Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

INTEGRIDAD Y CONTROL DE CONCURRENCIA

María Gertrudis López

Centro de investigación en Sistemas de Información
CISI.

INDICE

1 Control de concurrencia [1, 7, 9]

Si se da como entrada un estado correcto de la base de datos, una transacción individualmente correcta producirá un estado correcto de la base de datos como salida si es ejecutada en forma aislada. En un sistema multiusuario, con transacciones que se ejecutan concurrentemente, pueden haber interferencias entre ellas de forma tal que se produzcan resultados no correctos. Tales interferencias pueden tomar muchas formas, como por ejemplo el problema de la actualización perdida. Tales problemas pueden ser manejados mediante técnicas de control de concurrencia, tales como bloqueos o timestamping.

1.1 Correctitud en la ejecución concurrente de transacciones

Dadas las siguientes transacciones que se ejecutan concurrentemente:

TRANSACCION A	TIEMPO	TRANSACCION B
-		
-		
-		
FIND R	t1	-
copy RF into ATEMP		-
-	t2	FIND R
-		copy R.F into BTEMP
.		
-		-
-		-
UPD R:	t3	-
REPLACE RF		-
by ATEMP + 1		-
-		-
-	t4	UPD R
-		replace RF
-		by 2 * BTEMP
-		-

La transacción A se propone sumar 1 al campo F del registro R
La transacción B se propone multiplicar por 2 ese mismo campo.
Si el valor inicial de F es 4 se tiene que:
- Si la transacción A se ejecuta antes que la transacción B se tiene que F = 10, ya que

Transacción A: 4 + 1 = 5
Transacción B: 5 * 2 = 10

- Si la transacción B se ejecuta antes que la transacción A se tiene que F = 9, ya que

Transacción B: 4 * 2 = 8
Transacción A: 8 + l = 9

Cualquiera de los dos valores anteriores es considerado un resultado final correcto ya que se cumple que "Si se da como entrada un estado correcto de la base de datos, una transacción individualmente correcta producirá un estado correcto de la base de datos como salida si es ejecutada en forma aislada". A partir de esto se puede definir el concepto de correctitud en un ambiente de ejecución concurrente de transacciones.

Correctitud: Una ejecución concurrente de un conjunto de transacciones (t1, t2, ..., tn) es correcta si y sólo si existe una secuencia de dichas transacciones que ejecutadas serialmente darían los mismos resultados.

Se dice entonces que si el concepto de correctitud se cumple el conjunto de transacciones es serializable.

Para examinar la correctitud de la ejecución concurrente de un conjunto de transacciones hay que definir la precedencia existente entre las mismas. Entonces se tiene que una transacción A precede a una transacción B si la transacción B ve un dato que la transacción A modificó o si la transacción A ve un dato que la transacción B modificará.

En el ejemplo anterior se tiene que la ejecución concurrente de A y B no es correcta ya que A precede a B porque A ve un dato que B modificará y al mismo tiempo B precede a A ya que B ve un dato que A modificará. Entonces no se puede establecer una serialización posible entre ambas transacciones. Si ambas transacciones se ejecutan concurrentemente se obtiene que el valor de F es 8, ya que:

Transacción A: 4

+ 1 = 5

Transacción B: 4 *

2 = 8

Este resultado no se corresponde con ninguno de los obtenidos por la ejecución serial de ambas transacciones y se dice que en este caso se presenta el problema de la actualización perdida, ya que la actualización que hace A se pierde porque B la sobrescribe.

Es claro, entonces, que en un ambiente multiusuario es necesario algún mecanismo de control de concurrencia con el fin de evitar estos problemas y otros. El problema esencial aquí es que ambas transacciones A y B actualizan RF sobre la base del valor inicial del campo y ninguna ve la salida de la otra transacción. Para prevenir esta situación hay tres cosas básicas que puede hacer un mecanismo de control de concurrencia:

a) Puede impedir el FIND de B en el tiempo t2 en base al hecho de que la transacción A ya ha direccionado al registro R y puede ser que vaya a actualizarlo posteriormente.

b) Puede impedir la actualización de A en el tiempo t3 en base al hecho de que B ya ha direccionado al registro R y B ya ha visto el valor de R antes de la actualización.

c) Puede impedir la actualización de B en el tiempo t4 en base al hecho de que A ya ha actualizado a R y entonces la actualización de B va a estar basada en un valor obsoleto de R

1.2 Técnicas de Control de Concurrencia

1.2.1 Bloqueos

El bloqueo es una técnica de control de concurrencia que regula el acceso concurrente a objetos compartidos tales como los registros de una base de datos. Una transacción puede obtener un bloqueo sobre un registro haciendo un requerimiento a un componente del sistema llamado manejador de bloqueos. El bloqueo puede dar la idea de un bloque de control que incluye, entre otras cosas, la identificación del registro que es bloqueado y la identificación de la transacción que bloquea el registro. Si una transacción T mantiene un bloqueo sobre un registro R entonces se le dan a la transacción T ciertas garantías sobre el registro R, por ejemplo, T siempre va a tener garantizado el hecho que ninguna transacción concurrente va a poder actualizar R hasta que T levante su bloqueo. La naturaleza precisa de las garantías depende del tipo de bloqueo.

1.2.1.1 Bloqueo Exclusivo

Bloqueo Exclusivo: Si una transacción T mantiene un bloqueo exclusivo sobre algún objeto (digamos un registro de la base de datos), entonces ninguna transacción distinta T

puede adquirir un bloqueo de ningún tipo sobre ese objeto hasta que la transacción T libere su bloqueo.

El bloqueo exclusivo provee la base para resolver el problema de actualización perdida planteado anteriormente. Para hacer uso de este tipo de bloqueo se define un protocolo llamado protocolo PX que dice:

Protocolo PX: Cualquier transacción que intente actualizar un registro R debe primero ejecutar un XFIND R para obtener direccionabilidad sobre el registro y para adquirir el bloqueo exclusivo sobre él. Si el bloqueo no puede ser adquirido en ese momento la transacción entra a un estado de espera (el XFIND es acepta pero no es otorgado en ese momento); la transacción va a reasumir su ejecución cuando el registro este disponible y el bloqueo pueda ser garantizado.

El sistema debe garantizar que una transacción que ha sido forzada a esperar va a salir eventualmente de ese estado (a menos que esa transacción sea seleccionada como victima de un abrazo mortal). Una técnica simple para proveer tal garantía es servir a todos los requerimientos de, bloqueo para un objeto dado como una cola FIFO (First In - First Out). Si el sistema no provee tal garantía entonces es posible que una transacción caiga en un estado de espera indefinida.

