MÉTODOS DE ACCESO PARA LAS BASES DE DATOS HISTÓRICAS

Juan Fernando Vélez*

RESUMEN

Los sistemas que soportan el atributo de *tiempo de validez* de las tuplas de las relaciones son llamados Bases de Datos Históricas. Ellos permiten realizar operaciones históricas sobre la base de datos tales como la búsqueda, la inserción y el join. Estas operaciones necesitan de nuevos métodos de acceso que permitan satisfacer consultas con predicados que involucran el atributo de tiempo de validez. En este trabajo se presenta el estado del arte en los métodos de acceso temporales y luego se realiza un análisis de rendimiento del método más reciente: el índice temporal.

INTRODUCCIÓN

Los SGBD convencionales carecen de capacidad para almacenar la información histórica concerniente a las aplicaciones del mundo real. Estos sistemas solo dan una *visión instantánea* (última actualización) de los datos. Además, la actualización de las tuplas de una relación se hace "*sobre el sitio*" destruyendo la información existente.

La utilización de los SGBD convencionales no permiten informarse sobre el pasado ni tratar una secuencia de datos históricos. Si uno de estos sistemas dispone de un soporte temporal, los usuarios podrían realizar consultas históricas, operaciones de recuperación después de fallas, etc. Por tanto, es necesario concebir nuevos métodos de acceso y almacenamiento para relaciones temporales capaces de responder eficazmente a las consultas históricas.

En este trabajo se presenta: primero los conceptos de base concernientes a la integración de las dimensiones temporales con los SGBD Relacionales y segundo, los métodos de acceso temporales: Encadenamiento hacia atrás, Acceso por lista y el Índice temporal. Por ultimo, se realiza un análisis del rendimiento de la operación de búsqueda y de las consultas históricas con el método del Índice Temporal.

* **Docente Facultad de Ingeniería. Universidad de Antioquia. Centro de Investigación en Informática Universidad de París 1 - Sorbonne**

1. EL TIEMPO EN LAS BASES DE DATOS

La dimensión temporal puede ser representada como un conjunto de puntos de tiempo (instantes) consecutivos y equidistantes [GY88], denotado T = { 0, 1, ..., *now*} donde *0* representa el punto de partida de la aplicación real y *now* el tiempo corriente. La escala entre dos instantes de tiempo consecutivos puede ser: meses, días, horas, minutos, segundos.

Se define un *intervalo de tiempo* llamado *periodo de vida* (*lifespan*), denotado $[t_s, t_e]$, como un subconjunto de T donde t_s es el tiempo inicial y t_e es el tiempo final del intervalo. Un punto discreto de tiempo t es representado como un intervalo de tiempo $[t, t]$ o simplemente $[t]$.

Si se tienen dos intervalos de tiempo pertenecientes al intervalo universal T = [*0, now*], se pueden definir las operaciones entre conjuntos: unión, intersección, diferencia y el complemento de T. Se pueden también definir los predicados de comparación entre dos intervalos de tiempo utilizando:

$$=,\ \neq,\ \supseteq \text{ y } \subseteq.$$

Clasificación de las Bases de Datos

La taxonomía del tiempo en las Bases de Datos definida por Snodgrass [SA85] permite clasificar los SGBD con respecto su capacidad para soportar las dimensiones temporales.

El *tiempo de validez* (*valid time*) es el intervalo de tiempo durante el cual la información almacenada en la Base de Datos es valida en el mundo real. Esta dimensión permite informarse sobre el pasado o anticiparse en el futuro.

El *tiempo de transacción* (*transaction time*) da cuenta del momento en que la información es registrada en la Base de Datos. Esta dimensión traduce la historia de las modificaciones sobre la Base de Datos.

Por tanto, hay cuatro tipos de Bases de Datos:

1) Los SGBD clásicos que no soportan el tiempo de validez ni el tiempo de transacción, son llamados Bases de Datos Estáticas (*snapshop databases*). Estos sistemas solo dan una visión *instantánea* de los datos de la aplicación real. Es decir, la ultima modificación sobre la Base de Datos trayendo como consecuencia la destrucción de la información existente.

2) Cuando el SGBD soporta el *tiempo de validez* se habla de una Base de Datos Histórica (*historical databases*).

3) Cuando se dispone de un soporte para el *tiempo de transacción*, se habla de Bases de Datos Retrospectivas (*rollback databases*).

4) Finalmente, si el SGBD soporta simultáneamente las dos dimensiones temporales, se habla entonces de Bases de Datos Bitemporales (*bitemporal databases*). Ellas permiten realizar operaciones de actualización retroactivas (corrección de errores) en los datos históricos.

