 near term2	batman ; robin batman accum robin
WEIGHT	term * *n, donde n es un número de 0.1 a 10*	batman * 0.3

Tabla 4.4 Operadores que afectan la puntuación (score)

- Consulta por la raíz

El operador **STEM** expande una palabra en todas aquellas que tengan la misma raíz lingüística. Está disponible en los siguientes idiomas: español, francés, italiano, holandés y alemán. Su sintaxis se tiene en la tabla 4.5

- Consulta por sonidos similares

El operador **SOUNDEX** expande una palabra en todas aquellas que suenen de manera similar a ella. SOUNDEX se implementa usando el mismo algoritmo que el SQL estándar. Disponible sólo en inglés. Su sintaxis se tiene en la tabla 4.5

- Consulta difusa

El operador **FUZZY** expande una palabra dada en todas las palabras que se deletrean o tienen una ortografía similar a ella. Su sintaxis se tiene a continuación, en la tabla 4.5

Operador	*Sintaxis*	*Ejemplo*
STEM	$term	$auto
SOUNDEX	!term	!Meyer
FUZZY	?term	?toco

Tabla 4.5 Operadores de expansión)

- Consulta con expresiones regulares

Algunos caracteres pueden usarse en las consultas por medio de expresiones que permiten la expansión de las palabras buscadas por medio de patrones. Estos caracteres son:

Caracter	*Descripción*	*Ejemplo*
%	Cerradura Kleene	(texto, 'relacion%')
_	Especifica una posición en la cual, puede aparecer un caracter.	(texto, '_ ion')

Tabla 4.6 Caracteres para expresiones regulares

- Consulta con Thesaurus

ConText proporciona un marco de *thesaurus* que cumple con el estándar *ISO-2788*, permitiendo que cualquier otro pueda ser agregado. Los operadores para la extensión de términos permiten búsquedas basadas en sinónimos, términos preferidos, términos relativos, términos genéricos etc. Existen diez clases de operadores para el *thesaurus* que corresponden a los diez tipos de relaciones que pueden ser definidas en el estándar. Como se mostró en el capítulo III, los operadores más importantes, son: Sinónimos, preferencia, términos relacionados, asociaciones, etc.
Estos operadores son: SYN, PT, RT, TT, BT, NT, BTG, NTG, BTP y NTP. Para su implementación, s emplean funciones *PL/SQL* en las que se define el término, el nivel (para jerárquias) y el nombre del *thesaurus*.

- Contando los éxitos (hits) de una consulta

Se puede emplear la función CTX_QUERY.COUNT_HITS para obtener el número de éxitos de una consulta sin generar cuentas para los éxitos. Los documentos pueden estar almacenados en una base de datos local o remota. Contar los éxitos de una consulta es generalmente mucho más fácil que ejecutar una consulta completa y puede servir para auditorias de consultas muy grandes.

- Consultas resaltando el texto

ConText permite resaltar palabras o frases por medio del uso del procedimiento CTX_QUERY. HIGHLIGHT que requiere de una programación PL/SQL muy específica. Además, dependiendo de la salida para desplegado, se requerirá de SQL*Plus (para modo caracter) en el cual, deben definirse una serie de variables de ambiente. Si se desplegará en modo gráfico, se requiere de *ConText*

Cartridge Viewer Control (CTXV32.OCX) que permite a los usuarios ver los documentos resaltados en un ambiente Windows-32.

- Consultas almacenadas

Una expresión de consulta almacenada (*Stored Query Expression*, SQE) es una expresión de consulta que ha sido almacenada en las tablas de la base de datos junto con los resultados de la consulta. Se pueden combinar consultas referenciando una SQE en la expresión de consulta de otra consulta. Usando una SQE en una consulta, la ejecución de la misma es más rápida de ejecutar porque los resultados ya han sido almacenados en la base de datos.
Una SQE también puede ser usada para ejecutar consultas interactivas, en las cuales una consulta inicial refinada usando una o más consultas adicionales.
En el siguiente ejemplo, se crea una sesión SQE llamada *prog_lang*, consulta la columna de texto para la política *resumen_emp* y recupera todos los documentos que contienen el término *cobol*. Los resultados se almacenan en la tabla SQE de la política.

```
EXEC ctx_query.STORE_SQE('resumen_emp', 'prog_lang','cobol', 'session');
```

Así, *prog_lang* puede ser llamada en la expresión de consulta:

```
SELECT score, docid FROM emp
WHERE CONTAINS(resumen, 'sqe(prog_lang)' ) > 0
ORDER BY score;
```

Una consulta iterativa, sería la siguiente:

```
EXEC ctx_query.STORE_SQE('politicas', 'Q1','z%', 'session');
EXEC ctx_query.CONTAINS('politicas', 'SQE(Q1)| UNAM','ctx_temp');
```

- Consultas en bases de datos remotas

Para efectuar la consulta de dos pasos a una columna en una base de datos remota, se debe especificar la liga (*link*) dentro de los parámetros del procedimiento CONTAINS.

```
SELECT *
FROM text_tab@db1
WHERE CONTAINS (texto, 'politica') > 0;
```

En este ejemplo, de una consulta de un-paso, si la columna consultada está en la base remota, basta con indicarlo en el FROM. Se tiene una limitante en las consultas de un-paso, que no soportan consultar columnas remotas de tipo LONG o LONG RAW.

```
EXEC CTX_QUERY.CONTAINS ('MY_POL@DB1', 'politica,'CTX_TEMP');
```

En este ejemplo, de una consulta de dos-pasos, MY_POL y la tabla de resultados CTX_TEMP existen en una base de datos remota identificada por la liga DB1.

- Consultas paralelas

El procesamiento CONTAINS proporciona un argumento para procesar consultas de dos-pasos en paralelo. Este tipo de procesamiento de consultas ayuda a balancear la carga de los servidores *ConText* disminuyendo el tiempo de respuesta. El argumento PARALLEL se usa para especificar el

número de servidores. En el siguiente ejemplo, la consulta es procesada en paralelo por dos servidores *ConText* disponibles.

```
EXEC CTX_QUERY.CONTAINS ('Articulo', 'petroleo,'CTX_TEMP', parallel =>2);
```

- Operadores de conjuntos de recuperación

Permiten controlar qué documentos se deben recuperar de un conjunto de resultados de una consulta. No se pueden emplear de manera anidada.
THRESHOLD recupera aquellos documentos en el conjunto resultante cuya puntuación sea superior al valor definido. El operador **MAX** regresa una lista de éxitos de un tamaño menor que el especificado. El operador **FIRST/NEXT** habilita una lista incremental revelando una parte de la lista original.

