COLEGIO DE POSTGRADUADOS

INSTITUCIÓN DE ENSEÑANZA E INVESTIGACIÓN EN CIENCIAS AGRÍCOLAS

CAMPUS MONTECILLO

POSTGRADO DE SOCIOECONOMÍA, ESTADÍSTICA E INFORMÁTICA

CÓMPUTO APLICADO

"UN SISTEMA DE GESTIÓN DE PALABRAS CLAVES POR CONTEXTO PARA UN ACERVO DE DATOS"

NANCY HERNÁNDEZ NEGRETE

T E S I S

PRESENTADA COMO REQUISITO PARCIAL
PARA OBTENER EL GRADO DE:

M A E S T R A E N C I E N C I A S

MONTECILLO, TEXCOCO, EDO. DE MÉXICO

2010

La presente tesis, titulada: **"UN SISTEMA DE GESTIÓN DE PALABRAS CLAVES POR CONTEXTO PARA UN ACERVO DE DATOS"**, realizada por la alumna: **Nancy Hernández Negrete**, bajo la dirección del Consejo Particular indicado, ha sido aprobada por el mismo y aceptada como requisito parcial para obtener el grado de:

MAESTRA EN CIENCIAS

SOCIOECONOMÍA, ESTADÍSTICA E INFORMÁTICA

CÓMPUTO APLICADO

CONSEJO PARTICULAR

CONSEJERO: _____

Dr. Juan R. Bauer Mengelberg

ASESOR: _____

Dr. Paulino Pérez Rodríguez

ASESOR: _____

M.C. Margarita Cruz Millán

Montecillo, Texcoco, Edo. de México, Noviembre 2010.

AGRADECIMIENTOS

A Dios por prestarme la vida, por conservarme con salud y en especial por haberme guiado y cuidado a lo largo de mi vida.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) porque a través de su programa de becas para estudios de posgrado, fue posible que realizará con éxito y calidad mis estudios de Maestría en Ciencias.

Al Colegio de Postgraduados a través del Posgrado en Socioeconomía, Estadística e Informática – Cómputo Aplicado por haber confiado en mí y darme la oportunidad de realizar mis estudios de maestría en dicho posgrado. Así como por el apoyo tanto de los profesores como del personal administrativo.

A los miembros de mi Consejo Particular, por sus consejos, apoyo y comprensión en todo momento.

Un agradecimiento muy especial y sincero al Dr. Juan R. Bauer Mengelberg, por todo el tiempo dedicado al desarrollo de este proyecto, por sus sabios consejos y su paciencia. Por haber compartido conmigo este valioso e interesante proceso de aprendizaje, que me ayudo a enriquecer mi formación como profesional y como persona. Por su apoyo incondicional con respeto y admiración.

A lo largo de mi vida han sido muchas las personas que han contribuido a mi formación, deseo expresar mi mas sincero agradecimiento a familiares, amigos y profesores que con sus consejos y apoyo hacen que sea mejor cada día.

DEDICATORIA

A mis padres por ser el más grande ejemplo de amor, porque a lo largo de mi vida siempre han estado conmigo para apoyarme y guiarme para que sea una persona de bien, impulsándome siempre a seguir adelante y a no dejarme vencer aun cuando el camino parezca difícil. Porque su fortaleza y amor me han dado las bases que necesitaba para enfrentar los retos de la vida. Por ser las personas a quien mas admiro, porque gracias a ellos soy lo que soy. este logro también es suyo, los amo.

A mis hermanos por que siempre han estado conmigo y me han apoyado en todo momento, por el claro ejemplo de esfuerzo y dedicación.

Con mucho cariño, *Nancy*.

INDICE

INDICE DE FIGURAS

INDICE DE TABLAS

UN SISTEMA DE GESTIÓN DE PALABRAS CLAVES POR CONTEXTO PARA UN ACERVO DE DATOS

Nancy Hernández Negrete, M.C.
Colegio de Postgraduados, 2010

RESUMEN

El creciente volumen de datos que se usan en análisis de negocios y minería de datos ha ocasionado que se formularan y usaran tecnologías diferentes a las basadas exclusivamente en bases de datos relacionales. El uso de índices invertidos generalizados constituye un elemento fundamental de muchos productos que se ofrecen y desarrollan en la actualidad. El paquete que se describe proporciona precisamente una variación de tales índices. Sirve para indizar un conjunto cualquiera de elementos, tales como registros, pero no se circunscribe a hacerlo para una sola tabla, como es el caso en una base de datos.

Como una versión anterior del paquete no proporcionaba la eficiencia suficiente, especialmente en cuanto al número de operaciones de entrada-salida, pero también en cuanto al volumen de datos a almacenar en disco, se crearon e implementaron nuevos modos de gestionar las listas invertidas. En especial, en lugar de usar listas parciales y árboles B, se incluyeron nuevos modos de armar y almacenar los arreglos, y en especial, utilizar bitmaps – cadenas de bits – para las listas invertidas. Un ejemplo del uso del paquete por una aplicación se incluyó para aclarar los conceptos. Se describen los algoritmos para efectuar operaciones entre subconjuntos de números de registros resultantes de consultas formuladas con anterioridad.

Palabras clave: Árboles, Búsqueda y Recuperación de Información, Métodos de Indexación, Repositorios y Data warehouse.

A SYSTEM TO MANAGE KEYWORDS BY CONTEXT FOR A DATA COLLECTION

Nancy Hernández Negrete, M.C.
Colegio de Postgraduados, 2010

ABSTRACT

The increase in size of data collections used for business analytics and data mining has caused technologies which are not based exclusively on relational databases to be developed and used. The use of generalized inverted indices is one of the main elements of many products offered. The software product described here offers a variant of such indexes. It serves to index any set of elements such as records, but is not confined to a single table, as is the case in a database.

Since a previous version of the same product was not efficient enough, especially concerning input-output operations and the volume of data to be stored, new ways to store and manage inverted lists were created and implemented. Particularly, instead of using partial lists and B trees, new methods including the use of bitmpas were introduced to build and store the arrays corresponding to the inverted lists,

An example of an application that uses the software product was included to shed lights on the different concepts. Some of the algorithms used to perform the operations between subsets of the record numbers resulting from queries are described.

Keywords: Data warehouse and repository, indexing methods, information search and retrieval, trees.

CAPÍTULO 1. INTRODUCCIÓN

Durante la última década, diversos organismos y especialmente las empresas han visto que la información contenida en sus acervos de datos, combinados con las de otros, puede ser utilizada para obtener ventajas competitivas o aun ser imprescindible para ciertas funciones. Esto ha causado una explosión en el uso del técnicas tales como la Minería de datos y la Inteligencia y el Análisis de Negocios. El efecto de esta situación fue que los volúmenes de datos a procesar para obtener resultados ha crecido de un modo muy significativo: hoy hay colecciones de datos de muchos petabytes. La consecuencia de estos volúmenes fue que el manejo de datos usando bases de datos relacionales resultó sobrepasado: había que buscar alternativas. El problema fundamental se presentó por el número de accesos a unidades de almacenamiento, puesto que estas operaciones son lentas comparadas con las velocidades con las que las computadoras efectuan operaciones en memoria. De ese modo, además de aprovechar muchas investigaciones y tecnologías alternativas a los modelos relacionales, se ha producido una suerte de explosión en el número de productos de software (Bain T., 2010), (Conway D., 2010), (Diaz S., 2010), (Frost S., 2010), (IBM, 2010), (Infobright open source data warehousing, 2010), que utilizan diversos enfoques para proporcionar análisis de los datos y consultas formuladas con un acervo *enorme*, donde esto indica precisamente que las soluciones anteriores no proporcionaban una solución factible. Si se necesita un dato en un minuto y se lo obtiene en dos horas, el proceso que proporciona la información no satisfizo la restricción (la duración máxima tolerable).

En general, se puede decir que el uso de los datos para un fin específico se basa en un subconjunto de todos los datos. De ese modo, la primera etapa de un proceso será formar tal subconjunto. En ocasiones, este proceso se ha denominado el *subconjunteo (subsetting)* aunque ni el sustantivo (ni el verbo asociado, sub-conjuntear) existe. Sin embargo, su uso aclara mucho el concepto involucrado y se ha adoptado para la investigación que se describe.

Si la obtención de un subconjunto es el núcleo del problema, naturalmente los modos de reducir los tiempos de respuesta se han enfocado a su solución. Hay dos estrategias para lograr este propósito**:** o se aceleran los procesos de transferencia de datos o se utiliza alguna herramienta para no leer todos los datos. El primer enfoque está siendo investigado constantemente. Las velocidades de transmisión de datos se incrementan, pero no en forma suficiente para resolver el problema, puesto que todavía hay que leer *demasiados* datos, donde esto se refiere a que el número implicará duraciones intolerables. La duplicación de la velocidad de transferencia sería muy notoria en sus impactos, pero no sería suficiente para una situación en la que se requiere una reducción del orden de 100 veces menos.

Surgió una estrategia interesante para atacar este problema. El uso de lo que se ha denominado datawarehouse appliances, los pioneros en está estrategia son Netezza (Howard P., 2010). Consisten, esencialmente, en el uso de muchas computadoras especializadas en lectura (no hacen otra cosa). La consulta llega a una computadora, que podríamos llamar central, y esta reparte la tarea de leer los registros a N computadoras especiales, que constan exclusivamente de un procesador y una unidad de almacenamiento. Estas recuperan los datos solicitados, y los entregan a la computadora central, que los integra (junta) y procede a efectuar las operaciones solicitadas. Con esta estrategia se han logrado reducciones espectaculares de los tiempos de respuesta, pero a un cierto costo (especialmente de hardware). La otra estrategia consiste en usar artificios que permitan determinar cuáles datos se tendrán que leer, siempre con el objetivo de leer los menos posibles que de todos modos sean suficientes para un uso particular. Los índices de bases de datos han sido – y siguen siendo – los principales medios de lograr esto**:** en lugar de leer toda una tabla, se hace una selección basada en uno o más índices y solo se leen los registros resultantes de este proceso. Un ejemplo aclara este concepto: si alguien necesita un registro en particular, la presencia de un índice permite hacer esto en una sola operación, además del proceso que significa procesar el índice mismo. Del mismo modo, si

se necesita un subconjunto basado en elementos presentes en índices, se puede limitar el número de registros que se transfieren de disco a memoria.

La investigación que se describe se refiere precisamente a la situación en que se necesita un subconjunto de un acervo de datos para algún fin. Se describirá un producto de software denominado KBC (keyword by context) cuya función es precisamente la de indicar cuáles registros se tendrán que leer. De ese modo, como se verá, es una generalización del concepto de índice de una base de datos. La historia del desarrollo y evolución de este paquete será presentada después de la descripción de sus componentes principales, puesto que hubiera resultado difícil comprender las etapas sin conocer la terminología y los conceptos que describen. De ese modo, se ha organizado este trabajo de la siguiente manera. Se presentan algunos conceptos, tecnologías y métodos relacionados con bases de datos y almacenes de datos en el Capítulo 2. Para introducir el paquete KBC, en el Capítulo 3 se describe una aplicación típica del mismo para ilustrar lo que proporciona, es decir, para qué sirve. A continuación se dedica el capítulo 4 a la descripción de los elementos del paquete mismo, incluyendo las estructuras que se usan para almacenar las marcas, así como, un relato del desarrollo del KBC. Sin esto, no es posible entender el tema de esta investigación, que consistió en mejorar los elementos relacionados con el almacenamiento de los datos que se manejan, tanto en su dimensión (espacio en disco) como en el número de accesos a disco que implican los usos de los datos. En otras palabras, se trataba de crear o mejorar las estructuras de datos utilizadas para almacenar y usar los elementos de estos índices generalizados. El Capítulo 5 está dedicado a los tipos de listas de resultados, y el 6 a cómo se hacen las operaciones entre ellas. En el Capítulo 7 se incluyeron algunos comentarios técnicos que pudieran ser de interés. En el Capítulo 8 se presenta la descripción de algunas aplicaciones que indicaron o sugirieron funciones que debería proporcionar el paquete, para después finalizar con las conclusiones. En el apartado de Anexos, se describe de manera general el ejemplo utilizado para probar la versión actual del KBC, asi como los métodos principales para la generación de datos de prueba.

CAPÍTULO 2. MARCO TEÓRICO

Para la elaboración de la investigación que se describe, se utilizaron conocimientos y técnicas de diversos temas de la Ciencia de la Computación. Sólo se describen aquellos elementos que se utilizaron, para usarlos o compararlos con otros similares.

2.1 Bases de datos y su evolución

A través del tiempo, se han desarrollado diversos métodos y modelos que permiten almacenar y recuperar información de manera eficiente. Uno de los modelos más utilizados son las bases de datos relacionales postuladas por Codd E.F. (1970). La idea fundamental es el uso de "relaciones". Cada tabla está compuesta por registros (las hileras de una tabla), y campos (las columnas de una tabla). El lenguaje más habitual para construir las consultas a bases de datos relacionales es SQL, Structured Query Language o Lenguaje de Consultas Estructurado, el cual es un estándar implementado por los principales motores o sistemas de gestión de bases de datos relacionales.

Con el paso de los años, las necesidades de información tanto en cantidades como en tiempo respuesta al ejecutarse una consulta han ido creciendo de tal manera que un modelo relacional se ha hecho ineficiente. Para resolver dicho problema se han desarrollado diversos tipos de bases de datos, entre los que podemos mencionar: transaccionales, multidimensionales, orientadas a objetos, documentales, deductivas, distribuidas, entre otras.

En cada vez mayor medida, las empresas reúnen información de diversos sistemas de información operativos, con datos de otras fuentes (incluyendo las de otras empresas del sector) en un repositorio al que se denomina *data warehouse*, termino acuñado por Bill Inmon (2002), traducido como almacén de datos, o también, bodega de datos. Es una base de datos corporativa que se caracteriza

por integrar y depurar información de una o más fuentes distintas, para luego procesarla permitiendo su análisis desde infinidad de perspectivas y con grandes velocidades de respuesta. La creación de un almacén de datos representa en la mayoría de las ocasiones el primer paso, desde el punto de vista técnico, para implantar una solución completa y fiable de Inteligencia o Análisis de Negocios.

Naturalmente hay muchos modos de almacenar los datos de un almacén de datos. En particular, las diferencias entre ellos consisten en las estructuras que se usan para almacenar la información, entre los cuales, además de las más utilizadas, que son las bases de datos relacionales, se encuentran modelos de tablas en estrella, en copo de nieve, cubos relacionales, etc.

Cada uno de los tipos de bases de datos utiliza sus estructuras específicas para almacenar tanto los datos, como índices que sirven principalmente para acelerar procesos de consulta.

2.2 Índices comúnmente utilizados

De acuerdo con Rob P. y Coronel C. (2009), un índice es un arreglo ordenado utilizado para acceder lógicamente a las hileras de una tabla. Cada elemento del arreglo será un par (clave, posición) donde la clave es el valor del índice. El valor suele ser de un campo en particular, o una concatenación de campos. Sin embargo, algunas bases de datos permiten calcular índices, donde los valores se obtienen de una expresión o función. La posición es una identificación de la hilera involucrada, que pueden ser el número de fila o una referencia indirecta, por ejemplo, la clave de un índice agrupado (clustered). Aunque esta definición de índice se refiere a las tablas de una base de datos, puede ser aplicado a muchas colecciones de datos de otro tipo, siempre y cuando las claves apunten a un elemento de la colección. Por ejemplo, el índice de un libro apunta a una página o sección que contiene una clave particular.

Las estructuras de datos convenientes se utilizan para almacenar los valores de tal manera que se mantenga el orden de los elementos durante las actualizaciones. Las consideraciones que deben tenerse en cuenta al elegir la estructura de datos adecuada para un determinado índice son**:** el espacio necesario para almacenar todos sus valores, y el número de operaciones necesarias para actualizar o usar el índice. El tamaño del índice es importante, ya que por lo general se carga a memoria, además del espacio en disco.

Sin embargo, hay otro aspecto fundamental, especialmente cuando el índice se refiere a una colección de datos muy grandes**:** el número de operaciones de entrada-salida (E/S) que se necesitan para usar el índice, ya sea de carga total o parcial en memoria o para otros usos, es crucial, ya que las operaciones de E/S a menudo contribuyen de manera significativa al tiempo total de procesamiento requerido para una tarea en particular. Muchas estrategias se utilizan para reducir el número de operaciones de E/S, pero probablemente la que prevalece es el uso de paginación de algún tipo, donde los datos no se leen de forma individual, sino que se cargan en páginas de memoria. Shapiro J. (2006) define la paginación como el proceso de mover datos dentro y fuera de la memoria física, mientras que Null L. y Lobur J. (2006) lo interpretan como un método utilizados usualmente para la implementación de memoria virtual en donde la memoria principal es dividida en bloques de tamaño fijo y los programas son divididos en bloques del mismo tamaño (páginas).