2. ALMACENAMIENTO TEMPORAL COMPARTIDO

En los SGBD temporales, es difícil mantener almacenados grandes volúmenes de información histórica en una sola zona de memoria-disco [AS88]. La respuesta a consultas que implican la recuperación de datos históricos es ineficaz. Por tanto, el objetivo de esta sección es estudiar el *almacenamiento temporal compartido,* esta técnica consiste en almacenar los datos corrientes en una *zona corriente* y los datos históricos en una *zona histórica*. Se presentaran, en particular, las estructuras de almacenamiento para la zona

histórica: Encadenamiento hacia atrás y Acceso por lista.

Para facilitar la presentación del tema, se utilizara el modelo de almacenamiento basado en registros que soportan versiones de objetos [EWK90]. Cada registro es una versión de objeto compuesta de atributos clásicos mas un atributo intervalo $[t_s, t_e]$ llamado *tiempo de validez* donde t_s y t_e son el *tiempo de inicio* y de *fin de validez* respectivamente. Se considera que por un objeto e_i hay un conjunto de versiones $\{ e_{i1}, e_{i2},...,e_{ik} \}$, denotado e_{ij}. Se dice que una versión e_{ij} de un objeto e_i es corriente si y solamente si $e_{ij}.t_e$ = now.

El almacenamiento temporal compartido esta compuesto de dos zonas: la *zona de almacenamiento corriente* y la *zona de almacenamiento histórica.* La zona corriente contiene las versiones corrientes que satisfacen todas las consultas no temporales y la zona histórica guarda les versiones históricas que permiten responder a las consultas temporales. Esto permite mejorar el tiempo de acceso a las versiones de objetos.

Es posible utilizar los métodos de acceso convencionales para las dos zonas de almacenamiento tales como el Haching o los Arboles B. Pero, la zona histórica requiere la utilización de nuevas técnicas que permitan explotar la dimensión temporal: tiempo de validez.

2.1. Encadenamiento hacia atrás

Cuando se tienen consultas temporales sobre la Base de Datos, es necesario recuperar las versiones de objeto históricas. Para llevarlo a cavo evitando el barrido de toda la zona histórica, la primera técnica propuesta fue el *Encadenamiento hacia atrás*.

Todas las versiones de objetos e_{ij} de un objeto e_i son almacenados en *"orden inverso"*; siendo la primera, la *versión corriente* (ver figura 1). Una vez que la versión corriente es almacenada en la zona corriente, sus ancestros pueden ser recuperados sin barrer toda la zona histórica [BZ82].

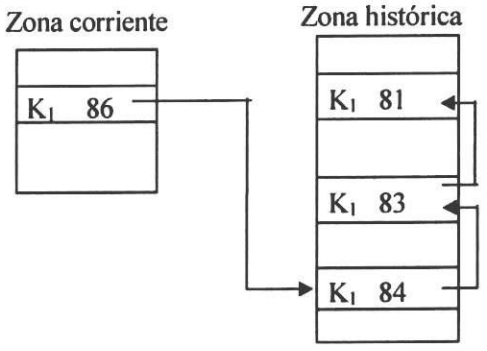

Fig 1. Encadenamiento hacia atrás

A cada *tupla* se le adiciona un campo *nvp* (apuntador hacia la próxima versión). Cuando se inserta la primera tupla en la relación, ella es almacenada en la zona corriente y el valor de *nvp* es puesto en *null*. Cuando la tupla es remplazada, la versión existente en la zona corriente es desplazada hacia la zona histórica y la nueva versión es guardada con el valor del campo *nvp* apuntando hacia la versión precedente. Este esquema permite conservar la cadena de la versión mas reciente a la mas vieja, y de mantener el orden cronológico de versiones en la zona histórica.

Para la operación de recuperación, la versión corriente puede ser localizada utilizando una estrategia de acceso disponible para la zona de almacenamiento corriente. Si la consulta es temporal, el campo *nvp* es examinado. Si *nvp* es *null* o la consulta es no temporal, no es necesario ir hacia la zona histórica. En el caso contrario, todos sus predecesores pueden ser encontrados

recorriendo la cadena de apuntadores hasta encontrar una versión con el campo *nvp* igual a *null*.

2.2. Acceso por lista

En la técnica de *Encadenamiento hacia atrás*, si la longitud de la cadena histórica crece mucho, su recorrido puede ser muy costoso, incluso cuando las consultas involucran una parte de la cadena histórica. Una alternativa es la de utilizar una *lista de acceso* entre la *zona corriente* y la *zona histórica* (ver figura 2).

