Objetos compartidos

Una de las características fundamentales de cualquier base de datos es el soporte que proporciona a la compartición de objetos.

En todas las situaciones en donde muchos usuarios necesitan acceder a los mismos datos se requieren controles para evitar la interferencia.

- El principio fundamental es que cualquier usuario debe ser capaz de completar una unidad lógica de trabajo, conocida en la terminología de base de datos como "transacción", sin que ningún otro usuario pueda alterar sus acciones.
- Las herramientas que un ODBMS proporciona para evitar la interferencia entre usuarios se conocen como "control de concurrencia" y "gestión de transacciones". Estas técnicas son también útiles en la implementación de la capacidad de recuperación de la base de datos orientada a objetos.

Muchas de las técnicas empleadas por los ODBMSs son las mismas que las utilizadas en los productos relacionales.

- Las técnicas tradicionales han sido perfeccionadas a lo largo de los años para dar soporte al proceso de transacciones *on-line* (OLTP, *On-Line Transaction Processing*), que comprenden actividades muy dirigidas y de corta duración (operaciones de crédito, débito, cuentas bancarias individuales, etc...).
- Los ODBMSs requieren actividades de mayor duración, tales como modificar una parte de un diseño completo o analizar el rendimiento de una cartera financiera bajo determinadas condiciones económicas. Las aplicaciones distribuidas y cliente-servidor requieren modificaciones adicionales al mecanismo de control de acceso tradicional.

Propiedades de una transacción

Una transacción de base de datos tiene tres propiedades fundamentales:

1. Está definida por la aplicación. Esto significa que responde a las reglas de consistencia establecidas por la aplicación. El DBMS no puede determinar qué es y qué no es una transacción lógica, esto es responsabilidad de la aplicación.
2. Responde al principio de "todo o nada". La transacción tiene lugar en su totalidad o no tiene lugar en absoluto. Si todas las actualizaciones llevadas a cabo por la transacción tienen lugar, se dice que la transacción ha sido validada (*commit*). Si alguna de las operaciones no se puede llevar a cabo, ninguna de las actualizaciones sobrevive y la transacción se dice que ha sido abortada (*abort*).
3. Es irreversible. Una vez que la transacción ha sido validada, no es posible deshacer los cambios realizados, de modo que su resultado sólo puede ser modificado mediante transacciones subsecuentes.

Una transacción sólo tiene, por tanto, dos posibles resultados: *commit* o *abort*.

Propiedades de una transacción ...

- Una transacción validada ha llegado a fin con éxito.
- Una transacción abortada ha encontrado alguna dificultad que impide su conclusión satisfactoria.
- No es permisible que una transacción que ha sido abortada deje en la base de datos alguna de las modificaciones que ha realizado.
- Si la transacción es abortada, el ODBMS dede tomar las acciones pertinentes para asegurar que ninguna de las operaciones realizadas es persistente.

Es responsabilidad de la aplicación especificar los límites de una transacción. Una transacción es una unidad lógica de trabajo y sólo la lógica de la aplicación puede determinar las condiciones en que ésta se realiza.

- En algunos casos una transacción puede ser abortada por el "sistema". Por ejemplo, puede haber fallos del *hardware* de almacenamiento, cortes de suministro eléctrico o simplemente conflictos entre acciones de diferentes usuarios.
- En otros casos es decisión del usuario abortar la transacción. Por ejemplo, un ingeniero puede decidir no actualizar unos planos después de haber realizado algunas modificaciones cuyos resultados no son satisfactorios (el diseño no está funcionando como esperaba).

Ejecución serializable

- En un DBMS las transacciones pueden ejecutarse en serie, de modo que es necesario ejecutar todas las acciones asociadas a una transacción antes de comenzar a procesar cualquier otra transacción. Cuando se ejecutan en serie, las transacciones no pueden interferir entre sí, ya que sólo una está activa en un momento dado.