De acuerdo con Rob P. y Coronel C. (2009) la mayoría de sistemas de gestión de base de datos implementan índices usando una de las siguientes estructuras de datos:

- **Función Hash:** se utiliza un algoritmo hash para crear un valor a partir del valor de una columna.
- **B-tree (lo llamaremos árbol-B):** es el tipo de estructura más común utilizado en bases de datos y será explicado a detalle mas adelante.

- **Bitmaps:** se utiliza principalmente en aplicaciones con grandes almacenes de datos (data warehouse) o en tablas con un gran número de filas en las que un pequeño número de valores en la columna se repite muchas veces. En Greenwald R., et. al. (2004), un índice implementado mediante un bitmap es aquel en donde cada bit del índice representa un ROWID. Si una hilera contiene un valor en particular, el bit que representa dicha hilera se "prende", es decir se pone en 1, en el bitmap para ese valor.

2.3 Árboles B (B-trees)

Weiss M. A. (1993) define un árbol - recursivamente - como un conjunto de n nodos, el cual puede estar vacío. De tal manera, un árbol consiste en un nodo distinguido R, llamado raíz, y cero o más subárboles, cada uno de ellos unidos por una arista a R. El árbol-B se definió por primera vez en Bayer R. y McCreight E. (1972) aunque hay una referencia precedente de los mismos autores en 1970, en la cual se establece: Sea $h \geq 0$, un número entero, y k un número natural. Un árbol dirigido T pertenece a la clase $T(k,h)$ de los árboles-B, si T está vacío $(h = 0)$ o tiene las siguientes propiedades:

i) Cada camino desde la raíz hasta cualquier hoja tiene la misma longitud h, también llamada altura de T, es decir, $h =$ número de nodos en el camino.

ii) Cada nodo, excepto la raíz y las hojas tiene por lo menos $k+1$ hijos. La raíz es una hoja que tiene al menos dos hijos.

iii) Cada nodo tiene a lo más $2k$ hijos.

Comer D., (1979) no fue el primero en definir un árbol B +, pero lo incluye como una descripción de las variaciones de los árboles-B utilizada especialmente para almacenar índices. En un árbol B +, todas las claves residen en las hojas. Los niveles superiores, los cuales están organizados como un árbol B, sólo consisten en un conjunto de índices (nodos), que permiten localizar las claves de

forma más rápida. La Figura 1 muestra la separación lógica de los índices y las claves en un árbol B+. Naturalmente, los nodos y las hojas pueden tener diferentes formatos o incluso diferentes tamaños. Las hojas son por lo general vinculadas entre sí de izquierda a derecha, como se muestra, para permitir un proceso secuencial de las hojas. La lista enlazada de las hojas se conoce como sequence set.

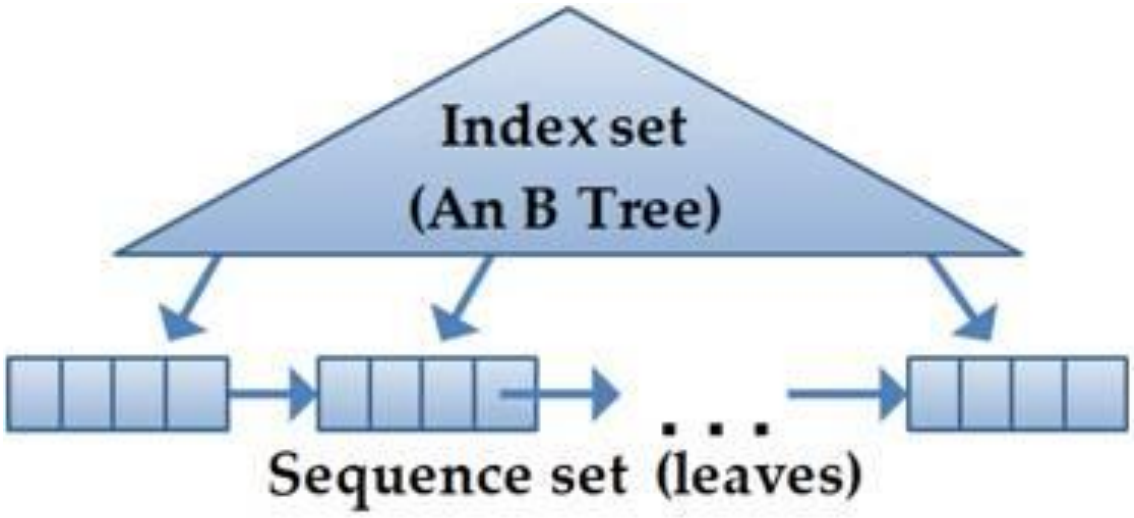


Figura 1. Un árbol B+

En este trabajo, como en la mayoría de la literatura relevante sobre índices, incluyendo libros de texto, los árboles B+ se denominan árboles B. Esto también se debe a que el KBC no usa árboles B en los cuales no todos las claves están en sus hojas, como es el caso en los B+.

2.3.1 Algoritmos para actualizar o usar un árbol-B

Se describen de manera simple los algoritmos necesarios para realizar búsquedas, inserciones y eliminaciones, de valores dentro de un árbol-B.

a) Algoritmo para buscar un valor dentro del árbol

Al realizar un proceso de búsqueda, si se trata de una actualización, se guarda en una pila el “camino” es decir los nodos que se leyeron para llegar a las hojas,

1. Iniciar en el nodo raíz
2. Dentro del nodo, recorrer las claves (se continua en el mismo nodo mientras el valor buscado sea mayor a la clave), cuando el valor buscado es menor a la clave, quiere decir que se encontró el siguiente apuntador SA.
3. Leer el elemento correspondiente a SA.
4. Si es un nodo repetir el proceso desde el paso 2.
5. Si es una hoja, buscar el elemento dentro de la hoja. El resultado es que lo encontró o no.
6. Si se trata de una actualización, hay que conocer la hoja en cuestión.

Observación: A pesar de que frecuentemente se usan algoritmos recursivos para esta actividad, no son convenientes desde ningún punto de vista práctico (solo simplifican la programación).

b) Agregar un elemento (valor) a un árbol-B

i. Busca el valor en el árbol. Esto resulta en:
   - ya estaba el valor (termina la función)
   - no estaba, pero ya se conoce la hoja en la que se debe agregar.

ii. Hacer una copia de esta hoja en una hoja temporal que es la que se usará para actualizar.

iii. Hacer un espacio en la posición adecuada.

iv. Insertar nuevo elemento.

v. Actualizar cuántas ranuras ocupadas.

vi. Si la hoja sobrepasó su capacidad, entonces:

vii. Dividir la hoja en 2 de tal manera que nos quedan: un nuevo apuntador (NA) y una nueva clave a insertar (NC) (que es el primer elemento de la hoja nueva).

viii. Los datos de NA y NC son enviados al nodo inmediato anterior (en la pila armada durante el recorrido inicial).

ix. Si el nodo aun tiene espacio se insertan NA y NC en la posición adecuada y el proceso de actualización termina.

x. De lo contrario, se crea un nodo temporal con una posición adicional (es decir, tendrá 5 claves, una de las cuales está vacía).
xi. Se agregan NC y NA en al posición “correcta” (para preservar el orden de las claves en el nodo).
xii. Se divide el nodo en 2. Se graban los nodos (el anterior y el nuevo), pero ahora hay que actualizar el “padre” del nodo afectado.
xiii. Se repite el proceso desde el paso vii, pero ahora se agregan NA = apuntador al nuevo nodo, y NC = la clave que no se incluye en ninguno de los dos nodos (es la clave “del medio" de los 5).
xiv. Si se llega a un punto en donde ya no hay un nodo con el cual combinar los valores de NA y NC, se crea un nuevo nodo raíz con la raíz anterior, NA y NC.

c) Algoritmo para eliminar un valor del árbol-B

Busca el valor en el árbol. Esto resulta en:

- no estaba el valor (termina la función)
- estaba, y se conoce la hoja en la que está

Se determina la posición que ocupa el valor en esa hoja. Después de eliminar ese valor, hay dos casos:

- La hoja permanece. Se quita el valor (a eliminar) de la lista de valores de la hoja. Si el valor era el primero de la hoja, se debe buscar en la pila el nodo que tenga dicho valor y actualizarlo con el “nuevo primer valor de la hoja”. Observe que para esto se recorre la pila en sentido inverso hasta encontrar el valor como el apuntado.
- La hoja se elimina (era el único valor de la hoja). Se busca en una hoja hermana (hija del mismo nodo) para repartir sus elementos. Si esto no es posible, es necesario recorrer el árbol (usando la pila guardada), de tal forma que se hace un reacomodo completo de los nodos. Al caer en este caso, hay muchas excepciones que se deben considerar por lo que no se describe a detalle el algoritmo.

2.3.2 Inserciones masivas en un árbol-B

Cuando ya tenemos un árbol-B armado y necesitamos hacer actualizaciones masivas, esto es, agregar “muchos” valores nuevos, resulta un tanto inconveniente (dependiendo del tamaño del árbol y del número de valores nuevos) hacer actualizaciones elemento por elemento, por lo cual desarrollamos un algoritmo que permite hacer el proceso de forma más eficiente.

Supongamos que la lista de nuevos valores llega como un arreglo clasificado en orden ascendente de los mismos. Hay dos casos generales:

a) Que todos los nuevos valores estén antes del primero o después del último elemento del árbol

Es necesario crear un nuevo subárbol-B. Para esto se siguen los siguientes pasos:

1. Calcular el número de hojas del subárbol: esto es
   cuantas_hojas = (cuantos-elementos-nuevos – 1) \ elementos por hoja)+1
   donde como siempre, la diagonal invertida “\” indica la división entera.
2. Armar todas las hojas con los nuevos valores.
3. Calcular cuantos nodos intermedios se necesitan para el ultimo nivel = (cuantas_hojas – 1) \ orden del árbol + 1.
4. Determinar la profundidad que tendrá el subárbol.
5. Para cada nivel del subárbol (de las hojas hacia arriba), se arman los nodos con los apuntadores correspondientes hasta llegar a la profundidad establecida.
6. Guardar la raíz del subárbol.
7. Se determina cuál de los árboles tiene mayor profundidad (el árbol original o el subárbol nuevo).
8. Agregar el de menor profundidad al otro, en el nivel correspondiente.

b) Inserciones intercaladas

Cuando los datos están intercalados hay 2 opciones. La selección del algoritmo a usar depende de la cantidad de instancias a agregar relativas al número que tenía el árbol antes de la operación. Si se agregan “pocos” valores (en este sentido) se agregan los valores uno por uno.

Si llega un número considerable de nuevos valores, se arma un árbol nuevo mezclando los datos nuevos y las instancias del árbol original o hacer inserciones una por una. Se usa el mismo proceso con el que se construye el subárbol descrito para la inserción masiva al principio o al final del árbol original. Es decir, se construyen todas las hojas, con una operación tipo intercalado de dos listas (la de las hojas, leídas una tras otra, y la lista de valores nuevos).

2.4 Índice GIN

La base de datos Adabas (Adabas Comprehensive Data Management System, 2010) fue el primer producto comercial que usó listas invertidas, pero el término de índice GIN no había sido definido, por lo que se presenta la definición establecida en PostgreSQL (The PostgreSQL Global Development Group, 2009): GIN significa (*Generalized Inverted Index)* un índice invertido generalizado. Se trata de una estructura de índice que almacena un conjunto de pares (*key*, *posting list*), donde un posting list es un conjunto o grupos de hileras en las que ocurre un key. Cada valor del índice puede contener muchas claves, así que el mismo identificador de la hilera puede aparecer en múltiples posting list. Internamente, un índice GIN contiene un índice construido con claves en forma de árbol-B, donde cada clave es un elemento del campo indexado y cada elemento dentro de una hoja tiene la clave y un apuntador que puede ser**:** a un árbol-B (PT, posting tree) o a una lista (PL, posting list), si la lista es suficientemente pequeña.

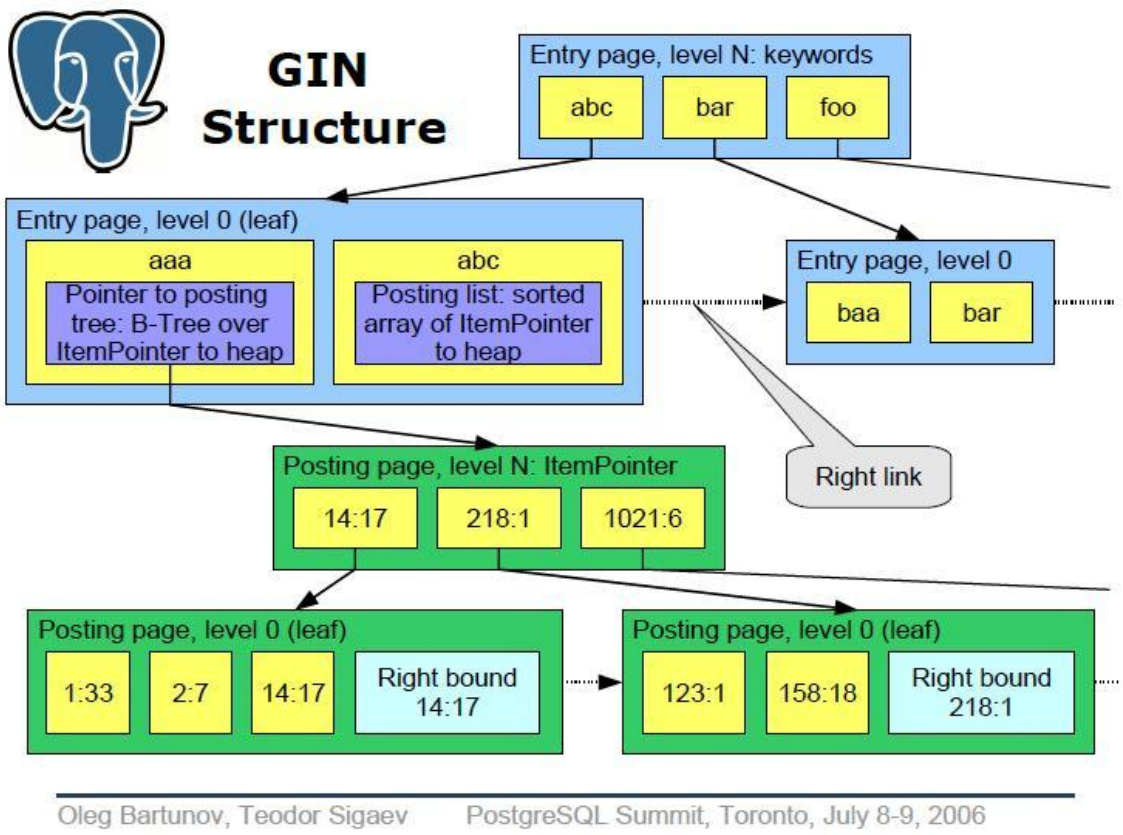


Figura 2. Un índice GIN

En la Figura 2 (Bartunov O. y Sigaev T., 2006.), se muestra un ejemplo de un índice GIN en donde las claves se almacenan con una estructura de árbol-B. En el caso ilustrado, las claves son aaa, abc, baa, bar y foo. En las hojas del árbol están todas las claves, pero además de la clave se almacena un apuntador: en el caso de la clave aaa, es un apuntador a un árbol-B, pero para la clave abc se guarda un apuntador a un arreglo ordenado de todas las hileras que están indizadas con esa clave.

En este trabajo, para evitar la confusión derivada de la utilización del mismo término para referirse al conjunto de elementos y a la estructura para almacenarla, vamos a utilizar el termino *arreglo* en lugar de lista, tal como se usa en el término posting list.

El uso de estas estructuras se ha generalizado, ya que son particularmente útiles para los índices en datos no estructurados. Por ejemplo, si en un acervo de datos de diversos tipos, están marcados los objetos que tienen la palabra clave “Rosa”, como se muestra en la Figura 3, al hacer una búsqueda en donde aparezcan todos los elementos que tengan dicha palabra clave, la consulta devolverá elementos que pueden no servir para el propósito de la búsqueda.

Para evitar esta situación, se agrega una etiqueta especifica a las marcas: por ejemplo en el caso del objeto 1, etiqueta=flor y en el objeto 2 etiqueta=apellido, de tal manera que las búsquedas se pueden limitar usando estas etiquetas para aumentar su eficacia. Esto es lo que actualmente se denomina semantic tagging, proceso que se usa principalmente para indizar datos no estructurados, lo que aquí significa que no tienen una estructura específica, como la tiene un elemento de datos cuyo significado o interpretación está dado por su posición dentro del registro, ya sea por el campo o por su ubicación física. Al proveer a un valor de una etiqueta, la interpretación se conoce a pesar de que era desconocida antes de hacerlo. Nos referimos a este tipo de registro como semi-estructurados. Esto quiere decir que se almacenan datos de diferentes tipos y no todos los datos son etiquetados de la misma manera.