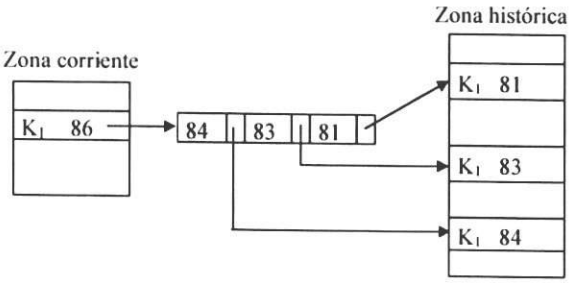

Fig 2. Acceso por lista

Con esta técnica, la primera tupla es insertada en la relación con un campo adicional *alp* (apuntador a una lista de acceso) puesto en *null*. Cuando una nueva versión reemplaza la versión corriente, la nueva versión es almacenada en la zona corriente con el valor del campo *alp* apuntando a la *lista de acceso*. Esta lista es también inicializada para que haga referencia a las versiones históricas después de la primera inserción en la zona histórica. Si otra versión de objeto es adicionada en el conjunto de versiones entonces se adiciona una entrada correspondiente a dicha versión en la *lista de acceso.*

Es deseable almacenar la información temporal (atributo de tiempo de validez) en cada entrada de la lista de acceso. Esto permite la evaluación de los predicados temporales sin acceder a las versiones históricas. Cuando los atributos temporales son almacenados en la lista de acceso, no es necesario guardar esta información en la zona histórica. En el *acceso por lista*, las versiones históricas no necesitan un apuntador adicional *nvp* como en el *encadenamiento hacia atrás*.

2.3.Operaciones de modificación, supresión y creación

Se considera un intervalo de búsqueda $I_s = [t_a, t_b]$, la operación de búsqueda de una versión de objeto e_{ij} es realizada sobre el conjunto $\{e_{11}, e_{12},... e_{1m}, e_{21}, e_{22}, ... e_{2m}, ..., e_{n1}, e_{n2},..., e_{nm}\}$ verificando que la intersección de $[e_{ij}.t_s, e_{ij}.t_e]$ y I_s sea no vacía.

Esta búsqueda puede utilizar una de los métodos de acceso ya estudiados (*encadenamiento hacia atrás y acceso por lista.* Por tanto, el costo de las operaciones de modificación, supresión y creación de una versión de objeto depende del algoritmo de búsqueda utilizado.

Operación de modificación

Cuando un objeto e_i es modificado por nuevos valores de atributos, la versión corriente e_{ij} de este objeto se convierte en la versión pasada mas reciente y una nueva versión $e_{i(J+1)}$ será creada para este objeto.

La operación de modificación es efectuada de la manera siguiente: sea t_u el tiempo de modificación de una versión de objeto,

Update(e_{ij})
{ $e_{ij}.t_e$ <— t_u -1;

crear una nueva versión de objeto $e_{i(j+1)}$ a partir de e_{ij} : $e_{i(j+1)}$ <— e_{ij} ;

for each (atributo convencional modificado A_i) **do**

$e_{i(j+1)}.A_i$ <— nuevo valor de atributo;

$e_{i(j+1)}.t_s$ <— t_u ;

$e_{i(j+1)}.t_e$ <— now ;

}

Operación de supresión

Se dice que una Base de Datos Temporal esta en modo de "***adición***" cuando las viejas versiones de objetos no son nunca suprimidas. La operación de supresión en el tiempo t_d es la siguiente:

Delete(e_{ij})

{ encontrar la versión corriente e_{ij} del objeto e_i ;

$e_{ij}.t_e$ <— t_d ;

}

Operación de creación

La operación de creación de una versión de objeto e_{i1} en el tiempo t_i es la siguiente:

Insert(e_{i1})

{ crear una versión inicial e_{i1} para e_i ;

$e_{i1}.t_s$ —> t_i ;

$e_{i1}.t_e$ —> now ;

}

3. EL ÍNDICE TEMPORAL

El objetivo de esta sección es presentar el *índice temporal* [EWK90], método que permite mejorar el rendimiento de cierto tipo de consultas temporales.

En las Bases de Datos Temporales, se tienen relaciones con atributos que son intervalos de tiempo (*tiempo de validez*). Es necesario entonces definir las técnicas para indexar los puntos de tiempo pertenecientes a tales intervalos. Es posible utilizar técnicas de indexación tradicionales como el Haching o los arboles B^+ en la indexación temporal, pero esto es muy difícil porque:

a) Los intervalos de validez de las versiones de objetos se solapan arbitrariamente, esto no permite definir un orden total sobre los valores de los intervalos.

b) Por la naturaleza, *en adición*, de las Bases de Datos Temporales. La mayoría de operaciones son operaciones de modificación que ocasionan la inserción (adición) de una nueva versión de objeto en la Base de Datos incrementando el valor del tiempo de fin de validez de las tuplas. Además, en estas Bases de Datos la supresión de versiones de objetos se hace raramente.

