UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

CONTROL DE CONCURRENCIA EN UN SISTEMA MANEJADOR DE BASES DE DATOS BINARIAS ASOCIATIVAS

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACION

P R E S E N T A

LAURA OSAWA VELASCO

México, D. F. **Noviembre de 1985**

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

Indice

Lista de Figuras

CAPITULO 1

INTRODUCCION

Desde que aparecieron las computadoras, el manejo y almacenamiento de información es cada vez más sencillo, sin embargo, existen varias formas de almacenamiento de la información, siendo un ejemplo los archivos.

El manejo de información en la forma tradicional se hace con programas de aplicación, cada uno de los cuales tiene sus propias versiones de archivos por lo que hay redundancia en la información. Al tener que actualizar la información, lo tenemos que hacer en todos los archivos en los que se le haga referencia y si nos olvidamos de uno, la información ya no es consistente ni confiable.

De aquí surgió la idea de concentrar la información en un solo lugar del que todas las aplicaciones pudieran extraerla y actualizarla teniendo así un control centralizado de las operaciones. Este es el origen de una Base de Datos que es una colección de datos almacenados en memoria secundaria interrelacionados y disponibles a varias aplicaciones con redundancia mínima. Los datos se encuentran ligados entre sí por medio de relaciones y asociaciones que también se

almacenan en la base de datos. Debido a esta característica, si alguna aplicación actualiza un dato, éste podrá ser consultado de inmediato por otra aplicación y sin necesidad de esperar a que termine la primera. Las ventajas de una base de datos son [DATC75]:

- Reducir la redundancia en la información evitando el desperdicio en el almacenamiento en disco.
- Evitar los problemas de inconsistencia en los datos almacenados. La inconsistencia no se presenta si se cumple con el punto anterior ya que al repetirse dos entidades, se puede dar el caso en que solo una de ellas se actualice y por consiguiente serán distintas.
- La información almacenada puede ser compartida no solo por las aplicaciones existentes sino por todas las que se desarrollen a futuro.
- Se uniformizará la forma de representar la información simplificando los problemas de mantenimiento y de intercambio de información entre instalaciones.
- Seguridad mediante restricciones para el acceso de la información. Pueden existir distintos procedimientos para la consulta, modificación y borrado.
- Integridad en la información, se logra mediante validaciones antes de almacenar la información así se evitará que se haga referencia a un dato que no exista.
- Se puede estructurar la base de datos de tal forma que dé un mejor servicio a las aplicaciones más importantes (rapidez de acceso).

La forma en que se utiliza la información

"lógicamente" no siempre corresponde a la forma en que está almacenada "físicamente". Para poder trabajar con la información almacenada en la Base de Datos, es necesario disponer de un Manejador de Base de Datos, es decir, de una interfase entre el usuario y la Base de Datos. Es un sistema que nos permite insertar, borrar, recuperar y modificar datos bajo ciertas reglas en una Base de Datos. La metodología a seguir es la siguiente:

- Definir el modelo de la información, es decir, se definen todas las estructuras que se van a usar.
- Definir la estructura de almacenamiento que se va a usar y la estrategia para utilizarla.
- Existencia de un sistema manejador de Bases de Datos que interceptará las instrucciones que desee hacer el programa de aplicación o el usuario, para traducirlas en instrucciones que utilicen la Base de Datos y regresar la respuesta.

Las Bases de Datos se encuentran clasificadas de acuerdo a la estructura de datos que usan y pueden ser: reticulares, jerárquicas y relacionales. Existe una variación de éstas últimas que son consideradas como relacionales de orden dos en el cual, el nombre de la relación especifica la asociación existente entre los dominios involucrados y reciben el nombre de binarias asociativas.

Se han llevado a cabo una secuencia de trabajos sobre Bases de Datos de la que la presente tesis forma parte. El primer trabajo define los lineamientos a seguir para el diseño lógico de una Base de Datos SIMBAD ("Schema Integration Methodology / Binary Associative Dictionary") [BUCA80, BUCA82]. Como apoyo a éste, fué desarrollado un segundo trabajo que es un Sistema Manejador de Bases de Datos Binarias Asociativas SMBDBA [PALW82] que tiene la limitación de que solo puede ser usado por una aplicación a la vez, lo que le resta poder. El objetivo de la presente tesis, es la de proporcionar una herramienta para acabar con esta limitación, haciendo que el Manejador trabaje en un ambiente multiusuario. Las principales características de este manejador se explican a continuación. Fué desarrollado en una microcomputadora ONYX de 16 bits con 512 Kbytes de memoria "RAM", un disco fijo con capacidad de 20 Mbytes y una unidad de cinta integrada. El sistema operativo es el IS/1 cuyo lenguaje principal es el "C" y la filosofía es la del Sistema Operativo UNIX cuyas principales características se encuentran en [CHEE81]. La estructura de datos utilizada para guardar la información es la de un árbol B+ prefijo. En estos árboles, las llaves residen en las hojas y sus ramas

sólo nos sirven como guía para llegar a la hoja correcta. Durante la operación del SMBDBA, al buscar una hoja, se va formando una pila con las ramas que va leyendo, ésto es con el fin de tener en memoria principal la trayectoria y que sean más rápidos los futuros accesos ya que solo se tiene que traer a memoria aquellas ramas que no estén en la pila. Este manejador nos permite utilizar la Base de Datos desde programas de aplicación en lenguaje "C".

El mayor problema surge cuando a dos usuarios se les otorga el mismo registro al mismo tiempo ya que el primero lo actualizará pero cuando el segundo lo reescriba, se habrá perdido la actualización del primero. La solución más fácil es la de otorgárselo al primero en forma exclusiva y hasta que éste lo desocupe, se le otorgará al segundo. Esto nos trae como consecuencia el problema de abrazo mortal (deadlock). Una solución a este problema es la de otorgar una sola vez registros en forma exclusiva, por lo que si se necesitan varios, éstos deberán ser pedidos todos juntos así que sólo se otorgarán cuando todos los registros estén disponibles y no por partes. Algunas veces se necesitan registros en forma exclusiva aunque no vayan a modificarse, simplemente para asegurarse de que otros no los modifican.

Para ello es necesario disponer de un control de concurrencia apropiado. Esto consiste en administrar la Base de Datos de tal forma que no exista interferencia entre las aplicaciones y que todas dispongan de información actualizada y confiable. Cuando una transacción está llevando a cabo algunas actualizaciones, antes de terminarlas, se dice que la Base de Datos es inconsistente y por lo tanto se debe impedir que otra transacción utilice la misma información. Este control debe administrar dos aspectos: el lógico (asociaciones) y físico (páginas en disco) cuyo detalle se explica más adelante.

La presente tesis consta de cinco capítulos, siendo el primero esta introducción. En el segundo capítulo, se encuentra una revisión bibliográfica de los conceptos necesarios para la comprensión del tema de esta tesis, así como de los métodos que se describen en el capítulo tres para los candados lógicos y en el capítulo cuatro para los candados físicos. En estos dos últimos capítulos se describe el método que se empleó y cuyos resultados y conclusiones aparecen en el capítulo cinco.

CAPITULO 2

REVISION BIBLIOGRAFICA

2.1. CONSISTENCIA

Como se mencionó en la introducción, una Base de Datos es una colección de información sin una organización en particular (reticular, jerárquica o relacional). Se dice que una Base de Datos se encuentra en un estado consistente cuando la información que se encuentra almacenada en ella, satisface una serie de condiciones definidas entre sus registros (consistency constraints) [KORH83].

Cuando se lleva a cabo una operación en la Base de Datos, ésta pasa de un estado a otro y no forzosamente se conserva la consistencia. Por lo anterior, una transacción se define como una serie de acciones secuenciales que se ejecutan sobre una Base de Datos. Cuando la Base de Datos es consistente al iniciarse una transacción, quedará consistente cuando ésta termine [GRAJ75].

Al tener dos o más transacciones independientes, se dice que hay interferencia entre ellas si al ejecutarse al mismo tiempo, producen resultados diferentes que

cuando son ejecutadas por separado [GRAG79]. Un manejador multiusuario de Base de Datos, debe garantizar que el resultado obtenido de una colección de transacciones que fueron ejecutadas en forma entrelazada (simultánea) es igual al que se obtendría si éstas hubieran sido ejecutadas en forma secuencial[RIED79].

Cuando un despachador (scheduler) no entrelaza las acciones de varias transacciones, se dice que es serial y sería como procesar en lote (batch). La consistencia puede perderse al entrelazar las acciones de varias transacciones [KORH83]. El despachador debe entrelazar las acciones de tal forma que puedan ejecutarse produciendo el estado consistente deseado (seriabilidad), y éste sería nuestro criterio para aceptar a un despachador [KORH82].

2.2. CONTROL DE CONCURRENCIA

El control de concurrencia es un mecanismo mediante el cual se garantiza la consistencia de la Base de Datos y consiste en coordinar el acceso simultáneo a un sistema de bases de datos multiusuario, es decir, permite que varios usuarios utilicen la base de datos y sin embargo parezca que cada transacción se ejecuta en forma individual. El obstáculo más importante para

lograrlo es el de evitar que las actualizaciones que hace un usuario le interfieran a otro que está leyendo [BERP81].

El problema de control de concurrencia se divide en dos aspectos que son el de lectura-escritura (read-write) y el de escritura-escritura (write-write). El problema de lectura-escritura, consiste en que el valor que es leído difiere del que está almacenado. Esto depende del orden en que fueron ejecutadas las instrucciones de lectura y escritura. El problema de escritura-escritura consiste en que al realizar dos escrituras en la misma información se conserva únicamente el valor de la última escritura perdiéndose la anterior.

Para que se comprenda mejor el problema, a continuación se presentan dos anomalías que pueden suceder en cualquier banco.

1. Pérdida de actualizaciones (escritura-escritura). Supongamos que dos clientes tratan de hacer un depósito a la misma cuenta al mismo tiempo (figura 2-1). Por no disponer de un control de concurrencia, para los dos se lee el mismo saldo, se calcularía su nuevo saldo en paralelo y se actualizará en ambos casos la Base de Datos. El resultado es incorrecto ya que solo uno quedaría registrado.

2. Información no confiable (lectura-escritura). Supongamos que dos clientes hacen las siguientes transacciones al mismo tiempo (figura 2-2). El primer cliente transfiere $1,000,000 de su cuenta de ahorros a su cuenta de cheques. El segundo cliente solicita el estado de cuenta del primero. La primera transacción (cliente 1), leerá el saldo de la cuenta de ahorros, le restará $1,000,000 y actualizará la Base de Datos. La segunda transacción (cliente 2) leerá el saldo de la cuenta de ahorros y de la de cheques imprimiendo el estado de cuenta. Finalmente la primera transacción terminará de calcular y actualizar el saldo de la cuenta de cheques.

A diferencia del primer problema, en el segundo se quedan almacenados correctamente los resultados pero el estado de cuenta no será correcto.

2.3. GRANULARIDAD

La granularidad se refiere al tamaño de los objetos individuales que una transacción puede asegurar, por lo que, la unidad mínima en una Base de Datos que podrá tener un candado es un grânulo. La granularidad de un candado se define como la cantidad de información asociada a él (en la sección 2.5.1 se explica el concepto de candado).