Operador	*Sintaxis*	*Ejemplo*
THRESHOLD	expresion > *n* term > *n*	'SQL Server > 75' 'perro | gato > 50' '(perro > 30) and gato'
MAX	expresion: *n*	'SQL:10'
FIRST/NEXT	expresion # *m-n*	'perro#1-10' 'gato#11-20'

Tabla 4.7 Operadores de conjunto

Soporte de letras

ConText permite el tratamiento de letras con acentos como el equivalente de sus letras base o no-acentuadas; lo cual es una característica importante para muchas lenguas como el francés y el español. Se tiene la opción de convertir caracteres a su representación base antes de indexarlos. Esto significa que palabras con acentos, diéresis, etc. son convertidas a su representación base antes de que sus *tokens* sean indexados. Este proceso también se realiza previamente al momento de efectuar consultas y comparar los términos.

Lingüística de Context

Se usa para analizar el contenido de documentos en inglés. Permite tener diversas vistas del contenido de un documento, lo cual permite al usuario revisar rápidamente el contenido de los documentos y determinar su relevancia. *ConText* puede generar las siguientes formas lingüísticas:

Tipo de salida	*Descripción*
Tema	Principales conceptos de un documento.
Esencia genérica	Párrafo(s) en un documento que mejor representa(n) de lo que trata un documento.
Esencia de punto de vista	Párrafo(s) en un documento que mejor representa(n) en un documento lo que mejor representa un tema dado en el documento.

Tabla 4.8 Salidas lingüísticas

Lenguaje de manipulación automática de datos (DML)

Usando *triggers* de bases de datos. *ConText* reconoce automáticamente cualquier operador DML sobre una tabla que contenga documentos y podrá actualizar el texto indexado o el índice por tema. La indexación puede ocurrir inmediatamente en el tiempo especificado o solo cuando se defina de manera explícita.

4.4.8 Carga de Datos Textuales

Oracle permite almacenar muchos tipos de texto, desde el estándar VARCHAR2 hasta documentos de *Microsoft Word* y *Adobe Acrobat*. Almacenar estos tipos de datos en una columna de tipo texto no es difícil ya que se puede hacer con la sentencia INSERT del SQL o bien, por medio de la utilería *SQL*Loader* de quien a continuación, se tiene un breve ejemplo de su uso:

- Agregando una columna de tipo BLOB a la tabla *Recetas*.

```
SQL> ALTER TABLE Recetas ADD
        (Modo_Preparar BLOB DEFAULT Empty_BLOB());
```

- La columna *Modo_Preparar* almacenará archivos PDF de Adobe Acrobat que contienen las más recientes revisiones del autor. Empleando el SQL*Loader, se tiene a continuación que el archivo *recetas.dat* contiene los datos para la inserción en la tabla, así como la ruta completa donde se encuentran los archivos de las recetas.

```
1,Tarta de Queso con Zarzamora,/u12/oracle/admin/dev/documentos/receta_uno.pdf
2, Sopa Azteca,/u12/oracle/admin/dev/documentos/receta_dos.pdf
3, Flan al Whisky,/u12/oracle/admin/dev/documentos/receta_tres.pdf
4, Pollo al Curry,/u12/oracle/admin/dev/documentos/receta_cuatro.pdf
5, Carlota Francesa,/u12/oracle/admin/dev/documentos/receta_cinco.pdf
```

- Se define el archivo de control usado por *SQL*Loader*

```
LOAD DATA
INFILE 'recetas.dat'
APPEND INTO TABLE recetas
FIELDS TERMINATED BY ','
(clave, nombre, EXTERNAL_FILENAME FILLER,
Modo_Preparar LOBFILE(EXTERNAL_FILENAME)
TERMINATED BY EOF)
```

4.4.9 Técnicas empleadas

Aunque este es un tema confidencial, algunas veces se publica cuáles son las técnicas que emplean los productos comerciales, *Oracle* ha publicado en algunas referencias técnicas lo que ellos aplican:

- Las búsquedas estándar de SQL que incluyen las cláusulas LIKE y la comparación contra patrones fijos, usan métodos de acceso sobre *árboles-B*.
- La máquina de búsqueda de *Oracle* emplea el *Algoritmo de Salton* (capítulo III) para la asignación de puntos a los documentos [ScBr01]; además de un algoritmo de frecuencia inversa que no se permite publicar.

4.4.10 Las nuevas propuestas

Para la fecha de terminación de este trabajo, *Oracle* ya tiene dos últimas propuestas:
La primera de ellas es ***inter*Media**, que es una característica de *Oracle8i* para manejar tipos de datos complejos como lo son el texto, video, audio e imágenes. ***inter*Media Text** es la nueva generación de los servicios de *Oracle ConText*.
La segunda y más reciente se presentará en el *Oracle OpenWorld* en junio 20 del 2001 en Berlín, cuando Larry Ellison, fundador del corporativo, presente de manera formal *Oracle9i*, en el cual, se ha hablado de que los componentes de búsqueda de texto se renombraron como ***Oracle Text*** y no forman parte de los servicios de *interMedia*. La funcionalidad de indexación de texto está ahora integrada al servidor de la base de datos. A diferencia de la opción *Oracle Context*, *Oracle Text* no es un producto por separado que requiere de licencia, sino una característica del manejador de la base de datos.

4.4.11 Los Clientes

Oracle cuenta con varios clientes en México que usan el cartucho de *ConText*, algunos de ellos son:

- Liverpool (en la tienda virtual)
- Periódico UNIVERSAL
- Petróleos Mexicanos (PEMEX)
- Secretaria de Hacienda y Crédito Público (SHCP)

Uno de los clientes en Norteamérica, que se ha visto beneficiado por las capacidades de *ConText* es el Departamento de Policía de Chicago. El sistema CHRIS es un conjunto de aplicaciones 7x24 que soportan importantes funciones como: historias y registros criminales, casos de detectives, patrulleos, crimen organizado, registro de armas, funciones especiales, desastres, pistas de pandillas e infractores juveniles. La base de datos de 500 GB (300 tablas, 5 millones de renglones inicialmente) está en tres Servidores *Sun Enterprise 4000*, trabajando con *Oracle8.04* en modo paralelo con replicación y particionamiento. Se tiene una red con fibra óptica con 1800 PCs con Win95 y 1400 terminales móviles con Win95. Las aplicaciones se construyeron con *Oracle Developer*, *Oracle Designer*, *Oracle Data Browser* y *Oracle Data Query*. El número de usuarios es de 16,200 (800 simultáneos).
Los beneficios inmediatos del sistema fueron:

- La eliminación de papeleo excesivo.
- Integración de la información de justicia criminal.
- Proporcionar a los detectives herramientas de análisis.
- Búsqueda y captura de narrativas de aprensión muy eficientes.
- Se puede decir que se ayudó a 90 policías-día en sus tareas administrativas.
- Acceso al FBI para búsquedas de datos.