En efecto de aplicar el protocolo PX al problema planteado en el punto 5.2.2 es:

TRANSACCION A	TIEMPO	TRANSACCION B
XFIND R	t1	-
copy R.F into ATEMP		-
-	t2	XFIND R
-		wait
		wait
UPD R:	t3	wait
REPLACE R.F		wait
by ATEMP + 1	-	wait
XRELEASE R	t4	wait
	t5	(recomienza) XFIND R
		copy R.F into BTEMP
-	t6	UPD R:
-		replace R.F
-		by 2 * BTEMP
-	t7	XRELASE R

En este caso B comienza a esperar en el tiempo t2 porque su requerimiento de bloqueo exclusivo sobre R entra en conflicto con el bloqueo exclusivo sobre R ya otorgado a la transacción A. La transacción B recomienza su ejecución después que la transacción A libera su bloqueo. El efecto es que se fuerza a B a ver el valor de R después de que ha sido actualizado por A.

Problemas de protocolo PX: Este protocolo puede producir problemas de abrazo mortal. El abrazo mortal es una situación en la cual dos o más transacciones están en un estado de espera simultaneo, cada una esperando por el hecho de que alguna de las otras libere un bloqueo antes de que se puedan seguir ejecutando. Un ejemplo de abrazo mortal entre dos transacciones se presenta en el siguiente ejemplo:

TRANSACCION A	TIEMPO	TRANSACCION B
XFIND R1	t1	-
-		
	t2	XFIND R2
XFIND R2	t3	-
wait	-	-
wait	t4	XFIND RI
wait .		wait
wait		wait

Como se sabe, el problema del abrazo mortal se ha estudiado profundamente en otros ambientes como por ejemplo en sistemas operativos, pero las soluciones encontradas en estos ambientes no son todas aplicables a los problemas de abrazo mortal en un ambiente de base de datos puesto que existen ciertos aspectos que lo diferencia de otros ambientes. Entre estos aspectos se pueden mencionar los siguientes:

- El conjunto de objetos bloqueables no es sólo muy grande (posiblemente millones de registros) sino también que este conjunto cambia dinámicamente (los registros se crean y destruyen constantemente) .
- Los objetos bloqueables (registros) son direccionados típicamente no por nombre sino por contenido, así que los requerimientos se pueden determinar a momento de ejecución.
- El alcance preciso del bloqueo (el conjunto especifico de registros requeridos) sólo es determinado usualmente en forma dinámica.

Todo esto permite ver que el sistema debe estar preparado para la posibilidad de abrazo mortal; entonces debe poder detectarlos y resolverlos.

El detectar abrazos mortales es básicamente un asunto de detectar un ciclo en un grafo de espera (esto es, el grafo de quien esta esperando por quien a nivel de bloqueos de transacciones). En un grafo de espera los nodos representan las transacciones y los arcos representan las esperas. El sistema coloca un arco desde el nodo Ti hasta el nodo Tj cuando la transacción Ti requiere un objeto que mantiene bloqueado la transacción a Tj y se borra al arco cuando Tj libera el bloqueo sobre el objeto en cuestión

El chequeo de abrazos mortales en un grafo de espera puede hacerse cuando un requerimiento de bloqueo causa una espera o cada cierto intervalo de tiempo. Resolver un abrazo mortal consiste en seleccionar una victima a partir de las transacciones que están envueltas en el abrazo mortal y hacerle rollback La victima no es necesariamente la que ha causado el abrazo mortal, sino que puede ser la más joven, la que tenga menos bloqueos adquiridos o la que haya hecho menos actualizaciones. El proceso de rollback envuelve no sólo terminar la transacción y echar para atrás todas sus actualizaciones sino también liberar todos sus bloqueos pudiendo ser utilizados estos recursos por las otras transacciones.

La pregunta es: ¿Qué le sucede a la victima después de que ha ocurrido un rollback? Algunos sistemas como el System R retornan el control a la victima con un código de error que indica que ha ocurrido un rollback. El programa de aplicación se sigue ejecutando y puede, por ejemplo, iniciar otra transacción para intentar rehacer las actualizaciones deshechas. En otros sistemas como IMS se reinicia la transacción automáticamente utilizando los mismos parámetros de entrada de tal manera que el usuario no se entera que un rollback ha ocurrido.

Otro problema que se puede presentar cuando se hace rollback a una transacción antes de que haya alcanzado una terminación exitosa es el problema de datos fantasmas. En el ejemplo siguiente la transacción B ve un dato que nunca existió o dato fantasma en el tiempo t4 y lo actualiza en el tiempo t5 en base, a ese valor fantasma ya que cuando se le hace rollback a la transacción A en el tiempo t6 se deshacen las actualizaciones realizadas por A sobre el registro R y recobra el valor original que tenia antes de que A se ejecutara.

TRANSACCION A	TIEMPO	TRANSACCION B
XFIND R	t1	-
UPD R	t2	XFIND R2
XRELEASE R	t3	
		-
-	t4	XFIND R
-	t5	UPD R
* * *ABORT* * *	t6	

El problema anterior se evita si se asegura que la transacción A no va a liberar el bloqueo adquirido sobre R en un punto donde se garantice que la actualización que realizó

no se le va a hacer rollback, esto es al momento del commit. Entonces surge el protocolo PXC el cual se deriva del protocolo PX más una regla adicional:

Protocolo PXC: Cualquier transacción que intente actualizar un registro R debe primero ejecutar un XFIND R para obtener direccionabilidad sobre el registro y para adquirir un bloqueo exclusivo sobre él. Si el bloqueo no puede ser adquirido en ese momento la transacción entre a un estado de espera (el XFIND es aceptado pero no es otorgado en ese momento); la transacción va a reasumir su ejecución cuando el registro esto disponible y el bloqueo pueda ser garantizado. Los bloqueos exclusivos son retenidos hasta el **final de la transacción (COMMIT o ROLLBACK).**

1.2.1.2 Bloqueo Compartido

Una transacción puede necesitar retener data bloqueada cuando no la vaya a actualizar. En el siguiente ejemplo se tienen dos transacciones trabajando sobre registro de tipo CUENTA La transacción A calcula un total de CUENTAS y la transacción B transfiere una cantidad de 10 desde la CUENTA 3 a la CUENTA 1. Al ejecutarse concurrentemente estas transacciones, la transacción A produce un resultado incorrecto de 110 en vez de 120 ya que vio a la CUENTA 1 con 40 en vez de 50.