Si la cantidad de información asociada a un candado es mucha, se dice que tenemos granularidad gruesa. Por el contrario, si la cantidad de información es poca, se

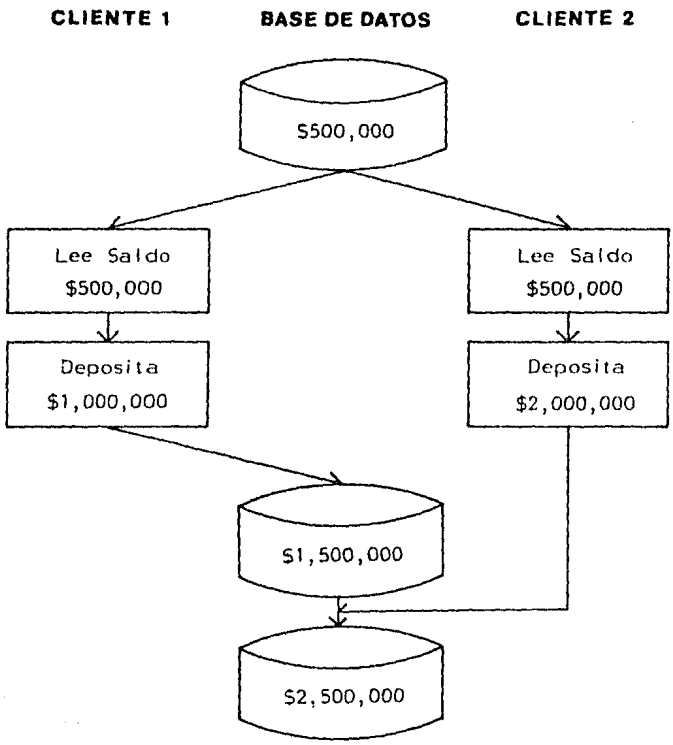


Figura 2-1: Problema Escritura-Escritura

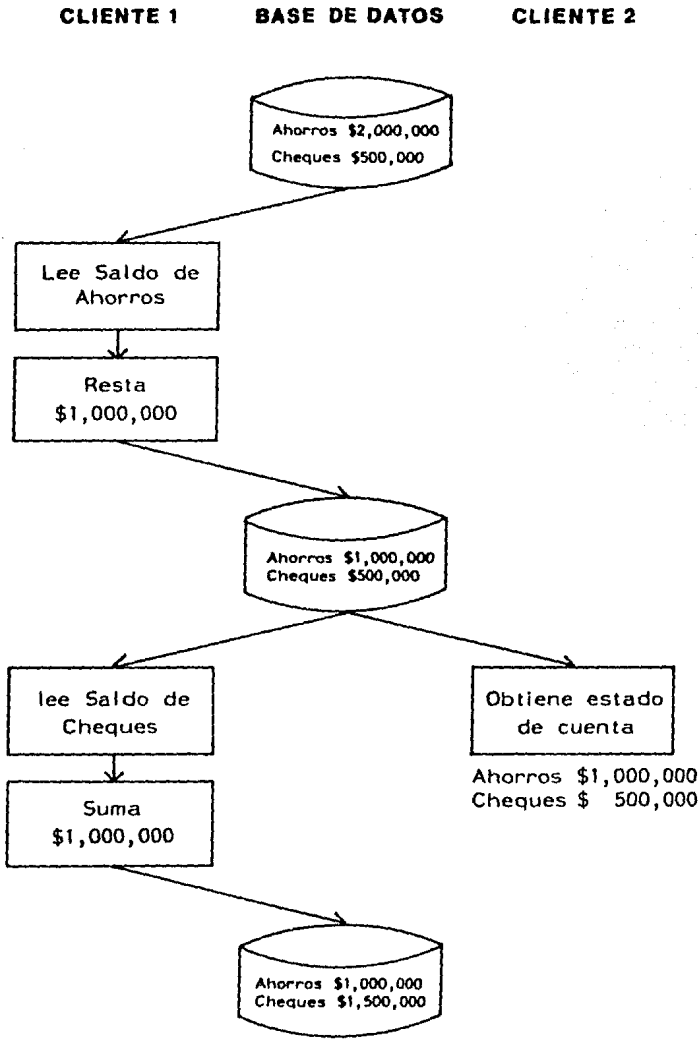


Figura 2-2: Problema Lectura-Escritura

dice que tenemos granularidad fina. La granularidad debe elegirse dependiendo de la transacción, ya que si hacemos referencia a pocos registros, el tamaño del grânulo debe ser de un registro para que no aseguremos información que no vamos a necesitar. Pero si vamos a usar casi todo un archivo, el tamaño del gránulo debe ser de un archivo ya que de lo contrario, necesitaríamos muchos candados para asegurarlo.

La concurrencia en una base de datos depende del tamaño del gránulo que se elija, incrementándose si se elige un gránulo muy chico como un campo de un registro. Sin embargo, el solicitar, esperar, administrar y almacenar candados lleva un costo asociado (overhead). Por ejemplo, si nuestro grânulo corresponde a un registro, entonces debemos de manejar una tabla de candados cuya longitud será igual al número de registros con que estemos trabajando. Por el contrario, si tuviéramos una granularidad muy grande nos reduciría el paralelismo pero se minimizaría el manejo de candados. Por ejemplo, si el gránulo es toda la base de datos (que es el más grande que puede haber), entonces sería muy sencillo porque manejaríamos un solo candado.

La solución a este problema es permitir que una

transacción utilice diferentes granularidades. En [RIED79] se muestra el proceso general que se lleva a cabo para asignar candados y una simulación para encontrar la óptima granularidad (figura 2-3)

Inicialmente las transacciones llegan en diferentes tiempos pasando por las siguientes etapas:

- Cuando una transacción llega, es colocada en la cola de pendientes y saldrá en el momento en que solicite sus candados. Si los candados son otorgados, la transacción es colocada al final de la cola de entrada/salida. Si los candados fueron negados, entonces la transacción es colocada al final de la cola de bloqueados.
- Cuando se ha completado la entrada/salida la transacción es colocada al final de la cola de CPU.
- Cuando se han completado las peticiones de CPU, la transacción libera sus candados y es pasada al final de la cola de pendientes. Todas las transacciones que estaban bloqueadas por la transacción que acaba de terminar se colocan al frente de la cola de pendientes.

Se puede observar que todos los candados que se iban a utilizar, fueron pedidos inicialmente por lo que es imposible que se genere un DEADLOCK.

En la simulación, un número fijo de transacciones que utilizaban la base de datos, fueron continuamente recicladas en el modelo de la figura 2-3.

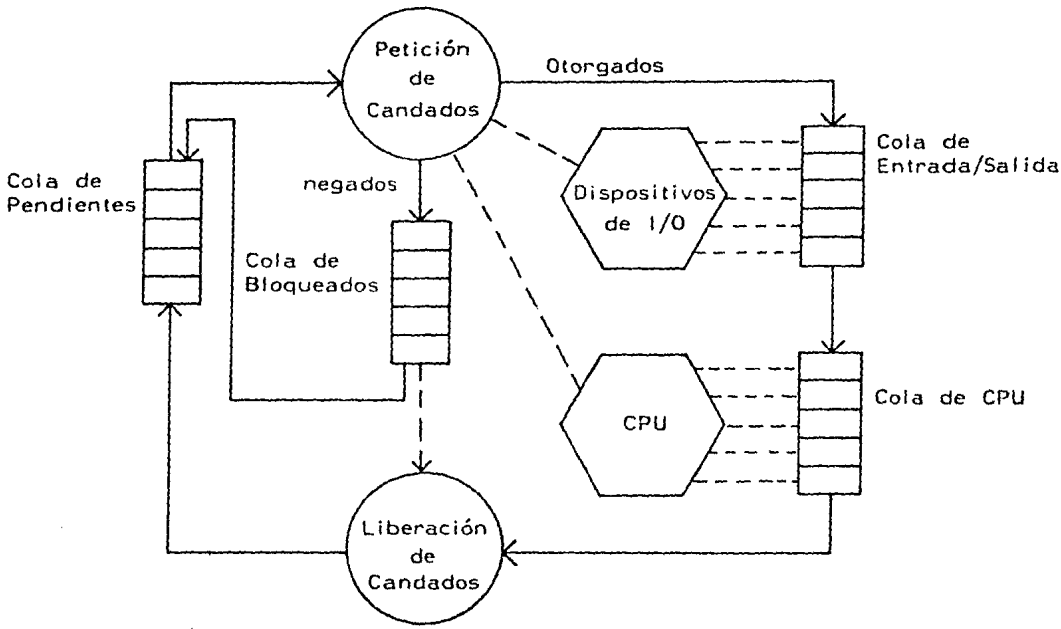


Figura 2-3: Simulación para obtener la óptima granularidad

Como resultado se encontró que la utilización de la máquina se incrementa conforme va aumentando el número de gránulos hasta llegar a un límite que es cuando empieza a descender, por lo que, se puede decir que el manejar gránulos grandes es casi óptimo y además es mucho más sencillo de implementar.

En investigaciones posteriores [RIED79] se encontró que una granularidad fina es mejor solo en los siguientes dos casos:

1. Cuando casi todas las transacciones son pequeñas, es decir, cuando utilizan una pequeña parte de la Base de Datos.
2. Cuando el acceso a la Base de Datos es aleatorio, sin ninguna secuencia, donde la probabilidad de utilizar cualquier entidad es la misma ya que no depende de las que han sido usadas previamente.

Sin embargo, si varias transacciones utilizan gran parte de la base de datos, entonces el manejar gránulos pequeños nos crea un costo adicional por lo que nuevamente es preferible la granularidad mayor. Esta conclusión nos será de utilidad para definir la granularidad en nuestra aplicación.

Una transacción puede iniciarse una vez que le hayan sido otorgados todos sus candados como en la

simulación de la figura 2-3. En algunas ocasiones, las transacciones no saben que registros van a necesitar hasta que se están ejecutando por lo que la solicitud y asignación de candados debe ser dinámica. Esto trae como consecuencia que el distribuidor de candados (scheduler) debe manejar situaciones de abrazo mortal (deadlock) que se explican en la siguiente sección.

2.4. ABRAZO MORTAL (DEADLOCK)

Cada vez que una transacción espera a que se le asigne un recurso que solicitó, corre el riesgo de esperar por siempre al cerrarse un círculo [GRAJ75]. Esto sucede cuando una transacción T1 tiene que esperar un recurso X2 que tiene asignada la transacción T2, y al mismo tiempo T2 espera a que T1 libere el recurso X1 que necesita. Esto se presenta en la gráfica de la figura 2-4.

Para detectar una situación de abrazo mortal, es necesario saber quién espera a quién. Las situaciones de abrazo mortal se detectan construyendo las gráficas de espera y localizando ciclos.

Existen dos reglas para la solución de situaciones de abrazo mortal que son: prevención y detección.