3.1. Descripción del índice temporal

El hecho de tener solapamientos de intervalos de tiempo no permite definir un orden total sobre los valores del índice. Con el fin de resolver este problema, se propuso la idea de utilizar un conjunto de *puntos de indexación* ordenados linealmente en el eje del tiempo [EWK90].

Un punto de indexación es definido por:

- el tiempo inicial de validez de cada versión de objeto,
- el punto inmediato que sigue al tiempo de fin de validez de cada versión de objeto,
- el tiempo corriente *now*.

Formalmente, se define el conjunto BP de todos los puntos de indexación como

$$BP = \{ t_i / \exists\, e_{ij} \in TDB\ ((t_i = e_{ij}.t_s)\ v\ (t_i = e_{ij}.t_e + 1)) \} \cup \{ now \}$$

Nom	Depto	Valid_Time
emp1	A	[0, 3]
emp1	B	[4, now]
emp2	B	[0, 5]
emp3	C	[0, 7]
emp3	A	[8, 9]
emp4	C	[2, 3]
emp4	A	[8, now]
emp5	B	[10, now]
emp6	C	[12, now]
emp7	C	[11, now]

Fig 1. Relación Temporal EMPLEADO

En la figura 2, se ilustra el concepto de puntos de indexación por la relación EMPLEADO mostrada en la figura 1. EL conjunto BP esta compuesto de nueve puntos de indexación,

$$BP = \{ 0, 2, 4, 6, 8, 10, 11, 12, now \}$$

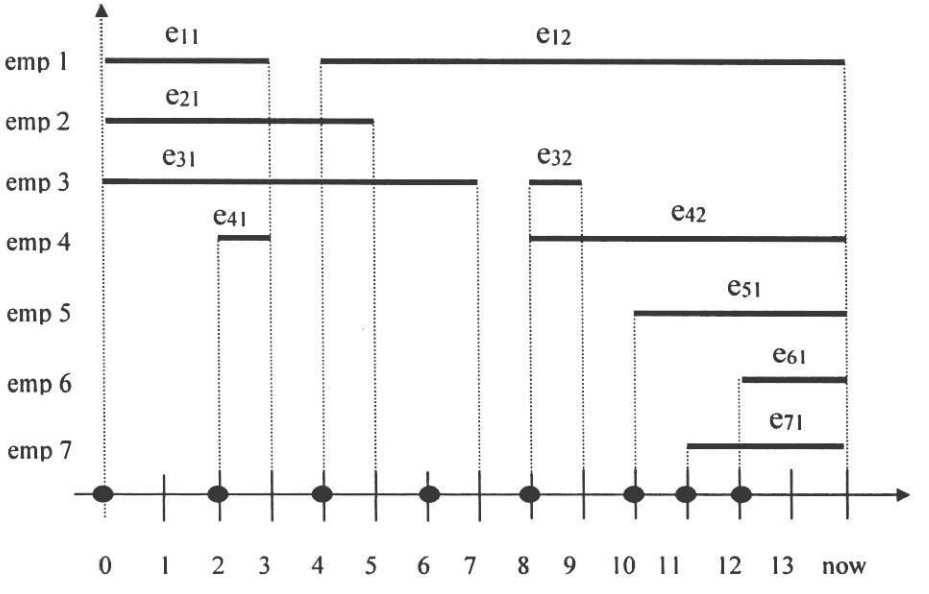

Fig 2. Puntos de Indexación

Sea t_j un punto arbitrario perteneciente (o no) al conjunto BP.

Se define t_j^- (t_j^+) como un punto en BP que se encuentra inmediatamente antes (después) del punto t_j.

Se define también t_j^- como

$$t_j^{-=} = \begin{cases} t_k & si \ \exists \ t_k \in BP \ / \ t_k = t_j \\ t_j^- & sino \end{cases}$$

Como todos los puntos de indexación t_i en BP son totalmente ordenados, entonces es posible utilizar un árbol B^+ [Com79] para indexar estos puntos de tiempo. Cada entrada de un nodo hoja es de la forma [t_i, *paquete*] donde *paquete* es un apuntador a las versiones de objetos tales que su *tiempo de validez* contiene el intervalo [t_i, t_i^+-1].

Mas formalmente, esta propiedad puede ser expresada como

$B(t_i) = \{ e_{ij} \in TDB \ / \ ([t_i, t_i^+-1] \subseteq e_{ij}.valid\_time) \}$

La figura 3 muestra el árbol B^+ de orden 3 que permite indexar el conjunto de puntos de tiempo concernientes a las versiones de objeto de la relación EMPLEADO. Cada nodo del árbol contiene al menos dos valores y tres apuntadores.