ACC1 = 40	ACC2 = 50	ACC3 = 30
TRANSACCION A	*TIEMPO*	TRANSACCION B
FIND ACC1 (40):	t1	-
sum = 40	-	-
FIND ACC2(50):	t2	-
sum = 90	-	-
-	t3	-
-	t4	XFIND ACC3 (30)
-		UPD ACC3: RESTAR 10 (30->20)
-	t5	XFIND ACC1 (40)
-	t6	UPD ACC1:
-	t7	SUMAR 10 (40->50)
		COMMIT
FIND ACC3 (20)		
sum = 110	t8	

La transacción A debería bloquear los registros CUENTA al menos hasta que la suma esté completa y así evitar que B actualice un registro que A ya ha visto. El bloqueo que necesita hacer la transacción A no tiene necesariamente que ser un bloqueo exclusivo ya que A puede permitir que otras transacciones vean el registro pero lo que no puede permitir es que lo actualicen mientras ella lo esta viendo. Entonces surge un nuevo tipo de bloqueo llamado bloqueo compartido (o bloqueo S).

Bloqueo compartido: Si una transacción T retiene un bloqueo compartido sobre algún objeto (digamos un registro de la BD), entonces una transacción distinta T' puede adquirir un bloqueo compartido sobre ese objeto, pero ninguna transacción distinta T' puede adquirir un bloqueo exclusivo sobre ese objeto hasta que todos los bloqueos compartidos existentes sobre ese objeto sean liberados.

Protocolo PS: Cualquier transacción que intente actualizar un registro R debe ejecutar primero un "SFIND R" para obtener direccionabilidad sobre el registro y adquirir el bloqueo compartido. Después de que la transacción, ha adquirido el bloqueo, cualquier intento subsecuente por actualizar el registro debe ser hecho mediante una operación "UPDX R", la

cual no solo actualiza el registro sino que también promueve el bloqueo compartido existente sobre el registro a bloqueo exclusivo.

Para solucionar el mismo problema que resuelve el protocolo PXC también se define un protocolo similar PSC.

En el caso del ejemplo anterior la transacción A debe hacer un bloqueo compartido para cada registro CUENTA y usar “SRELEASE ALL" para liberar los bloqueos compartidos después de completar la suma o puede liberarlos al final de la transacción pero, como se ve en la siguiente figura, el resultado final es que ocurre un abrazo mortal en el tiempo t6.

ACC1 = 40	ACC2 = 50	ACC3 = 30
TRANSACCION A	TIEMPO	TRANSACCION B
SFIND ACC1 (40):	t1	-
sum = 40	-	-
SFIND ACC2(50):	t2	-
sum = 40		
	t3	-
-	t4	SFIND ACC3 (30)
-	-	UPDX ACC3:
-	-	RESTAR 10
		(30 -> 20)
-	t5	SFIND ACC1 (40).
-	t6	UPDX ACC 1:
-	t7	wait
-	-	wait
SFIND ACC3 (20)	t8	wait
	-	wait
wait	-	wait

1.2.1.3 Bloqueo de Actualización

Como se vio anteriormente, el protocolo PS tiende a producir un gran número de abrazos mortales Por otro lado, el protocolo PX ofrece un bajo nivel de concurrencia. Por todo esto algunos sistemas (como el IMS) emplean un protocolo diferente, el protocolo PU. Este protocolo envuelve un nuevo tipo de bloqueo, el bloqueo de actualización o bloqueo U.

Bloqueo U: Representa una indicación de que una transacción puede querer actualizar el registro; es compatible con Bloqueos S pero no con otros bloqueos U ni con bloqueos X.

Protocolo PU: Cualquier transacción que intente actualizar un registro R debe ejecutar primero un "UFIND R" para obtener direccionabilidad sobre el registro y adquirir un bloqueo de actualización sobre él. Después de que la transacción ha adquirido el bloqueo cualquier actualización subsecuente del registro ("UPDX R") va a promover el bloqueo a nivel exclusivo.

También se define el protocolo PUC, similar al PSC y al PXC, estipulando que el bloqueo exclusivo sea retenido hasta el final de la transacción.

La interacción entre los bloqueos compartidos de actualización v exclusivos pueden expresarse mediante la siguiente matriz de compatibilidad.

T’ \ T	X	U	S	-
X	N	N	N	S
U	N	N	S	S
S	N	S	S	S
-	S	S	S	S

Por ejemplo, el siguiente caso de abrazo mortal se soluciona utilizando el protocolo PU:

TRANSACCION A	TIEMPO	TRANSACCION B
SFIND R	t1	-
copy RF into ATEMP		
-	-	-
-	t2	SFIND R
-		copy RF into BTEMP
UPD R:	t3	-
wait		-
wait		-
wait		-
wait	t4	UPD R:
wait		wait
wait		wait
wait		wait

TRANSACCION A	TIEMPO	TRANSACCION B
UFIND R	t1	-
copy R.F into ATEMP		-
	t2	UFIND R
		wait
-		wait
-		wait
UPDX R:	t3	wait
REPLACE R.F		wait
by ATEMP + I		wait
COMMIT		wait
		RECOMIENZO
		copy R.F into BTEMP
	t4	UPD R:
		replace R.F by 2 * BTEMP
		COMMIT

1.2.1.4 Bloqueo en dos fases

Cualquier transacción que después de liberar un bloqueo adquiere otro siempre corre el riesgo de producir resultados incorrectos. Esto es, siempre es posible definir una segunda transacción que pueda ejecutarse concurrentemente con la primera de manera tal que la ejecución intercalada o concurrente de ambas no sea serializable y por ende no correcta. Se define el siguiente teorema:

Teorema: Si todas las transacciones obedecen las siguientes reglas:

a) Antes de operar sobre cualquier objeto la transacción debe adquirir primero un bloqueo sobre ese objeto; y

b) Después de liberar un bloqueo la transacción no adquiere ningún otro bloqueo

Entonces, todas las ejecuciones intercaladas de esas transacciones son serializables.

Una transacción que obedece las reglas a) y b) se dice que satisface el protocolo de bloqueo en dos fases. Las dos fases son una fase creciente durante la cual los bloqueos son adquiridos, y una fase decreciente durante la cual los bloqueos son liberados.

El teorema anterior no establece que todas las transacciones tengan que estar en dos fases para que sean serializables. La condición de que todas las transacciones estén en dos fases es suficiente pero no necesaria para garantizar la seriabilidad.