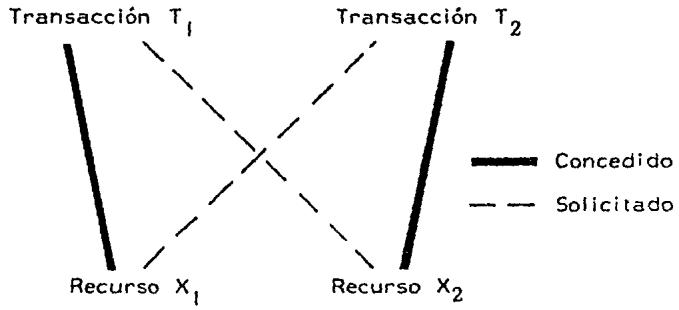


Figura 2-4: Situación de Abrazo Mortal

2.4.1. PREVENCION

Es una técnica cautelosa en la que una transacción es reiniciada cuando el sistema "teme" que puede suceder una situación de abrazo mortal. Cuando un recurso es negado a una transacción debido a que otra transacción lo tiene asegurado, se hace una prueba entre ambas que podría ser asignando prioridades. Si se pasa la prueba, la transacción que requiere el recurso puede esperar, de otra manera, una de las dos transaccioens se aborta. Si la transacción T1 que es quién solicita el recurso se aborta, entonces la transacción T2 que lo tiene asignado

no se interrumpe (nonpreemptive). Si se aborta la transacción T2 que tiene el recurso asignado, entonces éste se concede a la transacción T1 que lo solicitaba (preemptive).

Pueden existir "reinicios cíclicos" debidos a que una transacción puede ser contínuamente reiniciada al tener que abortar siempre, provocando que nunca termine. Para evitar esto, se propone usar sellos de tiempo (timestamps) como prioridades [BERP81]. De manera general, el sello de tiempo de una transacción es el momento en el cual inicia su ejecución, de tal forma que las transacciones antiguas tienen mayor prioridad que las nuevas.

En [BERP81] se proponen dos esquemas de prevención basados en sellos de tiempo y que evitan los reinicios cíclicos:

WAIT DIE — Supongamos que la transacción T1 espera a T2. Si T1 tiene menor prioridad que T2 por haber iniciado después, entonces se le permite que espere. De lo contrario, se aborta y posteriormente se reinicia conservando su sello de tiempo original.

WOUND DIE — Esta técnica es la contraria de la anterior, ya que solo permite esperar si la prioridad de T1 es mayor que la de T2.

El preordenamiento de recursos es una técnica para evitar situaciones de abrazo mortal que cancela reinicios. Esta técnica requiere de la predeclaración de los candados antes de la ejecución de todas las transacciones. Los datos necesarios (data items) son numerados y cada transacción obtiene todos los candados uno a la vez en orden numérico. La prioridad de una transacción es el número de candado mayor numerado que posee. Debido a que una transacción solo puede esperar por transacciones con mayor prioridad, no pueden ocurrir situaciones de abrazo mortal. La desventaja de esta técnica es que se forza a obtener los candados secuencialmente, lo cual tiende a incrementar el tiempo de respuesta.

2.4.2. DETECCION

Al detectar un ciclo en la gráfica de espera, una transacción llamada víctima, es abortada con lo cual se rompe la situacion de abrazo mortal. Para minimizar el costo de reinicio de la víctima, la selección se hace tomando en cuenta los recursos que utiliza cada transacción.

2.5. CANDADOS

2.5.1. DEFINICION

Los candados son marcas que se colocan en los gránulos para asegurarlos cuando se van a utilizar. Entre las acciones que hace una transacción, está la de asignar y liberar candados. Estas acciones no alteran el estado de la Base de Datos pero tienen el efecto de asegurar la información que van a utilizar de otras transacciones. La asignación de candados es un medio mediante el cual todas las transacciones saben cuando pueden usar un gránulo, o en su caso, saben que transacción la está utilizando, evitando así las situaciones de abrazo mortal.

En las siguientes subsecciones se muestran los tipos de candados o modos de acceso, así como el funcionamiento general de los protocolos (locking protocol), es decir, la forma en que estos candados son asignados. El colocar un candado en el gránulo "lógico" que vamos a utilizar no nos libera de conflictos "físicos", esto se debe a que la forma en que está almacenada la información "físicamente" no corresponde a la forma en que se encuentra "lógicamente". Es por ésto, que debemos tener diferentes protocolos para una misma

aplicación. Los candados físicos deben ser asignados por un protocolo que considere la estructura de datos en la que está almacenada la información.

2.5.2. MODOS DE ACCESO

Cuando dos transacciones solicitan un candado sobre la misma entidad, existe un conflicto. La consecuencia de ésto es que una de ellas debe esperar. Con el objeto de asignar eficientemente los candados en varias granularidades, los recursos y sus respectivos candados deben organizarse en una jerarquía [GRAJ75]. A cada nivel de la jerarquía le corresponde un tipo de nodo. Por ejemplo en la figura 2-5 se muestra que a la Base de Datos le corresponden varias áreas, a cada área varios archivos y a cada archivo, varios registros. Como se puede observar, cada nodo tiene solo un padre. Cada nodo de la jerarquía puede tener un candado.

Existen varios tipos de candados que una transacción puede obtener en un gránulo y ésto depende de la acción que quiera hacer sobre él. Cuando una transacción solicita un nodo para acceso exclusivo (X) o compartido (S), en el momento en que es otorgado, adquiere el acceso deseado en ese nodo e implícitamente en todos sus descendientes. Estos dos modos de acceso

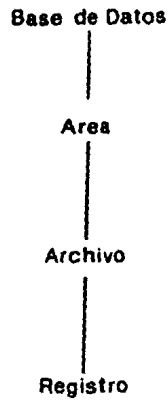


Figura 2-5: Estructura Jerárquica de la Información

aseguran todo el subárbol formado al tomar como raíz el nodo solicitado [GRAJ75]. Cuando una transacción adquiere un candado no solo adquiere los privilegios del modo (exclusivo o compartido) solicitado sino todos aquellos de los modos que sean menos exclusivos del asignado [KORH82].

Al colocar un candado en un subárbol con cualquiera de los modos, es importante evitar que más tarde otra transacción coloque candados a sus ancestros que podrían tener conflicto con el que ya fué asignado. Para evitar este problema, se genera un nuevo modo que es el modo de

intención (I). Se usa para marcar los ancestros de un nodo que tiene asignada una transacción T1. Si se permitiera que otra transacción T2 pudiera tener un candado en alguno de los ancestros, recordar que implícitamente el nodo de T1 lo obtendría provocándose un conflicto.

Se dice que dos peticiones al mismo nodo son compatibles, si pueden ser manejadas concurrentemente sin sacrificar consistencia [GRAJ75]. Los modos X, S e I son incompatibles entre sí. Una petición del modo compartido es compatible con otra del mismo modo, pero entre las de modo exclusivo no. El modo de intención es compatible con sí mismo. Para hacer compatible un acceso de intención con un acceso compartido, se generan dos nuevos modos: el modo de intención compartida (IS) y el modo de intención exclusiva (IX). Si tenemos un modo de intención compartida, se puede convertir en compartido únicamente. Lo que no es posible es convertir de intención exclusiva a compartida.

Existe un nuevo modo llamado compartido y de intención exclusiva (SIX) que puede ser usado para leer un subárbol y actualizar ciertos nodos. Pone candados de compartidos a los descendientes, pero algunos de ellos

pueden ser marcados con X, SIX o IX. Su gráfica de compatibilidad es la que se muestra en la figura 2-6.

El modo IS es la forma más débil de marcar un recurso. SIX tiene los privilegios de IX y de S. X es el de mayor privilegio ya que permite la lectura y escritura de todos los descendientes de un nodo sin que sea necesario poner un nuevo candado. La gráfica de privilegios se muestra en la figura 2-7.

Cuando el candado se pone en forma implícita (S y X) se hace de acuerdo con el protocolo siguiente:

- Antes de pedir un candado S o IS en un nodo, es necesario que todos los ancestros del nodo tengan modo IX o IS.
- Antes de pedir un candado X, SIX o IX en un nodo, es necesario que todos los ancestros del nodo tengan modo SIX o IX.
- Los candados deberán ser liberados al final de la transacción.

Los candados son puestos de la raíz a las hojas, y quitados de las hojas a la raíz. Un árbol es una gráfica acíclica dirigida así que para poner un candado a un nodo, ya sea en forma explícita o implícita, es necesario poner un candado a todos los ancestros del nodo. En la figura 2-8 se muestra una estructura con granularidades distintas.

COMPATIBILIDAD					
	IS	IX	S	SIX	X
IS	SI	SI	SI	SI	NO
IX	SI	SI	NO	NO	NO
S	SI	NO	SI	NO	NO
SIX	SI	NO	NO	NO	NO
X	NO	NO	NO	NO	NO

Figura 2-6: Gráfica de Compatibilidad entre los Modos

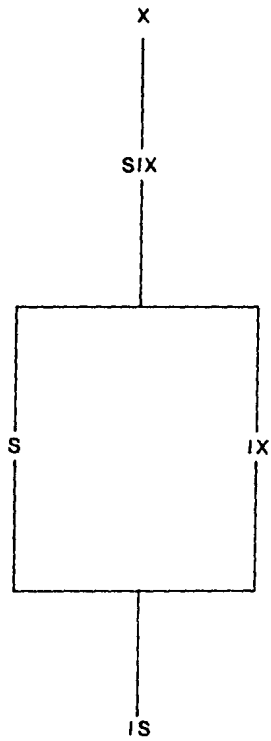


Figura 2-7: Orden de los Modos con sus Privilegios

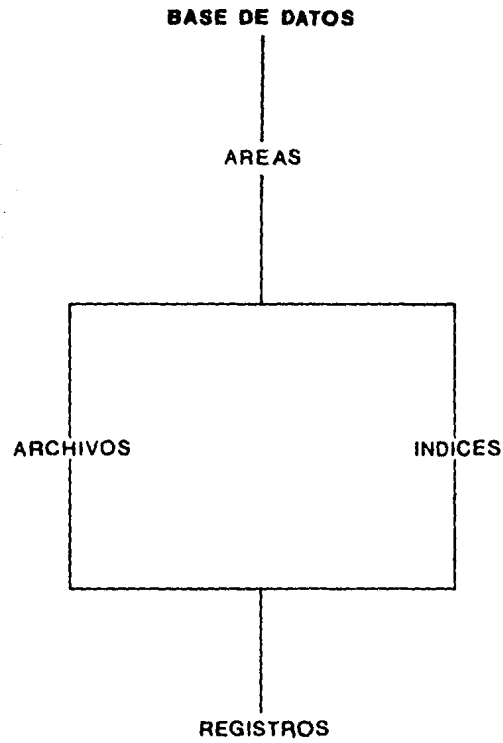


Figura 2-8: Estructura con Granularidades Distintas

Dentro de un área, cada archivo tiene un índice. El archivo nos dá el acceso secuencial a los registros mientras que el índice nos dá una ruta de acceso asociativo también de los registros y lo hace basándose en los valores del campo. Un nodo puede ser marcado en forma explícita o implícita. El protocolo para poner un candado en forma implícita es el siguiente:

- Un nodo adquiere el modo S implícitamente en una transacción cuando por lo menos uno de sus padres tiene modo S, SIX o X.
- Un nodo adquiere el modo X implícitamente cuando todos sus padres están en modo X.

El protocolo para poner un candado en forma explícita es el siguiente:

- Antes de pedir un candado S o IS se debe de pedir por lo menos que un padre tenga un modo igual o mayor que IS.
- Antes de pedir un modo IX, SIX o X se debe de pedir que todos los padres tengan modo igual o mayor que IX.
- Los candados deben ser quitados al final de la transacción en el orden de hoja a raíz.