  Esta es una de las soluciones adoptadas para garantizar la consistencia, sin embargo, generalmente no es una aproximación válida en sistemas multiusuario

- En lugar de esto, las acciones de múltiples usuarios se intercalan bajo el control del DBMS y con el objetivo de aumentar el índice de concurrencia en el sistema atendiendo a más usuarios en un periodo de tiempo dado mientras se mantiene la consistencia.

  La ejecución concurrente de transacciones no siempre tiene como resultado la aparición de inconsistencias. La ejecución se dice que es "serializable" cuando produce el mismo resultados que si se llevaran a cabo en serie (es decir, no concurrentemente).

  Un DBMS que ejecuta sólo transacciones serializables garantiza la consistencia de la base de datos.

Tipos de bloqueo (*Locking*)

Tanto los DBMSs relacionales como orientados a objetos proporcionan varios tipos de bloqueo. Las dos formas de bloqueo más importantes son:

- "Bloqueo exclusivo" (*exclusive lock*). En un momento dado, sólo una transacción puede disponer de acceso exclusivo sobre un objeto concreto. Además, esto impide que cualquier otra transacción pueda acceder a ese objeto.
- "Bloqueo compartido" (*shared lock*). Múltiples transacciones pueden disponer concurrentemente de un acceso compartido sobre un objeto particular. El bloqueo compartido permite a estas transacciones acceder concurrentemente al objeto. Sin embargo, disponer de este acceso no es suficiente para permitir a la aplicación realizar algún tipo de actualización. Una transacción debe disponer de bloqueo exclusivo sobre el objeto para poder actualizarlo.

El bloqueo exclusivo se denomina en ocasiones "bloqueo de escritura" mientras que el bloqueo compartido se denomina "bloqueo de sólo lectura".

- Es posible procesar múltiples demandas de acceso en sólo lectura sobre un objeto ya que no hay conflictos y el DBMS puede garantizar el acceso concurrente.
- Por otra parte, múltiples solicitudes de bloqueo de escritura sobre un objeto dan lugar a la aparición de conflictos y el DBMS sólo puede conceder una en un momento dado.

Tipos de bloqueo ...

Algunas aplicaciones no necesitan asegurar la consistencia de los objetos que están procesando.

- En estos casos y con objeto de aumentar la concurrencia, algunos DBMSs permiten un modo de acceso denominado *dirty read* que consiste en evitar cualquier mecanismo de bloqueo.
- La lectura en modo *dirty read* ignora el bloqueo que cualquier otra transacción pueda estar ejerciendo sobre el objeto y por tanto ignorando cualquier modificación que ésta pueda realizar.
- Por ejemplo, una aplicación puede estar accediendo a una lista de empleados para calcular el salario medio e ignorar cualquier modificación del salario concreto de algunos empleados durante el barrido (esto no afecta al resultado final).
- El empleo de *dirty reads* es una forma importante de incrementar la concurrencia. Depende de las características de la aplicación que se pueda o no utilizar este modo de acceso ya que el DBMS no puede tomar esta decisión.

Los DBMSs disponen de varios mecanismos para bloquear físicamente un objeto:

- Algunos DBMSs mantienen tablas con identificadores de los objetos bloqueados.
- Otros DBMSs modifican ciertos *bits* del objeto que actúan como señales de control.
- Por último, algunos utilizan una técnica que consiste en mover todos los objetos bloqueados a un área específica de memoria.

El mecanismo de bloqueo empleado puede tener una influencia considerable en las necesidades de cómputo y en el rendimiento de la base de datos.

Técnicas de auditoria (*logging*)

Los DBMSs relacionales y orientados a objetos normalmente emplean técnicas de auditoría (*logging*) tanto para abortar como para validar las transacciones.