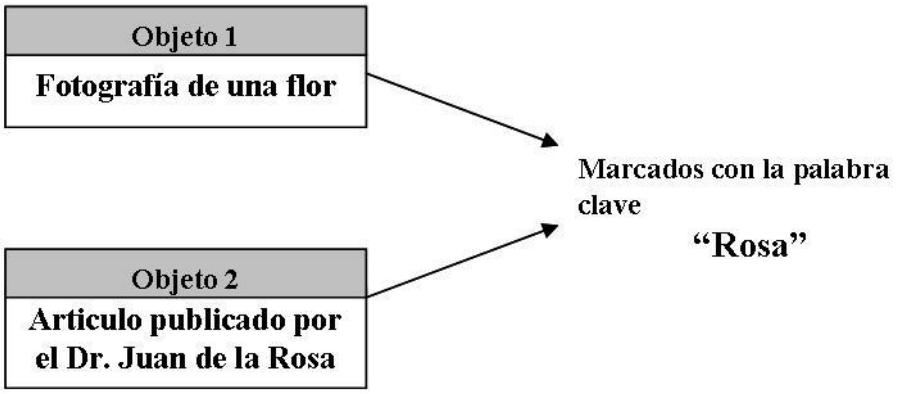


Figura 3. Ejemplo de marcado de palabras

2.5 El uso de bitmaps

Como se indicó, los bitmaps son cada vez comunes en aplicaciones relacionadas con el tema de índices. Diferentes productos pueden utilizar sus propias maneras de almacenar y utilizar un bitmap. En particular, KBC, el producto que se describe en esta tesis, que también utiliza los bitmaps para almacenar el equivalente a una posting list (llamada la lista de instancias de un valor de un contexto) implementa los bitmaps como se describe a continuación.

Los bitmaps están representados por arreglos de enteros de 4 bytes. Los bits se numeran en cada entero: el bit 0 es el bit menos significativo del entero, es decir, el que representa a 2^0, y el bit 31 es el bit del signo. Como se muestra en la Figura 4.

Bit	**31**	**30**	**.**	**.**	**.**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Valor del bit	1	0	.	.	.	0	0	1	1	1	0	1	0
+ significativo													**- significativo**

Figura 4. Un bitmap

Esto indica que cada entero en realidad está representando 32 posibles números de instancias, en donde si la instancia está presente en el valor correspondiente a dicho bitmap, el bit está encendido (tiene valor 1), de lo contrario está apagado. Para un bitmap se especifica el número de instancia representado por el primer bit, ya que éste determina las instancias presentes en el índice como se ve en la siguiente Figura 5, en la que se muestran dos bitmaps, uno correspondiente a las instancias 33 a 64 y el otro a las instancias 97 a 128.

Bit	**31**	**30**	**.**	**.**	**.**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Valor del bit	1	0	.	.	.	0	0	1	1	1	0	1	0
Instancias presentes si el bitmap inicia en **33**	44							38	37	36		34	
Instancias presentes si el bitmap inicia en **97**	128							102	101	100		98	

Figura 5. Ejemplo de dos bitmaps

La razón para adoptar esta estrategia en lugar de utilizar arreglos de enteros de 1 byte y sus 8 bits, como se hace frecuentemente, es que la cantidad de operaciones para procesar una lista se puede reducir considerablemente. Por ejemplo, al construir una lista de resultados, un valor de 0 de un entero permitirá saltarse 32 "posiciones" representadas por el número entero, en lugar de sólo 8 posiciones que resultarían de la utilización de enteros de un byte. Análogamente, se obtienen ahorros similares al realizar operaciones lógicas entre bitmaps o durante la conversión de un bitmap a un arreglo de instancias.

Los algoritmos utilizados para encender o apagar los bits o para determinar si está encendido, usan un arreglo de 32 constantes. Sus valores son los valores de los números enteros en los que sólo el bit indicado por la posición del arreglo está encendido. Llamamos al arreglo Only_bit_set (0-31) y se inicializa la siguiente manera:

Only_bit_set (I) = 2^I desde I = 0 hasta 30
Only_bit_set (31) = -2^{31}

Ejemplos del uso de este valor son:

- Si 2235 AND Only_bit_set(17) IS NOT = 0 determina si el valor del bit 17 (dentro del valor 2235) está encendido.
- 2235 OR Only_bit_set(17) encenderá el bit 17 (o permanecerá encendido)

Un arreglo similar de constantes con los complementos, es el que maneja los números en donde todos los bits están encendidos excepto el bit correspondiente al índice dentro del arreglo. Este arreglo se inicializa con las siguientes expresiones:

Only_bit_off (I) = - Only_bit_set (17) – 1 desde I = 0 hasta 30
Only_bit_off (31) = $2^{31} - 1$

De esta manera la operación 2235 AND Only_bit_off(17) apagará el bit 17 del numero. Una operación usada frecuentemente para determinar si todos los bits de un numero están encendidos, es comparándolo NUM=-1**.** Del mismo modo si NUM=0 quiere decir que todos los bits están apagados.

CAPÍTULO 3. INTRODUCCIÓN AL KBC

3.1 Introducción

El paquete KBC es un conjunto de métodos que se ofrecen a una aplicacíón. Su función es la de almacenar ternas de (contexto, valor, instancia) llamadas marcas, donde sólo la aplicación conoce el signficado concreto de los 3 elementos.

Una aplicación que se apoye en su funcionalidad requiere que el KBC arme subconjuntos de sus unidades de información (que en KBC se llaman instancias, pero en la aplicación pueden ser registros, fichas, objetos, etc). La condición fundamental para poder usar el KBC es que cada unidad de información tenga un número entero, distinto a los de todas las demás unidades.

Los subconjuntos se solicitan indicando la presencia de ciertas palabras contextuales (pares de valores contexto-valor). El KBC devolverá una lista de los números de las instancias que "contienen" las marcas indicadas como parte de la solicitud.

Las aplicaciones no conocen los detalles del KBC, ni dónde almacena los datos de todo tipo. Hacen una instancia de la CLASE_KBC, que se usa invocando una DLL. Como parámetro le pasa el directorio en el cual desea que se incluyan los archivos del KBC. En este directorio debe haber un subdirectorio que se llama KBCfiles. KBC prepara sus archivos a medida que los necesita, pero esto es transparente a la aplicación.

La aplicación construye una lista de contextos según lo que necesita. Es decir, representan la interpretación o el contexo en el cual se usaron los valores de los datos. Decimos que estos contextos constituyen una ontologia, donde aplicamos una definición relativamente limitada de este término: una ontología es un conjunto de definiciones de conceptos. Además de los datos (atributos) de los

contextos que use la propia aplicación, por ejemplo una descripción del contexto y cómo querrá usarlo, le tiene que informar a KBC dos elementos fundamentales y uno optativo: el número del contexto, el atributo o tipo de contexto y, optativamente, el tipo de estructura que usa el contexto o una estimación de cuantos valores tendrá el contexto y cuántas instancias habrá para los valores de ese contexto. Este último dato será una estimación del número de instancias de un valor “promedio”. KBC le informa a la aplicación el tipo de estructuras que usa para ese contexto, para que en los usos subsecuentes no tenga necesidad de buscar el dato en otro lado.

Los atributos de un contexto son:

Atributo	Significado o consecuencia
1	Normal: no tiene restricciones, es decir, cualquier VALOR que le llegue a KBC será válido.
2	Único (corresponde a un índice único). Cada valor tiene exactamente una instancia.
3	Tipo catálogo: no aceptará valores si no hay una instancia previa de dicho valor.
4	Es un tipo 3, excepto que se podrán introducir valores sin instancia (preparación del catalogo de valores).

Tabla 1. Atributos de un contexto

El marcado equivale a incluir palabras claves y descriptores del elemento de datos, sea un registro u otro tipo de elemento. Por ejemplo, se pueden describir archivos, objetos, bases de datos (cada base seria una unidad de información), etc.

Se proporciona un ejemplo de una aplicación para aclarar o presentar algunos elementos de KBC.

3.2 Ejemplo de un sistema que utiliza el KBC

El ejemplo seleccionado para ilustrar el uso del KBC, es un producto de software denominado SPRP, que implementa a los U-Books (Unstructured Book) introducidos por (Bauer Mengelberg J. R., 2007) como una colección de objetos (denominados ítems del libro), que pueden ser incluidos en sucesiones –conjuntos ordenados de ítems del libro- para proporcionar lecturas del libro. Los ítems pueden ser textos, imágenes, presentaciones, video-clips, referencias a páginas web u otros objetos. De este modo, el U-Book permite a sus autores, editores y lectores construir sucesiones o leer las de otros.

El historiador Stephen Sussna había reunido, a lo largo de casi 60 años, una gran cantidad de materiales relacionados con un evento histórico (Sussna S., 2008). Algunos de estos materiales ya residían en archivos en disco, mientras que muchos otros estaban siendo digitalizados. Se le sugirió que usara un U-book, puesto que la publicación de toda la colección era imposible, e incluir solo parte del mismo era desaprovechar mucho material juntado con mucho esfuerzo pero especialmente, una enorme dedicación y cariño por el tema. De hecho, apareció en forma de libro unos años más tarde, pero con un subconjunto de los materiales que tenía su autor.

A pesar de que la oferta fue rechazada, este libro no escrito de este modo inspiró muchas de las funciones que se agregaron o modificaron en SPRP. Por ejemplo, que el autor pudiera elaborar muchas sucesiones, lo que resultaría en que ofrecería a sus lectores la oportunidad de seleccionar la que más les gustaba o convenía para cierto propósito. Adicionalmente, los lectores también tendrían la oportunidad de elaborar sucesiones, ya sea modificando alguna existente o creando una nueva. Naturalmente hay muchas otras funciones de estos libros, pero no usan el KBC; esta componente ayuda a construir sucesiones proporcionando subconjuntos de los items con ciertas propiedades, aunque también permite ubicar un item de una colección muy numerosa.

Para su repositorio de ítems, su U-book, podría haber incluido, entre otros, los siguientes contextos:

1 = el año
2 = evento
3 = tipo de evento
4 = tipo de documento
5 = que muestra
6 = que muestra el mapa
7 = personas involucradas
8 = fuente de la información
9 = idioma
10 = autor.

De ese modo, los items se describen proporcionando palabras clave, pero estas se interpretarán en conjunción con el contexto en el que fueron indicadas. En el caso del historiador mencionado, esto hubiera constituido una tarea considerable, pero SPRP incluye varios módulos que le hubieran asistido a hacerlo con menos esfuerzo.

Los descriptores apropiados difieren de un tipo de ítem a otro. Naturalmente, el modelo de datos para contener los ítems debe contemplar esta circunstancia. Se adoptó un modelo basado en clases para plantear el problema. Se definió una clase básica de items, con algunos atributos comunes a todos los tipos. Los 2 campos fundamentales son: el número (único) asignado a cada uno, y un apuntador al archivo que tiene el contenido del item (el texto, la imagen, etc.). Los archivos con los contenidos se crean por separado; en otras palabras, alguien tiene que “escribir” el libro. Otros atributos comunes a todos los tipos incluyen: el tipo de objeto, una descripción (que puede ser un título u otra cosa), una fecha, que pudiera tener diferentes interpretaciones para diversos tipos, y probablemente algunos datos técnicos, especialmente el “extent” del archivo, necesario para mostrarlo al lector durante una sesión de lectura.

Sin embargo, los usuarios podrían necesitar información adicional sobre los ítems. Para un affidávit, por ejemplo, se podrían indicar el tema del asunto, el idioma, quién lo firmó y la fuente de la cual el autor lo obtuvo. Para la descripción de fotografías, por otro lado, se usarían otros atributos, tales como quién o qué está representado en la imagen, y la ocasión en la que se tomó la fotografía.

La solución conceptual más sencilla para esta situación es la creación de subclases de la clase básica para cada tipo de ítem. Sin embargo, la implementación de estas subclases presenta algunas dificultades, especialmente si se toma en cuenta que surgirán nuevos tipos de items. Hay que recordar que el SPRP no implementa solamente el libro del historiador, sino cualquier otra colección de datos donde el concepto del U-book puede ser de utilidad.

Una solución consiste en crear diversas tablas, con diferentes campos – los contextos – para tipos de ítems en particular. Sin embargo, la inclusión de un nuevo tipo de ítem implicaría la creación de otra tabla, es decir, un cambio al modelo de datos.

Otra desventaja de esta solución es que para cada contexto, se podría proporcionar un solo valor. Para poder registrar 2 o más autores de un libro, por ejemplo, se tendrían que incluir otros campos, o tablas para almacenar los “valores de un campo” – es decir, resolver la relación uno-a-muchos. En muchos casos, la imposibilidad de asignar a un ítem más de un valor a un contexto es una de las circunstancias que hacen que un modelo relacional no resuelva el problema.

Sin embargo, hay una desventaja mucho mayor: en una consulta que involucre a ítems de distintos tipos se tendrían que especificar todas las tablas necesarias como parte del comando SQL. Por ejemplo, si se quisieran obtener todos los ítems que tuvieran el par (idioma, francés), cada una de las tablas que tuvieran este atributo tendrían que ser incluidas individualmente en la consulta. Esto prácticamente descarta el uso de un modelo relacional para el SPRP, y

buscar otro tipo de estructuras de datos que no presenten esta desventaja.

Se formuló un modelo distinto, pero también basado en el modelo relacional, usando un RDBMS. Se agrega una tabla al modelo, con los campos: número de ítem, contexto y palabra clave. Su índice principal (único) es la concatenación de contexto-palabra clave-número de ítem, para ser de utilidad en consultas basadas en sus primeros dos elementos. Sin embargo, se descartó porque presenta algunas situaciones desfavorables, especialmente que las consultas formuladas resultarían bastante complicadas. La solución, sin embargo, es muy superior a la mencionada anteriormente, consistente en la inclusión de tablas individuales por tipo de ítem.

Este modo de almacenar información, donde se guardan pares contexto / valor (en la literatura se los llama Key/Value) se ha convertido en el más utilizado para situaciones similares a las que presentan las aplicaciones del KBC. Quizá el ejemplo más espectacular por su impacto, tamaño y divulgación sea la famosa Big Table (Big Table, 2010), donde hileras de la misma tabla pueden tener campos distintos entre sí.

Puesto que KBC proporciona precisamente las funciones necesarias para almacenar los datos de modo “utilizable”, sin tener que agregar tablas a la base de datos, se incluyó en SPRP una componente denominada RISP (Relate Items in Structured Publishing), que a su vez se apoya en KBC. RISP permite agregar descriptores a un ítem, y KBC hará las funciones que en una base de datos desempeñan los índices, aunque, como se verá, de un modo distinto y con mucho mayor funcionalidad. RISP ofrece las interfases para agregar a un item las palabras claves, siempre en uno de los contextos definidos para ese U-book.

SPRP, como parte de la componente RISP, creará una instancia de KBC, y le informará los contextos que usa. KBC a su vez prepara lo que necesita para aceptar y usar las marcas que le llegarán: a medida que se agregan ítems al libro,

y se indican marcas, éstas se envían a KBC y éste las almacena. Supongamos que el ítem #342 es una fotografía del Capitán XX, que el autor fue la Sra. YY, tomada en mayo de 1944 en un barco denominado Barcobandera. RISP almacena estas palabras clave en un arreglo que no forma parte de la base de datos, estos arreglos son de dimensión variable, puesto que puede haber ítems que tienen pocos descriptores, y otros que requieren un mayor numero de estos. Por último, los campos de la tabla ITEMS de la base de datos se pueden “marcar en forma automática”: esto corresponde a la definición de un índice para cada campo, excepto que ahora los maneja KBC y no el RDBMS. Una característica fundamental de esto es que un campo de la tabla puede tener más de un valor. Un valor se almacenaría en el campo de la tabla; los demás se agregarían como palabras clave. En el ítem #342, por ejemplo, para el contexto 5 (“qué se muestra”) se incluirían las marcas para (5, Capitán), (5, XX), (5, barco), (5, Barcobandera).

Como se ha mencionado, en un U-book, los elementos de una sucesión se presentarán al lector uno tras otro. SPRP ofrece diversos modos de construir tales sucesiones, entre las cuales se encuentra precisamente el hacer uso del RISP para aprovechar las “marcas” del ítem. Se formula una consulta a KBC, y se obtiene una “lista de resultados” que es una lista de los números de ítems que satisfacen la consulta, que se envía como una fórmula en la que se indican operaciones booleanas entre varios operandos. Los operandos pueden ser: un par (contexto-valor) u listas de resultados obtenidas previamente.

Para una conferencia sobre un tema relacionado con su investigación, el historiador podría construir una sucesión del siguiente modo: solicita una lista de todos los ítems del U-book que “tienen”: Año = 1945, evento = “SegundaGuerra”, (3 =”invasión” y “costa” y “francesa”), (4 = ”foto” o “mapa” o “affidávit” o “descripción) donde las comas indican intersecciones (es decir, conjunciones). Se da cuenta que este show durará 3 horas, y decide reducirlo eliminando todo el material en el que aparece él mismo. Invoca otra consulta: (la lista previa) AND NOT (personas involucradas = “John” y “Smith”). Puesto que aún es demasiado

larga la presentación, agrega (lista previa) AND NOT (4 = descripción y NOT 9 = ingles) con lo que elimina las descripciones que están en otro idioma.