Si un nodo fué otorgado en forma explícita o implícita a una transacción, no podrá ser otorgado a otra en forma incompatible. En estos protocolos, solo los candados con modo exclusivo necesitan que se marquen

a todos sus ancestros. En el candado de modo compartido solo se tiene que marcar una sola ruta a la raíz, porque si vemos la figura 2-8 nos damos cuenta que para leer un registro lo podemos hacer a través del índice o del archivo, no necesitamos los dos. Para meter, quitar o actualizar un registro R dentro del archivo F con el índice I, se debe de poner un candado de forma explícita o implícitamente a todos los ancestros de R.

En nuestra aplicación tenemos el diagrama de la figura 2-9. Si manejáramos la granularidad de toda la Base de Datos, no tendría caso ya que sería dejar el sistema como está (monousuario). Si utilizáramos el nivel de ternas, el costo adicional que tendríamos por el protocolo de asignación de candados no sería justificable, ya que los programas de aplicación no serán muy grandes ni la concurrencia por las limitaciones del equipo (ver capítulo 1). Por tal motivo, se decidió usar solo una granularidad siendo la elegida la asociación. Más adelante se explica su implementación.

2.5.3. OTORGAMIENTO DE CANDADOS

Un proceso puede asignar un candado cuando el nodo no tiene otro o si el candado que tenía es compatible con el que está asignado. Si el nodo ya tenía un candado

Figura 2-9: Gráfica de nuestra aplicación

que es incompatible con el nuevo, entonces la petición tiene que formarse en una cola de espera hasta que otro proceso libere el candado.

El conjunto de las peticiones hechas a un recurso, es guardado en una cola ordenada por un despachador (scheduler). Estas peticiones pueden atenderse con diferentes métodos siendo un ejemplo el FIFO (First In First Out). A la cabeza de la lista se encuentran las peticiones compatibles y recibe el nombre de grupo otorgado. Estas peticiones son manejadas concurrentemente. La compatibilidad de las peticiones de dos transacciones depende de sus modos de acceso y de acuerdo con la gráfica de compatibilidad de la figura

2-6. Cuando llega una nueva petición, el distribuidor la pone al final de la cola pudiendo existir dos situaciones: que otra petición esté esperando o que ninguna espere. Si nadie está esperando entonces es asignado inmediatamente. En caso de que existan otras peticiones esperando, entonces se analiza si el modo que está solicitando es compatible con los ya otorgados. En caso afirmativo, se otorga y pasa a formar parte del grupo otorgado.

La forma en que se decide si la solicitud es compatible con el grupo otorgado es definiendo un modo para ei grupo y de acuerdo con la tabla de la figura 2-6 se analiza la compatibilidad. En la figura 2-10 se muestran los modos de cada solicitud del grupo y el modo total del grupo. Los parêntesis significan que pueden existir varias solicitudes con ese modo.

Cuando una petición que se encuentra en el grupo otorgado termina, es posible que el modo del grupo cambie por lo que, se revisan las peticiones que están esperando para definir si alguna puede entrar al grupo.

Modos de las Solicitudes	Modo del Grupo
X	X
SIX, (IS)	SIX
S, (S), (IS)	S
IX, (IX), (IS)	IX
IS, (IS)	IS

Figura 2-10: Posibles Modos de los Grupos Asignados

2.5.4. CONVERSION DE CANDADOS

Existen casos en los que una transacción repite una petición pero cambiando el modo del candado. Un ejemplo para comprender la conversión de un candado sería la de asignar un candado de lectura en un nodo para revisar si existe cierta información. En caso de que exista la

información, se tendría que convertir el candado para poder borrarla ya que recordemos que cuando el candado es de lectura, no se permite hacer modificaciones. No se coloca el candado de actualización desde un principio porque si no existe la información que deseamos borrar, no se modificará nada.

Se debe disponer de un método de conversión de modos. Existen dos tipos de conversión, una es para obtener un modo más exclusivo y la otra para obtener uno menos exclusivo [KORH83]. Si el nuevo modo es igual que el anterior, entonces es otorgado inmediatamente. Si el nuevo modo es diferente del anterior pero es compatible con el modo del grupo otorgado entonces es otorgado pero haciendo una conversión de acuerdo con la tabla de la figura 2-11 [GRAJ75].

En cualquier caso diferente de los que se acaba de mencionar, la conversión solicitada deberá esperar a que el modo que solicita sea compatible con el del resto del grupo. Si dos conversiones están esperando al recurso que tiene la otra y los modos son incompatibles, entonces puede existir una situación de abrazo mortal. Las conversiones son otorgadas por el despachador antes que las nuevas peticiones. La figura 2-12 nos muestra

	MODO NUEVO				
	IS	IX	S	SIX	X
IS	IS	IX	S	SIX	X
IX	IX	IX	SIX	SIX	X
S	S	SIX	S	SIX	X
SIX	SIX	SIX	SIX	SIX	X
X	X	X	X	X	X

Figura 2-11: Conversión entre Modos

dos transacciones que generan una situación de abrazo mortal al no liberar los candados hasta el final. El paso n de la transacción T1 y el m de T2 muestran la conversión de candados. Este problema es crítico en un sistema ya que el 97% de las situaciones de abrazo mortal que pueden suceder son por esta causa [KORH83]. La frecuencia de estos casos puede reducirse si usamos

un modo de actualización U (update). La figura 2-13 nos muestra la compatibilidad entre los modos {R,U,W}.

```
T1:                              T2:
1. Asegura e1 con modo R         1. Asegura e2 con modo R
2. Asegura e2 con modo R         2. Asegura e1 con modo R
        .                                .
        .                                .
        .                                .
n. Asegura e1 con modo W         m. Asegura e2 con modo W
        .                                .
        .                                .
        .                                .
x. Libera e1,e2,...              y. Libera e1,e2,...
```

Figura 2-12: Abrazo Mortal Generado por Conversion

	R	U	W
R	true	true	false
U	true	false	false
W	false	false	false

Figura 2-13: Matriz de Compatibilidad {R,U,W}

Una transacción T1 puede obtener un modo X en una entidad e1 aunque exista la posibilidad de convertirlo a un modo Y más exclusivo. Si llega una transacción T2 podrá obtener un modo Z sobre la misma entidad e1 por ser compatible con X. El problema sucede cuando T1 trata de convertir del modo X al Y si Z no es compatible con

Y, T1 tendrá que esperar para siempre produciendo una situación llamada "livelock". De alguna forma T1 debe avisar a T2 de sus intenciones de conversión.

Existen unas gráficas de compatibilidad y convertibilidad que se usarán en secciones posteriores, donde se muestra la relación entre los candados. Un ejemplo sería la figura 2-14 cuya interpretación sería la siguiente: Las letras encerradas en círculos nos muestran los candados. Las líneas contínuas nos indican la compatibilidad entre los candados, y las punteadas nos muestran la convertibilidad. Esto significaría que podríamos tener varios candados de modo X al mismo tiempo que serían compatibles con uno Y (notar que no pueden existir varios de modo Y al mismo tiempo). El candado de modo Y puede convertirse al modo Z pero no al contrario.

2.5.5. CLASIFICACION

Se deben usar técnicas de asignación de candados para solucionar los problemas de sincronización, cubriendo el aspecto lógico (asociaciones, ternas, etc) y el físico (archivos, páginas, etc) [RIED79].

CANDADOS LOGICOS: Se usan con el objeto de evitar situaciones en las que se pierden actualizaciones o que

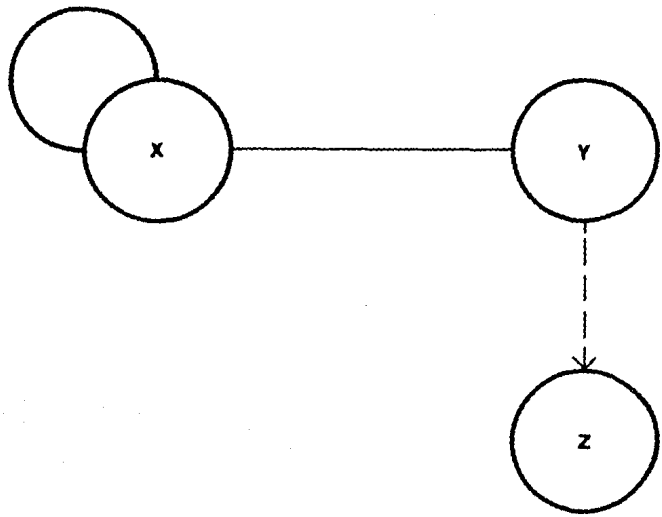


Figura 2-14: Gráfica de Compatibilidad y Convertibilidad

la información no es confiable (ver sección 2.2) [ASTM76]. Los candados lógicos son colocados en la parte de la Base de Datos que se necesita, es decir, son para reservar información a nivel lógico y que cumpla con ciertas características, por ejemplo, tipos de registros o registros que contengan atributos sobre los cuales se van a modificar. Estos candados no forzosamente corresponden a los candados físicos.

CANDADOS FISICOS: En este caso los candados son

asignados a particiones físicas de la Base de Datos, por ejemplo en registros, páginas, segmentos, archivos etc. Estos candados serán colocados sobre un grânulo de la Base de Datos. En ningún caso podrá ejecutarse una instrucción si el grânulo que se necesita tiene un candado de alguien más o que éste sea incompatible [RIED79]. Se necesitan manejar los candados físicos además de los lógicos, porque podrían existir conflictos al almacenar ternas de diferentes asociaciones dentro de la misma página [PALW82][ASTM76]. Los candados físicos tienen por objeto, además de evitar el acceso a unidades sobre las cuales va a operar una transacción, garantizar que no se va a destruir la pista de acceso en el caso de mecanismos de acceso dinâmicos, como lo son los árboles B cuyas características generales se encuentran más adelante.

2.6. GENERALIDADES DE ARBOLES B

Un árbol B de orden "m" tiene las siguientes propiedades [KWOY82]:

- Cada nodo tiene como máximo "m" hijos.
- Cada nodo (con excepción de la raíz y las hojas) tiene por lo menos m/2 hijos.
- La raíz tiene por lo menos dos hijos.

- Un nodo que no sea hoja contiene separadores y apuntadores a sus hijos.
- Todas las hojas aparecen en el mismo nivel. Las llaves aparecen en las hojas en orden ascendente cuando se leen de izquierda a derecha.

Los árboles con los que se trabajó se llaman B+ prefijos y tienen la variación de que la búsqueda se hace secuencial en las hojas, por lo que en los nodos ramas solo se encuentran separadores.

La figura 2-15 muestra la división lógica de los separadores y las llaves. Los formatos de los nodos rama y hoja pueden ser diferentes. Los nodos hojas apuntan al siguiente de izquierda a derecha por lo que se pueden leer de forma secuencial. Esta es una de las ventajas respecto a las otras variantes de árboles B [COMD79].

En los árboles B+ prefijos al dividir una hoja en dos, se emplea como separador entre ambas hojas a la cuerda prefija mínima necesaria para diferenciar la última llave de la hoja izquierda de la primer llave de la hoja derecha. Esta variación se emplea cuando las llaves son cuerdas de caracteres [COMD79]. Otra de las ventajas de los árboles B+ prefijos es que al emplear separadores más cortos se requiere menos espacio y se incrementa el número de separadores por nodo rama, consiguiéndose disminuir la altura del árbol.