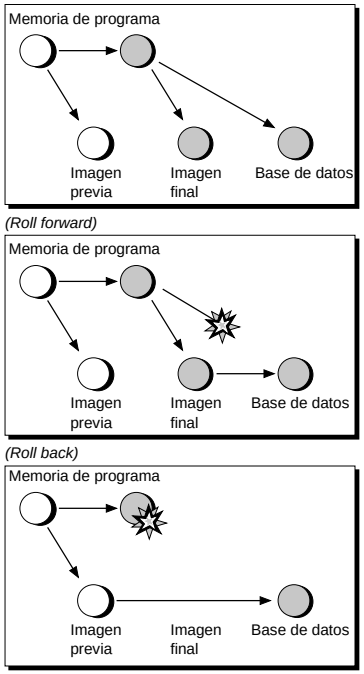
- La auditoría consiste simplemente en un registro adicional utilizado sólo por el DBMS para determinar el estado del objeto y que es invisible a las aplicaciones.
- Esta técnica, consistente en mantener un registro de incidencias, es esencial para la gestión de transacciones y la capacidad de la base de datos de recuperación frente a fallos.
- En realidad, se trata de una base de datos auxiliar que es empleada por el DBMS para asegurar la recuperabilidad de las acciones de actualización.

Los registros que el DBMS escribe en el fichero de incidencias o auditoría son fundamentalmente de dos tipos:

- Una imagen previa (*before-image*) del objeto. Es una copia del estado del objeto antes de aplicar las actualizaciones realizadas por la transacción.
- Una imagen final (*after-image*) del objeto. Es la copia del estado del objeto tras las operaciones de modificación realizadas por la transacción.

Técnicas de auditoria ...

- La imagen previa se utiliza para deshacer cualquier actualización que pudiera haber escrito en almacenamiento una transacción abortada. El DBMS sólo reemplaza el objeto actualizado por el copiado en la imagen previa.
- Las imágenes finales se emplean para validar aquellas transacciones que han sido interrumpidas durante el proceso de validación.
- Los registros de incidencias incluyen normalmente un identificador del programa o usuario asociado, un registro de la fecha y hora de la operación y otra información pertinente.



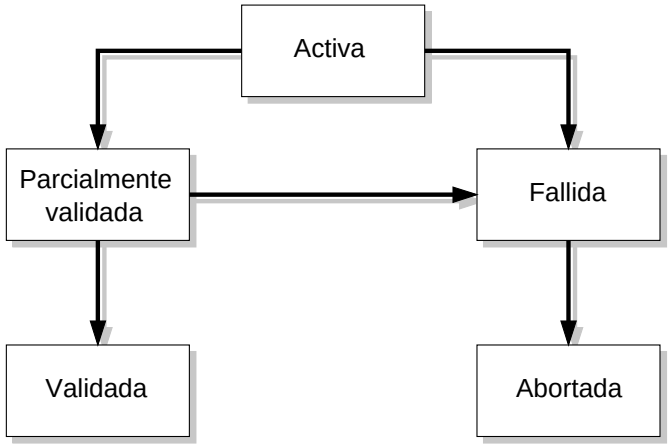
Técnicas de auditoria ...

Algunos DBMSs escriben los registros de incidencias en almacenamiento persistente lo antes posible (esta técnica es conocida como *write-ahead logging*), con el objeto de asegurar que están disponibles para recuperar la transacción.

- Esta técnica consiste simplemente en un protocolo que requiere que el DBMS escriba siempre los registros imagen en el fichero de incidencias antes que los resultados de la actualización del objeto.
- La forma más simple de entender cual es la necesidad del *write-ahead logging* es considerar qué sucedería si esta técnica no se utilizara.
- Supongamos que primero se actualiza la base de datos y después se escribe el registro de auditoría. Si tuviera lugar un fallo, por ejemplo la red deja de funcionar en un entorno cliente-servidor, justo después de actualizar la base de datos pero antes de que se escriba el registro de incidencias no dispondríamos del registro que nos permitiría recuperar la transacción si fuera necesario. En estas condiciones, el DBMS perdería la capacidad de recuperar las modificaciones de la base de datos frente a una transacción cancelada.
- Empleando la técnica *write-ahead logging* el DBMS siempre cuenta con el registro de incidencias necesario para recuperar una transacción antes de que las actualizaciones de ésta se reflejen en la base de datos.