Por ejemplo, la entrada de la hoja para el punto de tiempo 4 se calcula como

$B(4) = \{ e_{ij} \in TDB \ / \ ([4,5] \subseteq e_{ij}.valid\_time) \} = \{ e_{12}, e_{21}, e_{31} \}$

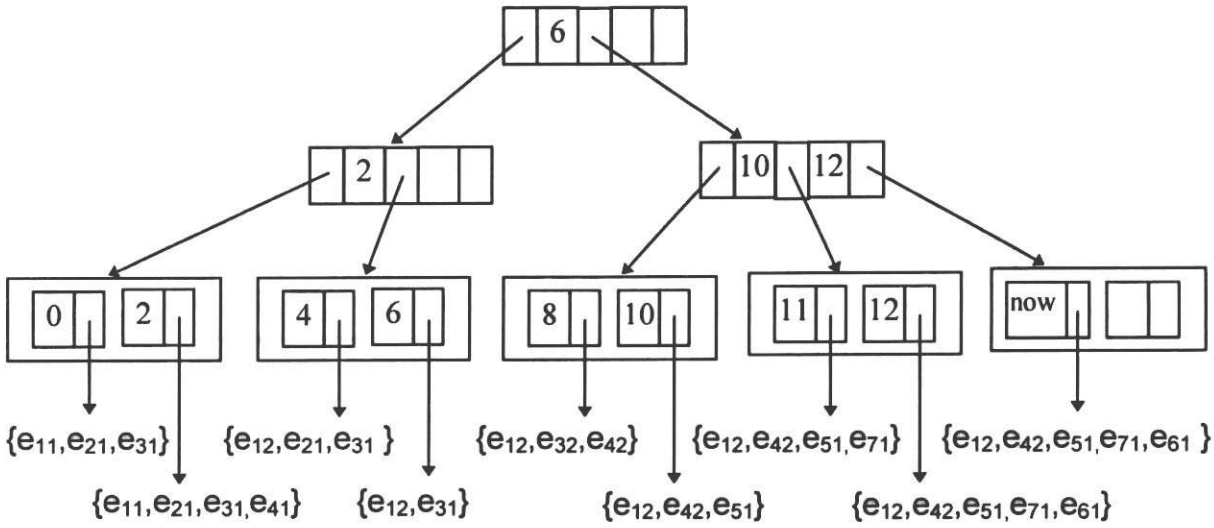

Fig 3. Indice Temporal

El problema del índice temporal se presenta cuando se tienen paquetes con un gran numero de versiones de objeto. Varias versiones de un paquete pueden repetirse en los paquetes siguientes. Por ejemplo, la versión de objeto e_{21} aparece en varios paquetes consecutivos. Para resolver este problema, apareció la idea de utilizar el índice incremental.

3.2. El Índice Incremental

El *índice incremental* es utilizado para reducir la redundancia de versiones de objeto existentes en los paquetes permitiendo así reducir su tamaño. El principio de esta solución es el siguiente:

El paquete referenciado por la primera entrada de cada nodo hoja del árbol es conservado lleno. En los paquetes referenciados por las entradas siguientes, solo se guardan los cambios increméntales.

Por ejemplo (ver figura 4), la entrada al punto 10 almacena $\{+e_{51}, -e_{32}\}$ en su paquete incremental. Esto indica que e_{51} comienza en el punto de indexación 10 y que e_{32} termina en el punto inmediatamente antes de 10.

El paquete incremental $B(t_i)$ referenciado por una entrada (que no es la primera) en el tiempo t_i puede ser calculado como

$$B(t_i) = B(t_1) \cup \left(\bigcup_{t_j \in BP,\ t_1 < t_j \le t_i} SS(t_j) \right) - \left(\bigcup_{t_j \in BP,\ t_1 < t_j \le t_i} SE(t_j) \right)$$

donde

- $B(t_1)$ es el paquete referenciado por la primera entrada del nodo hoja en el cual el punto t_i es localizado,
- $SS(t_j)$ es el conjunto de versiones de objeto que tienen un tiempo inicial t_j, y
- $SE(t_j)$ es el conjunto de versiones de objeto que tienen un tiempo final $t_j - 1$.

Por ejemplo, el paquete incremental para el punto de tiempo 6 es calculado como

$$B(6) = B(4) \cup \left(\bigcup_{tj \in BP,\ 4 < tj \le 6} SS(tj) \right) - \left(\bigcup_{tj \in BP,\ 4 < tj \le 6} SE(tj) \right)$$

$$= \{ e_{12}, e_{21}, e_{31} \} \cup SS(6) - SE(6) = \{ e_{12}, e_{21}, e_{31} \} \cup \Phi - \{ e_{21} \}$$

$$B(6) = \{ e_{12}, e_{21}, e_{31} \} - \{ e_{21} \}$$

La figura 4 muestra el índice incremental de la relación EMPLEADO.