Desafortunadamente, el teorema anterior es un poco engañoso. Las reglas a) y b) son establecidas apropiadamente, pero debe entenderse que algunas veces el objeto a ser bloqueado no existe. Por ejemplo:

Sean F y G campos en la base de datos y sean las transacciones A y B definidas como:

```
(A) if F exists
        then G : = 1;
        else G : = 0;
(B) if F does not exist then
        do;
           create F;
           G := 1;
        end;
```

Supongamos que inicialmente F no existe y se considera la siguiente ejecución intercalada de A y B:

TRANSACCION A	TIEMPO	TRANSACCION B
FIND F not found	t1	-
	t2	FIND F not found
-	-	
-	t3	INSERT F
-	t4	XFIND G
-	t5	UPD G: replace G by 1
-	-	
-	t6	COMMIT
XFIND G	t7	
UPD G: replace G by 0 COMMIT	t8 -	

Si A y B se ejecutan serialmente hay un sólo resultado correcto posible ya que
Si A entonces B: F existe, G = 1
Si B entonces A: F existe, G = 1

Sin embargo, la ejecución intercalada de ambas transacciones produce que F existe, G := 0 y esto las hace no serializables y por ende se tiene un resultado incorrecto. Para forzar la seriabilidad, se necesita impedir que B cree el registro F ya que A vio anteriormente que no existe. Para evitar que B cree un fantasma A necesita aplicar un bloqueo a la no-existencia de F en el tiempo t1. Los dos "FIND F" deben ser cambiados por "SFIND F". El SFIND de la transacción A va a ser exitoso y F va a poder ser bloqueado.

1.2.1.5 Calendarización de transacciones (Transaction Scheduling)

Este enfoque envuelve la calendarización de las transacciones para que su ejecución sea de una forma tal que dos transacciones no van a correr concurrentemente si sus requerimientos de data están en conflicto. Esto obviamente requiere que cada requerimiento de data sea conocido antes del momento de ejecución, lo que implica la declaración explicita de esos requerimientos o un análisis automático del programa a tiempo a compilación. Sin embargo, los requerimientos precisos de datos de una transacción dada, por lo general, no son conocidos hasta el momento de ejecución; entonces, este enfoque tiende a ser innecesariamente pesimista puesto que si se sabe que una transacción va acceder registros tipo X, el requerimiento de data de la transacción X se expande al conjunto completo de registros tipo X no sólo a los registros específicos a acceder por lo que, cualquier otra

transacción que vaya a acceder registros tipo X no podrá ejecutarse concurrentemente con la primera aunque los registros a acceder por cada una sean registros tipo X diferentes.

En general se puede decir que esta técnica es realmente un esquema de bloqueo en el cual a) La unidad bloqueable es el conjunto completo de registros en vez de un registro individual y b) los bloqueos son aplicados al momento de inicio del programa en vez de hacerlo durante la ejecución.

1.2.1.6 Rechazo de requerimiento

Una segunda forma de prevenir abrazos mortales es rechazar cualquier requerimiento de bloqueo que si fuera aceptado pudiera causar un abrazo mortal, retornando una indicación de requerimiento denegado a la transacción. Se dice que el aceptar un requerimiento de bloqueo causa un abrazo mortal si y solo si el bloqueo no puede ser otorgado inmediatamente, la transacción entra en estado de espera y al añadir el arco correspondiente al grafo de espera este cierra un ciclo en él.

Esta solución es un poco más flexible que la técnica de solución de abrazos mortales (que implica seleccionar una victima y hacerle rollback). Con la técnica de rechazo de requerimiento la transacción a la que se le negó el requerimiento puede esperar un tiempo corto a intentarlo de nuevo y por ejemplo, si se cumple su lapso de espera permitido puede generar el mensaje de salida especifico convirtiéndose así en un rollback explicito.

1.2.1.7 Retroceso de la transacción

Esta técnica fue propuesta en el contexto de sistemas distribuidos, pero puede usarse también en sistemas centralizados. Tiene dos versiones "Espera-Muere" (Wait-Die) y "Hiere-Espera" (Wound-Wait). La idea básica en esta técnica es evitar la creación de un ciclo en el grafo de espera pero no por inspección del grafo sino a través del uso de un protocolo que hace que la formación de ciclos sea imposible. El protocolo trabaja de la siguiente manera:

- Cada transacción se marca con un tiempo de inicio (fecha ó # de transacción).
- Dos transacciones diferentes no tienen el mismo tiempo de inicio.
- Cuando una transacción A requiere un bloqueo sobre un registro que ya ha sido bloqueado anteriormente por otra transacción B entonces:

a) En "Espera-Muere" (Wait-Die), A espera si es más vieja que B, en otro caso A muere (se le hace rollback y se recomienza).

b) En "Hiere-Espera" (Wound-Wait), A espera si es más joven que B, en otro caso A "hiere" a B implicando esto que a B se le hace rollback y es recomenzada.

Esta técnica esta dentro de las técnicas de bloqueos aunque están envueltos tiempos de inicio. “Espera-Muere" (Wait-Die) garantiza que todas las esperas consisten de transacciones más viejas esperando por otras más jóvenes; "Hiere-Espera" (Wound-Wait) garantiza que todas las esperas consisten de transacciones más jóvenes esperando por unas más viejas. Así se asegura que no se puede encontrar una secuencia de espera donde T1 espere por T2, T2 espere por T3 y T3 espere por T1. Rosenkrantz probó que bajo este protocolo no ocurren esperas indefinidas y todas las transacciones a las que se les hace rollback se les hace un numero finito de veces.

1.2.1.8 Granularidad de bloqueo

Los bloqueos pueden aplicarse a diferentes unidades de data como por ejemplo bases de datos, archivos, páginas registros o campos. Estos son diferentes niveles (de mayor a menor) de granularidad a los que se les pueden aplicar bloqueos. Mientras más fina sea la granularidad mayor será el nivel de concurrencia pero se eleva el overhead existente al tener que otorgar una gran cantidad de bloqueos. Si la granularidad es más gruesa menor es el nivel de concurrencia existente y también el overhead ya que el número de bloqueos a ser otorgados es más bajo que en el caso anterior. Si una transacción X tiene bloqueados todo el conjunto de registros de un tipo, no necesita bloquear registros individuales lo cual reduce el número total de bloqueos; por otro lado ninguna transacción concurrente podrá acceder dicho conjunto de registros, Io cual hace que disminuya el nivel de concurrencia.

El concepto lógico de bloquear registros individuales es frecuentemente implementado bloqueando páginas que contienen varios registros. Esto ilustra un principio general (principio general de implementación n° 1) que dice: **El sistema puede bloquear en forma segura más del mínimo requerido**. La frase "en forma segura" significa que se garantice que no se compromete la integridad aunque con esto puede sufrir el nivel de concurrencia y se puede incrementar la posibilidad de abrazo mortal. Otro principio general de implementación (principio de implementación n° 2) dice que **el sistema puede retener en forma segura un bloqueo más de lo requerido**. Si no se hace esto un programa puede que espera un largo tiempo antes de readquirir el conjunto base de bloqueos si otra transacción ha tomado ese conjunto base en el tiempo transcurrido entre la liberación y la nueva adquisición de bloqueos por el mismo programa. Este es un buen argumento a favor del principio n° 2.