Recuperación frente a fallos

Una transacción se ejecuta de acuerdo con el siguiente diagrama de estados:



- Cuando una transacción comienza su estado cambia a "Activa".
- Una vez que la transacción a ejecutado todas las instrucciones, incluyendo cualquier comprobación de integridad pertinente, pasa al estado "Parcialmente validada".
- Mientras la transacción se encuentra en el estado "Activa" puede fallar por un gran número de motivos, entrando por tanto en el estado "Fallida". Entre esos fallos se encuentran, por ejemplo, las caidas del sistema, la imposibilidad de satisfacer todas las restricciones de integridad o la necesidad del DBMS de abortar la transacción debido a que ha detectado un *deadlock*.
- Una transacción puede fallar aún en el estado "Parcialmente validada" ya que sus resultados no se han escrito de forma segura en disco...

El protocolo de validación en dos fases

El protocolo de validación en dos fases dicta la secuencia de pasos que permiten al DBMS determinar si las acciones de la transacción deben de validarse en todos los sitios que participan en ella o bien si ninguna de estas acciones debe de almacenarse permanentemente. En la discusión que sigue emplearemos esta terminología:

- La transacción como un todo se denomina "transacción distribuida" (*distributed transaction*).
- Aquella parte de una transacción distribuida que se ejecuta en un determinado nodo (*site*) se denomina "subtransacción local" (*local subtransaction*).
- La parte del DBMS que coordina la transacción distribuida se denomina "coordinador" (*transaction coordinator*).
- La parte del DBMS que reside en un nodo particular y gestiona la subtransacción local se denomina "gestor local" (*local transaction manager*).

El protocolo de validación en dos fases ...

Cuando un aplicación solicita la validación de una transacción distribuida, el coordinador de la transacción realiza las siguientes tareas:

Fase 1: Comprobación de la disponibilidad para validar.

- El coordinador envía un mensaje a cada gestor local que participa en la transacción preguntando si está preparado para validar su subtransacción local.
- Cada gestor local comprueba si está preparado para validar su subtransacción y responde al coordinador. Estar preparado significa que las actualizaciones de la subtransacción local han sido correctamente escritas en el registro de incidencias. El gestor local debe asegurarse de que, pase lo que pase, hay suficiente información como para permitir la validación de la parte local de la transacción. A partir de este momento, el gestor local se mantiene a la espera de nuevas instrucciones.

Fase 2: Validación o cancelación de la transacción.

- El coordinador recoge todas las respuestas de los gestores locales.
- Si todos los gestores locales responden afirmativamente, el gestor envía un mensaje indicando que el proceso de validación puede tener lugar, escribiendo las modificaciones pertinentes en almacenamiento. Si cualquiera de los gestores locales no responde o indica que no puede concluir su subtransacción local, envía un mensaje a todos y cada uno de los gestores locales solicitando que se aborte la transacción.
- Los gestores locales validan o abortan la transacción de acuerdo con las instrucciones del coordinador.

El protocolo de validación en dos fases ...

Son muchos los problemas que surgen al diseñar e implementar eficientemente este protocolo de validación en dos fases.

- El diseño debe tener en cuenta la posibilidad de un fallo en cualquier punto del proceso y por tanto, el rendimiento puede resultar seriamente afectado cuando se dan estas circunstancias de fallo.
- Por ejemplo, si el coordinador falla mientras los gestores locales están esperando instrucciones para proceder a la fase 2, deben mantenerse a la espera hasta que el coordinador vuelva a estar disponible.
- Si la conexión con un gestor local falla durante la fase 2 antes de que reciba las instrucciones del coordinador, éste deberá esperar hasta que la conexión vuelva a estar disponible.

Control de concurrencia

- Cuando dos o más usuarios o programas intentan compartir el acceso a un objeto aparece la posibilidad de que sus acciones interfieran.
- El objetivo del control de concurrencia es mantener un nivel adecuado de consistencia en la base de datos idependientemente del número de usuarios o programas que están accediendo al mismo objeto.
- Esto debe ser cierto incluso frente a caidas del sistema.