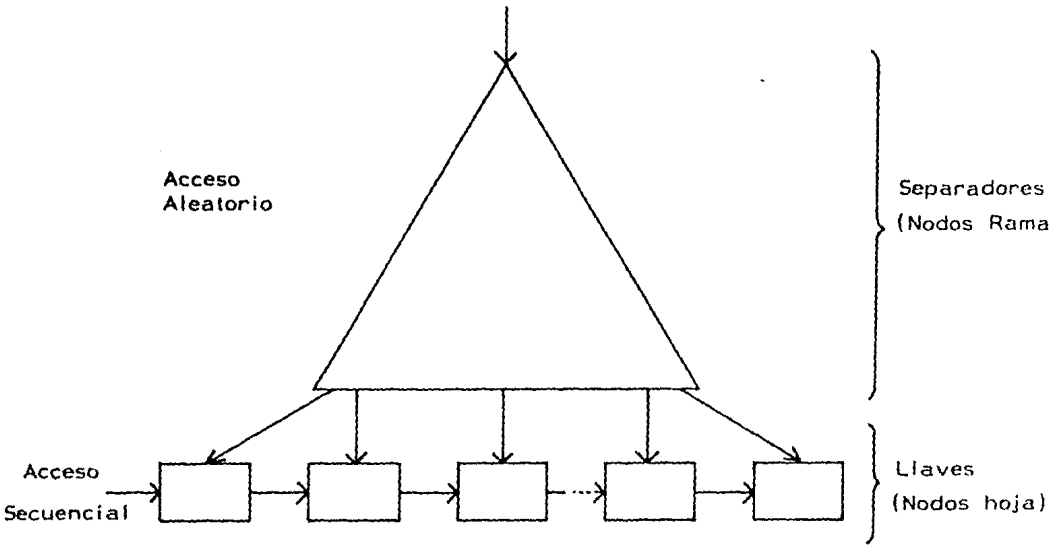


Figura 2-15: Representación de un Arbol B+

Otro término que se definirá es el de profundidad en un árbol B. La profundidad de un nodo "N" en un árbol es el número de nodos que se encuentran en la ruta de la raíz hasta "N".

Al permitir que un número arbitrario de procesos operen en forma concurrente sobre una base de datos almacenada en un árbol B, se asume que esos procesos son asíncronos, cada uno de los cuales se encuentra

avanzando a una velocidad indeterminada y que sólo puede estar ejecutando una de las siguientes instrucciones [KWOY82]:

SEARCH (K,T): Es un lector que se usa para determinar si la llave K existe dentro del árbol T de acuerdo con el acoplamiento entre candados. Si existe, se reporta éxito, en caso negativo, se reporta fracaso.

INSERT (K,T): Es un lector actualizador que utiliza candados de escritura en su trayectoria hasta llegar a las hojas. Si el argumento que iba a adicionar ya se encuentra, termina y libera los candados, de lo contrario, actúa como un escritor iniciando la reestructuración.

DELETE (K,T): Es un lector actualizador que elimina la llave K del árbol T si existe e inicia la reestructuración.

El proceso de búsqueda (SEARCH) únicamente lee los nodos a lo largo de una ruta desde la raíz hasta una hoja en particular. No realiza ninguna actualización a la información ni a la estructura del árbol. Por esta razón se le llama lector-puro. Un proceso de inserción (INSERT) o de remoción (DELETE) puede modificar la información así como la estructura del árbol. Por eso se le llama actualizador. Un actualizador generalmente pasa por las siguientes dos fases:

1. Búsqueda: Lee desde la raíz hasta las hojas (top-down) buscando el lugar apropiado para insertar o remover una llave.

2. Reestructuración: Aumenta o remueve una llave y después rebalancea el árbol en caso de que sea necesario.

La segunda fase no siempre es necesaria, por ejemplo, cuando se desea hacer una inserción y resulta que la llave ya existía. Otro caso es cuando se desea remover una llave y ésta no existe. Un actualizador se llama por consiguiente lector-actualizador. Una vez que se encuentra reestructurando el árbol, se dice que es un actualizador-escritor. La ruta utilizada por un proceso desde la raíz hasta la hoja es llamada "ruta de acceso". Para regular el acceso concurrente, se necesita un control de concurrencia que debe definir los tipos de candados usados por los procesos que operan sobre un nodo, y las gráficas de compatibilidad y convertibilidad especificando la relación que existe entre los candados.

Los procesos pueden manipular los candados mediante tres operaciones que son: asignar (lock), liberar (unlock) y convertir como se vió anteriormente.

Para las actualizaciones, es importante definir el concepto de nodo seguro. Este nodo es la raíz de un subárbol dentro del cuál podrán hacerse modificaciones que no podrán propagarse en el resto del árbol. La seguridad de un nodo es diferente para el caso de inserción que para el caso de remoción.

Un nodo es seguro para inserción, si no está saturado, es decir, tiene menos de m-1 llaves. Esto significa que si se aumenta otra llave, el nodo no tendrá que dividirse en dos por lo que su padre no se modificará. Un nodo es seguro para remoción cuando no tiene el número mínimo de llaves que puede tener, es decir, debe tener más de [m/2]-1 llaves. Si quitamos una llave no será necesario juntar dos nodos para que reúnan las características de los árboles B por lo que el padre permanecerá sin ninguna modificación.

CAPITULO 3

CANDADOS LOGICOS

El Control de Concurrencia se ha estudiado en los últimos años y se ha aceptado que una solución standard para este problema es la de "two phase locking" [BERP81]. Los algoritmos que existen para solucionar cada uno de los dos problemas que existen en el control de concurencia (lectura-escritura rw y escritura-escritura ww), son variaciones de dos técnicas básicas que son:

- Two Phase Locking.
- Timestamp Ordering.

3.1. TWO PHASE LOCKING

Esta técnica [BERP81] sincroniza las operaciones de lectura y escritura detectando y previniendo posibles conflictos al actuar concurrentemente. El manejo de candados se hace de acuerdo con dos reglas. La primera de ellas consiste en que dos transacciones no pueden tener simultáneamente candados con conflicto. La segunda de ellas es que cuando una transacción ha otorgado un candado, ninguna otra puede asignar un candado adicional sobre el mismo dato, hasta que sea liberado.

Existe conflicto entre dos candados en la sincronía de rw, si ambos tratan de actuar sobre el mismo dato o si uno es de lectura y el otro de escritura. Para la sincronía de ww existe conflicto cuando dos candados tratan de actuar sobre el mismo dato o si ambos son de escritura.

El método de "two phase locking" (2PL), como su nombre lo indica, consiste en dos fases. En la primera, la transacción va obteniendo todos los candados que necesita durante su ejecución sin liberar ninguno. En la segunda, los libera y ya no puede volver a solicitar otro. Cuando la transacción termina o aborta, todos los candados que le quedaban son liberados automáticamente.

La variación más restrictiva a este método es la llamada "predeclaración", en la cual la transacción obtiene todos los candados antes de empezar a ejecutarse. Se analizaron cuatro variaciones en su implementación en bases de datos distribuídas, ésto se hizo porque nuestra idea fué la de no modificar el manejador de Bases de Datos con que se disponía. Cada usuario tiene su propio manejador pero todos trabajarán sobre una sola Base de Datos, lo que simula proceso distribuído.

3.1.1. BASIC 2PL IMPLEMENTATION

Consiste en tener un despachador (scheduler) al que las transacciones le van a enviar las peticiones de los candados o los avisos de que han sido liberados y deberá controlar que se cumplan con las reglas del 2PL (primera fase: otorgar candados, segunda fase: liberarlos). Cada transacción tendrá su área de trabajo en la que se harán los prewrites de la base de datos. Existe redundancia de información ya que varias transacciones pueden tener copia de una misma página de la base de datos por lo que cuando uno lo actualiza tiene que avisar a todos los que la tienen y nadie puede usarla hasta que se haya terminado de actualizar.

3.1.2. PRIMARY COPY 2PL

A diferencia del anterior, en éste método se usa una copia primaria en la que todas las transacciones tienen que registrar sus candados.

3.1.3. VOTING 2PL

Aprovecha la redundancia de la información pero solo puede ser utilizada en la sincronización ww. Consiste en solicitar un candado y esperar a que se le informe que en todas las copias que existen ya ha sido asignado el candado para que pueda realizar su operación

sin ninguna interferencia. Es inapropiada para la lectura ya que en lugar de poner un candado en su copia, tendría que esperar a que todas las copias lo tuvieran para poder seguir.

3.1.4. CENTRALIZED 2PL

Consiste en tener un solo despachador central al que todos los manejadores pedirán sus candados y avisarán cuando los liberen. No existirán copias por lo que habrá una mayor comunicación entre los procesos.

3.2. TIMESTAMP ORDERING

Es una técnica [BERP81] en la que se ordenan las acciones por sello de tiempo y en ese orden se ejecutan. Cuando existe un conflicto entre candados, se le asigna a aquel que tenga más antiguedad. Por este motivo, cuando a una transacción se le niega un candado, la siguiente vez que lo solicite tendrá mayor prioridad por tener más tiempo en el sistema. Cada transacción tiene asignado un sello se tiempo único otorgado por el manejador.

3.3. IMPLEMENTACION DEL CONTROL DE CANDADOS LOGICOS

La granularidad que se eligió fué el de una asociación ya que si nuestro gránulo fuera más pequeño nos ocasionaría un costo adicional en nuestro sistema. Otra razón es que el tamaño de nuestro sistema no es lo suficientemente grande como para que amerite un control de concurrencia con granularidad fina. Como se mencionó en la sección 2.3, tener una granularidad fina no es conveniente en la mayoría de los casos. Las asociaciones son fáciles de detectar por nuestro sistema y por consiguiente se puede llevar fácilmente su control de concurrencia.

Por otro lado, la asignación dinámica de candados no era conveniente en nuestro caso ya que como se dijo anteriormente, el tamaño de nuestro sistema (aproximadamente 7 procesos concurrentes), no hace necesario este tipo de asignación que además de incrementar considerablemente el tipo de proceso, haría muchos accesos a disco debido a la forma en que se tienen que comunicar nuestros procesos. Más adelante se explican con detalle estos puntos.

Como se vió en el capítulo anterior, en el método de two phase locking no podemos pedir un candado si ya

fue liberado alguno de los que teníamos. Es por eso que si lo queremos implementar tendríamos necesidad de un algoritmo que detectara o previniera la situación de abrazo mortal (deadlock). Esto es muy costoso además de que para cada candado que necesitemos, tenemos que pedírselo al despachador y esperar a que éste nos lo otorgue. Debido a las características de nuestro sistema operativo, no podemos comunicarnos directamente entre dos procesos por medio de tuberías (pipes) ya que tienen la limitación de que los dos procesos deben tener un ancestro en común. Por tal motivo, es necesario usar un archivo externo para comunicación por lo que la operación interactiva entre ambos procesos sería muy lenta.

Por lo anterior, el método elegido es una variación de two phase locking, llamado predeclaración que es el caso más restrictivo como se mencionó anteriormente. Esto significa que un programa empieza a ejecutarse cuando ya se le han concedido todos los candados que solicitó.

El primer paso para trabajar en forma concurrente, consiste en preprocesar el programa fuente. Esto es, se ejecuta un programa llamado "preprocesa" al que se le

pasa como parámetro el nombre del programa a preprocesar. El preprocesador hace un recorrido del programa seleccionando los candados que utilizará durante su ejecución. Los candados se colocan a nivel asociación. Existen dos tipos de candados que son de lectura y de escritura. Los candados de lectura (rlock) son aquellos que no modificarán la base de datos como get, exist etc. Los de escritura (wlock) son aquellos que modifican la base de datos como store, delete etc. El nombre de la asociación debe ser una constante escrita entre comillas ("asociación").