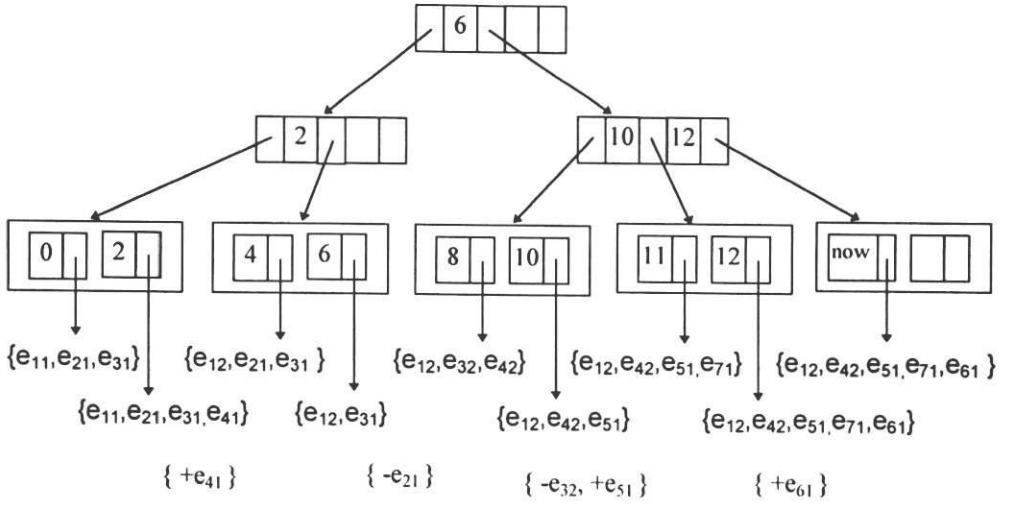

Fig 4. Indice Incremental

3.3. Algoritmo de búsqueda

Sea $I_s = [t_a, t_b]$ el intervalo de búsqueda. El algoritmo calcula primero el conjunto de puntos de tiempo en BP correspondientes a I_s:

$$S(I_s) = \{ t_i \in BP \ / \ t_a \leq t_i \leq t_b \} \cup \{ t_a^{-} \}$$

Para calcular el conjunto de versiones $R(I_s)$. Se recorre el árbol B^+ de la raíz a un nodo hoja que contiene el punto *a*. Luego, se continua el recorrido secuencial de las hojas (hacia la derecha) hasta llegar a una hoja que contenga el punto *b*.

Formalmente, se tiene $R(I_S) = \bigcup_{t_i \in S} B(t_i)$.

4. ANÁLISIS DE RENDIMIENTO

En esta ultima parte, se propone un modelo de evaluación del índice temporal con la variante incremental [EWK90]. El análisis se hará con el enfoque probabilista de colas de espera [GLOT96].

Primero, se consideran los parámetros del sistema con el fin de calcular el grado de un nodo del árbol. Se estimara el costo de almacenamiento de una relación temporal que es medida por el numero de paginas de disco utilizadas para almacenar el archivo. Luego, se estimara el costo de consultas sobre un intervalo de tiempo [a, b].

4.1. Parámetros del sistema

Con el fin de efectuar las estimaciones, se dispone de los siguientes parámetros del sistema:

- el tamaño de un paquete (una pagina disco), denotado B, es igual a 4 Koctetos,
- el tamaño de un apuntador, denotado p, es igual a 32 octetos,
- el tamaño de un entero, denotado t, est igual a 8 octetos. Se supone que los puntos de tiempo y las coordenadas sobre los ejes vertical y horizontal son enteros.

En general, un árbol B^+ de grado m tiene máximo 2m entradas. El grado del árbol puede ser estimado fácilmente sabiendo que el tamaño de un paquete es igual a la suma de los tamaños de las entradas de un nodo. Es decir:

B = 2m (talla_clave + talla_apuntador) + talla_apuntador.

Para el índice temporal que tiene entradas (t, p) donde t es un punto de tiempo y p un apuntador. Se tiene la ecuación $2m_e(p+t) + p = B$ que da un grado $m_e \cong 50$, es decir, cada nodo del árbol tiene al menos 50 y máximo100 entradas.

4.2.Costo de almacenamiento de una relación temporal

Cuando un árbol B^+ es utilizado como índice, es posible estimar el espacio de memoria en disco para almacenar una relación de N tuplas.