Determinar si una colección de objetos es o no consistente significa determinar si esta colección cumple una serie de reglas definidas por la aplicación.

Mantener en todo momento una colección de objetos consistente es imposible si hay actualizaciones y cambios en los objetos:

- Una base de datos no siempre puede cumplir las reglas de consistencia tras acciones individuales del usuario o la base de datos.
- En lugar de esto, una aplicación normalmente espera que sea un conjunto de acciones las que mantengan la consistencia.
- Cada conjunto de acciones lleva los objetos desde un estado consistente a un estado temporal probablemente inconsistente y de nuevo a un estado consistente.
- Un conjunto de acciones de este tipo se considera que es una unidad lógica de trabajo atómica que se denomina transacción.
- La gestión de transacciones es la tarea de asegurar que las transacciones preservan la consistencia.

Control de concurrencia ...

Los ODBMSs generalmente requieren que las aplicaciones hagan todos los accesos, creaciones, modificaciones y borrados de los objetos persistentes dentro de una transacción.

- Las operaciones de base de datos deben ejecutarse bajo el control de los límites de la transacción.
- Los ODBMS proporcionan al programador funciones para especificar los puntos de inicio y fin de una transacción. Un final de transacción (*commit* o *abort*) es un punto de sincronización.
- El ODBMS grarantiza en dicho punto de sincronización que los objetos se encuentran en un estado consistente, bien garantizando que todas las modificiaciones realizadas por la aplicación se han realizado con éxito, bien asegurando que ninguna de ellas se ha llevado a cabo.

El problema del acceso concurrente es relativamente fácil de gestionar para una base de datos orientada a objetos siempre que los usuarios no cambien el estado de los objetos.

Sin embargo, cuando múltiples programas o usuarios acceden concurrentemente a una colección de objetos y al menos uno de ellos está modificando el estado de uno o más objetos, pueden aparecer interferencias que den lugar a inconsistencias.

Control de concurrencia ...

Tres son los tipos de inconsistencias que pueden evitarse mediante técnicas de control de concurrencia apropiadas:

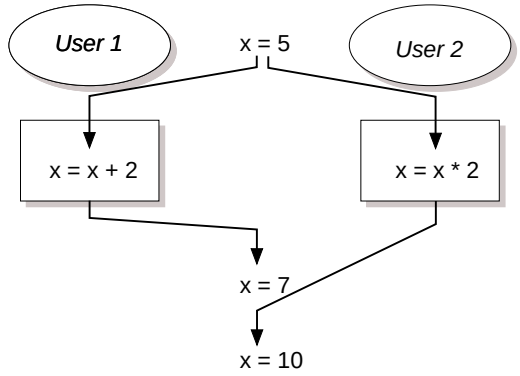
- Pérdida de modificaciones.
- Actualizaciones supuestas.
- Lecturas inconsistentes.

Todas estas son situaciones en las que la transacción es correcta cuando se ejecuta aisladamente pero que producen resultados incorrectos cuando se solapan con otras transacciones.

- La interferencia tiene su origen en el solapamiento de las acciones de las dos transacciones.
- El control de concurrencia está fundamentalmente orientado a evitar las situaciones de solapamiento que pueden dar lugar a problemas de consistencias.
- Estos problemas de consistencia son bien conocidos por los sistemas de gestión de bases de datos relacionales.

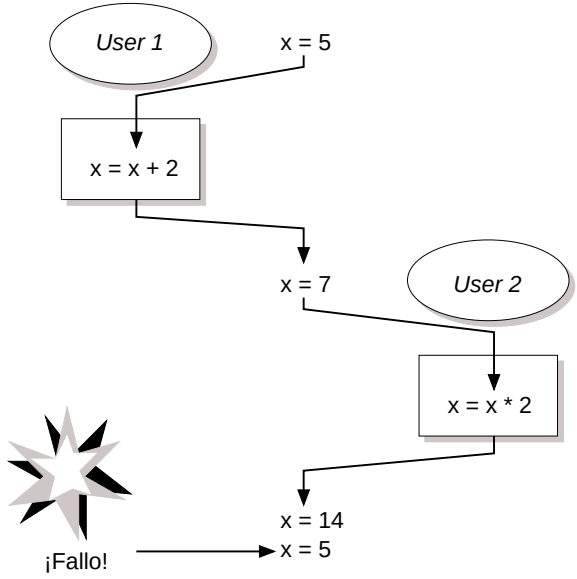
Control de concurrencia ...