1.2.1.9 Intención de bloqueo

Supongamos que una transacción T quiere procesar algún conjunto base T y supongamos que requiere que B sea estable (no puede tolerar que ocurran cambios sobre B como resultado de actualizaciones de transacciones concurrentes). Esto se puede lograr mediante el protocolo de Intención de Bloqueo. Dicho protocolo requiere que las transacciones obtengan el nivel apropiado de acceso al conjunto base antes que le sea permitido operar sobre cualquier registro del conjunto base. El nivel de acceso para un conjunto base dado es expresado como la combinación de dos cosas: Nivel de procesamiento (PROCLEVEL) y nivel de compartición (SHARELEVEL).

El PROCLEVEL especifica tipo de procesamiento que la transacción intenta realizar sobre el conjunto en cuestión y puede tomar valores REFERENCIA si solo va a referenciarlo, o CAMBIO si va a hacer actualizaciones sobre él.

El SHARELEVEL especifica los requerimientos de estabilidad de la transacción con respecto al conjunto base, es decir, el procesamiento que la transacción puede tolerar sobre el conjunto base de parte de otras transacciones concurrentes. Los posibles valores son NINGUNO si no va a permitir ningún tipo de acceso sobre el conjunto base por parte de otras transacciones concurrentes, REFERENCIA si va a permitir que otras transacciones concurrentes hagan referencia al conjunto base o CAMBIO si va a permitir que otras transacciones concurrentes realicen modificaciones sobre d conjunto base.

Los diferentes tipos de bloqueos vistos anteriormente (Bloqueos S, X y U) pueden expresarse mediante intención de bloqueo. Por ejemplo, una solicitud de un bloqueo S por una transacción T sobre un conjunto base B puede expresarse como (PROCLEVEL = Referencia, SHARELEVEL = Referencia). Un bloqueo X seria (PROCLEVEL = Cualquiera, SHARELEVEL = Ninguno). Un bloqueo U no tiene sentido expresarlo así ya que ese tipo de bloqueo se define a nivel de registro y no de conjunto base.

La interacción entre las intenciones de bloqueo pueden expresarse mediante la siguiente matriz de compatibilidad

T / T'	NINGUNO	REFERENCIA REFERENCIA	REFERENCIA CAMBIO	CAMBIO REFERENCIA	CAMBIO CAMBIO	NO SOLICITADO
NINGUNO	N	N	N	N	N	S
REFERENCIA REFERENCIA	N	S	S	N	N	S
REFERENCIA CAMBIO	N	S	S	S	S	S
CAMBIO REFERENCIA	N	N	S	N	N	S
CAMBIO CAMBIO	N	N	S	N	S	S
NO SOLICITADO	S	S	S	S	S	S

Mediante la intención de bloqueo se pueden definir nuevos tipos de bloqueos adicionales a los nombrados anteriormente que aplican solo a nivel de conjunto base. En general la definición de los diferentes tipos de bloqueo en base a la intención de bloqueo son:

S: La transacción puede tolerar lectores concurrentes pero no actualizadores concurrentes sobre el conjunto base a bloquear y la misma transacción T no puede actualizar ningún registro del conjunto. En la matriz anterior este tipo de bloqueo es equivalente a la entrada Referencia Referencia.

X: La transacción T no puede tolerar ninguna transacción concurrente que accede al conjunto base como un todo; ella misma si puede actualizar registros individuales del conjunto base. En la matriz anterior este tipo de bloqueo es equivalente a la entrada Proclevel = Cualquier cosa, Sharelevel = Ninguno.

IS (Intención de bloqueo compartido): T intenta colocar bloqueos tipo S sobre registros individuales del conjunto base, así que puede permitir que otras transacciones hagan cambios sobre los registros a los que no quiere bloquear tipo S pero que pertenecen al conjunto base. En la matriz anterior este tipo de bloqueo es equivalente a la entrada Proclevel = Referencia, Sharelevel = Cambio

IX: similar al IS pero la transacción T puede querer actualizar registros individuales en el conjunto base. En la matriz anterior este tipo de bloqueo es equivalente a la entrada Proclevel = Cambio, Sharelevel = Cambio.

SIX: Combina los tipos de bloqueo S e IX, es decir, T puede tolerar lectores concurrentes, pero no permite actualizadores concurrentes. Sin embargo, la transacción T' puede actualizar registros individuales en el conjunto base y bloquearlos de manera exclusiva. En la matriz anterior este tipo de bloqueo es equivalente a la entrada Proclevel = Cambio, Sharelevel = Referencia.

Si un bloqueo S o X es aplicado a nivel de conjunto base, no es necesario ningún bloqueo a nivel de registro. Si alguna de las intenciones de bloqueo es aplicada a nivel de conjunto base, el sistema debe aplicar automáticamente bloqueos de registros como sigue:

- IS: Cada FIND debe colocar un bloqueo tipo S sobre el registro encontrado.

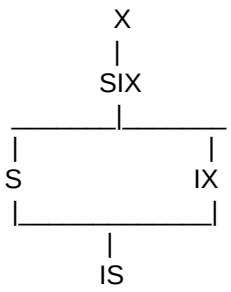
- IX: Cada FIND debe colocar un bloqueo sobre el registro encontrado. El bloqueo puede ser S, U o X. Si la transacción actualiza el registro, el bloqueo S o U debe ser promovido a un bloqueo X.

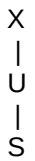
- SIX: El FIND no necesita ningún tipo de bloqueo pero los registros a actualizar deben ser bloqueados de manera exclusiva.

Una definición general del protocolo de intención de bloqueo seria:

Antes de que una transacción dada pueda adquirir un bloqueo de un tipo dado sobre un objeto dado debe adquirir primero una intención de bloqueo apropiada (IS, IX o SIX). Por ejemplo, para adquirir un bloqueo X sobre un registro la transacción debe adquirir al menos un bloqueo IX o SIX sobre el conjunto base que contiene el registro.

Los diferentes tipos de bloqueos existentes se pueden ordenar en un grafo de precedencia que muestra la fortaleza relativa de cada uno de ellos. Se tiene que un tipo de bloqueo B1 es más débil (implica que esta más abajo en el grafo) que otro bloqueo B2 sii siempre que haya una N en la columna de B1 en la matriz de compatibilidad en una fila dada, también hay una N en la columna B2 en la misma fila. Esta definición de fortaleza relativa implica que si un requerimiento de bloqueo falla para un tipo de bloqueo dado entonces fallará para un tipo de bloqueo mas fuerte: De acuerdo a esto; el grafo de precedencia de los bloqueos es el siguiente:





El concepto de precedencia de bloqueo origina el principio de implementación n° 3: **El sistema puede, de manera segura, tratar un requerimiento dado de bloqueo como un requerimiento más fuerte**.