4.5 Otros criterios importantes

4.5.1 Criterios de administración

A lo largo de este capítulo, se pudo apreciar que con el uso de algunos métodos, se tiene una extensión sencilla del SQL, basado en el SFQL. Esto es una gran ventaja para los desarrolladores de aplicaciones cuya curva de aprendizaje, no es difícil de alcanzar en poco tiempo. Pero algo que no se debe olvidar, son las tareas del administrador de la base de datos (DBA) que aunque trabajar con datos tipo texto, no debería alterar su esquema de trabajo, sí requiere de atención especial, pues no sólo deberá respaldar la base de datos, sino todos los archivos a nivel sistema operativo que sean referenciados por las columnas de las tablas, además de conocer las tablas del diccionario de datos que involucren el manejo de texto.
También debe otorgar permisos, actualización de versiones, etc. Todo esto marca otro aspecto diferente a la simple administración de una base de datos relacional sin texto. Los espacios deben calcularse con mucho cuidado, datos ejemplo son: Usualmente los archivos índice requieren de más espacio. El tamaño del archivo índice es de 70% de la cantidad del texto indexado. El *DB2 Text Extender* requiere un 200% adicional del tamaño del archivo índice para espacio temporal de trabajo durante la indexación.
Es importante considerar aquí, que existe otra tarea importante: La afinación (*Tuning*), donde se debe conocer perfectamente la arquitectura de cada manejador para poder otorgar diferentes valores a los parámetros de inicio de la base de datos, espacios de memoria, espacios físicos, etc. y la fragmentación es un problema común que debe ser atendido.

Es preciso y así lo recomiendan los corporativos vendedores de ORDBMS, que se tenga una persona encargada de administrar los datos tipo texto.

4.5.2 Criterios de Interfaces

A lo largo de este capítulo, se mantuvo una perspectiva desde el punto de vista del desarrollo de aplicaciones, lo que se puede conseguir con programación. Así, un programador puede plantear las consultas en un lenguaje extendido del SQL; se pudo apreciar que si existen buenas soluciones para muchas necesidades de consulta pero es sumamente importante resaltar que un usuario final no plantea de ese modo sus búsquedas, sino que existe una interfaz entre él y el sistema de búsqueda y recuperación.

El planteamiento de la consulta, como se dijo en el capítulo II, requiere de una recuperación y navegación, en la que el usuario plantea búsquedas en su lenguaje natural y desde su perspectiva, por lo que las interfaces deben hacer transparente toda la teoría involucrada en la máquina de búsqueda y no exigir que se comprendan conceptos como la lógica booleana y mucho menos, la difusa, por mencionar algunos ejemplos.

Aunque en la actualidad se cuenta con ambientes gráficos (ventanas, íconos, menús, etc), podría pensarse que esto favorece al diseño de aplicaciones, pero la realidad es otra, ya que muchas veces no se obtiene lo que desea debido a algunos factores que intervienen, entre ellos: el manejo del lenguaje natural es complejo, sobre todo con aquellos idiomas tan ricos como lo es el español.
Otro factor es que muchas veces, no se aprovecha la herramienta al máximo debido a que los parámetros no son proporcionados debido a que no hay una interfaz que los pida y entonces se emplean los valores por omisión.

Se tienen muchos tipos de búsquedas, algo importante de analizar, diseñar y desarrollar las interfaces que permitan plantear tales tipos de consultas.
En el capítulo 10 de [BaRi00] se presentan a detalle las problemáticas, principios de diseño de interfaces y algunas evaluaciones. Aquí, el alcance del trabajo es dar la pauta para el desarrollo de esas aplicaciones idóneas que aprovechen la capacidad de las máquinas de búsqueda.

4.6 Una gran restricción

Se puede decir que hay grandes avances con los principales manejadores de bases de datos en el mercado actual pese a que las necesidades son muchas, las soluciones propuestas son buenas.
Sin embargo, algo importante que se puede decir de estas herramientas, es que pese a que el SQL se ha visto extendido en el SELECT y en el DDL (se permiten definir tablas con varios dominios de tipo texto e índices), no se tiene la posibilidad de efectuar alguna instrucción de DML empleando la función/paquete CONTAINS en la cláusula WHERE de un UPDATE, INSERT o DELETE. Esto sigue siendo una limitante que no debería presentarse, pues la manipulación de texto debería ser transparente sobre una base de datos relacional, ya que si no es así, pues entonces no es un producto 100% relacional como desde hace algunos años lo ha dicho E.F. Codd.

4.7 Otros productos

Aunque existen otros productos que proponen soluciones compatibles a las de las herramientas presentadas en el capítulo, se considera que los manejadores sobre los que trabajan, no compiten con los analizados anteriormente. Algunos de los primeros productos mencionados son:

- **TextDb** (*WebBesi*) — Accesa bases de datos *Access* y *Fox Pro*.
- **SQL Servers Full Text Index** *(Microsoft)* — Accesa bases de datos en *SQL Server*
- **Excalibur Technologies** y **Verity Inc.** — Aunque accesan diversos sistema manejadores de bases de datos, el hecho de que no se tenga una integración "completa", lleva a muchos problemas de administración y no se tiene una integridad en datos responsabilidad del manejador, pues son servidores independientes.

4.8 Comentarios de comparación

Como se dijo al inicio del capítulo, se presentaron las capacidades de algunos de los productos comerciales que apoyan a los ORDBMS en la búsqueda y el manejo de texto.