- **Pérdida de la modificación**. Consideremos la situación ilustrada en la figura.
  - El usuario 1 pretende sumar 2 a la variable x, que es una variable de estado de cierto objeto.
  - El usuario 2 intenta multiplicar el valor de x para ese mismo objeto por 2.
  - Ambos leen el valor de x (5) y actualizan el valor de la variable (el usuario 1 para obtener $x = 7$ y el usuario 2 para obtener $x = 10$).
  - El valor final de x depende exclusivamente de cual de ambos procesos finalice el último, de modo que la modificación realizada por uno de los dos procesos se pierde.
  - Más aún, ninguno de los dos usuarios tiene en cuenta el resultado de las acciones del otro.
  - La ejecución secuencial de las modificaciones evita la interferencia y la pérdida de las actualizaciones de los usuarios 1 y 2.



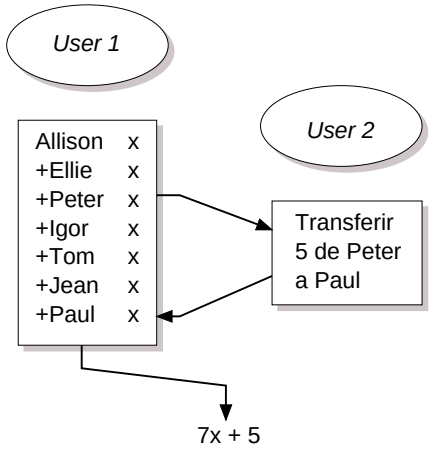
Control de concurrencia ...

- **Actualización supuesta**. Consideremos otro posible caso de interferencia ilustrado en la siguiente figura.
  - Supongamos que los usuarios 1 y 2 pretenden realizar las mismas acciones que en el caso anterior.
  - Las acciones se ejecutan en serie (y no de forma interlazada), pero las acciones del usuario 1 se abortan debido a un fallo no determinado.
  - Antes de que se produzca el fallo, el usuario 2 accede al dato y utiliza el valor modificado por 1.
  - El efecto es que el usuario 2 supone que las acciones llevadas a cabo por 1 se han completado satisfactoriamente, incluso cuando la actualización es descartada posteriormente.



Control de concurrencia ...

- **Lectura inconsistente**. Consideremos la situación ilustrada en la figura.
  - En este caso, el usuario 1 está recorriendo una colección ordenada de objetos y sumando el valor de una cierta variable de estado x.
  - Simultáneamente, el usuario 2 actualiza un par de objetos, sustrayendo 5 de `Peter` y transfiriendo esa misma cantidad al objeto `Paul`.
  - El usuario 1 suma la contribución de `Peter` antes de que se produzca la transferencia y la contribución de `Paul` después de la transferencia, el resultado de la suma es 5 unidades mayor que si el proceso no hubiese tenido lugar concurrentemente con la actualización llevada a cabo por el usuario 2.



Control de concurrencia ...

- En todos estos casos, la actualización de un usuario interfiere con la lectura o modificación realizada por otro.
- Es responsabilidad del ODBMS asegurar que no tengan lugar este tipo de interferencias.