Naturalmente SPRP ofrece modos sencillos para armar tales fórmulas y para utilizar las listas de resultados que obtiene como respuesta a las consultas. KBC recibe la fórmula como una lista de operandos, que le llega en forma de un arreglo, y la fórmula misma en notación postfija o con la inclusión de los paréntesis necesarios (en este caso, la convierte a notación postfija antes de ejecutarla). Se hacen las operaciones, no sin antes construir la lista de resultados de cada uno de los operandos. KBC devuelva la lista en un archivo en disco. El usuario usa la lista de resultados como desee, en particular, para armar una sucesión, a la que SPRP le permite agregar elementos, cambiarlos de orden o combinarlos con otras listas.

CAPÍTULO 4. DESCRIPCIÓN DEL KBC

El KBC es un programa que almacena marcas (palabras clave por contexto) enviadas por una aplicación. Le llegan como ternas: (contexto, valor, instancia). El KBC no conoce el significado que tienen estos elementos en la aplicación. La “instancia” es el número de registro u otro número de unidad de información (una instancia de los elementos de datos de la aplicación).

También se encarga de armar subconjuntos de elementos de la colección indizada, misma que le llegan como “consultas”. Estas son de dos tipos:

- Armar la lista de las instancias marcadas con un par (contexto-valor). Las llama lista de resultados.
- Ejecutar una “formula” que indica ciertas operaciones entre listas de resultados o marcas.

4.1 Cómo se usa el KBC

El KBC se ofrece como una DLL, que consiste de una clase con los métodos "Públicos" (los que puede usar la aplicación) y otros elementos de programación propios: módulos y módulos de clase. El KBC no usa formas, puesto que no tiene elementos interactivos. Un usuario no usa KBC, sino usa una aplicación que a su vez se apoya en KBC: una analogía evidente es la de un índice de una tabla de una base de datos. El usuario no usa – directamente - el indice, pero sí la tabla.

4.2 Las funciones del KBC ofrecidas a sus aplicaciones

Los tres métodos fundamentales de KBC, implementados como funciones, se describen a continuación. Se muestran los parámetros y aquellas explicaciones que se consideraron imprescindibles para aclarar los parámetros mismos y la naturaleza de la función. Se utilizó una notación tipo Visual Basic. *Integer* designa

a un entero de 2 bytes, mientras que *long* designa uno de 4 bytes.

4.2.1 Actualizar una marca

Public **function actualiza una marca** (operación as string, contexto as integer, valor as string, instancia as long) as long

Parámetros:

operación A=agregar, DE=delete (eliminar), V=verificar si está
contexto (el número de contexto)
valor (la palabra clave, máximo 16 caracteres)
el número de instancia (como entero de 4 bytes.)

Regresa (resultado de la función):
Éxito:

El NUEVO número de instancias del valor solicitado.
Para la operación V, devuelve 1 si estaba la marca y 0 = sí no estaba.

Fallo: un valor negativo (hay códigos específicos para el tipo de error) incluyendo especialmente:

-1 = ya estaba
-2 = no está
-3 = valor inválido (tipo catálogo)

Los otros valores (negativos) son técnicos y no se documentan aquí.

Descripción:

1. Accesa el árbol de valores del contexto, y ubica el valor indicado. Si no está, regresa -3 y termina.
2. Se divide el proceso en 3 casos (se entenderán después de leer la sección que explica las estructuras).
    a. Si las instancias se guardan en una lista única, se usa el apuntador (que tiene la hoja del árbol de valores) y se recupera la lista del archivo.

b. Si es un “bitmap único”, se accesa el archivo por nombre (el nombre se arma con el contexto y el valor representado).
c. Si las instancias se almacenaron usando un árbol B, se accesa el primer nodo del árbol de instancias del valor (obtiene la dirección del árbol de valores).

3. Se busca la instancia en la lista de acuerdo al método apropiado. Para lista única, se usa una búsqueda binaria, Para el árbol, se ubica la instancia o el bitmap en el que está contenido.
4. Si se determina que no está, si es AGREGAR, se agrega, de lo contrario se informa con el código asociado.
5. Si se encuentra la instancia en la lista, si es Delete, se lo elimina.

4.2.2 Armar una lista de resultados

Public **function arma-lista-resultados** (contexto, valor, mínima-instancia, máxima-instancia, tipo-de-lista as string) as long

Parámetros:
Instancia mínima y/o máxima, indican que la lista no incluira números que caen fuera de ese intervalo. Mínimo = 0 o Máximo = 0 indican que no se proporcionan criterios de ese tipo.
El tipo de lista es optativo: "L" , "I", "B", "" (nada). Los valores se aclararán cuando se decriban los tipos de listas de resultados.

Regresa:
El número de instancias incluidas en la lista solicitada.

Descripción:
Encuentra la lista de instancias (como se describió anteriormente); instancia una lista del tipo apropiado; arma la lista en memoria; la graba en un archivo en disco.
El nombre del archivo lo proporciona la aplicación antes de invocar esta función. El default (si no lo hace) es TT3, que es un nombre de archivo "temporal" que el KBC

coloca en una carpeta llamada TEMPORAL. dentro de la carpeta KBCfiles.

4.3 Las fórmulas

Para solicitar la ejecución de una fórmula, el programa invocador proporciona uno tras otro los operandos. Cuando le llega el operando #1, inicializa todo lo necesario.

Sub **ahi_van_los_operandos** (cuantos as integer) ' dimensiona lo que necesita
Informa al KBC cuántos operandos le enviará (uno por uno):
Para no causar confusión en los que preparan la invocacion del método, se separó en dos subrutinas: una permite informar un operando consitente de una "marca" (contexto, valor) y la otra se usa para operandos que usan una lista de resultados anterior.

Public sub **este_es_otro_operando-marca** (Numero-operando, contexto, valor)
agrega el operando (marca) a la lista de los mismos que usará la fórmula
Public sub **este_es_otro_operando-lista** (Numero-operando, path-de-la-lista)
agrega el operando (lista) a la lista de los mismos que usará la fórmula

Public **function ejecuta_una_formula** (la-fórmula as string, minima-instancia as long, máxima-instancia as long, tipo-de-lista as string) as long

Parámetros: se explicaron para otras funciones.
La fórmula le llega en notación postfija en el modo en el que la necesita para ejecutarla, Por ejemplo, O1 Unión O2 Unión O3, es decir, le informa qué operandos usa, y para ello emplea los números de los operandos.

El KBC tiene una función adicional que transforma una fórmula proporcionada con paréntesis a notación postfija.

Regresa:

Devuelve el numero de instancias en la lista de resultados final

Si falla indica el motivo con un código (entero negativo)

-1 = sintaxis equivocada

-2 = contexo invalido

-3 = no estaba una lista usada como operando

- 100 etc los motivos técnicos que pudieron haber causado la falla.

La fórmula:

Se pueden incluir las operaciones: intersección, unión, "not", "XOR" (or exclusivo), "UNION ALL" (union con repetidos) y diferencia (A y NOT B)

Los operandos se incluyen en la fórmula numerados en forma consecutiva (O1, O2, etc.).

4.4 Otras funciones incluidas para comodidad de los programas que invoquen ejecuciones de operaciones

De hecho, hay otras 3 funciones que sirven cuando se trata de una operación entre dos operandos. Pueden ser dos marcas, una marca y una lista o dos listas. Las 3 regresan el numero de instancias de la lista de resultados. La operación es UNION, INTERSECTION, XOR, UNIONALL y DIFFERENCE, pero tambien acepta "OR", "AND", "XOR", "OR ALL", "AND NOT"

Function operacion-dos-marcas (operacion as string, contexto1, valor1, contexto2, valor2, minima-instancia, máxima-instancia) as long

Function operacion_una_marca_y_una_lista (operacion as string, operando-1-es as string, contexto, valor) as long

El operando 1 vale M (marca) o L (lista). Siempre usa la LISTA1. Para AND NOT (diferencia) hay que indicar cual es la lista "original" y cual es la que se usa para excluir valores (A - B no es lo mismo que B - A).

Function operacion-dos-listas (operacion as string, mínima-instancia, máxima-instancia) as long
' usa LISTA1 y LISTA2. Antes de invocar la función, se indica (con un metodo) el nombre del archivo.

El KBC almacena sus marcas en estructuras de varios tipos, descritas en la sección correspondiente.

4.5 Otros métodos del KBC

No se describen todos los métodos; solo se describen los que contribuyen al entendimiento de los métodos que se apoyan, a su vez, en estos métodos complementarios. Se agrupan para efectos de esta explicacíon en diversos tipos de métodos, aunque esto no sucede en la clase misma.

Métodos relacionados con contextos:

- recibe lista de contextos de la aplicación
- que estructura usa (para un contexto)

Decisión tipo de lista; que tipo de lista de resultados armó

Métodos diversos:

- actualización MASIVA de tercias (contexo, valor, instancia)
- cambiar una lista de tipo
- eliminación de una instancia (caro) de todas las listas en las que está
- proporcionar los valores de un contexto (con filtros)

Métodos relacionados con el manejo de los archivos

Observación: el KBC usa muchos tipos de archivos (todos son archivos planos, aunque algunos los usa en modo “random”). Como los métodos usados no se vieron afectados por la investigación que se describe en esta tesis, no se

proporcionan detalles, mismos que serán publicados cuando haya una versión final y completa del KBC).

Los archivos difieren en varios aspectos. Se los incluye en carpetas de acuerdo al tipo de información que se almacena en ellos. Los nombres los proporciona KBC (con excepción de listas de resultados solicitados por un usuario) usando una serie de reglas de nomenclatura que son parte del diseño del paquete. Ademas, muchos archivos se particionan (en varios archivos físicos) para lo cual se agrega un "consecutivo" al nombre del archivo. Esto a su vez hace necesario que el paquete decida el archivo físico que contiene la información que necesita o actualizará.

Estos métodos no forman parte de la clase KBC misma (la de interfaz), sino están incluidos en clases y módulos que forman parte del programa.

- Bautiza un archivo (cual as integer, path as string)

  L1, L2 y L3 son las listas que ofrece KBC para esto. CUAL = 1 proporciona el nombre que tiene L1, etc.

  Como se dijo antes, L3 siempre es la lista de resultados que se regresa (que se grabará como "TT3" o con el nombre que le pidieron antes de invocar la fórmula,
- Abre archivos
- Cierra archivos
- Arma nombre de un archivo
- Ve si existe un archivo
- Determina cual archivo: este es un método complicado, puesto que el KBC usa varios "tipos de archivo"
- Determina la posición de algo en un archivo
- Graba en un archivo
- Lee de un archivo
- Depuración de archivos (elimina archivos vacíos o no utilizados)

- Depuración de un archivo
- Reorginazación de datos (compresión).

Métodos de apoyo:

- Referencia = Instanciar una lista de resultados (tipo-de-lista, numero-de-elementos)
- Destruir una instancia de resultados (referencia)

4.6 Estructuras de datos que utiliza el KBC

El KBC almacena los contextos de forma independiente uno de otro. Para cada contexto, se almacena un árbol-B de sus valores y una lista de todas las instancias de cada uno de los valores (en una estructura independiente). En otras palabras, para cada contexto se utiliza un índice GIN - un árbol para las claves y una colección de valores para cada clave. Aquí usamos el término colección en lugar de lista, ya que la estructura utilizada para almacenar dichos elementos puede ser: un árbol, un arreglo o un bitmap. Cabe señalar que en la terminología del índice GIN, los pares se conocen como clave / valor, donde el último se refiere a la posición, por lo general un número de fila. En KBC, estos pares son (valor, número de instancia), es decir, la palabra clave se llama valor.

Una de las propiedades de un índice GIN es que la misma instancia puede estar presente en más de una de estas listas. Por ejemplo, en una aplicación, la misma instancia puede estar presente en varias marcas (contexto**:** autor, valor**:** Smith) y (contexto**:** autor, valor**:** Jones). Nos referimos a esto diciendo que el contexto puede aplicarse a varios elementos de una instancia.

El árbol de valores de un contexto esta representado en la Figura 6.

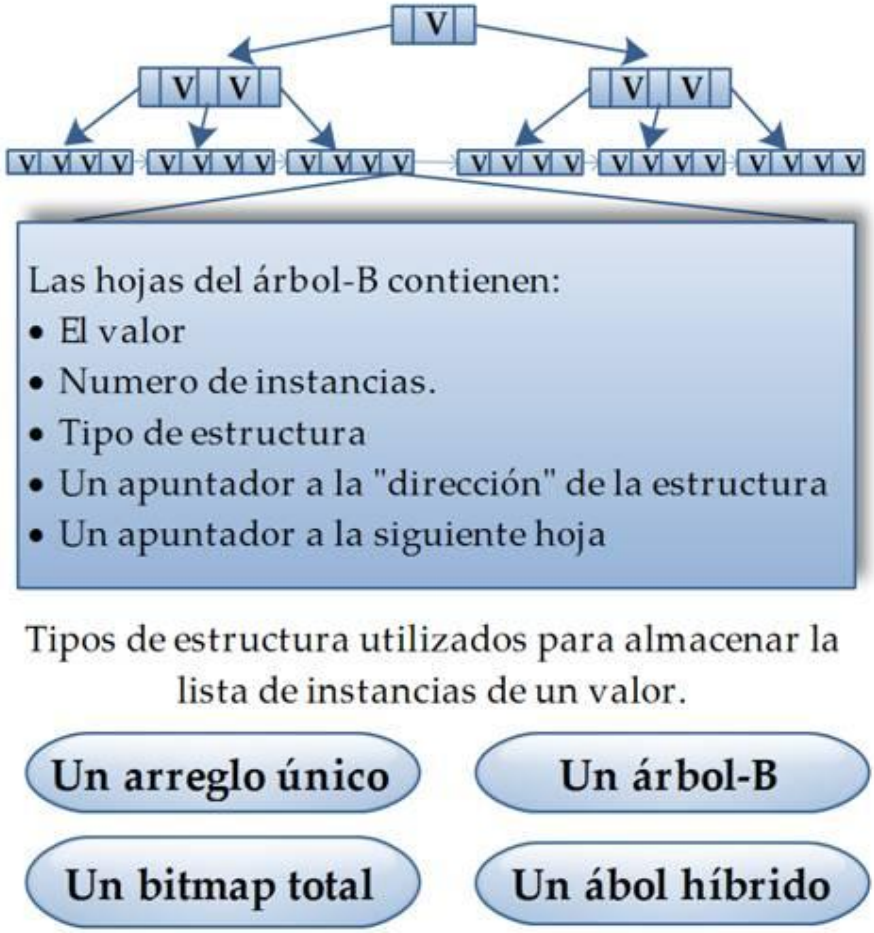


Figura 6. Árbol B de los valores de un contexto.

Los nodos del árbol B de los valores de un contexto tienen los campos ilustrados en la Figura 7, considerando que la longitud máxima de un valor dentro del KBC es de 16 caracteres.

Nodos (152 bytes)
slots_ocupados As Integer 4 bytes **apuntadores_a_los_hijos** (0 To 4) As Integer 4 bytes **las_claves** (0 To 3) As String * 16

Figura 7. Estructura de un nodo del árbol-B de valores

Tenga en cuenta que las claves son los valores del contexto. Como puede verse se decidió que estos árboles serían de orden 5. El árbol esta formado por nodos (solo guían la búsqueda a través del árbol) y hojas (contienen todos los valores).

La estructura de las hojas de un árbol que almacena los valores de un contexto, se muestra en la Figura 8.

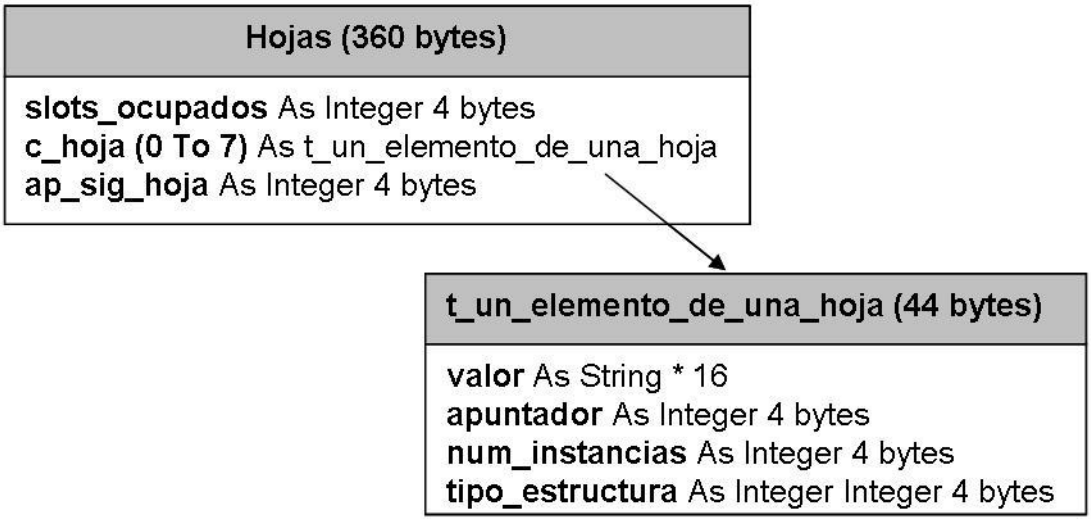


Figura 8. Estructura de las hojas del árbol-B de valores

La interpretación del campo apuntador en las hojas del árbol B depende del tipo de estructura usada para un valor en particular. Por lo que en la explicación de cada tipo de estructura, se incluye también dicha interpretación.