Sea S el numero total de paquetes necesarios para almacenar N tuplas. Utilizando el árbol B^+, se tiene $S = S^i + S^b$

donde

- S^i es el numero de paquetes utilizados por la estructura del índice y

- S^b es el numero de paquetes necesarios para almacenar N tuplas.

Con el fin de estimar el costo de almacenamiento de S^i, se supone que la inserción de tuplas en el árbol B^+ sigue un proceso de Poisson. Por tanto, el tiempo entre llegadas sigue una ley exponencial de media $1/\lambda$. Se supone también que el tiempo de

servicio (o tiempo de validez) es uniformemente distribuido sobre el intervalo [0, 2μ] con una duración media μ.

Primeramente, se define el modelo de costo de almacenamiento del índice temporal. El índice es construido sobre el conjunto ordenado de puntos de indexación BP, el tamaño del índice depende del numero de elementos distintos de BP, denotado | BP|.

Para calcular |BP|, se define X como una variable aleatoria Poissoniana que representa el numero de llegadas por unidad de tiempo. Es decir, la variable X da el numero de tuplas a insertar en el arbol B^+. Por tanto, se tiene

$$| BP| = 2N \times Pr(X > 0) = 2N (1 - (1 - e^{-\lambda})) = 2 N e^{-\lambda}$$

Si se toma un tasa de llenado para los nodos del árbol igual a $\ln 2 \cong 0.7$ [Yao78], el numero de nodos hoja es $\left\lceil \frac{2N e^{-\lambda}}{2m_e \ln 2} \right\rceil = \left\lceil \frac{N e^{-\lambda}}{m_e \ln 2} \right\rceil$

Teniendo en cuenta los nodos no hojas de un árbol B^+ de altura h, se obtiene

$$S_e^i = = \left\lceil \frac{N e^{-\lambda}}{m_e \ln 2} \right\rceil \left(1 + \frac{1}{2m_e \ln 2} + \frac{1}{(2m_e \ln 2)^2} + \cdots + \frac{1}{(2m_e \ln 2)^{h-1}} \right)$$

$$S_e^i = \left\lceil \frac{N e^{-\lambda}}{2m_e \ln 2} \right\rceil \left(\frac{2m_e \ln 2}{2m_e \ln 2 - 1} \right)$$

Para estimar el numero de entradas *ne* en el paquete referenciado por la primera entrada de un nodo hoja del árbol, se debe sumar:

- el numero esperado de llegadas durante un periodo m, dado por $m / (1/l) = ml$ y

- el numero de entradas incrementales por cada nodo hoja (igual al numero esperado de puntos de indexación en cada nodo hoja) multiplicado por el numero esperado de llegadas en un punto de tiempo. Se tiene entonces el producto $(2m_e \ln 2)\, l$.
  Por tanto, se tiene $ne = ml + (2m_e \ln 2)\, l$.

Suponiendo que las versiones de objeto son almacenadas lo mas cerca posible, se obtiene,

$$S_e^b = \left\lceil \frac{N e^{-\lambda}}{2m_e \ln 2} \right\rceil \left(\frac{\mu\lambda + 2m_e \lambda \ln 2}{2m_e} \right)$$

Finalmente, el costo de almacenamiento del índice temporal es

$$S_e = \left\lceil \frac{N e^{-\lambda}}{2m_e \ln 2} \right\rceil \cdot \left[\left(\frac{2m_e \ln 2}{2m_e \ln 2 - 1} \right) + \left\lceil \frac{\mu\lambda + 2m_e \lambda \ln 2}{2m_e} \right\rceil \right]$$

La ultima expresión indica que para un numero N fijo de versiones de objeto, el costo de almacenamiento del índice temporal es dependiente de las características de los datos temporales. Esto quiere decir:

- si el tiempo entre llegadas es pequeño entonces habrá llegadas múltiples al mismo tiempo y por tanto el numero de puntos de indexación será reducido.Como consecuencia se tendrá un árbol B^+ pequeño y un tasa baja de utilización del disco;

- si el tiempo de validez (duración) de cada tupla es grande comparada con el tiempo entre llegadas entonces la misma tupla podrá ser duplicada en varios paquetes referenciados por los primeras entradas de los nodos hojas. Por tanto se tendrá sobrecarga de paginas disco.

4.3. Rendimiento de consultas

El estudio del rendimiento de consultas consiste en estimar el numero de accesos a las paginas del disco necesarias para responder a una consulta dada. En particular se hará el análisis de una consulta sobre un intervalo de tiempo [a, b] y se calculara su costo.