El preprocesador escribe en un archivo el nombre del programa y los candados que necesita, como se muestra en la figura 3-1.

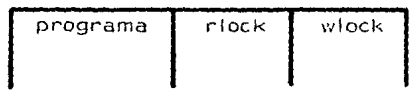


Figura 3-1: Archivo de programas preprocesados

Una vez preprocesado el programa, se puede ejecutar las veces que se desee sin que tenga que volver a preprocesarse.

Para ejecutar un programa, se corre un "lanzador" al que se le pasa como parámetro el programa a ejecutarse. Este "lanzador" busca en el archivo de preprocesados el programa y sus candados. Si no lo encuentra, manda mensaje de que falta preprocesar el programa. Si lo encuentra, lo copia a un archivo con el mismo formato, en el que se encuentran los programas que van a ejecutarse concurrentemente.

Se tiene un despachador que continuamente revisa el archivo que contiene los programas a ejecutarse. Cuando encuentra un programa, lo borra del archivo y lo incluye en una tabla en la que maneja los candados lógicos con el formato de la figura 3-2.

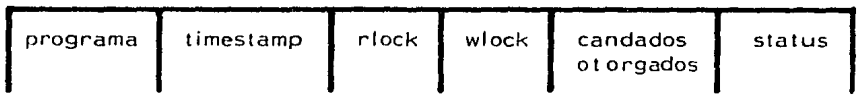


Figura 3-2: Tabla del despachador para control de candados lógicos

El "timestamp" o sello de tiempo, es otorgado por el despachador a cada programa en el momento en que se incluye en la tabla. El despachador otorga los candados bajo el siguiente criterio: Si es un rlock, lo otorga si

la asociación no tenía ningun candado o si tenía otro rlock puesto. Esto se debe a que dos o más programas pueden leer la misma información al mismo tiempo sin interferirse ya que ninguno está modificando la base de datos. Si la asociación tuviera un wlock, no podríamos otorgar el rlock porque estaríamos leyendo información no actualizada. Si es un wlock el que se está solicitando, sólo se otorga si la asociación no tiene ningún candado puesto.

Cuando el despachador ha otorgado todos los candados a un programa, le pone el status de "run", de lo contrario, le pone "wait". Una vez que el status es de "run", el despachador lanza a ejecución el programa de aplicación para que éste corra de manera independiente, empleando el siguente lanzador:

```
IF (FORK () == 0)
     EXECL (.....)
WAIT (&STATUS).
```

Este programa espera a que termine la aplicación para avisar al despachador. El aviso lo hace escribiendo en un archivo el nombre del programa que ha terminado. El despachador lee este archivo y a cada programa que ahí encuentra, le libera sus candados y lo borra de la tabla. Los candados liberados, son otorgados a aquel

programa que lo solicite siempre que el candado que está solicitando sea compatible con el que tenga la asociación, en caso de que tenga alguno y cuyo timestamp sea el más antiguo de todos los que lo estén solicitando.

CAPITULO 4

CANDADOS FISICOS

Los métodos para el manejo de candados físicos pueden catalogarse en dos tipos:

1. El primer tipo corresponde a aquellas soluciones en las que el dominio del actualizador no se modifica, lo que implica que un actualizador debe poner un candado a todo su dominio para que ningún otro actualizador pueda entrar.

2. El segundo tipo corresponde a aquellas soluciones en las que el dominio de un actualizador puede ser modificado durante la reestructuración. Unicamente se pondrán candados en los nodos afectados durante cada paso de la reestructuración. Estos nodos se conocen como el dominio local del actualizador.

El tipo que a nosotros nos interesa es el primer tipo, porque es el que garantiza que no se destruye la pista de acceso. Los métodos que aquí se analizarán corresponden a este tipo. En estas soluciones, el dominio del actualizador deberá tener un candado antes de que pueda empezar cualquier reestructuración. Este candado impedirá la entrada a cualquier otro actualizador al dominio mencionado, sin embargo, éstos podrán formarse en la cola de espera del nodo solicitado. Dependiendo de la solución en particular, los lectores puros, podrán en algunos casos entrar al

dominio. Existen 6 soluciones propuestas en la literatura para el tipo 1 [KWOY82], cuya explicación se encuentra a continuación.

4.1. SOLUCIONES DE SAMADI y PARR

La primera solución [KWOY82] que se propuso, fué la de Samadi en 1976, que se basa en la búsqueda de nodos en los árboles B. En 1977, apareció una solución parecida pero con búsqueda de hojas en árboles B, ésta fué propuesta por Parr. En estas dos soluciones se maneja un solo tipo de candado que es usado tanto por lectores puros como por actualizadores en los que dos procesos no podrán asignar un candado a un nodo simultáneamente. El control de concurrencia es muy sencillo y su gráfica de compatibilidad y convertibilidad se muestra en la figura 4-1.

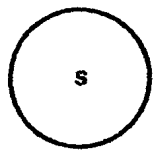


Figura 4-1: Gráfica de Compatibilidad y Convertibilidad para la solución de Samadi y Parr

El dominio para un actualizador se asignará de la siguiente forma: Primero el actualizador colocará un candado en la raíz y conforme va avanzando asigna candados en los nodos y los analiza. Si el nodo es seguro, se liberan los candados de todos sus ancestros.

El dominio para los lectores puros es el siguiente: El lector colocará un candado en la raíz y colocará el siguiente conforme avance en el árbol liberando el anterior, es decir, liberará un candado en el momento que haya asignado el siguiente.

Se deben tener ordenados todos los procesos que van a utilizar una ruta del árbol en común. Cuando un actualizador ha puesto un candado en su dominio, no podrán existir lectores puros ni actualizadores dentro de él. Ningún proceso podrá estar en la cola de espera de ninguno de los nodos que no sean el nodo seguro ya que ésto impilcaría que en sus ancestros hay candados y el actualizador no hubiera podido asignar el suyo.

4.2. SOLUCIONES DE BAYER y SCHKOLNICK

Existen tres soluciones propuestas por Bayer y Schkolnick [BAYR77] para el control de candados físicos, las cuales pertenecen al tipo uno de soluciones. En

todos los casos los candados son otorgados por el despachador y son libres de caer en una situación de abrazo mortal. La primera solución es muy parecida a la de Samadi y Parr excepto que existen dos tipos de candados que son de lectura y exclusivo. Un nodo no puede tener un candado de lectura y uno exclusivo simultáneamente. La gráfica de compatibilidad y convertibilidad se muestran en la figura 4-2.

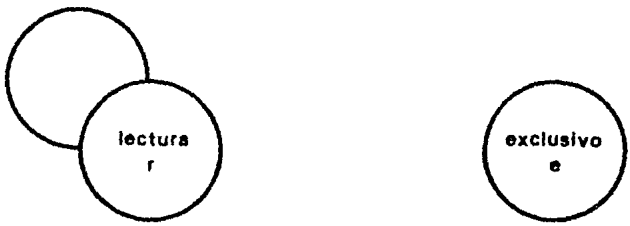


Figura 4-2: Gráfica de Compatibilidad y Convertibilidad primera y segunda solución de Bayer y Schkolnick

Se puede tener más de un lector puro leyendo un nodo simultáneamente. No existe un orden predefinido que deban de seguir los lectores por lo que pueden rebasarse entre ellos. Entre los actualizadores que estén usando la misma ruta de acceso si debe existir un orden para que no puedan meterse al dominio que tiene asignado otro

actualizador. Cuando un actualizador haya colocado un candado de tipo exclusivo, los lectores no podrán entrar.

Se puede observar que este método es más flexible que el de Samadi y Parr porque podemos tener varios lectores utilizando el mismo nodo.

El protocolo para los lectores es el siguiente:

```
        0) Place read-lock on root;
        1) Get root and make it the current node;
MAIN LOOP:
        2) While current node is not a leaf do
             {Exactly one read-lock is held by process}
             begin
             3) Place read-lock on appropiate son of
                current node;
             4) Release read-lock on current node;
             5) Get son of current node and make it
                current;
             end mainloop
```

Ejecutando este protocolo, el lector recorrerá el árbol B+ empezando en la raíz y siguiendo hacia abajo.

El protocolo para un actualizador es el siguiente:

```
        0) Place exclusive-lock on root;
        1) Get root and make it the current node;
MAIN LOOP:
        2) While current node is not a leaf node do
             {number of exclusive locks held >= 1}
             begin
             3) Place exclusive-lock on appropiate son
                of current node;
             4) Get son and make it the current node;
```

```
        5) If current node is safe then
              Release all locks held on ancestors
              of current node
        end
```

En la segunda solución de Bayer y Schkolnick se usan los mismos candados que en la primera solución. El protocolo para los lectores es el mismo que en la primera solución. La segunda solución de Bayer y Schkolnick se basa en la observación de que la reestructuración debida a los actualizadores ocurre una vez cada m/2 actualizaciones (donde m es el orden del árbol). Por lo que manejan a los actualizadores como si fueran lectores puros hasta llegar al nodo hoja. En ese momento se analiza si el nodo es seguro. En caso afirmativo, se coloca un candado exclusivo en el nodo hoja y se hace la actualización. En caso negativo, se usa el tratamiento de los actualizadores del método anterior.

El protocolo para los actualizadores es el siguiente:

```
        0) Place read-lock on root;
        1) Get root and make it the current node;
MAIN LOOP:
        2) While current node is not a leaf node do
              begin
              3) If son is not a leaf node then
                    Place read-lock on appropiate son
                 else
```

```
                Place exclusive-lock on appropiate
                son;
        4) Release lock on current node;
        5) Get son and make it the current node
        end mainloop;

6) (A leaf node has been reached)
   If current node is unsafe then
      Release all locks and repeat access to
      the tree, this time using the protocol
      for an updater as in solution 1;
```

El tercer método necesita un control de concurrencia que utiliza tres tipos de candados que son de lectura, de esritura y exclusivos cuya gráfica de compatibilidad y convertibilidad se muestra en la figura 4-3.

Los lectores puros y los lectores actualizadores se manejan igual que en la solución uno excepto que los lectores actualizadores utilizan un candado de escritura en lugar del candado de tipo exclusivo. En este método, cuando un lector actualizador ha puesto un candado de escritura en su dominio, entonces no se permitirá la entrada a otro lector actualizador, pero si podrán entrar los lectores puros. Antes de que se reestructure, el actualizador debe convertirse en escritor convirtiendo así los candados de escritura en exclusivos asegurándose así de que los lectores puros quedan fuera del dominio. Esto se logra colocando la petición de conversión en el primer lugar de la cola de espera de

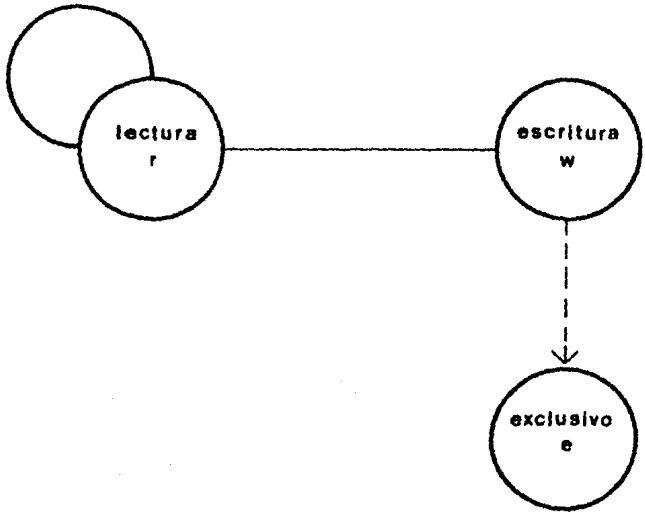


Figura 4-3: Gráfica de Compatibilidad y Convertibilidad tercera solución de Bayer y Schkolnick, solución de Ellis y solución de Yat-Sang Kwong Derick Wood

candados a ese nodo asegurando así que la conversión se lleva a cabo antes que cualquier otra asignación de ese nodo. Lo más importante de éste método es que los actualizadores, en su etapa de búsqueda permiten la entrada de los lectores dentro de su dominio y solo los sacan cuando van a reestructurar.