Para cada contexto definido por la aplicación, KBC utiliza la información acerca de dicho contexto para decidir las estructuras de datos necesarias para almacenar las instancias de sus valores. Sin embargo, KBC puede cambiar estas decisiones, si se determina que la estructura inicial no es la mejor para una situación particular. Las estructuras utilizadas actualmente por KBC son:

- Un árbol-B cuyas claves (elementos) son las instancias del valor. También lo llamamos árbol puro ya que contiene la colección de instancias almacenadas en la estructura de un árbol-B. Para todas las instancias de

los valores de un contexto, hay 2 archivos (debido a que la estructura es diferente): el que guarda los nodos y el que guarda las hojas. El campo apuntador se interpreta como la dirección del nodo raíz del árbol-B que corresponde a las instancias de un valor.

- Un arreglo único. La colección de instancias del valor se almacena como un arreglo ordenado y único de números enteros. El primer elemento (índice 0) del arreglo indica su dimensión, ya que, puede ser conveniente almacenar arreglos con algunos lugares disponibles para utilizarlos al momento de realizar actualizaciones. Para todas las instancias de los valores de un contexto se almacena un solo archivo, por lo cual el campo apuntador indica la posición adecuada del inicio del arreglo de instancias dentro del archivo.
- Un bitmap único para el rango de todos los posibles números de instancia involucrados. Se guarda un archivo por bitmap (el nombre está dado por el contexto y el valor). En este caso, el campo apuntador indica el número de instancia correspondiente al primer bit del bitmap, la longitud del bitmap se almacena en el archivo junto con el bitmap mismo.
- Un árbol B híbrido: Se usa un árbol B, pero las instancias no se almacenan en sus hojas, sino en intervalos de bitmaps que llamaremos bitmap intervalo. Cada uno de estos intervalos está definido por un elemento de la hoja del árbol hibrido, donde se indican dos datos: el primer número de instancia representado por el intervalo (bitmap intervalo) y la dirección en la que almacenó el bitmap intervalo. Como en el caso del árbol puro, tenemos 2 archivos (nodos y hojas) para almacenar el árbol y uno más por cada contexto valor para almacenar los bitmaps. Las claves del árbol corresponden a los bitmaps intervalos. Es decir, en vez de construir un bitmap único para todos los números de instancias posibles, se utilizan varios bitmaps intervalos. Este tipo de estructura se usa por primera vez dentro del KBC – por lo menos, a pesar de una búsqueda muy intensa no se encontraron tales estructuras en la literatura - y es una de las principales diferencias con respecto al índice GIN. Se decidió que todos

estos intervalos de un valor del contexto serán de la misma longitud, que se indica como un atributo del contexto. El campo apuntador indica como en el caso del árbol puro, la dirección a la raíz del árbol híbrido.

Observe que hay un caso especial que se da cuando hay una sola instancia en un valor, la instancia misma es almacenada en el campo apuntador y no se apunta a una estructura diferente. Esto pasa principalmente cuando un valor es agregado al árbol-B con su instancia. Cuando llegan más instancias de dicho valor, se crea la estructura necesaria para almacenarlas y ahí es cuando se actualiza el apuntador y el tipo de estructura.

3	1	32,001	3,200,001	Available	77543
	43,255	865,437	1	Available	

Figura 9. Estructura de la hoja de un árbol-B híbrido

En la Figura 9, se muestra la estructura de una hoja de un árbol híbrido, observe que solo se muestran 4 “espacios” pero en el KBC una hoja tiene 8 de ellos. Supongamos que los bitmaps para el valor representado corresponden a intervalos de 32,000 números (un parámetro determinado por el contexto y se aplica a todos sus valores). Este número debe ser múltiplo de 32, puesto que cada entero representa 32 números, como se explicó en una sección anterior. Los 32,000 números de instancia se almacenan en 1,000 enteros de 4 bytes. Para evitar operaciones adicionales durante la actualización, se les impone una restricción adicional a los intervalos: deben comenzar en un múltiplo de 32,000 +1.

El valor 3 en el primer campo indica que esta hoja sólo contiene tres valores (de los 8 disponibles). La última columna contiene el apuntador a la ubicación física de la hoja siguiente, misma que se proporciona como un número, puesto que las hojas se guardan como páginas de un archivo.

La primera fila indica el inicio del bitmap. Por ejemplo, el primer bitmap de la hoja representa números del 1 al 32.000, mientras que el tercero contiene los indicadores para los números 3,200,001 a 3,232,000. La segunda fila es la ubicación de ese bitmap-intervalo en el archivo donde se almacenan esos intervalos para el valor del contexto al que se refieren. Los nodos de árboles B híbridos contienen los campos: cuántas ranuras ocupadas, apuntadores-a-los-hijos (0-4) y las_claves (0-3), donde las claves son números de instancia.

4.7 Resumen de las estructuras utilizadas para almacenar la lista de instancias de un valor

Estructura para lista	Interpretación del apuntador	Tipo de estructura	Cómo se guarda (archivos)
Árbol B	Al nodo raíz.	En forma de árbol-B.	En un archivo, todos los árboles de los valores del contexto.
Árbol híbrido	A nodo raíz.	En forma de árbol-B, pero las hojas apuntas a bitmaps intervalos.	En un archivo, todos los árboles de los valores (en realidad uno para nodos y otro para hojas) y en otro archivo separado los bitmaps intervalos.
Arreglo único	A dirección en archivo.	En forma de un arreglo ordenado.	En un archivo único las listas de todos los valores de un contexto.
Bitmap único	El primer numero de instancia representado por el bitmap.	En forma de un bitmap único.	En un archivo por bitmap (el nombre está dado por el contexto y el valor).
Caso especial	Contiene el número de la (única) instancia.		Es cuando el valor tiene solo una instancia.

Tabla 2. Resumen de estructuras para almacenar una lista de instancias

4.8 Algoritmo para agregar una marca dependiendo del tipo de estructura utilizado

1. Buscar el valor en el árbol correspondiente al contexto a actualizar.
2. Si no está ese valor, se lo agrega como valor “Nuevo” con la instancia correspondiente (almacenada en el campo apuntador), es decir, no apuntará a ninguna estructura especifica ya que cuenta únicamente con una instancia.

   Observación: si el contexto es tipo catálogo (atributo 3), la operación se rechaza.
3. Si el valor está, el proceso depende del tipo de estructura al que apunta:

   a. Arreglo de instancias:
      i. Leer la dimensión del arreglo guardada en el índice 0 del mismo.
      ii. Leer el arreglo completo haciendo una operación de “get” al archivo.
      iii. Hacer una búsqueda binaria (dentro del arreglo) para determinar la posición del nuevo elemento. Si se encuentra el número de instancia “nuevo” en la lista, se rechaza la operación.
      iv. Hacer un lugar dentro del arreglo (en la posición correcta para colocar el nuevo elemento) con una instrucción de copiado de memoria. Se copia desde la posición que se desea desocupar hasta el fin de la lista a la posición siguiente de la que quedará vacía. Si es necesario, se redimensiona antes el arreglo.
      v. Insertar el nuevo elemento en la posición adecuada.
      vi. Se regraba la lista. Si hay espacio en disco en el lugar en que residía la lista antes de la operación, se graba la lista ahí mismo.
      vii. De lo contrario, se busca un lugar disponible (mayor o igual a la nueva dimensión) dentro del archivo para colocar el arreglo y se graba. En la sección de comentarios técnicos se comentan los dispositivos que usa el KBC para la reutilización de segmentos no utilizados en disco, como el que se produce en una situación como

la que describimos aquí. Para entender y recordar estos conceptos, es útil pensar en tales segmentos como “agujeros”.

b. Árbol B

Aquí se ejecuta el procedimiento descrito en el marco teórico, en donde se plantea el algoritmo para la actualización de un árbol B.

c. Un bitmap total

i. El nombre del archivo está dado por el contexto y el valor. Se leen primero los primeros bytes del archivo y se determina la dimensión del arreglo a leer. Observe que hay otro modo de hacer esto: se divide la longitud del archivo físico entre 4 (tras restarle los 4 bytes iniciales).
ii. Leer el bitmap como un arreglo de enteros.
iii. Determinar si la instancia esta dentro del bitmap.
iv. Si es necesario aumentar las dimensiones del bitmap.
v. Si el aumento es al inicio ejecutar la instrucción de copiado de memoria, de lo contrario solo redimensionar sin eliminar los datos anteriores. Observe que esta operación puede ampliar considerablemente el bitmap original.
vi. Determinar el entero (es decir, el índice del arreglo) que refleja el número de instancia nuevo, y el bit dentro del entero que representa la instancia a agregar.
vii. Utilizar el procedimiento descrito en una sección anterior, esto es, al entero correspondiente aplicarle la operación “OR” con Only_bit_set (bit correspondiente).
viii. Guardar el bitmap en el archivo.

d. Un árbol hibrido

Los pasos son los mismos que en un árbol puro (descritos en el marco teórico), solo que al llegar al intervalo correspondiente, se debe manejar como un bitmap, en donde se puede:

i. Insertar la instancia en el intervalo tal y como esta, si está el intervalo que contiene el número de la instancia a agregar.
ii. Crear un intervalo nuevo y actualizar el árbol, es decir, apuntar a este nuevo intervalo.
iii. Aumentar la longitud de un intervalo para poder incluir la nueva instancia. Se comenta que en la presente versión del KBC esto no es posible, puesto que todos los intervalos para todos los valores de un contexto tienen la misma longitud. Sin embargo, se contempla eliminar esta restricción, es decir, modificar ligeramente las estructuras utilizadas.

4. Se actualizan los datos dentro del árbol B de valores. Especialmente, se aumenta en 1 el número de instancias y el apuntador (si cambió). Puede cambiar en el caso de un arreglo único (lista) de instancias que se grabó en otro lugar del archivo, o cuando cambia el nodo raíz del árbol de instancias del valor.

4.9 ¿Cómo decide KBC las estructuras a utilizar en cada caso?

Hay cuatro maneras posibles para almacenar las instancias de un valor particular de un contexto. Cada una tiene sus ventajas y desventajas en términos de: espacio en disco (en especial el espacio no utilizado) y el número de operaciones de E/S para llevar a cabo las funciones necesarias (actualización de una marca o construcción de lista de resultados). El impacto de este número de operaciones es significativo, puesto que son mucho más lentas que las que se efectúan en memoria, y por lo tanto casi siempre representan la mayor parte del tiempo total de procesamiento.

Un arreglo de todas las instancias tiene la ventaja de que pueden ser cargados a memoria en una operación única de E/S (actualmente dos operaciones, ya que la dimensión se almacena como el primer elemento del arreglo). Dado que sus elementos se almacenan en orden ascendente, las operaciones de búsqueda y actualización se hacen mediante búsquedas binarias, y una inserción de un nuevo elemento al arreglo se realiza con una operación única de copia. Por ejemplo si el nuevo elemento se colocará en la posición 45 de un arreglo de 144 elementos, 400 bytes - 100 posiciones del arreglo - se copian de la A (45) a A (46), lo que hace un espacio disponible en A (45) para la nueva entrada. Naturalmente, la dimensión del arreglo se debe aumentar en 1. Tenga en cuenta que la posibilidad de copiar un bloque de memoria elimina una de las razones que han conducido a la adopción de estructuras de datos diferentes, como las listas enlazadas en lugar de un arreglo o matriz.

Sin embargo, la mayor desventaja del uso de estos vectores consiste en el espacio en disco. Cuando la dimensión aumenta, el arreglo se tiene que almacenar en un área diferente del archivo y el espacio anterior ahora será inutilizado. Para las listas muy largas, este espacio en disco añadido puede resultar excesivo. Por supuesto, los dispositivos para reutilizar dicho espacio se incluyeron en el KBC, pero no obstante, la situación de espacio no utilizado es una de las principales preocupaciones que deben tenerse en cuenta antes de elegir esta estrategia.

Los árboles B son adecuados si hay un número suficiente de instancias que hacen impractico el uso de un arreglo único: las listas pueden ser demasiado largas para su manejo como arreglos. Las desventajas de estos árboles es que por lo general tienen un porcentaje de espacios no utilizados y las operaciones de búsqueda y actualización implicarán la lectura de nodos en el orden indicado por el árbol. Por supuesto, la paginación evitará operaciones individuales de E/S en muchos casos.

Un bitmap total, es decir, un bitmap único para todo el rango de números posibles, reducirá el número de operaciones de E/S, pero puede resultar en muchos bits no utilizados. Si las instancias son escasas, esto puede ser un factor crítico. Intervalos de mapas de bits se incluyeron precisamente por esta razón: es como hacer grupos de números de instancias. Además, para algunas operaciones, sólo uno de estos intervalos tendrá que ser cargado a memoria.

La decisión sobre la elección de la estructura utilizada para almacenar las instancias de un determinado valor, o en general, las de un contexto dado, se verá reflejada en la construcción de las listas de resultados. Por los motivos expuestos, la selección se basará en diversos factores. Dado que la información proporcionada sobre los contextos (por la aplicación que usa el KBC) sólo puede ser parcial o basada en estimaciones a priori, KBC implementa las rutinas que pueden determinar la necesidad de modificar las estructuras utilizadas para almacenar las instancias de un valor dado. Por ejemplo, si la información inicial indicaba el uso de un arreglo único, se debe cambiar la estructura si el número de instancias que se almacenan actualmente es mayor que la máxima dimensión de un arreglo, el cual será determinado como parámetro de la aplicación pero en la versión actual del paquete es 100,000. Al actualizar un árbol-B que fue elegido como la estructura conveniente, se reúne información adicional, especialmente el número de instancias de algunos de sus valores. Esto a su vez permite determinar si conviene cambiar de estructura, es decir, en lugar de un árbol B, usar alguna de las otras estructuras.

La aplicación también puede indicar que los valores de un contexto particular tendrán pocas actualizaciones. Por ejemplo, esto ocurrirá cuando se cargan los datos de otras fuentes, y no se harán cambios en el futuro. Otro elemento que se puede tomar en cuenta es cuántas actualizaciones ocurrirán para un determinado valor. Si esto sucede de forma individual, un árbol-B puede ser óptimo. En caso de actualizaciones en lotes, KBC incluye un método que acepta un arreglo de instancias para actualizar un par particular (contexto valor). El

método usa los algoritmos para la actualización de un lote de claves en un árbol B descritos en la sección sobre estos árboles.

4.10 Historia del KBC

4.10.1 Primera versión de KBC

El KBC surge en los 80's como solución a una necesidad específica. Sin embargo, no es hasta el 2002 que se elabora una versión total, es decir, que proporciona todos los métodos necesarios para una aplicación. Se trataba de gestionar una colección de datos, que contenía elementos no estructurados – textos – en el sentido de poder obtener los subconjuntos deseados de los mismos. Un trabajo que resultó en la tesis (Cruz M., 2003), incluyó un producto denominado Marte (marcado y recuperación de textos). Se señalaban (marcaban) ciertas palabras del texto como palabras clave, pero con una indicación del significado de la palabra. Esto permitiría obtener subconjuntos sin la presencia de textos no deseados. Por ejemplo: armar el subconjunto de los textos en los cuales está marcada la palabra "ROJO" para obtener el material relacionado con el artista Rojo, devolvería textos sobre un vestido rojo, o un automóvil de ese color.

Se creaba una lista de contextos (en aquel paquete se llamaban tipos de marca) y además de marcar una palabra, se lo hacía con un color asociado con el tipo de marca. Para gestionar estas "marcas" (tercias de contexto-valor-instancia) donde la instancia era el número de texto indizado, se creó un paquete, el actual KBC, aunque tenía otro nombre.

El KBC manejaría las marcas como listas de instancias para cada valor de cada contexto. Es lo que ahora se llama un índice invertido generalizado, como se ha dicho antes. Para ello, para cada contexto se guardan los valores en un árbol B+ (como se aclaró en la sección sobre árboles, aquí los llamaremos B). En cada hoja de este árbol, además del valor, se incluye un apuntador que indica dónde está la lista de sus instancias. Las instancias, a su vez, también se guardaban en un árbol B+, aunque con una variante, podría ser lo que ahora llamamos un árbol

B híbrido: sus hojas podrían ser valores o apuntadores a listas de instancias. Por la urgencia con la que se desarrolló una versión que funcionara, se decidió usar bases de datos relacionales para *simular* los árboles B. Se almacenan los valores de un contexto, junto con sus instancias, en una tabla de una base de datos relacional (BDR). Hay una tabla para cada contexto (es decir, se usan tantas instancias de la tabla como contextos se manejan para una aplicación).