Por ejemplo, un sistema que no soporta el bloqueo tipo S puede interpretar todos los requerimientos para tales bloqueos como requerimientos de tipo X.

1.2.2 Time-Stamping

El timestamping no es una técnica de bloqueo ya que mientras que los bloqueos sincronizan la ejecución intercalada de un conjunto de transacciones de manera tal que sea equivalente a alguna ejecución serial de esas transacciones, el timestamping sincroniza la ejecución intercalada de forma tal sea equivalente a una ejecución serial especifica (la ejecución definida por el orden cronológico de los timestamps de las transacciones completadas exitosamente). En timestamping los conflictos son resueltos asignándoles a las transacciones que recomienzan un nuevo tiempo de inicio.

Método de control de concurrencia TIME STAMPING

1. Al iniciarse una transacción se marca con un tiempo de inicio.
2. No hay dos transacciones con igual tiempo de inicio.
3. Se difieren las actualizaciones hasta el fin de la transacción.
4. Si una transacción entra en conflicto con la base de datos se le hace rollback y se reinicia con un nuevo tiempo de inicio.
5. Conflictos posibles:
   a) Una transacción t ve un dato que una transacción más joven modificó
   b) Una transacción t modifica un dato que una transacción más joven ya ha visto.

Para poder verificar la existencia de los conflictos anteriores, los datos están marcados con dos tiempos:

FMAX: Es el timestamp de la última transacción que ejecutó exitosamente una operación FIND sobre ese dato.

UMAX: Es el timestamp de la última transacción que ha ejecutado exitosamente una operación UPDATE sobre ese dato (la actualización no es considerada exitosa hasta el momento del commit).

6. Manejo de conflictos:.

a) Si una transacción T con timestamp t hace un FIND R, hacer:

```
SI t >= UMAX ENTONCES
   %No existe conflicto de tipo a)%
SINO
  %conflicto tipo a%
  restart T
FSI
```

b) Si una transacción T con timestamp t hace un UPDATE R, hacer:

```
SI (T >= FMAX) AND ( t >= UMAX) ENTONCES
  %No existe conflicto tipo b%
  UMAX: =t
SINO
  %conflicto tipo b%
  restart T
FSI
```

Por ejemplo, dadas las siguientes transacciones:

TRANSACCION A	TIEMPO	TRANSACCION B
FIND R	t1	
copy RF into ATEMP		
	t2	XFIND R

		copy R.F into BTEMP
UPD R: REPLACE R.F by ATEMP + 1	t3	
	t4	UPD R: replace RF by 2 * BTEMP

Si ts(A) > ts(B) entonces A es mas joven que B y según el protocolo la actualización de B falla por que existe un conflicto tipo b) y es recomenzada B en el tiempo t4.

Si ts(A) < ts(B) entonces A es más vieja que B y según el protocolo la actualización de A falla porque existe un conflicto tipo b) y A es recomenzada en el tiempo t3.

1.2 Casos de Estudio [12, 13]

1.2.1 Sybase

ASE (Adaptive Server Enterprise) tiene dos niveles de bloqueo:

- Bloqueo de tabla. ASE lo utiliza para proveer un bloqueo mas eficiente cuando una tabla completa o una gran cantidad de páginas o filas van a ser utilizadas por una operación.
- Bloqueo de página y fila: este tipo de bloqueo es menos restrictivo que el bloqueo de tabla; un bloqueo de página bloquea todas las filas en una página de datos o página de índices.

A su vez cada una describe unos tipos de bloqueos:

- Bloqueo de página y fila:
    - Bloqueo Compartido: ASE aplica este tipo de bloqueo para operaciones de lectura.
    - Bloqueo Exclusivo: ASE aplica este tipo de bloqueo para operaciones de modificación de datos.
    - Bloqueo de Actualización: ASE aplica este tipo de bloqueo durante la fase inicial de una operación de UPDATE, DELETE o FETCH, mientras la página o fila esta siendo leída. Este bloqueo permite bloqueos compartidos sobre la pagina o fila, pero no permite bloqueos de actualización o exclusivo.
- Bloqueo de tabla:
    - Intención de Bloqueo: indica que un bloqueo a nivel de paginas o a nivel de filas esta actualmente actuando sobre una tabla.
    - Bloqueo Compartido: similar al bloqueo compartido anterior solo que este afecta a toda la tabla.
    - Bloqueo Exclusivo: similar al bloqueo exclusivo anterior solo que este afecta a toda la tabla.
- Bloqueo por demanda: ASE lo utiliza para indicar que una transacción es la próxima en la cola de bloqueo para una tabla, pagina o fila.

1.2.2 Oracle

Se pueden definir dos tipos de contiendas de E/S: E/S concurrente y de interferencia. La contienda de E/S concurrente se produce cuando tienen lugar múltiples accesos en el mismo dispositivo simultáneamente. Éste es el tipo de contienda que se elimina aislando las tablas de los índices asociados a ellas. La contienda de interferencia se produce cuando las lecturas o escrituras de otros archivos del mismo disco interrumpen las escrituras secuenciales en un archivo, incluso si esas lecturas o escrituras se realizan en un momento que las lecturas y escrituras secuenciales. Existen tres procesos de segundo plano de la base de datos que acceden activamente a los archivos de la base de datos situados en el disco: Database Writer

(DBWR), Log Writer (LGWR) y Archiver (ARCH). El proceso DBWR lee y escribe archivos de una forma bastante aleatoria, LGWR los escribe secuencialmente y ARCH los lee y escribe también secuencialmente. Si se eliminan las posibilidades de contienda entre estos tres procesos de segundo plano, se elimina eficazmente todas las contiendas que se producen en el nivel de base. Por último, la división de una tabla o un índice en particiones puede mejorar el rendimiento de las operaciones realizadas con los datos, al eliminar las particiones no utilizadas del plan de ejecución de la operación.

La arquitectura de la opción de consulta en paralelo (Parallel Query Option PQO) de Oracle permite que prácticamente la totalidad de las operaciones se procesen en paralelo. El número máximo de procesos concurrentes de servidor para consultas en paralelo se fija mediante el parámetro PARALLEL_MAX_SERVERS en el archivo init.ora, mientras que el número mínimo se define mediante el parámetro PARALLEL_MN_SERVERS.