- ☆ Se puede concluir que todos los productos no funcionan de manera aislada, sino que se integran a su correspondiente motor de base de datos para la consulta sobre datos estructurados (con consultas clásicas) y sobre datos textuales no estructurados.
- ☆ La recuperación de textos depende completamente de los índices, con formatos totalmente diferentes a los construidos sobre los dominios estructurados.
- ☆ Todos permiten el uso de una lista de parada, la cual puede ser predefinida o definida por el usuario.
- ☆ Las búsquedas booleanas que se efectúan son aquellas en las que se utilizan los conectores lógicos AND, OR y NOT entre los términos buscados, pero con una importante diferencia respecto al modelo clásico puro, ya que se asignan pesos a los documentos que verifican la consulta (no solo indican si el documento satisface o no la consulta).
- ☆ Las búsquedas por pronunciación parecida dependen del idioma.
- ☆ Permiten la utilización de *thesaurus*, e incluso incorporan uno para el idioma inglés.
- ☆ *Oracle* no lo hace tan transparente para el desarrollador, pues lejos de tener una extensión simple del SQL, presenta la obligación de usar *PL/SQL* y ejecución de procedimientos empaquetados.
- ☆ *ConText* también permite búsquedas difusas, aunque no permite seleccionar los tipos de errores a considerar, sino que utiliza un módulo (diferente según el idioma de los textos) para decidir qué palabras tienen una ortografía similar al término buscado. Actualmente, incorpora módulos para el alemán, coreano, chino, español, francés, holandés, inglés, italiano y tailandés.

En las siguientes tablas, se hizo un resumen del panorama general de las capacidades de los productos, los datos hablan por sí mismos. Un factor para la elección de una de las herramientas sería analizar las necesidades particulares del usuario así como los costos del manejador de base de datos y la extensión de la máquina de búsqueda.

Idioma soportado	***Text Extender***	***Excalibur***	***Verity***	***ConText***
Alemán	✓		✓	✓
Árabe	✓			
Catalán	✓			
Chino (tradicional y simplificado)	✓			✓
Danés	✓			
Español	✓		✓	✓
Finlandés	✓			
Francés	✓		✓	✓
Franco Canadiense	✓			
Hebreo	✓			
Holandés	✓		✓	✓
Inglés Británico	✓			
Inglés Norteamericano	✓	✓*	✓	✓
Islandés	✓			
Islandés	✓			
Italiano	✓			✓
Japonés	✓			✓
Koreano				✓
Noruego	✓			
Portugués	✓			
Ruso	✓			
Sueco	✓		✓	
Suizo	✓			

Tabla 4.9 Idiomas soportados

✓* No se encontraron explícitamente otros idiomas.

Los códigos soportados en general son: *EBCDIC, ASCII, CCSID* (se extendió para documentos en Unicode), *DBCS* (doble byte character), *UNICODE, UCS2* para soportar lingüística arábiga e ISO-8859-1 que es el estándar de 8 bits.

- *Excalibur* permite construir un conjunto de caracteres propio.
- *Text Extender* permite emplear formatos no establecidos usando USER-EXITS.

Formato	*Text Extender*	*Excalibur*	*Verity*	*ConText*
Adobe Acrobat PDF		✓	✓	✓
AMI Pro	✓		✓	✓
Aplix Words 4.2			✓	
ASCII	✓	✓	✓	✓
Corel Word Perfect			✓	
Excel		✓	✓	
FFT	✓			
HTML	✓	✓	✓	✓
Lotus 1-2-3			✓	
Lotus AMI Pro			✓	
Lotus Word Pro			✓	
PowerPoint		✓	✓	
RTF	✓			
SGML			✓	
Word	✓	✓	✓	✓
WordPerfect	✓	✓	✓	
XML			✓	

Tabla 4.10 Formatos soportados

Tipo de Consulta	*Text Extender*	*Excalibur*	*Verity*	*ConText*
Difusa	✓	✓	✓	✓
Texto Libre	✓		✓	
Thesaurus	✓	✓	✓	✓
Término raíz	✓		✓	✓
Proximidad	✓	✓	✓	✓
Expresiones regulares	✓		✓	✓
Booleanas	✓	✓	✓	✓
Frase en general	✓	✓	✓	✓
Frase exacta	✓	✓		
Frase aproximada	✓	✓		
Por tema (Sólo en inglés)			✓	✓
Sonido similar	✓		✓	✓

Tabla 4.11 Tipos de consultas

Otras capacidades particulares:

ConText

- Consultas paralelas.
- Consultas a bases de datos remotas.
- Control del número de registros recuperados.
- Relación Maestro-Detalle en el almacenamiento.
- Creación de índices en paralelo.
- Capacidad de resaltar términos en documentos recuperados.

Verity

- Creación de índices fragmentados.
- Permite el almacenamiento de múltiples lenguajes en una columna.
- Control del número de registros recuperados.
- Permite que se obligue la diferenciación entre mayúsculas y minúsculas.
- Capacidad de resaltar términos en documentos recuperados.
- Consultas limitadas por párrafo.
- Consultas limitadas por enunciado o sentencia.

Conclusiones

Conclusiones

Hasta hace algunos años, parecía que las bases de datos eran la panacea para todos los problemas que existían con el manejo de grandes volúmenes de datos almacenados en archivos, sus beneficios convencieron a todo tipo de usuarios: Analistas, desarrolladores, administradores y sobre todo, a los usuarios finales que veían satisfechas sus necesidades de obtención de información a partir del almacenamiento de datos relevantes de sus organizaciones. Las grandes inversiones de capital en la adquisición del software manejador de bases de datos, se justificaban con los beneficios alcanzados. Sin embargo, las nuevas necesidades de información, no sólo se obtienen del procesamiento de datos tipo estructurado, sino también de los datos no estructurados como el texto, que representa un 90% de la información digital que se pretende administrar y consultar por medio de un manejador de bases de datos.
El texto puede contener información de misión crítica, pero es complejo debido a que aparece en diversos formatos y sobre todo, porque es la forma de representar un lenguaje natural. Las consultas sobre texto son una necesidad actual, más aún, con la popularidad del Web.
La solución para este tipo de consultas no es un problema trivial, requiere de la aplicación de las bases teóricas de la recuperación de información y su integración con el sistema manejador de bases de datos. Por un lado, la teoría de la recuperación de información define técnicas y algoritmos cada vez más complejos y eficientes; por otro, las bases de datos relacionales han llegado a una madurez plena junto con el SQL; por ello, es una buena alternativa pensar en explotar con todos estos medios al texto.
Muchas aplicaciones requieren de estas tecnologías, como por ejemplo: bases de datos de bibliotecas, librerías, páginas Web, reglamentos jurídicos institucionales, bases de datos biológicas, envío de correos electrónicos a un usuario específico dependiendo de su contenido, bases de datos biológicas (DNA), etc.