El diseño de la tabla se muestra y comenta a continuación, no sin antes mencionar que los valores tienen un máximo de 16 caracteres, y las instancias (se pueden conceptualizar como número de registro o de texto) son enteros de 4 bytes. El diseño incluía dos usos diferentes de "las instancias". Para listas relativamente cortas de instancias, se indicaban sus respectivos números en estos campos. En cambio, cuando un valor del contexto tenía un número mayor que un parámetro (fijado en aquella versión como variable, pero con un valor estimado de 10,000) se usarían los campos de otro modo. Se usaban las 16 posiciones como dos arreglos "paralelos". En uno de ellos, se guardaba el valor de la primera instancia de una lista de 200 instancias, y en el otro arreglo se colocaba un apuntador a la lista, que residía en archivos especiales, manejados en forma independiente del manejador de base de datos. De ese modo se implementaron los árboles híbridos en su primera versión. Cabe la observación que este diseño de la base de datos viola casi todas las reglas de normalización de una base de datos relacional. Esto naturalmente se debe a que no era necesaria la normalidad de la base, sino el diseño estaba enfocado al espacio total para almacenarla y la eficiencia máxima de uso para las funciones para las cuales fue diseñada.

Esta versión se usó durante varios años, y su desempeño fue lo suficientemente bueno para obviar alguna modificación. Sin embargo, a medida que el producto se usó para otras aplicaciones, la duración de ciertas operaciones y el espacio en disco consumido resultaron excesivas, comparadas con lo que se podía imaginar que era el mínimo requerido para lograr el mismo propósito. Esto resultó particularmente importante para una investigación sobre el desempeño de

una aplicación del KBC, pero que involucraba un conjunto numeroso de registros (aproximadamente 200 millones). Se decidió elaborar una nueva versión del KBC, en la cual se reflejarían los dos objetivos fundamentales: reducir el número de accesos a disco necesarios para ciertas funciones fundamentales, y aprovechar al máximo el espacio en disco de modo de reducir el tamaño total de un acervo de datos que incluyera las marcas administradas por el KBC. Como se verá, esto consistió en dos aspectos fundamentales: formular diferentes modos de almacenar las marcas, y diversos modos de armar y usar las listas de resultados. Anteriormente, desde la concepción del producto, se había planeado el uso de conjuntos de bits que reflejarían la inclusión de una instancia por la posición del bit en la cadena. Es lo que en una sección anterior se describió con el término bitmap. Listas almacenadas de este modo acelerarían considerablemente las operaciones booleanas entre ellas, pero por otro lado, resultarían en un número de bits "superfluos" puesto que no correspondían a elementos del conjunto indizado. Para no dejar al lector la tarea de imaginarse la tabla descrita, se reproducen aquí sus campos.

4.10.2 Diseño de la tabla para almacenar las instancias de un contexto

Nombre	Tipo de dato	Significado
ocup_este_reg	Byte	De las 8 (o 16) instancias que caben
V1	Long	La primera instancia (de este registro)
V2 --- v8	Long(s)	Otras 7 instancias
V9	Long	Depende de v-es-instancia (ver abajo)
V10 --- v16	Long(s)	Mismo significado que V9
Pal	String 16 cars.	El valor (del contexto)
cota_superior	Long	El mayor numero de instancia que puede estar en este registro
min_posible	Long	El menor número de instancia que puede estar en este registro
Cuantas_instancias	Long	Solo se usa para el primer registro del valor: indica el numero total de instancias de ese valor
V_es_instancia	Si/no	SI= la lista son 16 instancias; no hay apuntadores

Tabla 3. Diseño de una base de datos para almacenar marcas

El índice (principal y único) estaba formado por pal & min_posible.

4.10.3 Interpretación de los campos en el diseño de base de datos

Si v-es_instancia es Verdadero se guardan los números de instancia en estos campos (es decir, es un árbol).

Si ve-es-instancia es FALSO, se trata de un árbol híbrido. Se interpretan como pares de valores V1 y V9, V2 y V10, etc. El primer dato es el menor número de instancia de la lista de números que están “en la dirección” que se anota como segundo elemento del par.

Ejemplo v1 = 405, V9 = 314567 v2 = 1498, v10 = 7843

Hay una lista (todas tienen 200 elementos, de los cuales puede haber algunos vacíos) en la dirección 314567 del archivo en disco, y el primer numero de instancia de la lista es el 405.

El manejo de la tabla para actualizaciones y consultas no se muestra en la tesis, puesto que de todos modos, ya fue reemplazado por los métodos expuestos anteriormente.

CAPÍTULO 5. LISTAS DE RESULTADOS EN KBC

Cuando KBC recibe una solicitud para construir un conjunto con las instancias de un par (contexto-valor), devuelve una lista de resultados. Como se verá, estas listas también se pueden utilizar como operandos de operaciones lógicas.

KBC utiliza tres tipos de resultados listas. **Listas-L** son arreglos de números enteros. **Listas-B** consisten en un bitmap único y **Listas-I**, que se almacena como un arreglo de intervalos y el bitmap correspondiente a cada uno de ellos. Tenga en cuenta que un Lista-B es una Lista-I con un solo intervalo.

5.1 Ejemplo de listas de resultados en KBC

Tendiendo una lista de instancias que representa los múltiplos de 2, del 2 al 20 y del 100 al 110, esto es (2, 4, 6, 8, …..20), utilizando los tipos de listas que maneja el KBC, cada estructura se almacenaría como se muestra a continuación:

Lista-L. Almacena un arreglo ordenado de números enteros (de 4 bytes) de instancias, como se muestra en la Figura 10.

2	4	6	8	10	12	14	16	18	20	100	102	104	106	108	110

Figura 10. Ejemplo de una Lista-L

Lista-B. Almacena un bitmap total para todas las posibles instancias de 1 a 120, como se muestra en la Figura 11. Es importante recordar que, enteros de 4 bytes, representan 32 instancias por número entero, deben terminar en un múltiplo de 32. El bitmap se almacena y usa como un arreglo de enteros de 4 bytes.
Inicio del bitmap = 1. Esto significa que el primer bit del bitmap corresponde a la instancia número 1 (del acervo de datos al que se refiere).

Figura 11. Ejemplo de una Lista-B

Observe que del 33 al 96 todo está lleno de ceros, esto quiere decir que no hay instancias presentes en ese rango, pero como es un bitmap total, es necesario guardarlo completo, lo que provoca cierto desperdicio de espacio en disco principalmente.

Lista-I. En lugar de un arreglo de enteros únicos (el arreglo es el bitmap) la lista está compuesta de varios arreglos de enteros, como se muestra en la Figura 12. El archivo está dividido en 3 partes. En el inicio (que es común para todos los tipos de lista) se guardan ciertos datos descriptivos. A continuación está la lista de intervalos. Para cada uno de ellos, se incluyen: la dimensión del bitmap correspondiente y el número de instancia que refleja el primer bit del bitmap. Por ejemplo, si en lugar de un bitmap único (lista-B) se decide guardar la lista como lista-I, entonces resultaría lo siguiente.

Los intervalos son:

Intervalo 1: longitud 1 (1 entero). Primer bit es instancia 1.

1-16	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
17-32	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

Intervalo 2: longitud 1 (1 entero). Primer bit es instancia 97.

97-112	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	0
113-128	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 12. Ejemplo de una Lista-I

5.2 Cómo se implementan las listas (en los programas)

Para implementar las listas de resultados, KBC utiliza las clases y subclases que se describen a continuación. Los comentarios y explicaciones están en cursiva. Las funciones se indican con: el tipo de datos que devuelven = el nombre de la función. La descripción detallada de los métodos se incluyó sólo en caso de que parecía necesaria para entender bien el método o la forma en que se ejecuta. Los nombres de algunas variables y métodos están en idioma inglés puesto que ése es el idioma utilizado para desarrollar el KBC. Los "convenience fields" son campos superfluos que se incluyen con algún propósito principalmente de eficiencia computacional.

La clase base de la cual heredan las dos subclases es:

Class Cls-List (Mustinherit): '*solo sus dos subclases pueden ser instanciadas*

Fields:

- List_type_of_this_instance '*valores "I" o "L".*
- Number_of_elements_of_the_array
- Dimension_of_the_array *'Observe que el numero de elementos indica la ultima posición ocupada en el arreglo, mientras que la dimensión marca el tamaño total del arreglo ya que en algunos casos es conveniente almacenarlos con huecos que son utilizados durante las actualizaciones, para evitar ser copiado a otra posición dentro del archivo.*

Convenience fields *(se pueden calcular pero son útiles):*

- last-instance-provided-to-caller
- smallest-instance-number
- largest- instance-number *´Los datos de: pimer y ultimo numero de instancia son muy utiles para poder determinar rangos en caso de un bitmap total, o para saber si es posible descartar o incluir (a la lista de resultados) una lista completa o parte de la misma.*

La subclase utilizada para almacenar Listas-L es:

Class Cls_L_List (inherits Cls_list) *'para Listas L*
Fields: • array_of_instances () *'Enteros correspondientes a números de instancias.* **Methods:** • New (how-many-elements) *'El constructor dimensiona el arreglo en how-many-elements.* • Integer = next_instance_of_list (not-less-than) *'Devuelve el siguiente numero de instancia de la lista. El parámetro indica que no se devolverá un número de instancia menor, si este parámetro es 0, se devolverá la siguiente instancia. Si ya no hay mas instancias (fin de lista), el método devolverá el valor de the-biggest-number, el cual es* $2^{31} - 1$. • Add-an-instance (instance-number) '*Si es necesario, redimensionar el arreglo (agregarle 100 posiciones), determinar donde se debe colocar el nuevo elemento, hacerle espacio (con una instrucción de copiar en memoria) y actualizar el arreglo con el nuevo valor.* • Delete-an-instance (instance-number) *'Determinar la posición del elemento a eliminar, copiar memoria de (posición +1) hasta el final de la lista, decrementar Number_of_elements_of_the_array en 1.* • Load_from_disk_file (path) '*Lee la dimensión del arreglo y pasa los datos de disco al arreglo.* • Store_in_disk_file (path) *'Escribe la primera parte del archivo (como se describe más adelante) a continuación, añadir lo del arreglo al archivo.*

La subclase utilizada para almacenar Listas-I es:

Class Cls_I_List (Inherits Cls_List) *'Lista- I (y Listas- B como un caso especial)*

Structure An_Interval_Bitmap *'Almacena el bitmap de un intervalo*

- number_of_integers_in_this_bitmap *'Longitud del bitmap*
- First_bit_corresponds_to *'El numero de instancia representado por el primer bit*
- Last_bit_corresponds_to *'Puede ser calculado pero es incluido por eficiencia*
- Reference_to_bitmap as Cls_bitmap *'Es la referencia a la clase Cls_Bitmap*

Fields:

- array_of_intervals () as An_Interval_Bitmap *'Del tipo de la estructura definida*
- Interval_index *'Utilizado cuando se procesara una instancia*
- current-index *'El ultimo usado*

Methods:

- New (elems) *'El constructor dimensiona el arreglo de intervalos*
- Load_the_I_list (path_of_file) *'Carga el arreglo de intervalos y para cada intervalo crea una instancia de la clase Cls_Bitmap (con la dimensión correcta) y su constructor lee el bitmap de un archivo*
- Int = Next_instance_of_list (not_less_than) '*Usa el current index, pide al bitmap la siguiente instancia y si llega al final del bitmap actualizar el current-index + 1*
- Determine-interval-index (instance-number) *'Utiliza una búsqueda binaria para determinar el bitmap que contiene la instancia*
- Is_on=Process_an_instance (inst, oper)
  'Determine-interval-index (instance-number);
  'is-on = Preguntarle al bitmap de ese intervalo si esta prendido (instance-number)
- Add new instance_to list (instance-number)
  'Determine-interval-index (instance-number), si se encontró, invocar SET de su bitmap; de lo contrario, crear intervalo (en la posición correcta) con un solo entero, los bitmaps "cortos" después son juntados.
- Delete-instance-from-list (instance-number) *'Determine – internal – index (instance-number, si se encontró, invocar TURN_OFF en el bitmap correspondiente.*
- add_new_integer_to_a_bitmap (the_integer, first bit is instance) *'Determine-interval-index (instance-number); de acuerdo con first-bit-is-instance, agregar un nuevo bitmap si es necesario y llamar al método de su bitmap.*
- delete_an_integer_from_a_bitmap (the_integer, first bit is instance) *'determine-interval-index; de acuerdo con first-bit-is-instance, llamar al método de su bitmap.*
- Send_List_To_File (Path-of-file-to-be-created) *'Escribe la primera parte del archivo (como se describe más adelante), copia del arreglo de intervalos al archivo, a continuación, agregar uno detrás de otro, los bitmaps como matrices de enteros.*

La clase utilizada para almacenar cada bitmap es:

Class Cls_Bitmap *'Para un bitmap- intervalo*

Fields:

- Array of integers ()
- first_bit_corresponds-to
- integer-index
- bit-position
- dimension-of-array-of-integers

Methods:

- New (how_many_integers) *'Dimensionar el arreglo de enteros*
- Load-bitmap-from-file (first_byte_of_file_this_bitmap)
- Compress-a-bitmap (first_instance as reference, dimension as reference) *'Es llamado con ceros ya que en los parámetros se almacenarán los nuevos valores, si la dimensión es cero, significa que puede ser eliminado*
- Operation_on_bit (instance-number, operation) *'Prender, apagar o hacer un SWITCH; switch (si está prendido se apaga y viceversa)*
- True/False = is_the_bit_on (instance-number)
- Integer=Operation_on_an_integer
  (first-instance-represented, operand-value, oper)
  '*UNION, INTERSECCION, XOR, DIFERENCIA*
  'Realizar la operacion Operand-value OPER array_of_integers (index_integer); y regresar el nuevo valor del entero ya afectado.
- Change_dimension_of_array (beginning-or-end, how- many-positions) *'Redimensionar el arreglo, hacer un espacio y si los espacios deben ser al inicio, se realiza un copiado de memoria*
- Determine-the-integer-and-bit (instance-number)
  integer-index=(instance-number –first-bit-corresponds-to)\ 32
  *' **** indica división entera,*
  ' enteros y bits son numerados desde 0
- *bit-position=(instance-number - first-bit-correspons-to) mod 32*

5.3 Proceso de construcción de una lista de resultados para un par Contexto-Valor

El caso más sencillo es cuando las instancias se guardaron en una lista de instancias (arreglo). En ese caso, se lee el arreglo del archivo. SI la estructura

utilizada era un bitmap único, el proceso es un "GET" de todo el bitmap, que a su vez es un arreglo de números enteros.

Si se usó una de las otras estructuras (un árbol B o un árbol Híbrido). Se procesa el árbol-B de los números de instancia para un par (contexto-valor) para crear una lista de resultados. La inclusión de apuntadores de cada hoja a la siguiente proporciona una forma eficiente de hacerlo, esto es: buscar la primera hoja del árbol-B, incluir los números reflejados en esa hoja en la lista, y luego usar los punteros a la hoja siguiente. Observe que en caso de un árbol hibrido, para cada valor almacenado en la hoja habrá otro dato, que apunta a la ubicación del bitmap intervalo correspondiente.

El tipo de lista que se construirá en memoria está determinado por la forma en que fueron almacenadas las instancias: un árbol B puro o una lista única se traducirá en una Lista-L, mientras que los otros casos resultarán en una Lista-I (para un bitmap único, esta tendrá un solo intervalo).

La aplicación que solicita la lista puede especificar el tipo de lista que necesita: en este caso, la lista de resultados se transforma si es necesario. Las rutinas para hacer esto no se incluyen en las clases, ya que implica la lectura de un archivo que contiene una lista, y la creación de un archivo diferente para la lista transformada.

Para crear una lista de resultados, se instancia la clase apropiada y se invocan sus métodos por la rutina que procesa las instancias.

5.4 Archivos de disco que contienen las listas de resultados

Los primeros bytes del archivo contienen una descripción de su contenido, especialmente, la dimensión del arreglo que almacena y el tipo de lista (L o I). Los números de primera y última instancia de la lista se incluyen por conveniencia, puesto que se podrían obtener de los datos mismos, pero esto implicaría leer el

resto del archivo de datos. Además se guardan otros datos informativos, tales como la fecha de creación, un número de identificación del usuario que creó la lista, y una descripción de los contenidos, es decir, el par o fórmula que produjo la lista de resultados. A continuación (después de este primer segmento que es común a todos los tipos de listas) se almacenan los datos como se indicó anteriormente.