Sea Q_e el numero de paginas necesarias para leer las tuplas en las cuales su tiempo de validez esta incluido en [a, b]. Una consulta sobre un intervalo de tiempo es satisfecha en dos etapas:

1) se recorre el árbol B^+ para localizar el nodo hoja que contiene el punto de tiempo a,

2) siguiendo el encadenamiento de los nodos hoja, se continua el recorrido a la derecha y al mismo tiempo se recuperan las entradas incrementales en los paquetes hasta que se llega al punto b.

Se asume que el punto de tiempo a es seleccionado de manera aleatoria en el intervalo de tiempo [0, now]. Se asume también que la longitud (b-a) del intervalo de tiempo [a, b] es une variable aleatoria que sigue una ley exponencial de media γ.

Para obtener la estimación del numero de puntos distintos de tiempo en BP que corresponden al intervalo [a, b], se considera una cola de espera donde los procesos de llegada siguen una ley de Erlang.

Sea L_n una variable aleatoria de media n/λ y que representa el n^{esimo} tiempo (ordenado) entre llegadas. Para estimar el valor de n, se supone que la n^{esima} llegada coincide con el tiempo medio γ en el intervalo [a, b]. Por tanto, se tiene $n/\lambda = \gamma \Rightarrow n = \lambda\gamma$

El número total de puntos de tiempo pertenecientes al conjuntos BP y que caen en el intervalo [a, b] est $2\lambda\gamma \times \Pr(X > 0) = 2\lambda\gamma\, e^{-\lambda}$.

La estimación del numero de nodos hoja leídos esta dada por

$$\left\lceil \frac{2\lambda\gamma\ e^{-\lambda}}{2m_e \ln 2} \right\rceil = \left\lceil \frac{\lambda\gamma\ e^{-\lambda}}{m_e \ln 2} \right\rceil$$

Para cada nodo hoja, se debe leer une pagina disco de tuplas. Si se desprecian los nodos no hoja, se tiene

$$Q_e = \left\lceil \frac{\lambda\gamma\ e^{-\lambda}}{m_e \ln 2} \right\rceil \left(1 + \frac{\mu\lambda + 2m_e\lambda \ln 2}{2m_e}\right)$$

Se puede deducir que el valor de Q_e se hace muy grande para grandes valores de $\mu\lambda$.

CONCLUSIÓN

En este articulo se han descrito algunos métodos de almacenamiento para las Bases de Datos Temporales. Se ha efectuado también el análisis de rendimiento de estos métodos con un enfoque probabilistico.

Primero, se han presentado los métodos de almacenamiento tradicionales basados en el *almacenamiento compartido*, donde se utilizan una zona corriente y una zona histórica para guardar las versiones de objeto corrientes e históricas respectivamente. Esta técnica no es eficaz para consultas sobre un intervalo de tiempo.

Luego se presento el Índice Temporal, este método construye un conjunto ordenado de puntos de tiempo a partir de los intervalos de tiempo de

validez de las tupras, entonces es posible utilizar un árbol B^+ para indexar estos puntos. Esta técnica permite mejorar el costo de almacenamiento y el rendimiento de las operaciones asociadas a la base de datos con respecto a la técnica de almacenamiento compartido.

El inconveniente del índice temporal es que su rendimiento depende de las características de los datos temporales. Es decir, los solapamientos entre intervalos de tiempo de validez obligan a que se almacene varias veces una misma tupla degradando así el rendimiento de las consultas.

REFERENCIAS BIBLIOGRÁFICAS

[AS88] SNODGRASS, R. and AHN, I. Partitioned Storage for temporal databases.Information Systems, 13(4):369-391, December 1988.

[BaM72 BAYER, R. and MCCREIGHT, E. Organization and Maintenance of Large Ordered Indexes, Acta Informatica, Vol. 1, No 3, 1972.

[BZ82 BEN-ZVI, J. The Time Relational Model. Ph.D. Thesis, Computer Science Departament, UCLA, 1982.

[EW90 ELMASRI, R. and WUU, G. A temporal model and query langage for ER databases. In IEEE Data Engineering Conference, February 1990.

[EWK90] ELMASRI, R. and WUU, G. Kim. The time Index: An acces structure for temporal Data. In Proceeding of the conference on Very Large Databases, Brismane, Australia, August 1990.

[GLOT96] GOH, C. H. LU, H. B. C. Ooi and K. L. Tan. Indexing temporal data using existing B^+ trees. In Data & Knowledge Engineering 18, pages 147-165, 1996.

[GY88] GADIA, S. K. and YEUNG, C. S. A generalized model for a relational temporal database. In Proceedings of ACM SIGMOD International Conference on Management of Data, pages 251-259, Chicago IL, June 1988.

[SA85] SNODGRASS, R. and AHN, I.A taxonomy of time in databases. In Proceedings of ACM SIGMOD International Conference on Management of Data, pages 236-246, Austin, TX, May 1985.