Es así, como en este trabajo se presentaron las principales técnicas de la recuperación de la información en texto y cómo se aplican conjuntamente con los principales sistemas manejadores de bases de datos (*Oracle*, *Informix* y *DB2*), obteniendo así, buenas propuestas comerciales para la recuperación y búsqueda no trivial de texto.
Se encontró que existen muchas semejanzas en las capacidades de búsqueda de estas herramientas como por ejemplo, las necesidades de almacenamiento del texto como un dominio de una relación, la creación de índices y la utilización del SQL en el planteamiento de las consultas, de las cuales, sobresalen las difusas por permitir una mayor flexibilidad semántica y no sólo una generalidad que recupera datos no deseados.
Respecto a los modelos clásicos, las búsquedas booleanas que permiten todos las herramientas, asignan pesos a los documentos para verificar la consulta y no sólo indican si el documento satisface o no la consulta.
En cuanto al manejo de lenguaje natural, se encontró que aún es limitado en cuanto a las búsquedas por temas, pues no todas las herramientas lo permiten y las que lo hacen, se limitan al idioma inglés que no es tan flexible ni posee un complejo desarrollo morfológico, tal como el español o el ruso.
Con relación a las tareas de administración que permiten el uso de estas herramientas, se puede decir que requieren de mucho detalle para poder disfrutar de sus bondades, pudiendo mencionar entre ellas: la definición de *thesaurus*, listas de parada e índices.
Respecto al SQL extendido que se aplica para la búsqueda de texto, se vio que en todos los productos, se carece de la posibilidad de efectuar alguna instrucción de UPDATE, INSERT o DELETE que incluya el CONTAINS en la cláusula WHERE. Esto, además de ser una necesidad de los usuarios no resuelta, muestra una falta importante del SQL-extendido para texto con relación al modelo relacional que considera las operaciones de manipulación de datos.

Si se tuviera que elegir entre una de las cuatro opciones presentadas, se consideraría como primera opción a *Verity Text Search DataBlade Module*, trabajando sobre IDS de *Informix*, ya que aplica una gran mayoría de los conceptos teóricos de manera sencilla para el usuario programador que es quien traduce a código SQL las necesidades de un usuario casual que interactúa con una interfaz. Pero realmente, una elección depende de muchos factores, entre ellos: Las capacidades del manejador de base de datos (requerimientos, capacidades de concurrencia, manejo en paralelo, tolerancia a fallas, técnicas de respaldo y recuperación, costo, etc), el tipo de búsquedas que se plantearán, el costo del producto (como extra del DBMS), la capacidad de integración con los productos con que se desarrollarán las aplicaciones, etc.

Puntualizando, entre las aportaciones de esta tesis se tienen:

- Se presenta un trabajo que resuelve muchas necesidades actuales en el campo de las bases de datos textuales.
- La unificación de dos temas muchas veces tratados y analizados por separado: Los sistemas de bases de datos y la recuperación de la información.
- Resaltar la complejidad del tratamiento del lenguaje natural en texto, sobre todo que dada la riqueza del idioma español, éste presenta muchas limitantes en los productos comerciales.
- Recopilación teórica que soporta a los productos comerciales.
- Comparaciones de cuatro productos comerciales poco conocidos.
- Presentación del SQL extendido para búsqueda en texto.
- Hacer conciencia en la necesidad de atender las interfaces que faciliten el planteamiento de las consultas del usuario para una mayor satisfacción a su búsqueda.

Dentro de las líneas de investigación que se pueden continuar a partir de este trabajo, se tienen:

- La construcción de buenas y mejores interfaces para el planteamiento de consultas.
- Tratamiento del lenguage natural, sobre todo del español, que al ser un idioma tan rico en sintaxis, semántica y morfología, se ha dejado de lado.
- Dar mayor transparencia a la administración de una base de datos con texto, no debería ser una carga para el DBA; sino una extensión a la capacidad del manejador.
- Mayor desarrollo del optimizador del manejador de bases de datos para las tareas con texto y proporcionar así, técnicas de afinación de aplicaciones y de ejecución de este tipo de bases de datos.

Anexo A

Tipos de datos para almacenar texto en los principales RDBMS

Tipo de dato	*RDBMS*	*Descripción*
CHAR	DB2	Tipo de dato caracter de longitud fija. La longitud máxima es de 254.
VARCHAR	DB2	Tipo de dato caracter dinámico. La longitud máxima es de 254.
LONG VARCHAR	DB2	Tipo de dato caracter de longitud variable. DB2 determina la longitud máxima de la columna.
GRAPHIC, VARGRAPHIC LONG VARGRAPHIC	DB2	Estos tipos especifican cadenas gráficas y es comparable a los tipos caracter de una cadena. Sin embargo, n especifica el número de caracteres doble-byte. Su valor máximo es de 127.
CLOB	DB2	Tipo de dato caracter para almacenar texto de hasta 2 GB.
		[IBMDB299], [IBM_W01]
CHAR	INFORMIX	Almacena cadenas de texto single-byte o multibyte con una longitud máxima de 32,767 bytes.
CHARACTER	INFORMIX	Es un sinónimo del tipo CHAR reconocido por ANSI.
CHARACTER VARYING	INFORMIX	Son sinónimos del tipo VARCHAR reconocido por ANSI.
LVARCHAR	INFORMIX	Almacena cadenas de longitud variable con una longitud máxima de 32 kilobytes.
NCHAR	INFORMIX	Almacena cadenas de texto single-byte o multibyte con una longitud máxima de 32,767 bytes. Los caracteres almacenados dependen del código con que se creó la base de datos.
NVARCHAR	INFORMIX	Almacena cadenas de texto single-byte o multibyte de longitud variable mayor a 255 bytes. Los caracteres almacenados dependen del código con que se creó la base de datos.
VARCHAR	INFORMIX	Almacena cadenas de texto single-byte o multibyte de longitud variable hasta de 255 bytes. Los caracteres almacenados dependen del código con que se creó la base de datos.
		[INF_W01], [INF_W02]