CAPÍTULO 6. OPERACIONES ENTRE LISTAS DE RESULTADOS

6.1 Operaciones y fórmulas

KBC recibe peticiones para realizar operaciones como fórmulas en notación postfija, o con los paréntesis necesarios para agrupar operaciones. Una fórmula consistirá de operaciones y operandos. Los operandos son de dos tipos: los que están indicados por un par (contexto-valor) o a través de una referencia a una lista anterior, ya sea en memoria o en un archivo de disco.

La ejecución de las operaciones se hace siempre sobre listas de resultados en memoria. De ese modo, si hace falta, se construye la lista resultante de un par (contexto-valor) antes de usar el operando en una operación. El KBC acepta las operaciones: "O", "Y", "NO", "XOR" y UNION ALL, donde XOR significa "o exclusiva " (miembros que pertenecen exactamente a uno de los operandos), y UNION ALL indica que, si un elemento está presente en más de un operando, se incluye más de una vez en la lista de resultados. El uso principal de la operación unión all es reemplazar la función de agregación "Count" de SQL. Puesto que los lenguajes de programación ofrecen las operaciones "O", "Y", "NO" y "XOR", los algoritmos correspondientes están basados exclusivamente en estos operadores para procesar números enteros que representan 32 instancias, en el caso de bitmaps.

KBC creará una lista de resultados como una instancia de la clase correspondiente, y luego la guardará en un archivo de disco. Si la aplicación especificó un tipo de lista KBC transformará la lista durante el proceso de almacenamiento. El nombre de este archivo lo proporciona la aplicación (antes de invocar la fórmula) o se crea con el nombre “TT3”, como se describió previamente.

Un intérprete de fórmula lleva a cabo la tarea de dividir las operaciones de una fórmula en etapas, cada una de las cuales puede o no consistir en una operación única. Así, ciertas combinaciones de las operaciones no se producirán.

Una operación puede implicar M operandos, pero la operación debe ser única. En otras palabras, las ejecuciones no mezclan intersecciones y uniones. Si la operación es “diferencia”, el resultado es el conjunto de elementos de la lista como primer operando que no están presentes en ninguna de las listas de otro operando. Es decir, R = RL1 - RL2 - RL3 ...

El siguiente paso consiste en dividir las operaciones, si se considera conveniente. El objeto es producir operaciones con 2 operandos, salvo si todos los operandos son del mismo tipo de lista. En el caso especial de una operación de diferencia, las listas de excluidos son combinadas (se hace primero la unión) de modo que la diferencia siempre implica exactamente dos operandos: A - B - C = A - (BUC).

6.2 Los algoritmos implementados en KBC

KBC tiene 6 algoritmos básicos para ejecutar las operaciones. La elección del algoritmo apropiado depende tanto de la operación como del tipo de lista involucrada.

Como se verá, algunos de estos algoritmos se pudieron haber concentrado en uno solo. Sin embargo, esto hubiera incluido algunas instrucciones “IF” o “CASE” que se tendrían que ejecutar muchas veces, mientras que al separar los algoritmos, un único “IF” los elimina de los algoritmos.

También crea un arreglo de "nombres de listas" L (1), L (2), ... L (M): que son las referencias a los operandos de la fórmula. Observe que no se trata de construir listas ni instancias: se trata de “bautizar” las listas que intervendrán en una operación con un nombre genérico, adicional al que tiene la lista, aplicable a todas las operaciones de una fórmula, para poder formular los algoritmos precisamente basados siempre en los mismos nombres de listas..

La determinación de la primera lista, L (1) se puede resumir de la siguiente manera:

- En las diferencias, la lista original se denomina L (1), independientemente de su tipo.
- En todas las demás operaciones, la primera lista debe ser del mismo tipo que la que se obtendrá como resultado. Sin embargo, si se producirá una lista_B, L (1) deberá ser una lista de tipo I.

El principal objetivo de la selección del algoritmo correspondiente es evitar el procesamiento de los elementos de una Lista-I (bitmaps), uno por uno. La única operación para la cual esto no puede evitarse es en Unión-all involucrando una Lista-I como uno de sus operandos: las instancias representadas por el bitmap tienen que ser procesados individualmente ya que algunos números de instancias se deben incluir más de una vez en la lista: el resultado de la lista no puede ser un bitmap.

Los algoritmos implementados en KBC para realizar operaciones entre listas de resultados son:

1. Recursive-unions-of-lists: Utilizado solo en uniones de M listas L o para union-all de M listas de cualquier tipo.
2. Produce_a_B-List. Para uniones de cualquier número de listas I.
3. Produce_an_I-list_as_union_1_I-list _ and_1_L-list
4. Produce_an_I-list_intersec_or_diff_of_I-lists.
5. Produce_I-list_difference_I-list_minus_L-list
6. Produce_an_L-list_intersection_or_difference.

6.2.1 Recursive-unions-of-lists

Se utiliza para uniones de: solo Listas-L, o para una Unión-all de M listas de cualquier tipo. La lista de resultados final se crea como una Lista-L (vacía). Los elementos de L (1) invocan la rutina recursiva para agregar las instancias menores a los de la L(1) y a continuación, se agrega el valor de L(1). No se incluyen valores repetidos excepto en una operación de Union-all. Se muestra el código puesto que es más fácil describirlo de este modo.

```
Sub Recursive_unions_of_lists()

For ind_lista = 2 To CUANTOS_OPERANDOS
        ULTIMO_VALOR_DE_LISTA(ind_lista) =
        otra_lista_como_clase(ind_lista).next_instance_of_list(0)
Next ind_lista
val1 = 0
While val1 < THE_BIGGEST_NUMBER
        val1 = otra_lista_como_clase(1).next_instance_of_list(0)
        Call RecursiveUnion(2, val1)
        If val1 < THE_BIGGEST_NUMBER Then
                esta_ya_quedo_en_la_lista_final (val1)
Wend
End Sub

Sub RecursiveUnion(n_list As Integer, min As Long)
If n_list > CUANTOS_OPERANDOS Then Exit Sub
Dim value As Long
value = ULTIMO_VALOR_DE_LISTA(n_list)

If value > min Then
        Call RecursiveUnion(n_list + 1, min)
          Exit Sub
End If

While value <= min
        Call RecursiveUnion(n_list + 1, value)
        If value = THE_BIGGEST_NUMBER Then Exit Sub
        If (value < min) Then esta_ya_quedo_en_la_lista_final (value)
        value = otra_lista_como_clase(n_list).next_instance_of_list(0)
        ULTIMO_VALOR_DE_LISTA(n_list) = value
Wend
End Sub
```

6.2.2 Produce_a_B-List

Se usa para uniones de cualquier número de Listas-I. Se construye un intervalo está construido para incluir todos los intervalos de las listas-I de sus operandos. La razón de esta decisión arbitraria no es técnica. El algoritmo original que se había utilizado durante muchos años para la construcción de la lista de resultados I en estas operaciones, fue - increíblemente - patentada por Oracle en el año 2000 (Ozbutun C., et. al. 2000), por lo que se decidió no utilizarlo ya que esto se "inventó" hace muchos años. Este bitmap único se denota como [de primera instancia, a última instancia] y está determinado por:

first-instance = min **{**(L(i). smallest-instance-number, i= 1 to M**}**
last-instance = max **{**L(i).largest instance number, i = 1 to M**}**

Uno tras otro, se procesan los bitmap intervalos de los operandos. Para cada uno de los bitmaps, se procesan, uno tras otro. Los enteros del arreglo, y para cada entero, se efectúa una operación “OR” con el entero del bitmap resultado correspondiente al mismo rango de instancias. Naturalmente se contemplan todos los artificios que permiten evitar operaciones superfluas, como procesar enteros con valor 0 (no representan instancias).

6.2.3 Produce_an_I-list_as_union_1_I-list _ and_1_L-list

La lista de resultados final se construye como una copia de L (1), que será la lista-I, y se agregan las instancias de la otra lista (la Lista-L). Siempre que sea necesario, se agrega un intervalo de bitmap (cuando la instancia de una Lista-L no está en un intervalo de la lista final).

6.2.4 Produce_an_I-list_intersec_or_diff_of_I-lists

La lista de resultados final se crea como una lista vacía, es decir, una Lista_I con 0 intervalos. Cada intervalo de L (1) es procesado, y el intervalo se agrega a la lista final si contiene al menos una instancia después de efectuar las

operaciones pertinentes. En realidad, antes de ser agregado, es reducido si es posible. Esta rutina utiliza un bitmap temporal (un arreglo de enteros) con el fin de no afectar al original de L (1). Para cada intervalo de L (1), se crear el bitmap temporal como una copia del arreglo: cada número entero se “verifica” con las otras listas, por supuesto que con la operación apropiada: "y" en las intersecciones o "y no" en las diferencias. En la clase Cls-I-lista hay métodos para proporcionar el siguiente número entero del bitmap, así como el número de instancia (que será el representado por el primer bit de ese número entero).

6.2.5 Produce_I-list_difference_I-list_minus_L-list

La lista final se crea con una copia de L (1). Los elementos de L(2) son eliminados – uno tras otro - de la lista final. La lista final se comprime, esto es, los intervalos de bitmaps que no contengan instancias son eliminados. También se pueden “acortar” los intervalos cuando los arreglos comienzan o terminan con varios ceros.

6.2.6 Produce_an_L-list_intersection_or_difference

Se invoca para las intersecciones de una Lista-L y una Lista-I, o la diferencia L (1) de tipo L - L (2). La lista final se crea como una copia de L (1). Sus elementos se verifican en las otras listas y sobreviven o no. La lista L se comprime cada vez que se elimina uno de sus elementos. Esto se hace con una operación “copymemory”.

CAPÍTULO 7. COMENTARIOS TÉCNICOS

7.1 Etapas de desarrollo del paquete KBC

El paquete KBC constantemente sufre modificaciones, en especial mejoras, pero también adiciones de funciones complementarias. Además, hay dos versiones, una en Visual Basic 6.0 y la otra en Visual Basic .Net. No sólo difieren en el lenguaje y en las estructuras de programación (el VB no permite el uso de subclases, mientras que la versión .Net sí las implementa) sino que tienen grados distintos de avance en cuanto a la inclusión de las mejoras, como las mencionadas en este trabajo.

La intención es tener dos versiones idénticas en cuanto a la funcionalidad, para ofrecer el producto en ambas tecnologías, a pesar de que existe la posibilidad de usar una versión en uno de los lenguajes para una aplicación desarrollada en el otro.

En cuanto a la funcionalidad, hay varios temas que son motivo de constante investigación, entre los que destacan dos de ellos: el aprovechamiento de la paginación para reducir el número de operaciones de entrada-salida, y los elementos que proporcionan confiabilidad en ambientes multi-usuario, es decir, contemplar todos los aspectos de la concurrencia. A pesar de que ya contiene elementos para evitar algunos conflictos que se presentan por la concurrencia, especialmente con dispositivos de “lock” de archivos, estas componentes podrán ser mejoradas considerablemente con la aplicación de otros modelos.

7.2 Cómo se almacenan en disco los árboles, los arreglos y los bitmap

7.2.1 Los árboles B

Todos los elementos de los árboles se almacenan en archivos planos. El conjunto de índices (sus nodos) y el de “sequences” (las hojas) se graban en archivos separados. Todos los árboles de instancias de valores de un mismo contexto usan el mismo archivo, aunque esto se refiere a un archivo lógico. Los

archivos en ocasiones se particionan en varios archivos físicos, para evitar archivos demasiado “pesados”, especialmente para los procesos de respaldos parciales de los datos.

El archivo para guardar un conjunto de nodos de un árbol B, al igual que el que guarda sus hojas, se usa por páginas. El tamaño de estas páginas lo determina el uso del árbol, y la cantidad de memoria que se desea dedicar a las páginas recuperadas de archivos.

7.2.2 Como se guardan los arreglos de números de instancia

Todos los arreglos de un mismo contexto usan el mismo archivo. Cuando la actualización de una tal lista resulta en una de tamaño mayor, frecuentemente no habrá espacio para regrabarla en el mismo lugar. En ese caso, se graba el arreglo en otro lugar del archivo, lo que produce un “agujero”: el espacio que ocupaba anteriormente está desaprovechado. En la sección sobre gestión de espacio se explican los dispositivos de software para evitar desperdicio de espacio en disco.

7.2.3 Como se guardan los bitmaps

Los intervalos de bitmaps se guardan en un archivo, pero bitmaps de valores diferentes usan archivos separados. En otras palabras, los bitmaps se guardan en un archivo para cada valor. Si se trata de un bitmap total, constituye todo el archivo, es decir, se guarda el bitmap en un archivo que no contendrá otros elementos.

Puesto que todos los bitmaps de un mismo contexto tienen la misma longitud, el único caso en que habrá espacios intermedios del archivo no utilizados se da como consecuencia de la eliminación de un bitmap. Una vez más, referimos al lector a la sección siguiente sobre la reutilización del espacio en disco.

7.3 Reutilización de espacio en disco

Como consecuencia de las nuevas estructuras usadas para almacenar la información, se produjeron situaciones en las cuales habrá segmentos de los archivos que no contengan información. Esto hará que se desperdicie espacio en disco, lo que a su vez significará una perdida de eficiencia en cuanto al aprovechamiento de estos espacios.

Por lo tanto una parte de la investigación consistió en disminuir este desperdicio. A continuación se describen los artificios usados para conseguir este propósito.

Se presentan dos situaciones de espacio en disco ya no utilizado: todos estos espacios son de la misma longitud (en árboles y bitmaps intervalo) o pueden ser de tamaño diferente (almacenamiento de arreglos de números).

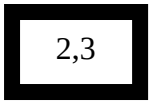
7.3.1 Áreas desalojadas del mismo tamaño

Para el primer caso, se indica (como primeros 4 bytes del archivo correspondiente) la dirección del primer nodo o posición de bitmap disponible. Como las posibles posiciones donde comienzan estos elementos están numeradas, se indica el número correspondiente a la posición desalojada. Cuando se libera un área de disco se agrega la dirección del espacio liberado como primer miembro de la lista enlazada de tales elementos.

Sea $n1$ el número indicado de posición del principio de la lista (el que está grabado en disco). Es un apuntador a un nodo o posición libre. Si se libera otro, se reemplaza $n1$ por el nuevo, $n2$, y se indica, en los primeros bytes del espacio $n2$, el apuntador a $n1$. Cuando se necesita agregar un elemento, se ocupa el lugar indicado por la cabeza de la lista, y se graba como nueva cabeza de la lista enlazada el apuntador que estaba en posición $n1$. Las siguientes figuras ilustran este proceso. Las celdas indican nodos o bitmaps; las marcadas con X están

“ocupadas”. El número de celda está indicado como un par (hilera, columna) para facilitar la comprensión de la imagen.

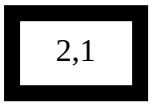
El recuadro muestra la primera posición disponible (2,3). A su vez, esta “posición” apunta a la siguiente libre, que es la (2,1) Si el apuntador en una posición es nulo, lo indicamos aquí con (0,0) es que se acabó la lista enlazada: no hay posiciones disponibles adicionales. En ese caso, se agrega al final del archivo.



	1	2	3	4	5	6
1	**x**	**x**	**X**	**x**	(0,0)	**X**
2	***(1,5)***	**x**	***(2,1)***	**x**	**x**	**X**

Figura 13. La lista enlazada de posiciones libres

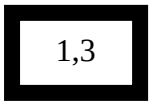
La lista enlazada es (INICIO), (2,3), (2,1), (1,5) (fin)
Si llega un nodo (o bitmap, según el caso, puesto que el dispositivo es el mismo) se colocará en la posición (2,3), y se elimina este elemento de la lista, es decir, se coloca en el cuadro que indica el inicio de la lista el apuntador que estaba en la posición (2,3) es decir, (2,1).



	1	2	3	4	5	6
1	**x**	**X**	**X**	**X**	(0,0)	**X**
2	***(1,5)***	**X**	**X**	**X**	**x**	**X**

Figura 14. Eliminar un espacio “libre” de la lista enlazada

Si se eliminara un elemento que está en posición (1,3), por ejemplo, se cambia por lo siguiente:



	1	2	3	4	5	6
1	**x**	**X**	**(2,1)**	**X**	(0,0)	**x**
2	***(1,5)***	**X**	***X***	**X**	**x**	**x**

Figura 15. Agregar un espacio “libre” a la lista enlazada

7.3.2 Áreas desalojadas de tamaño variable

En este caso, no funciona este dispositivo, puesto que el área a grabar puede ser mayor que el disponible indicado como el siguiente disponible. Por lo tanto, se usa un modo diferente de reusar los espacios. Se construye un árbol B para los espacios libres. La “llave” (de ordenamiento) del árbol es el tamaño (en posiciones, puesto que son enteros los elementos de los arreglos a almacenar) del espacio desalojado. Observe que podrá haber duplicados en este árbol.