Un criterio formal, desarrollado en el campo de las bases de datos relacionales, nos permite determinar cuando un conjunto de operaciones se han ejecutado de forma consistente:

> *Se dice que una base de datos es consistente si su estado después de la ejecución de un conjunto de transacciones es exactamente el mismo que tendría si esas transacciones se hubieran realizado secuencialmente.*

- Esta definición surge directamente de la tecnología de base de datos convencional y se puede emplear tanto en los DBMS relacionales como orientados a objetos.
- La definición no exige que la ejecución tenga lugar necesariamente de forma secuencial, únicamente requiere que el resultado final sea el mismo.
- La consistencia, definida de esta forma, resulta un tanto ambigua, ya que permite que la ejecución de un conjunto de transacciones proporciones un resultado final diferente dependiendo del orden.
- Sin embargo, el ODBMS no puede determinar cual es la secuencia correcta de operaciones, esto es parte de la lógica de la aplicación y no del ODBMS.

Empleo de bloqueos (*locks*)

La aproximación más común utilizada por los ODBMSs para garantizar la ejecución secuencial es la técnica conocida como bloqueo (*locking*):

- Está directamente inspirada en la técnica empleada en los DBMSs relacionales.
- En su forma más simple, bloquear un objeto evita que otras transacciones utilicen dicho objeto hasta que se retira el bloqueo.

Es responsabilidad del ODBMS gestionar los bloqueos:

- Respondiendo a las solicitudes de bloqueo de las transacciones a medida que estas los solicitan.
- Siguiendo la pista de qué transacciones bloquean a qué objetos en un momento dado.
- Detectando cuando las solicitudes de bloqueo pueden interferir entre sí.
- Levantando el bloqueo cuando la transacción así lo requiere.

Sin embargo, es responsabilidad de la aplicación de base de datos especificar qué objetos deben ser bloqueados.

Empleo de bloqueos ...

Es normal que una aplicación mantenga un número alto de bloqueos, tantos como objetos son accedidos por la aplicación.

- Algunos de estos bloqueos pueden ser exclusivos y otros compartidos.
- Para simplificar el trabajo del programador, algunos ODBMS solicitan el bloqueo automáticamente cuando un objeto es accedido, haciendo que el mecanismo de bloqueo sea invisible para el programador.
- En otras situaciones es más apropiado que sea la propia aplicación la que solicite explícitamente los bloqueos del tipo adecuado.
- En general, los ODBMSs suelen conceder implicitamente un bloqueo compartido a todos los objetos referenciados excepto en el caso que la aplicación solicite explícitamente bien un bloqueo exclusivo, bien ningún tipo de bloqueo.

Cuando una aplicación necesita solicitar explícitamente un bloqueo es conveniente para el programador disponer de la posibilidad de solicitar el bloqueo de una colección completa de objetos (en lugar de tener que solicitar el bloqueo de cada uno de los elementos individualmente).

Sin embargo, siempre es necesario establecer un compromiso entre la conveniencia de bloquear un gran conjunto de objetos (incluso toda una base de datos) y el hecho de que un número grande de bloqueos limita el acceso del resto de otras transacciones.

Utilización de las técnicas de bloqueo para aumentar la concurrencia

Cuanto mayor es el número de objetos que bloquea una transacción particular, menor es el nivel de concurrencia que la base de datos puede mantener durante la transacción.

- Cualquier objeto bloqueado en exclusiva no está disponible para el resto de transacciones, ni en lectura ni en escritura.
- Los objetos bloqueado en modo compartido no están accesible en escritura para el resto de las transacciones (pero si en lectura).

El bloqueo sólo de los objetos requeridos en una determinada transacción aumenta el grado de concurrencia de la base de datos.

Otra forma de aumentar la concurrencia consiste en solicitar el bloqueo menos restrictivo que cumpla con los requisitos de la transacción.

- Si la transacción va a actualizar un objeto, necesita acceder a éste mediante un bloqueo exclusivo.
- Si la transacción no va a modificar el objeto, no necesita solicitar este tipo de bloqueo.
- En general, solicitar el mismo tipo de bloqueo para todas las referencias a objetos tiene como resultado limitar la posibilidad de concurrencia.
- Solicitar el bloqueo adecuado a cada tipo de acceso aumenta esta posibilidad.

Utilización de las técnicas de bloqueo para aumentar la concurrencia ...