Tipos de datos para almacenar texto en los principales RDBMS

Tipo de dato	*RDBMS*	*Descripción*
TEXT	INFORMIX	Almacena datos de tipo texto sin acceso aleatorio por fragmentos.
CLOB	INFORMIX	Almacena datos de tipo texto con acceso aleatorio por fragmentos.
		[INF_W01], [INF_W02]
VARCHAR2	ORACLE8	Tipo de dato caracter de longitud variable, siendo su longitud máxima definida por el parámetro t. El tamaño mínimo es 1 y el máximo es 4000.
NVARCHAR2	ORACLE8	Tipo de dato caracter de longitud variable teniendo una longitud máxima en caracteres o bytes, dependiendo del código de caracteres nacionales utilizados. El tamaño máximo está determinado por el número de bytes requeridos para almacenar cada carácter en un límite máximo de 4000 bytes.
LONG	ORACLE8	Tipo de dato caracter de longitud variable hasta de 2 gigabytes, o $2^{31}-1$
CHAR	ORACLE8	Tipo de dato caracter de longitud fija, siendo su longitud de tamaño dado en bytes. El tamaño máximo es de 2000 bytes. Su tamaño mínimo y por omisión es de 1 byte.
NCHAR	ORACLE8	Tipo de dato caracter de longitud fija en bytes o caracteres, dependiendo del conjunto de caracteres nacionales utilizados. El tamaño máximo está determinado por el número de bytes requerido para almacenar cada carácter, con un límite de 2000 bytes. Su tamaño mínimo y por omisión es de 1 byte dependiendo del conjunto de caracteres utilizado.
CLOB	ORACLE8	Objeto caracter largo que contiene caracteres de un byte. No se soporta la variación de anchura de los caracteres de conjuntos. Su tamaño máximo es de 4 GB.
		[ORAC97]

Tipos de datos para almacenar texto en los principales RDBMS

Tipo de dato	*RDBMS*	*Descripción*
NCLOB	ORACLE8	Objeto caracter de longitud larga que contiene caracteres de anchura-fija en caracteres multibyte. No se soporta la variación de anchura de los caracteres de códigos. Su tamaño máximo es de 4 GB. Almacena datos de códigos de caracteres.
LOB	ORACLE8	Objeto binario largo. Su tamaño máximo es de 4 gigabytes.
LONGRAW	ORACLE8	Puede usarse para almacenar gráficas, sonido, DOCUMENTOS y arreglos de datos binarios; la interpretación depende de su uso.
BFILE	ORACLE8	Tipo de dato que contiene un localizador para un archivo binario largo almacenado fuera de la base de datos. Habilita el byte stream de acceso de entrada/salida (I/O) para acceso externo de LOBs que residen en el servidor de la bases de datos. Su tamaño depende del sistema operativo pero no debe exceder los 4 gigabytes.
		[ORAC97]
CHAR	SQL Server 7	Tipo de dato de longitud fija, con un máximo de 8,000 caracteres ANSI.
VARCHAR	SQL Server 7	Tipo de dato de longitud variable, con un máximo de 8,000 caracteres ANSI.
NCHAR	SQL Server 7	Tipo de dato de longitud fija con un máximo de 4,000 caracteres Unicode (los cuales requieren para su almacenamiento 2 bytes por caracter; incluye a todos los caracteres internacionales).
NVARCHAR	SQL Server 7	Tipo de dato de longitud variable, con un máximo de 4,000 caracteres Unicode.
		[Coff99]
CHAR	SYBASE	Tipo de dato de longitud fija, con un máximo de 255 bytes.
VARCHAR	SYBASE	Tipo de dato de longitud variable, con un máximo de 255 bytes.
TEXT	SYBASE	Tipo de dato para almacenar texto, hasta 2 Gb.
		[SYB_W01]

Tipos de datos para almacenar texto en los principales RDBMS

Tipo de dato	***RDBMS***	***Descripción***
NCHAR	SYBASE	Tipo de dato de longitud fija con un máximo de 4,000 caracteres
NVARCHAR	SYBASE	Tipo de dato de longitud variable con un máximo de 4,000 caracteres
		[SYB_W01]

Anexo B

a	about	above	across	after
again	against	all	almost	alone
along	already	also	although	always
among	an	and	another	any
anybody	anyone	anything	anywhere	are
area	areas	around	as	ask
asked	asking	asks	at	away
b	back	backed	backing	backs
be	because	became	become	becomes
been	before	began	behind	being
beings	best	better	between	big
both	but	by	c	came
can	cannot	case	cases	certain
certainly	clear	clearly	come	could
d	did	differ	different	differently
do	does	done	down	downed
downing	downs	during	e	each
early	either	end	ended	ending
ends	enough	even	evenly	ever
every	everybody	everyone	everything	everywhere
f	face	faces	fact	facts
far	felt	few	find	finds
first	for	four	from	full
fully	further	furthered	furthering	furthers
g	gave	general	generally	get
gets	give	given	gives	go
going	good	goods	got	great
greater	greatest	group	grouped	grouping
groups	h	had	has	have
having	he	her	herself	here
high	higher	highest	him	himself
his	how	however	i	if
important	in	interest	interested	interesting
interests	into	is	it	its
itself	j	just	k	keep
keeps	kind	knew	know	known
knows	l	large	largely	last
later	latest	least	less	let
lets	like	likely	long	longer
longest	m	made	make	making
man	many	may	me	member
members	men	might	more	most
mostly	mr	mrs	much	must

my	myself	n	necessary	need
needed	needing	needs	never	new
newer	newest	next	no	non
not	nobody	noone	nothing	now
nowhere	number	numbered	numbering	numbers
o	of	off	often	old
older	oldest	on	once	one
only	open	opened	opening	opens
or	order	ordered	ordering	orders
other	others	our	out	over
p	part	parted	parting	parts
per	perhaps	place	places	point
pointed	pointing	points	possible	present
presented	presenting	presents	problem	problems
put	puts	q	quite	r
rather	really	right	room	rooms
s	said	same	saw	say
says	second	seconds	see	seem
seemed	seeming	seems	sees	several
shall	she	should	show	showed
showing	shows	side	sides	since
small	smaller	smallest	so	some
somebody	someone	something	somewhere	state
states	still	such	sure	t
take	taken	than	that	the
their	them	then	there	therefore
these	they	thing	things	think
thinks	this	those	though	thought
thoughts	three	through	thus	to
today	together	too	took	toward
turn	turned	turning	turns	two
u	under	until	up	upon
us	use	uses	used	v
very	w	want	wanted	wanting
wants	was	way	ways	we
well	wells	went	were	what
when	where	whether	which	while
who	whole	whose	why	will
with	within	without	work	worked
working	works	would	x	y
year	years	yet	you	young
younger	youngest	your	yours	z