Para reducir al mínimo la cantidad de datos inactivos y en uso de los segmentos de anulación, hay que saber en que momento se están ejecutando consultas de larga duración. Si estas consultas se producen concurrentemente con múltiples transacciones, se aumentaría de manera constante elementos de segmentos de anulación inactivos y en uso. Con independencia de lo grandes que sean los segmentos de anulación, esta mala distribución de las transacciones provocará finalmente que las consultas fallen.

Este problema se soluciona, aislando todas las consultas de gran tamaño de forma que se ejecuten en aquellos instantes en los que se estén llevando a cabo menos transacciones. De esta manera, se minimiza la cantidad de datos inactivos y en uso de segmentos de anulación, mientras que, a la vez, se evitan potenciales contiendas debidas a operaciones de E/S concurrentes entres las consultas y las transacciones.

2 Integridad [11]

El término integridad es usado en el contexto de bd con el significado de exactitud, correctitud validez. El objeto del control de integridad es asegurar que la data en la bd sea correcta resguardando a la bd de actualizaciones inválidas. Otro término que es usado por integridad es consistencia pero es preferible hablar de consistencia cuando se requiere que dos o más valores en la bd tienen que ser consistentes uno con el otro en alguna forma.

2.1 Reglas de integridad

Para que el subsistema de integridad de un SMBD pueda:

- Monitorear las transacciones y detectar violaciones de integridad
- Si ocurre una violación tomar las acciones necesarias

dicho subsistema debe ser provisto con un conjunto de reglas de **integridad** constituidas por siguientes partes:

- Una **condición de activación**: dice cuando chequear la regla (al momento de inserción, actualización ó eliminación)
- El **predicado** a chequear: expresa la condición que deben cumplir los datos involucrados en la actualización
- **Acciones a tomar** en caso de que no se cumpla el predicado: indica que es lo que se va a hacer en caso de que no se cumpla el predicado exigido (rollback o colocar un valor por defecto, por ejemplo)

Por ejemplo, la restricción:

```
SIR1:   AFTER UPDATING S.STATUS:
                S. STATUS > 0
        ELSE
                DO;
                set return code to "regla SIR1 violada";
                REJECT;
                END;
```

Posee una condición de activación "AFTER UPDATING S.STATUS" que indica cuando se va verificar la reglas de integridad, un predicado "S.STATUS > 0" que indica la naturaleza del chequeo y la cláusula "ELSE" que indica que hacer cuando el predicado no se cumpla.

Las reglas de integridad son compiladas y almacenadas en el sistema diccionario por el compilador de reglas (componente del subsistema de integridad).

Las reglas de integridad pueden ser divididas en dos grandes categorias:

- **Reglas de integridad de dominios**: tienen que ver con la admisión de un valor dado como valor candidato para un atributo dado, independientemente de su relación con otros valores en la bd.
- **Reg!as de integridad de relaciones**: tienen que ver con la admisión de una tupla dada como candidata para la insersión en una relación dada o con la relación entre tuplas de una relación o de varias relaciones.

2.1.1 Reglas de integridad de dominio

Cada atributo de cua!quier relación tiene un dominio subyacente, identificando en la definición de relación. Sea el atributo A de la relación R definido sobre el domimo D. Cualquier valor v candidato para ser un valor de R.A debe pertenecer a D. Entonces, la definición y verificación del dominio D constituye una regla de integridad que debe ser chequeada en todas las inserciones y actualizaciones que envuelvan cualquier atributo definido sobre D.

Una sintaxis BNF para reglas de dominios seria:

```
<definición-de-dominio>::= DCL<nb-de-dominio> [PRIMARY] DOMAIN
                                <restricción> <terminador>
<restricción>::= <tipo-de-dato> [predicado]
<terminador>::= ; I ELSE <respuesta-a-la-violación>
```

<respuesta-a-la-violación>::=<unidad-de-código-ejecutable>

Algunos ejemplos de este tipo de regla serian:

```
DCL P#          PRIMARY DOMAIN CHARACTER(6)
                SUBSTR (P#, 1, 1) ='P'
                AND IS NUMERIC(SUBSTR(P#, 2,5) )
                ELSE
                DO;
                set return code to "Regla de dominio de P# violada";
                REJECT;
                END;

DCL PNAME       DOMAIN CHARACTER (20) VARYING;

DCL COLOR       DOMAIN CHARACTER (6) COLOR IN ('RED', 'GREEN',
'BLUE', `YELLOW', BROWN', PURPLE');

DCL WEIGHT      DOMAIN FIXED (5,1) WEIGHT > 0 AND WEIGHT < 2000;

DCL QTY         DOMAIN FIXED (5) QTY > = 0;

DCL LOCATION DOMAIN CHARACTER (20) VARYING LOCATION IN
('London', 'Paris', Rome', 'Atenas');
```

La razón principal por la cual en este tipo de restricciones no se definen condiciones de activación exp!icitamente es que son siempre validadas al momento de hacer UPDATE del campo o al momento de hacer INSERT de una nueva tupla Este tipo de reglas tampoco necesitan un nombre diferente del nombre del dominio que definen, ya que se identifican unicamente por ese nombre del dominio.

El hecho de colocar la clausula PRIMARY a un dominio D implica que:

- Debe existir al menos una relación *R* en la bd que contenga su clave primaria definida sobre D.
- Sea una relación *R1* que tiene un atributo A definido dobre D y que no es; necesarimente la clave primaria de *R1.* En cualquier momento, cada valor de R1.A o debe ser nulo o igual a k, donde k es el valor de un atributo que es clave primaria de alguna tupla en una relación R2 con clave primaria definida sobre D, R1 y R2 no necesarimente son diferentes.

Las posibles repuestas a las violaciones de las reglas pueder incluir:

- Rechazo simple de la operación
- Corrección del valor inválido colocando un valor por defecto
- Forzar un rollback de la transacción que esta haciendo esa operación.