El protocolo para los actualizadores es:

0) Place a write lock on the root;

```
        1) Get the root and make it the current node;
MAIN LOOP:
        2) While current node is not a leaf node do
             {number of write locks held >=1}
             begin
             3) Place a write lock on appropiate son of
                current node.
             4) Get so n and make it the current node;
             5) If current node is safe then
                    Release all locks held on ancestors
                    of current node
             end mainloop;

        6) {A leaf node has been reached. At this time
           we can determine if update can be succesfully
           completed}

           If the update will be successful
           convert, top dowm, all read locks in
           exclusive locks.
```

4.3. SOLUCION DE ELLIS

Existen dos soluciones propuestas por Ellis [ELLC80] cada una de las cuales pertenece a un tipo distinto. La solución de tipo uno se basa en la tercera solución propuesta por Bayer y Schkolnick (ver figura 4.3) en la que los lectores y actualizadores recorren el árbol asignando los candados de lectura y escritura respectivamente. Cuando un lector actualizador ha colocado un candado de escritura en su dominio, entonces prohibe la entrada a otros actualizadores, sin embargo, los lectores si podrán entrar. Cuando el lector actualizador se convierte en escritor, en lugar de convertir sus candados en exclusivos y sacar a los

lectores, se les permite que permanezcan a éstos últimos en la mayor parte del dominio durante la reestructuración. En cada paso de la reestructuración el escritor cambia los candados a exclusivos en su dominio local, el cual consta de dos nodos y lo mantiene fuera de los límites de los lectores. Para continuar con el siguiente paso de la reestructuración, el dominio local es recorrido hacia arriba usando una técnica de acoplamiento de candados hasta que se llega al nodo seguro más profundo. La idea principal de este método es la de permitir a los lectores puros la entrada al dominio del actualizador durante las etapas de búsqueda y de reestructuración.

4.4. SOLUCION DE YAT-SANG KWONG y DERICK WOOD

La solución que proponen Kwong y Wood [KWOY82] se basa en las soluciones presentadas por Bayer, Schkolnick y Ellis. En esta solución se usan tres técnicas que son:

- Acoplamiento de candados.
- Mantener fuera a los lectores.
- Rama lateral (side-branching).

Las dos primeras técnicas ya fueron explicadas en los métodos anteriores. La que ahora se explicará es la

de "side branching". Es una técnica que se basa en hacer copias de las llaves y apuntadores para dejar su dominio intacto hasta el último momento. Como se puede observar, esta solución proporciona un alto grado de concurrencia entre los procesos porque no hace ninguna modificación a la estructura de árbol B y sus detalles de implementación permanecen invisibles. Esta solución utiliza los tres tipos de candados que son de lectura, de escritura y el exclusivo. Siendo su gráfica de compatibilidad y convertibilidad la misma de la tercera solución de Bayer y Schkolnick (ver figura 4-3).

En todas las soluciones propuestas en la literatura, una llave y un apuntador, siempre son adicionados en cada paso de reestructuración, empezando en la hoja. Si el nodo no es seguro, se vuelve sobresaturado y tiene que dividirse en dos "mitades" que consisten de [m/2]-1 y m-[m/2] llaves respectivamente. Esto nos trae como consecuencia que una hoja y una llave tengan que adicionarse al padre. Esto se repite hasta llegar al nodo seguro más profundo.

En el método que se presenta, en lugar de escribir una llave y un apuntador en cada paso de la reestructuración, un escritor usa la llave para

determinar si lo que se va a añadir debe ir en la mitad de la izquierda o de a la derecha del nodo hoja. Una vez elegida la mitad correspondiente, ésta es copiada en un nuevo nodo en el que es incrementada la llave y el apuntador y se va reestructurando esta rama del árbol hacia arriba pero sin tocar el árbol original por lo que otros procesos pueden estar leyendo al mismo tiempo la información del árbol original. Cuando se ha llegado al nodo seguro más profundo, entonces a éste se le convierte su candado de escritura a exclusivo, impidiendo así la interferencia de otro proceso en el momento en que se le adiciona la llave y el apuntador. Es importante hacer notar que la copia de la rama que se está actualizando no puede ser usada por ningún otro proceso.

Después de haber actualizado la mitad de los nodos correspondientes, solo falta borrar esas mitades de los nodos involucrados.

El proceso de inserción puede ser resumido en los siguientes pasos:

1. Poner un candado de escritura en su dominio.
2. Construir una rama lateral.
3. Agregar la rama lateral en el nodo seguro más

profundo después de convertir su candado de escritura a exclusivo.

4. Quitar la mitad de cada nodo que fué sustituída por la rama lateral, después de que se sacaron a todos los lectores del dominio del actualizador.

El algoritmo que se seguiría para la función de búsqueda es:

```
function SEARCH (K,T): boolean;
{determinar si la llave K está en el árbol T}

   begin
   {busca de arriba hacia abajo usando una
    técnica de acoplamiento de candados}

   r-lock(raíz de T);
   current := raíz de T;
   READ current (para encontrar el hijo adecuado
   que será al siguiente que se buscará);
   while not (current is a leaf) do
        r-lock(son);
        r-unlock(current);
        current := son;
        READ current to find appropiate son
   endwhile;
   SEARCH := K is in current;
   r-unlock(current);

   end



procedure INSERT (K,T);
{adicionar la llave K en el árbol T si K no
 existe}

   begin
   {procede a poner candados en el dominio
    usando una técnica de acoplamiento de
    candados}
```

```
w-lock(raíz de T);
current := raíz de T;
READ current (encuentra el hijo adecuado);
while not (current is a leaf) do
     w-lock(son);
     current := son;
     if current is i-safe then
             w-unlock ancestors of current
     endif;
     READ current to find the appropiate son
endwhile; {at this point the scope is
           w-locked}

{check if K is already present in T}
if K is in current then
     report K is already present;
     w-unlock current and its ancestors
else
     {restructure by side-branching}
     while not(current is i-safe) do
             CREATE a new node and append it
             to side branch;
             current := father of current
     endwhile;

     {add side-branch to the deepest safe
      node}
     convert(current,w,e);
     ADD side-branch to current;
     e-unlock(current);

     {remove redundant copies of keys and
      pointers}
     while not(current is a leaf) do
             current := son of current on
                        access path;
             convert(current,w,e);
             REMOVE appropiate half of
                    current;
             e-unlock(current)
     endwhile;
     report K has been inserted
endif

end
```

En los algoritmos anteriores se usan cuatro procedimientos que son: READ, ADD, CREATE y REMOVE. READ se usa por un lector para leer un nodo en la presencia de otros lectores. Su función es la de determinar el hijo apropiado que será leído posteriormente. ADD se usa por un escritor para agregar una llave y un apuntador a un nodo que al que ya le fué asignado un candado de modo exclusivo. CREATE se usa en un procedimiento de inserción para construir la rama lateral. Un nodo es creado al copiar la mitad apropiada del dominio que tiene asignados candados de escritura. Esta asignación la hizo el proceso. Una llave y un apuntador son agregados al nodo que es inaccesible para los demás procesos. El procedimiento REMOVE se usa para borrar una llave y un apuntador del nodo que tiene candado exclusivo.

Un proceso de borrado, primero actúa como un proceso lector-actualizador de manera semejante a la inserción. Si el argumento que se desea borrar no existe en el árbol, únicamente se tienen que liberar los candados y ahí termina. Si lo encuentra, entonces tiene que reestructurar el árbol. Si dos nodos hermanos tienen las suficientes llaves que al borrar una no lleguen al valor mínimo, entonces no hay necesidad de unirlos, sin

embargo, si alguno de ellos queda con un número menor al valor mínimo de llaves que puedan tener, entonces tienen que unirse dos nodos modificando así la llave del padre.

Las operaciones más importantes para un proceso de borrado, consiste en los siguientes pasos:

1. Colocar un candado de escritura en su dominio.
2. Reestructuración por medio de una unión de nodos y/o el cambio de una llave usando una rama lateral.
3. Borrar la rama descartada.

El algoritmo que se utiliza es el siguiente:

```
procedure DELETE (K,T);
{remove the key K from the tree T if K is
 present}

   begin
   {proceed to lock the scope using
    lock-coupling technique}
   w-lock(root of T);
   current := root of T;
   deepest-safe-node := current;
   READ current to find the appropiate son;
   while not (current is a leaf) do
        w-lock(son);
        current := son;
        if current is d-safe then
           deepest-safe-node := current;
           w-unlock all ancestors of current
        endif
        READ current to find the appropiate son
   endwhile; {at this point the scope is
              w-locked}
```

```
{check if K is in the tree}
if K is in current then
    {restructure by side-branching}
    rotate := false;
    while not (current is d-safe) do
        {perform node merging or rotation}
        w-lock (brother of current);
        if not (brother is d-safe) then
            {node merging}
            convert (brother,w,e);
            MERGE current into brother;
            e-unlock(brother);
            current := father of current
        else
            {rotation}
            rotate := true;
            convert (brother,w,e);
            convert (current,w,e);
            ROTATE brother,father and
            current;
            e-unlock (brother);
            if depth (current) > depth
            (deepest-safe-node) then
                {remove unnecessary
                 locks}
                e-unlock (father);
                w-unlock all ancestors
                of father
        endif
    endif
endwhile;

{detach redundant branch from a d-safe
 node and free storage}
if not (rotate) then
    convert (current,w,e)
endif;
REMOVE redundant branch from
       current;
e-unlock (current);
while not (current is a leaf) do
        current := son of current on
                   access path;
        convert (current,w,,e);
        e-unlock (current);
endwhile;
        free storage for current
{at this point restructuring is
```

```
        completed}
        report K has been removed
else
        {K is not in the tree}
        report K is not present;
        w-unlock current and its ancestors
endif

end
```

Se usan cuatro procedimientos en el algoritmo para remoción (delete). Estos son: READ, REMOVE, MERGE y ROTATE. Los procedimientos de READ y REMOVE fueron explicados con el algoritmo de INSERT. MERGE se usa para adicionar llaves y apuntadores de un nodo a su hermano que tiene un candado exclusivo con el separador correspondiente en el padre. ROTATE es el responsable de recorrer las llaves y apuntadores de un hermano y del padre para hacer que los nodos sean d-safe, los nodos involucrados deberán tener un candado exclusivo.