Se indica como dato de cada una de estas llaves (en las hojas del árbol) el apuntador a la posición inicial de dicho espacio. Cuando se desocupa un espacio, se agrega el par (cuantas posiciones, donde estaba) al árbol. Cuando haya que grabar un arreglo, se busca el menor de los intervalos libres que no sea menor al espacio necesario, se elimina el elemento del árbol y naturalmente, se graba el arreglo en la posición indicada. Si “sobra” espacio, se agrega un nuevo elemento al árbol de espacios disponibles indicando su tamaño (diferencia de disponible – utilizado) y la posición donde inicia el espacio no utilizado.

7.4 Los archivos en KBC

La mayoría de los archivos que usa el paquete se pueden considerar como archivos lógicos, puesto que pueden estar segmentados en varios archivos. La partición se basará en los criterios aplicables para cada tipo de archivo; por ejemplo, para los archivos en los que se almacenan nodos de árboles podrá ser diferente a los usados para arreglos o intervalos. De hecho, hay dos circunstancias que pueden requerir que un archivo se divida físicamente en varios. Una es el tamaño total, que puede resultar inconveniente para algunas operaciones, típicamente para respaldos o para su traslado de una computadora a otra.

La otra causa para dividir un archivo estriba en el uso de enteros de 4 bytes: con ellos no se puede apuntar a una dirección mayor que $2^{31} – 1$, puesto

que el KBC no usa números negativos para indicar direcciones. Este número, que vale 2, 147,483, 647, si bien es “grande”, puede resultar insuficiente para aplicaciones que envían muchas marcas al KBC. Podría haber un archivo donde un arreglo comienza en un byte cuyo desplazamiento del principio del archivo fuera mayor que dicho número. En ese caso, se divide el archivo en segmentos, por ejemplo, de 500 megabytes. Ahora los apuntadores se manejan por archivo físico, y de ese modo se obvia el problema que causa la cota superior de los apuntadores posibles. De hecho, en KBC no se usará una estructura tipo “list única de instancias” si el valor tiene más de 100,000 instancias (aunque este número es un parámetro que se puede modificar: lo fija la aplicación). En los otros casos, para árboles y bitmaps, a éstos se les apunta por un número que refleja la posición relativa del nodo u hoja, y no la posición física en el archivo. Y no sucederá que un árbol tenga un número de elementos que hagan que se rebase este número: los nodos de los árboles son de 40 bytes, las hojas de 44 bytes, y los bitmaps también tendrán un tamaño considerable, digamos 1Kb cada uno.

CAPÍTULO 8. APLICACIONES DEL KBC QUE INFLUYERON EN EL DISEÑO DEL PAQUETE

Además del paquete Marte, mencionado anteriormente como la primera aplicación del KBC, y que de hecho motivó la implementación del mismo en su forma inicial, hubo otras dos aplicaciones que indicaron funciones adicionales que debería ofrecer el paquete. Una de ellas es un manejador de bases de datos, que por ahora tiene el nombre DBB. No se ha publicado como tal, pero algunos de sus aspectos están siendo investigados e implementados por un equipo de investigadores del Colegio de Postgraduados y se incluirán en una tesis futura.

Se trata de un manejador de bases de datos híbrido; no es relacional y está basado precisamente en el uso de palabras clave por contexto. Se puede decir que aprovecha todo lo que ofrece el KBC. Almacena registros lógicos, denominados UBI (unidad básica de información) cada uno de los que pertenece a una de las clases definidas para una instancia del DBB. Las UBI no se almacenan como registros físicos, como en una base de datos, sino en segmentos por tipo de datos; por ejemplo, los enteros de 2 bytes se guardan por separado, y para las cadenas de caracteres sucede lo mismo. Las clases se pueden considerar como las equivalentes a las tablas de una base de datos, pero difieren en un aspecto fundamental: todas se gestionan como parte de un archivo (lógico) único, y las UBI de cada clase se numeran con un número único como parte del conjunto de todas ellas, no por clase como sucede en una base de datos relacional, y en muchas otras. Las clases se definen con un proceso análogo al del ofrecido por un DDL típico, pero se pueden incluir como “campos” otro tipo de elementos: objetos, vectores o estructuras de diversos tipos como vectores paralelos (los valores de los vectores están asociados a los de la misma posición en los otros vectores de la estructura). No permite índices; en su lugar, se pueden marcar (por contexto) los campos de una clase. Varios campos de una clase se pueden marcar con el mismo contexto, y un mismo contexto puede ser usado por diversas clases. Eso dio lugar a la designación del DBB como un sistema de gestión de datos

generalizado e híbrido. El DBB inspiró numerosas funciones incluidas sucesivamente en el KBC, en particular, la actualización “masiva” de marcas de un par (contexto-valor). También se vio reflejado en la necesidad de eficientar el almacenamiento de las marcas, lo que resultó en la definición del objetivo de la investigación descrita en este trabajo de tesis.

La otra aplicación que sugirió muchos cambios al KBC es la que hemos denominado METINDX, que está todavía en etapa de diseño, puesto que se apoya a su vez en el DBB. Permite la construcción de un conjunto de apuntadores a elementos heterogéneos, especialmente bases de datos que difieren entre sí en cualquier aspecto, pero también a otras colecciones de datos. Fue concebido como alternativa a una bodega de datos.

El cambio más significativo es que ahora se podrán incluir marcas virtuales: se indica una palabra clave (naturalmente por contexto) para una UBI, pero esa palabra no está incluida como campo en los registros. La inclusión de estas marcas permitirá a la aplicación armar listas de resultados sin requerir la presencia de las marcas involucradas en los archivos de la bodega de datos. El problema que presentan estas marcas virtuales es que no se pueden modificar como las “normales”: en estas, cuando cambia un valor, se da de baja la marca anterior y se agrega una nueva. Cuando la marca es virtual, el valor “anterior” no está disponible como campo, de modo que o no se puede hace la operación o se obtiene el valor anterior por inferencia o de una fuente externa, por ejemplo, del proceso de actualización. De este modo, puede ser que haya una marca equivocada registrada en KBC: apunta a un registro al que no debe apuntar. Esta situación se puede remediar por la aplicación, por ejemplo, determinar que un registro no debería estar en un subconjunto armado vía sus marcas, o ejecutando un proceso carísimo (en tiempo de proceso): se busca la instancia en todos los valores del contexto y se la elimina cuando se la encuentra. A pesar de que KBC tiene ese método, no se recomienda su uso, particularmente para contextos que pueden tener muchos valores.

CONCLUSIONES

El objetivo del proyecto era producir una versión eficiente del paquete de nominado KBC, descrito en el trabajo. En particular, se trataba de determinar las estructuras de datos para almacenar las marcas enviadas por una aplicación en disco, de tal modo que tanto las actualizaciones de las marcas como su uso para el armado de listas de resultados minimizara las operaciones, especialmente las de entrada y salida, pero también las que se realizan en memoria. Se reemplazaron los árboles B híbridos de la versión anterior, en los cuales las hojas apuntaban a arreglos de instancias almacenados en disco, por intervalos de bitmaps, que harán más eficientes los procesos que los usan.

Adicionalmente, se tenían que implementar los métodos para la creación y almacenamiento de listas de resultados en los tres tipos de lista definidos para el paquete, las llamadas Lista-L (una arreglo de instancias), Lista-I (una arreglo de bitmap intervalos) y Lista-B, que es un caso particular de una Lista-I (con un solo intervalo.)

Se elaboraron todos los diseños, y los algoritmos de manejo de espacio en disco para reutilización de áreas no utilizadas de los archivos. Se crearon algoritmos eficientes para efectuar las operaciones entre listas de resultados de acuerdo al tipo de lista que resultará de la operación, mismo que se determina a partir de la naturaleza de la operación y los tipos de listas de sus operandos. Se creó una aplicación – ficticia -, y con ella se probaron los métodos que ofrece el KBC.

Como resultado de la investigación, se determinó que las listas de marcas que usaba el KBC eran casos particulares de los índices GIN (generalized inverted index) que aparecieron después de la creación de la versión anterior del KBC. Por lo tanto se incorporó la nomenclatura como GIN, aunque con la generalización que resulta del uso de los árboles B híbrido.

La investigación futura incluye especialmente aspectos de concurrencia en actualizaciones multi-usuario. Los mecanismos que incluye el KBC son eficaces, pero consumen relativamente muchos recursos, comparados con los que usan los procesos sustantivos.

También se planea reemplazar el proceso que ejecuta una fórmula, que ahora efectúa las operaciones una tras otra, para contemplar la agrupación de varias operaciones similares que resultarán en procesos más eficientes.

REFERENCIAS

Adabas Comprehensive Data Management System. 2010. ADABAS [Base de datos]. Disponible en: http://www.softwareag.com/corporate/products/adabas _ 2010/default.asp

Bain T. 2010. Data Analytics for the Masses [Base de datos]. Disponible en: http://www.readwriteweb.com/enterprise/2009/01/data-analytics-for-the-massesp2.php

Bartunov O. y Sigaev T. 2006. PostgreSQL Summit, Toronto, July 8-9 [Base de datos]. Disponible en: http://www.sigaev.ru/gin/Gin.pdf

Bauer Mengelberg J. R. 2007. "The concept of an unstuctured book and the software to publish and read it", Information and Beyond: Part II Part II: Journal of Issues in Informing Science and Information Technology, Vol 4. Santa Rosa, CA: Informing Science Press, pp. 801-810.

Bayer R. y McCreight E. 1972. "Organization and Maintenance of Large Ordered Indices", Acta Informatica Vol. 1 Fasc. 3, pp. 173-189.

Big Table. 2010. [Base de datos]. Disponible en: http://en.wikipedia.org/wiki/Big Table

Codd E.F. 1970. A Relational Model of Data for Large Shared Data Banks. IBM Research Laboratory, San Jose, California, Communications of the ACM Volume 13 / Number 6 / June, pp 377-287.

Comer D. 1979. "The ubiquitious B-tree", Computing Surveys, VoL ll, No 2 11, pp. 121-137.

Conway D. 2010. Latest Teradata Database Release Supports Big Data and the Convergence of Advanced Analytics [Base de datos]. Disponible en: http://www.teradata.com/t/News-Releases/2010/Latest-Teradata-Database-Release-Supports-Big-Data-and-the-Convergence-of-Advanced-Analytics/

Cruz Millán M. 2003. "Desarrollo de una interfase para establecer criterios para la recuperación de información a partir del uso de claves", Tesis de maestría. Colegio de Postgraduados. México.

Diaz S. 2010. Greenplum pushes enterprise data cloud with new releases [Base de datos]. Disponible en: http://www.zdnet.com/blog/btl/greeplum -pushes-enterprise-data-cloud-with-new-releases/32965

Frost S. 2010. DATA-Beat - A DATAllegro Blog [Base de datos]. Disponible en: http://www.beyeblogs.com/DATAllegro/archive/2008/ 04/who_i_am_and_why_im_here.php

Greenwald R., Stackowiak R. y Stern J. 2004. Oracle Essentials: Oracle Database 11g. Boston, MA: O'Reilly.

Howard P. 2010. Netezza surprises with technical capabilities [Base de datos]. Disponible en: http://www.theregister.co.uk/2006/10/03/netezza_annual_ conference_roundup/

IBM. 2010. Data Warehouse InfoSphere Warehouse: Insight without Boundaries [Base de datos]. Disponible en: http://www.ibm.com/us/en/

Inmon W.H. 2002. Building the Data Warehouse. NY: Wiley Computer Publishing.

Infobright open source data warehousing. 2010. Infobright Enterprise Edition [Base de datos]. Disponible en: http://www.infobright.com/Products/Features/

Null L. and Lobur J. 2006. The essentials of computer organization and architecture. n.p.: Jones and Bartlett, pp. 741-742.

Ozbutun C., Cohen J. I., Depledge M., Hyde J., Jakobsson H., Kremer M. yTranQuery Q. T. 2000. Processing Using Compressed Bitmaps. (Oct. 31). Patent 6, 141, 656 [Base de datos]. Disponible en: http://www. freepatentsonline.com/6141656.pdf

Rob P. y Coronel C. 2009. Database Systems: Design, Implementation, and Management. Boston, MA: Course Technology, pp. 90-91.

Shapiro J. 2006. Microsoft SQL Server 2005: the complete reference. NY: Mc Graw Hill, pp. 54-55.

Sussna S. 2008. Defeat and Triumph: The Story of a Controversial Allied Invasion and French Rebirth. Jerusalem, Israel: Xlibris Corporation.

The PostgreSQL Global Development Group. 2009. PostgreSQL 8.4 Official Documentatio, Volume V. Linbrary, pp. 161-162.

Weiss M. A. 1993. Data structures and Algorithm Analysis. Reedwodd City, CA: The Benjamin Cummings, pp. 133-138.

ANEXO I SISTEMA ELABORADO PARA LA PRUEBA DE LOS MÉTODOS DE LA CLASE KBC

1.1 Descripción General

1.1.1 El ejemplo

Se creó un archivo con 999,999 descripciones de los números enteros del 1 al 999,999. A pesar de que se creó el registro para el número 1, no se usa).

Cada registro tiene el "tipo" (la estructura)

- descripcion-5 as string * 5
- descripcion-55 as string * 55
- los_primeros_factores as string * 36
- un indicador está marcado o no (las primeras 5 letras)

donde la descripción del número 96 es "noventa y seis".

Se incluyen solo los primeros factores primos (hasta llenar los 36 caracteres).

Observe que:

- Si el número es primo, en los-primeros-factores dice "es primo"
- Para cada número, se generan marcas virtuales para el contexto #1: sus factores primos.
- No se genera el 1 (evidentemente)
- Para los números mayores de 100,000 no se incluyen en "los_primeros_factores" los primos menores que 203.
- Se pueden marcar (con el contexto #2) las primeras 5 letras del nombre del número, pero estas marcas se introducen en forma "manual".

MARCA VIRTUAL; se refiere a un dato que no está físicamente en el registro, sino que está asociado por algún motivo, criterio, etc. en general durante la creación del acervo, pero se puede invocar cuando sea conveniente.

1.1.2 Los contextos

Los contextos que se incluyeron para el ejemplo son:

1) factores primos
2) primeras 5 letras de la descripción.

Ejemplos de valores marcados.

1) 2, 3, 5, 7, 11, 13, 17 19, etc. los números primos del 2 al 499.979
(es el mayor primo menor que 500,000).
2) Siete, cuatr, cuare, dos m, etc.

1.1.3 Tipos de estructura

Los tipos de estructura que se usan (dependen del contexto) son:

Contexto 1 - factores primos: sus valores: usa un ARREGLO ÚNICO

Contexto 2 – primeras 5 letras: usa ÁRBOLES B

1. 2 El programa que simula la aplicación

OBSERVACIÓN; este programa está como Anexo 2 (en Visual Basic)

1.2.1 La generación del acervo

Se crea un archivo (se usará con Acceso RANDOM)

1. NNN = 999,999 ‘ *para poder generar más o menos números*
2. Para num = 1 to NNN
    a. Se genera la descripción del numero (protección de cheques)
    b. Se incluye en el registro como un campo de 60 caracteres
    c. Se determinan sus factores primos
        i. Para cada factor primo, se invoca el marcado (contexto 1)
    d. Se incluyen los factores primos separados por un espacio en el campo correspondiente (hasta llenar los 36 bytes)
    e. Si el número es primo, se indica “es primo” en este campo
3. Graba el registro en el archivo.

1.3 Marcado del acervo

Se presentan dos formas de marcado que son:

- La primera es automática y es la que se utilizó al generar los datos y las marcas necesarias para poder realizar consultas.
- La segunda se invoca indicando un número y marcando los elementos que correspondan con su descripción-5.

1. 4 Consultas

Ver un número (cuántos factores primos tiene y los primeros de éstos)

Durante una sesión de uso del acervo:

Se crean listas de resultados (temporales.) Se destruyen cuando inicia una sesión los que se habían generado anteriormente. Sin embargo, se pueden guardar, para los cual se les cambia el nombre.

Cuando se hace una consulta, se agrega una lista a este conjunto.
Se pueden eliminar listas del conjunto si se desea.
Hay 2 tipos de consulta:

- Obtener la lista de resultados de un par (contexto, valor)
- Indicar una fórmula (en esta aplicación, sólo se pueden incluir varios operandos con una única operación).

Ver una lista de resultados (mostrar las instancias de la lista).

Si hay más de 200, solo se muestran los números de instancia (son los números) en una matriz paginada, pero se pueden ver los datos del número en forma individual. Si hay menos de 201 instancias en la lista, se muestran los números incluyendo la descripción de cada número y sus (primeros) factores primos.

Observación: el programa muestra las duraciones de una consulta (el armado de la lista de resultados) desde el momento en que se la solicita hasta el instante en que la lista queda grabada en disco.

ANEXO II EL PROGRAMA QUE GENERÓ EL ACERVO DE DATOS

Este programa se incluyó con una carpeta en el CD de la tesis.
La carpeta se llama “Prog Genera Acervo Números”, que a su vez está en la carpeta “ANEXOS DE LA TESIS”