Para ver !a utilización de los dominios definidos explicitamente a través de las reglas de definición de dominios, la definición de la tabla P (partes) sería:

```
DCL P  RELATION ATRIBUTES (
P#              DOMAIN (P#),
PNAME           DOMAIN (PNAME),
COLOR           DOMAIN (COLOR),
WEIGHT          DOMAIN (WEIGHT),
CITY            DOMAIN (LOC ATION) )
PRIMARY KEY (P#);
```

2.1.2 Reglas de integridad sobre relaciones

Una sintaxis BNF para definir reglas de integridad sobre relaciones es:

regla-de-integridad-sobre-relaciones::=
<etiqueta> : [{<condición-de-activación>}:]
<restricción>
[; I ELSE <respuesta-a-la-violación>]

<condición-de-activación>::= WHEN COMMITTING I
BEFORE <cambio-before> I
AFTER <cambio-after>

<cambio-before>::= INSERTING <parámetro-de-registro>
[FROM <nombre-de-estructura>]
I UPDATING <parámetro-de-registro>
[FROM <nombre-de-estructura>]
I UPDATING <parámetro-de-campo>
[FROM [<nombre-de-estructura>.]<nombre-de-campo>]
| DELETING <parámetro-de-registro>

<cambio-after>::= INSERTING <parámetro-de-registro>
| UPDATING <parámetro-de-registro>
| UPDATING <parámetro-de-campo>
| DELETING <parámetro-de-registro>

<parámetro-de-registro>::= [<nombre-de-cursor>→]<nombre-de-registro>

<parámetro-de-campo>::=[<nombre-de-cursor>→]<nombre-de-registro>.
<nombre-de-campo>

<restricción>::= <predicado>

<respuesta-a-la-violación> ::= <unidad-ejecutable>

Algunos ejemplos de este tipo de restricciones son:

SIR5 : BEFORE UPDATING S.STATUS FROM NEW STATUS:
NEW_STATUS > S.STATUS

EIRX7:AFTER INSERTING E1 → EMFLLADO
AFTER UPDATING E1 → EMPLEADO.MGR# :
EXISTS (E2 → EMPLEADO WHERE
E2 → EMPLEADO.EMP# = E1 → EMPLEADO.MGR#) ;

IR12: WHEN COMMITTING:
SETSUM (CUENTA.BALANCE) = UNIQUE (SUMMARY.TOTAL) ;

La regla de integridad:

L : BEFORE cambio : predicado ELSE respuesta

Es logicamente equivalente a la secuencia de instrucciones

```
L : PROC:
      IF se intenta,ejecutar cambio THEN
            IF ¬ (predicado) THEN
                  ejecutar respuesta;
      END;
```

Si la respuesta es ejecutada e incluye la operación REJECT, entonces el cambio no es ejecutado; en otro caso si se ejecuta.

La regla de integridad:

L : AFTER cambio : predicado ELSE respuesta

Es lógicamente equivalente a la secuencia de instrucciones:

```
L : PROC;
            ejecutar cambio; .
            IF ¬ (predicado) THEN
                  ejecutar respuesta;
            END;
```

Si la respuesta es ejecutada e incluye la operación REJECT, el cambio es deshecho en ese punto.

Las reglas de integridad sobre relaciones pueden clasificarse en dos grandes grupos, a saber:

- **Restricciones de transición (dinámicas):** este tipo de restricciones involucran una comparación los estados antes y después de la bd. Un ejemplo de este tipo de restricción es la regla SIR5.
- **Restricciones de estado (estáticas)**: este tipo de restricciones involucran un sólo estado de la base de datos para verificar su validez independientemente de los otros estados.

Otro tipo de clasificación de las reglas de integridad es según donde se apliquen, a saber:

- **Restricciones sobre tuplas**: son restricciones que pueden ser evaluadas examinando una sola tupla de la relación. Por ejemplo, la restricción SIR5.
- **Restricciones sobre conjuntos**: son restricciones que involucran un conjunto completo de registros en vez de una única tupla. Por ejemplo, la restricción EIRX7.

Según el momento en que se aplican, las reglas de integridad se pueden clasificar en:

- **Restricciones inmediatas**: estas son aplicadas inmediatamente cuando se ejecuta las instrucciones INSERT, UPDATE ó DELETE, según sea el caso. Por ejemplo las reglas SIR5 y EIRX7.
- **Restricciones diferidas (de consistencia)**: son restricciones que no pueden ser satisfechas en todo momento y se evaluan al finalizar una transacción especifica que debe hacer que la bd cumpla con la restricción (la condición de activación de ellas es WHEN COMMITTING). Un ejemplo de esto es la regla IR12.

Las restricciones inmediatas, de estado, aplicadas sobre conjuntos incluyen reglas para asegurar unicidad de la clave, reglas para asegurar la integridad referencial, reglas para asegurar dependencia funcionales adicionales. La unicidad de la clave es esencial y puede ser soportada con un índice o un algoritmo de hashing. La integridad referencial también es extremadamente importante y puede ser soportada a través de un índice doble o una estructura de sets (apuntadores).

2.2 Reglas de integridad y algunos sistemas existentes [12]

Actualmente los SMBD proveen el uso de triggers que permiten representar las reglas de integridad de relaciones.

En cuanto a las reglas de integridad de dominio:

- Oracle provee la claúsula CHECK como palabra reservada dentro de la instrucción de la creación de la tabla para especificar restricciones de integridad de dominio.
- Sybase permite crear reglas que luego son enlazadas a las tablas. Dentro de estas reglas podemos especificar restricciones en cuanto a los posibles valores que puede tomar un valor dado.

3 REFERENCIAS BIBLIOGRAFICAS

[1] Date C.J. "An Introduction to Database Systems". 7th edition, Addison-Wesley, 2000.
[2] Gio Wiederhold. "Database Design". 2a edición. Nueva York, N.Y.; McGraw-Hill (1983).
[3] T.H. Merret. "Relational Information Systems". Reston, Va: Reston Publishing Company Inc. (1984).
[4] Michael Stonebraker. "Operating System Support for Database Management". CACM 24, núm 7. Julio 1981.
[5] Pratt P. and Adamski J. "Database Systems Management and Design". Third Edition. Boyd & Fraser publishing company. 1994.
[6] R. Bayer y C. McCreight. "Organization and maintenance of Large Ordered Indexes". Acta Informática 1, núm 3 (1972).
[7] Elmasri / Navathe. "Sistemas de Bases de Datos. Conceptos fundamentales". Addison Wesley. Segunda Edición 1997.
[8] Öszu, Tamar and Valduriez, P. "Principles of Distributed Database Systems". 2nd Ed. Prentice Hall, 1998.
[9] Korth H., Silberschatz A, Sudarshan, S. "Fundamentos de bases de datos". Tercera edición. McGraw-Hill. 1998. ISBN 84-481-2021-3
[10] Leon-Hong B., Plagman B., "Data Dictionary Directory Systems". J. Wiley, 1982.
[11] Fernandez E., Summers R., Wood C. "Database Security and Integrity". Addison Wesley, 1981.
[12] Loney, Kevin. "ORACLE 8. Manual del administrador". McGrawHill. Primera Edición 2000.

FUENTES ELECTRÓNICAS

[13] Sybase. "Fast Track to Adaptive Server Enterprise 11.9.2". 2001.
URL: www.sybase.com.