4.5. IMPLEMENTACION DEL CONTROL DE CANDADOS FISICOS

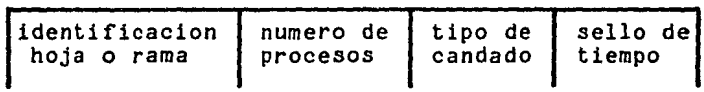
Por el tamaño de nuestra aplicación no es necesario un algoritmo muy sofisticado en cuanto a la asignación de candados ya que no sería muy eficiente por el costo adicional utilizado para su control. Por otro lado, el algoritmo de Samadi y Parr nos restringe demasiado al no permitir que dos transacciones lean simultáneamente el mismo gránulo. Por tal motivo, la solución elegida es la

primera de Bayer y Scholnick en la que se permite que dos o más lectores utilicen un gránulo simultáneamente (ver sección 4.2).

Con el sistema monousuario, cada vez que el manejador requería de una página (hoja o rama), las rutinas que hacían el acceso a éstas, revisaban si la página solicitada ya existía en la pila (ver capítulo 1). En caso afirmativo no realizaban ninguna actividad, mientras que en caso contrario, la traían a la pila sustituyendo alguna de las existentes, por medio de una política de LRU (Least Recently Used), es decir, se elimina la página que lleva más tiempo sin haber sido usada. Cada página en memoria se encontraba identificada en una estructura llamada hoja o rama según el caso.

Como se mencionó anteriormente este trabajo se hizo tratando de modificar lo menos posible la versión original, por lo que cada proceso tiene su propia copia del manejador y su propio buffer por lo que si un proceso modifica una página y ésta se encuentra en la pila de otro proceso, la información no será real. Recordamos que por las características de nuestro sistema operativo, no podemos comunicarnos directamente

entre dos procesos por medio de tuberías (pipes) ya que tienen la limitación de que los dos procesos deben tener un ancestro en común. Por tal motivo, para comunicarnos es necesario usar un archivo externo al cual se decidió llamarlo "candfis" en el que se encuentra la identificación de la página, el número de procesos que la tienen asignada (en el caso del candado de lectura rlock), el tipo de candado asignado y el sello de tiempo de la última vez que fué modificada. A cada página que se encuentra en la pila, se le asigna un sello de tiempo del momento en que se trajo a memoria.



Las rutinas que se encargan de hacer el acceso de las páginas se modificaron para que al revisar si una página se encuentra en la pila, revisen también si la versión que tienen está actualizada. Esto se hace comparando el sello de tiempo de la página de la pila con el sello de tiempo del archivo "candfis", es decir, con el que tiene la página en disco. En el caso en que la página solicitada no se encuentre en la pila o si la

que tenemos no está actualizada, entonces se trae de disco.

Al terminar de hacer referencia a una página se revisa si ésta fué modificada. En caso afirmativo, se copia a disco y se actualiza el sello de tiempo. Esto se hace antes de liberar el candado para asegurarnos de que a nadie le sea asignada la página antes de ser actualizada.

Al asignar un candado se incrementa en uno el número de procesos que tienen asignada esa página y se actualiza el tipo de candado de acuerdo con el algoritmo. Como en la pila solo cabe un número pequeño de páginas y los candados se colocan a lo largo de la ruta de acceso de acuerdo con el algoritmo, en ocasiones se tendrán asignados algunos candados en páginas fuera de la pila por lo que es importante controlarlos y es debido a ésto que se usó una estructura epecial para manejar los candados de cada proceso. La información que utiliza es la identificación de la página, el sello de tiempo de cuando fué asignado el candado y el tipo de éste.

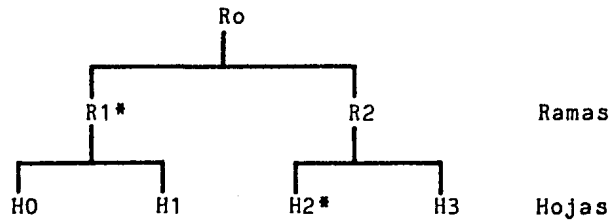
identificación hoja o rama	sello de tiempo	tipo del candado

Mediante el uso de esta estructura se pueden liberar todos los candados anteriores a un nodo seguro, así como la liberación de todos los candados al finalizar el proceso.

Al liberar un candado se disminuye en uno el número de procesos que tienen asignada esa página por lo que cuando llega el valor de cero, se borra el tipo de candado quedando libre totalmente para poder ser asignado posteriormente a otro proceso que lo solicite. Cabe hacer notar que cuando el candado es de escritura (write-lock) el número de procesos solo podrá ser uno.

Ejemplo:

Supongamos que tenemos la siguiente base de datos:



* Nodo Seguro

La transacción T1 solicita un candado de lectura para H0 en el tiempo 1 y es asignado porque nadie lo tiene.

tabla del manejador 1

ident	tiempo	tipo
H0	1	rlock

tabla general

ident	procesos	tipo	tiempo
H0	1	rlock	0

El cero en el dato tiempo de la tabla general significa que la hoja no ha sido modificada.

La transacción T2 solicita un candado de lectura para H0 en el tiempo 2 y es asignado porque un candado de lectura es compatible con si mismo.

tabla del manejador 1

ident	tiempo	tipo
H0	1	rlock

tabla general

ident	procesos	tipo	tiempo
H0	2	rlock	0

tabla del manejador 2

ident	tiempo	tipo
H0	2	rlock

Como se puede observar, el 2 en el numero de procesos de la tabla general, significa que hay dos procesos que

están utilizando la hoja para leerla en forma simultánea.

La transacción T1 libera el candado de lectura de H0 por lo que queda vacía la tabla del manejador 1 y se decrementa el número de procesos de la tabla general.

tabla del manejador 2

ident	tiempo	tipo
H0	2	rlock

tabla general

ident	procesos	tipo	tiempo
H0	1	rlock	0

La transaccion T2 libera el candado de lectura de la hoja H0 quedando las tablas sin candados.

La transaccion T1 solicita el candado de escritura en la hoja H1.

tabla del manejador 1

ident	tiempo	tipo
R1	3	wlock
H1	4	wlock

tabla general

ident	procesos	tipo	tiempo
R1	1	wlock	0
H1	1	wlock	0

El candado que existe en R1 es porque éste es el nodo seguro más cercano a la hoja H1.

La transacción T2 solicita un candado de escritura en H0, la tabla del manejador dos está vacía por lo que se podría pensar que los candados pueden ser otorgados, sin embargo, al consultar la tabla general, observamos que la rama R1 que necesitamos está asignada a otro proceso por lo que se tiene que esperar a que sea liberada.

La transacción T1 actualiza la hoja H1 en el tiempo 5, por lo que solo se actualiza la tabla general ya que las tablas de los manejadores no tienen cambios.

tabla general

ident	procesos	tipo	tiempo
R1	1	wlock	0
H1	1	wlock	5

Cada vez que el manejador trae una hoja o una rama a la pila, la marca con un sello de tiempo. Esto se hace con el fin de saber si la version que tenemos es la más reciente ya que se compara este sello de tiempo con el que tiene la tabla general (que en este caso tiene el valor de 5).

CAPITULO 5

CONCLUSIONES

Mediante el desarrollo de la presente tesis se logró el objetivo que se perseguía "Control de concurrencia en un sistema manejador de Bases de Datos binarias asociativas", el cual se basó en los algoritmos de Two Phase Locking con la variación de predeclaración y la primera solución de Bayer y Schkolnick para el control de candados lógicos y físicos respectivamente. A continuación se presentan las puntos más importantes del sistema así como sugerencias de posibles mejoras al mismo.

5.1. CARACTERISTICAS RELEVANTES DEL SISTEMA

Es necesario preprocesar el programa de aplicación para identificar los candados que necesita.

El nombre de las asociaciones deben ir entre comillas en el programa de aplicación, eliminando la posibilidad de poder asignarlo a una variable.

El control de concurrencia "lógico" se hace de manera independiente del manejador original ya que todo se hace antes de que éste se empiece a ejecutar.

El proceso que controla la concurrencia se

encuentra "dormido" en el sistema y se "despierta" en el momento en que se quiere ejecutar la primera aplicación para volverse a "dormir" cuando la última ha concluído. Esto es transparente para el usuario.

El control de concurrencia "físico" se hace dentro del manejador original utilizando las rutinas del archivo "mbdconcur"

Cada programa de aplicación utiliza su propia copia del manejador. La comunicación entre ellas es lenta ya que se hace a través de archivos (ver secciones 3.4 y 4.5).

5.2. EVOLUCION DEL SMBDBA

Los algoritmos utilizados se seleccionaron por creerse que son los más adecuados para la aplicación, sin embargo, sería interesante modificar estos algoritmos para comparar los resultados y saber en realidad si es el óptimo para nuestra aplicación. Esto sería posible ya que las rutinas que contienen estos algoritmos se incorporaron en forma modular al manejador, por lo que no implica ninguna dificultad su sustitución.

Sería de utilidad el optimizar la comunicación

entre los procesos cuando el sistema operativo lo permita para incrementar la rapidez del sistema.

REFERENCIAS

[ASTM76] Astrahan, M.M., et al.;"System R: Relational approach to database management", ACM Trans. Database Systems, Vol.1, No.2, June 1976, 97-137.

[BAYR77] Bayer, R. and Schkolnick, M.; "Concurrency of Operations on B-Trees", Acta Informatica, 1977, 1-23.

[BERP81] Bernstein, P.A. and Goodman, N.; "Concurrency Control in Distributed Database Systems", Computing Surveys, Vol.13, No.2, June 1981, 185-221.

[BUCA80] Buchmann, A.P.;"A Methodology for Logical Design of Databases for Project Engineering", Ph.D. Thesis, Dept. of Chemical Engineering, The University of Texas, Austin, May 1980.

[BUCA82] Buchmann, A.P., Leesley, M.E., Dale, A.G.;"The role, structure and design of database for project engineering", Chemmical Eng. Comm., to appear.

[COMD79] Comer, D.;"The Ubiquitous B-tree", Computing Surveys, Vol.11, No.2, June 1979, 121-137.

[CHEE81] Cherlin, E. "The UNIX operating system: portability a plus", MINI-MICRO SYSTEMS, April 1981, 153-159.

[DATC75] Date, C.J.;An introduction to database systems, Reading Mass., Addison-Wesley 1975.

[ELLC80] Schlatter Ellis, C.;"Concurrent Search and Insertion in 2-3 Trees", Acta Informatica 14, 1980, 63-86.

[GRAG79] Gardarin, G.; "A Unified Overview of Algorithms for Updating Partially Redundant Distributed Databases", NCC 1979, 293-318.

[GRAJ75] Gray, J.N., Lorie, R.A. and Putzulu, G.R.;"Granularity of Locks in a Shared Database", Proceedings of the International Conference on Very Large Data Bases, Vol.1, No.1, September 1975, 428-451.

[KORH82] Korth, H.F.;"Deadlock Freedom Using Edge Locks", ACM Trans. Database Systems, Vol.7, No.4, December 1982, 632-652.

[KORH83] Korth, H.F.;"Locking Primitives in a Database System", Journal of the Association for Computer Machinery, Vol.30, No.1, January 1983, 55-79.

[KWOY82] Kwong, Y., and Wood, D.;"A New Method for Concurrency in B-Trees", ACM Trans. Soft. Engineering, Vol. SE-8, No.3, May 1982, 211-222.

[PALW82] Palacios Castrillo, W.; Sistema manejador de bases de datos binarias asociativas. Tesis Lic. Ing. en Computación, Fac. de Ingenieria, UNAM., Mexico D.F., mayo 1982.

[RIED79] Ries, D.R., Stonebraker, M.;"Locking Granularity Revisted", ACM Trans. Database Syst. Vol.4, No.2, June 1979, 210-227.