Anexo C

Adverbios

bien	entonces	bastante
mal	jamás	lejos
durante	mientras	hay
siempre	quizá	pronto
como	dentro	mañana
nunca	ahora	

Pronombres

Nominativos	**Dativos**	**Acusativos**
yo	mí	me
tú	tí	te
él	nos	lo
ella	les	los
nosotros		la
nosotras		las
ellos		
ellas		

Ablativo Compuesto	**Reflexivo**
conmigo	se
contigo	sí
consigo	

Pronombres Posesivos

mío	tuyo	su	nuestro
mía	tuya	sus	nuestros
míos	tuyos	suyo	nuestra
mis	tuyas	suya	nuestras
mi	tu	suyos	
	tus	suyas	

Pronombres Relativos

quien	que	cuyo	cuya
cuyos	cuyas	cual	cuales

Pronombres Relativos

Acompañados de un artículo:

el cual
la cual
los cuales
las cuales

Pronombres Demostrativos

éste	ése	aquél	ésa	eso
éstos	ésos	aquellos	aquella	
ésta	esto	éstas	aquello	

Pronombres Indefinidos

algo	alguien	otro	tal
todo	uno	cada	alguno
ninguno	cualquier	cualquiera	quienquiera
ambos	mengano	fulano	nada
nadie			

Artículos

el	la	los	las

Preposición

a	en	por	sin	para
con	entre	según	sobre	hacia
desde	hasta			

Conjunciones

e	y	o	u
que	aunque	pues	porque
como	pero	ni	cuando
si no	luego	cuanto	por que
mas	donde		

Bibliografía

Bibliografía

[AbCA00] Abbey, Michael; Corey, Michael J.; Abramson, Ian. "*Oracle8i. Guía de aprendizaje*". Osborne Mc Graw Hill. España. 2000.

[AdMC93] Adad, Rubén; Medina, Miguel Ángel; Careaga,Alfredo. "*Fundamentos de las Estructuras de Datos Relacionales*". Megabyte, LIMUSA. 1993

[AhUl98] Aho, Alfred V.; Ullman Jeffrey D. "*Compiladores. Principios, Técnicas y Herramientas*". Pearson Educación, 1998.

[Baez94] Baeza- Yates, Ricardo. "*Búsqueda de Información: Un Tour Algorítmico*". Quinta Escuela Internacional de Invierno en Temas Selectos de la Computación. Zacatecas, México. 24-28 octubre de 1994.

[Barb98] Barba, Jorge. *Seminario de Soporte Técnico Oracle. "Herramientas. ConText Cartridge"*. 2 de marzo de 1998. México, D.F.

[BaRi99] Baeza-Yates, Ricardo; Ribeiro-Neto, Berthier."*Modern Information Retrieval*". ACM Press, Addison Wesley. New York. 1999.

[BoGe99] Bolshakov, Igor A. & Gelbukh, Alexander F. *"Enriquecimiento de la base de combinaciones de palabras por medio de un tesauro jerárquico"*. Laboratorio de Lenguaje Natural. Centro de Investigación en Computación del IPN. Memorias del Congreso Internacional de computación 99. Editor: Pedro Galicia Galicia.

[Broo93] Brookshear, J. Glenn. "*Teoría de la Computación, Lenguajes Formales, Autómatas y Complejidad*". Addison Wesley Iberoamericana. 1993.

[NBBC91] Naylor, Thomas H. & Balintfy, Joseph L. & Burdick, Donald S. & Chu, Kong. "*Técnicas de simulación en computadoras*". LIMUSA. México. 1991

[Cama98] Camacho Cancino, Sara."*Análisis de Algoritmos*". Universidad Nacional Autónoma de México. ENEP Acatlán. 1998.

[Codd69] Codd, E.F. *"Derivability, Redundancy, and Consistency of Relations stored in Large Data Banks"*. IBM Research Report.RJ599. #12343. August 19, 1969.

[Codd70] Codd, E.F. *"A Relational Model of Data for Large Shared Data Banks"*. ACM 13, 6. pages: 377-387. June 1970.

[Codd72] Codd, E.F. "*Relational Completeness of Data Base Sublanguages*", Randall J. Rustin (ed.), Data Base Systems, Courant Computer Science Symposia Series 6. Englewood Cliffs, N.J. Prentice Hall. 1972.

[Codd82] Codd, E.F. *"A Relational Database: A practical foundation for Productivity"*. Communications of the ACM. Volume 25. Number 2. pages: 109-117. February 1982.

[Codd85] Codd, E.F. *"How Relational Is Your Database Management System?"*. Computerworld. October 14 and 21. 1985.

[Codd90] Codd, E.F. *" The Relational Model for Database Management "* Version 2. Addison Wesley Publishing Company.1990.

[CONA97] *Oracle8 ConText Cartridge. "Administator's Guide"*. Release 2.0 .Part No. A54628-01. Oracle Corp.June 1997.

[COND97] *Oracle8 ConText Cartridge. "Application Developer's Guide"*. Release 2.0 .Part No. A54630-01. Oracle Corp.June 1997.

[Coff99] Coffman, Gayle. *"SQL Server 7 , Manual de referencia"*. Mc Graw Hill. 1999

[Cohe86] Cohen, Daniel I. A. *"Introduction to Computer Theory"*. John Wiley & Sons, Inc. 1986.

[Date89] Date, Chris J. *" A Guide to the SQL Standard " 2nd* edition .Appendix F, An Annotated SQL Critique. Codd and Date International Addison Wesley.1989.

[Date90] Date, Chris J.*" Relational Databse Writings 1985-1989"*.Why Relational?. Addison Wesley. 1990.

[Date93] Date, Chris J.*" An Introduction to Database Systems " Vol. I* Addison Wesley. 1993.

[Date01] Date, Chris J. *"Introducción a los Sistemas de Bases de Datos"*. 7ª edición. Pearson Educación. 2001.

[EXCA99] *"Excalibur Text Search DataBlade Module. User's Guide"*. Version 1.2. Part. No. 000-5401. Informix Press. March 1999.

[FrBa92] Frakes, William B.; Baeza-Yates Ricardo. *"Information Retrieval. Data Structures & Algorithms"*. Prentice Hall PTR, Upper Saddle River, New Jersey 07458. 1992.

[GaBG99] Galicia-Haro, Sofía N., Bolshakov, I. A. & Gelbukh, A.F. *"Aplicación del formalismo de la Teoría Texto ⇔ Significado al análisis de textos en español, introduciendo análisis estadístico"*. Laboratorio de Lenguaje Natural. Centro de Investigación en Computación del IPN. Memorias del Congreso Internacional de computación 99. Editor: Pedro Galicia Galicia.

[GaGexx] Gardarin, Georges & Gelenbe, Erol. *"New Applications of Data Bases"*. Academic Press. Pags. 85-138, 185-244

[Gelb99] Gelbukh, Alexander F. *"Approximate Initial Substring Search Under Access Locality Requirements"*. Laboratorio de Lenguaje Natural. Centro de Investigación en Computación del IPN. Memorias del Congreso Internacional de computación 99. Editor: Pedro Galicia Galicia.

[Guzm98] Guzmán Arenas, Adolfo." *Clasitex +: Una herramienta para el análisis de Textos*".INFORME TECNICO. No.6 Serie: Verde Centro de Investigación en Computación.Subdirección de Vinculación. IPN. Junio 98.

[Guzm97] Guzmán Arenas, Adolfo. Artículo: *"Hallando los Temas Principales en un artículo en Español"*. Parte II. Centro de Investigación en Computación del IPN. Soluciones Avanzadas. Septiembre de 1997.No.5 Vol 47 (15 julio 97) pags. 58-63 No.5 Vol 49 (15 septiembre 97) pags.66-71

[Heap78] Heaps, H.S. "*Information Retrieval: Computational and Theoretical Aspects*". Academic Press. New York. 1978.

[HoUl79] Hopcroft, John E.; Ullman Jeffrey D. "*Introduction to Automata Theory, Languages, and Computation*". Addison Wesley Publishing Company. 1979.

[Jone88] Jones, Edward. "*Aplique el dBASE III plus*". Osborne / Mc Graw Hill.1988.

[Jone91] Jones, Susan. "*Text and Context: Document Processing and Storage*". Springer – Verlag. Pags. 1-79. 1991

[MaAS99] Makagonov, P., Alexandro M., Sboychakov K. *"Searching in full text Data Bases by using text patterns"*. Memorias del Congreso Internacional de computación 99. Centro de Investigación en Computación del IPN. Editor: Pedro Galicia Galicia.

[McGo94] Mc Goveran, David." *The Relational Model Turns 25 ... and we're still trying to get it right* " .DBMS Magazine. Pag. 46 –60.October 1994.

[MeSW86] Mendenhall, William & Scheaffer, Richard L. & Wackerly, Dennis D. " *Estadística Matemática con Aplicaciones*". Grupo Editorial Iberoamérica. México. 1986.

[Meye86] Meyer, Paul L. "*Probabilidad y Aplicaciones Estadísticas*". Addison Wesley Iberoamericana. México. 1986.

[Oliv97] Olivera Mendez, Rafael. *"La transición de Bases de Datos Relacionales a Bases de Datos Orientadas a Objetos"*. Ponencia del Oracle User Group en el Oracle OpenWorld 97. Ciudad de Mexico.1997.

[Orac92] Oracle7 Server. "*SQL Language Reference Manual*".Part No. 778-70-1292.Oracle Corporation. December 1992.

[Orac97] Oracle's Server Technologies Division.*"Oracle8 Server. On line Generic Documentation CD-ROM"*. Oracle Corporation. Oracle Parkway Redwood Shores, CA. USA.1997. Part No. A54647-01 .

[SaMc83] Salton,Gerard; McGill, Michael J. " *Introduction to Modern Information Retrieval*". McGraw Hill International Book Company. 1983.

[Sand85] Sanders, Donald H." *Informática: Presente y Futuro* ". Mc Graw Hill. 1985.

[Sand01] Sandsmark, Fred. *Oracle Set to Launch Oracle9i Database*. Oracle Magazine. Volume XV. Number3. May/June 2001. pages 27-34

[Sedg95] Sedge, Robert. "*Algoritmos en C++*". Addison Wesley, 1995.

[ScBr01] Scherer, Douglas & Carol, Brennan. *Oracle Text*. Oracle Magazine. Volume XV. Number 2. March/April 2001. pages 89 – 96.

[SiKS98] Silberschatz, Abraham; Korth, Henry F. y Sudarshan, S. " *Database System Concepts* ". Mc Graw Hill. 1998.

[TeST95] Teufel, Bernard; Schmidt, Stephanie; Teufel, Thomas. "*Compiladores. Conceptos Fundamentales*". Addison Wesley Iberoamericana. 1995.

[TEXT99] *"Text Extender: Administration and Programming"*. Version 6. First edition. Serie: SC26-9646-00. IBM Corp. June, 1999.

[vanR79] van Rijsbergen, C.J. "*Information Retrieval*". Butterworths & Co. Ltd., Second edition, 1979

[Vega94] Vega Hernández, Gerardo. "*Los Sistemas de Recuperación de Información por indización de triadas, una experiencia*". Investigación Bibliotecológica v.8. No.17 julio/diciembre 1994.

[Vega95] Vega Hernández, Gerardo. " *Técnicas de Búsqueda* ", Entrevista con Ricardo Baeza-Yates. Soluciones Avanzadas 20. Año 3. Número 20. pag. 55-57. México, 15 de abril de 1995.

[VERI00] *Verity Text Search Datablade Module.* " *User's Guide* ". Version 1.3. Part No. 000-8245. Informix Press. December 2000.

Direcciones electrónicas

[IBM_W01] *http://sasdocs.siu.edu/sashtml/accdb/z0267720.htm*

[INF_W01] *http://www.dbcenter.cise.ufl.edu/triggerman/InfoShelf/extend/01.fm1.html#109*

[INF_W02] *http://www.dbcenter.cise.ufl.edu/triggerman/InfoShelf/sqls/intro.fm1.html*
Informix Guide to SQL: Syntax, version 9.1
Copyright © 1998, Informix Software, Inc. All rights reserved.

[INF_W03] *http://www.informix.com/answers/english/docs/datablade/8245.pdf*

[INF_W04] *http://www.informix.com/answers/english/pdbm.htm*

[SYB_W01] *http://sasdocs.siu.edu/sashtml/accdb/z0439559.htm*

Información de Derechos Reservados