- En algunas ocasiones, dos o más transacciones se bloquean esperando a que cada una levante el bloqueo sobre el objeto que está esperando la otra.
- Ninguna de las dos puede continuar hasta obtener el acceso deseado al objeto que bloquea la otra y puesto que no puede continuar, tampoco levanta los bloqueos propios.
- Esta situación se conoce con el nombre de lazo mortal (*deadlock*).

Los ODBMSs disponen de varios mecanismos para evitar este tipo de situaciones.

- Una técnica consiste en que el ODBMS obligue a cualquier transacción a adquirir todos los bloqueos que va a necesitar en una única operación indivisible al principio de la transacción. Esta aproximación es poco realista, ya que en muchos casos las transacciones no saben a qué objetos van a acceder. Cuando se emplea esta técnica, la transacción generalmente bloquea toda la base de datos.
- Una técnica menos limitante consiste en hacer que el ODBMS mantenga una serie de grafos con la información sobre:
  - Las transacciones que han solicitado un bloqueo sobre un determinado objeto.
  - Los bloqueos que ha adquirido cada una de las transacciones en curso.

  De este modo, el ODBMS es capaz de determinar cuando la concesión de un bloqueo va a dar lugar a un *deadlock* y evitarlo. Esta técnica es difícil de implementar sobre todo en entornos distribuidos.

Registro de incidencias

El fallo del sistema es uno de los motivos más frecuentes de que una transacción se aborte. Todos los sistemas fallan en alguna ocasión y es responsabilidad del ODBMS habilitar los mecanismos de recuperación frente a estos fallos.

En un entorno de base de datos no es responsabilidad del usuario o de la aplicación determinar como se debe actuar en estas circunstancias.

- En muchos ámbitos de las aplicaciones informáticas, ante una situación de fallo el usuario no tiene más que reiniciar la máquina y las aplicaciones volviendo al estado correspondiente a la última vez que se salvaron los ficheros.
- En un entorno de base de datos compartido la base de datos no pertenece a un único individuo sino que es accedida y utilizada por varios usuarios y aplicaciones. El ODBMS tiene la responsabilidad de realizar copias de respaldo automáticas y de recuperar la base de datos en caso de fallo.
- Para llevar a cabo esta tarea se emplean varias técnicas dependiendo de la extensión del fallo: si sólo falla una transacción, el ODBMS sólo tiene que recuperar dicha transacción. Sin embargo, frente a un fallo del sistema el ODBMS debe ser capaz de recuperar todas las transacciones.

Registro de incidencias ...

- Abortar una transacción significa que las modificaciones que ésta ha realizado no son escritas en el almacenamiento permanente. Pueden darse dos situaciones:
  - Las modificaciones se realizaron sólo en memoria principal y no es necesario escribir nada en el registro de incidencias o en la base de datos.
  - Algunas de las modificaciones ya se han escrito en el registro o en la base de datos. El ODBMS debe deshacer estar modificaciones empleando las copias previas.
- La recuperación de todas las transacciones es una tarea mucho más compleja, ya que cada transacción podría encontrarse en uno de varios estados:
  - La transacción no ha alcanzado todavía un punto de *commit* o *abort*. Ninguna de las actualizaciones de esta transacción en proceso está en almacenamiento. El ODBMS gestiona esta situación como si abortara una transacción individual (utilizando la imagen previa del fichero de incidencias).
  - La transacción estaba siendo abortada en el momento del fallo. El ODBMS debe asegurarse de que este proceso se completa adecuadamente. Para ello aplica los cambios documentados en la copia previa a los objetos afectados hasta deshacer todos los cambios en curso (*Roll back*.
  - Algunas transacciones podrían estar en proceso de validación en el momento del fallo. El ODBMS debe garantizar que este proceso se completa. Para reconstruir estas modificaciones el ODBMS hace uso de las *after-images* del registro de incidencias (*Roll forward*).

Consideraciones acerca del rendimiento

El mecanismo de recuperación frente a fallos tiene un serio inconveniente debido al número de accesos a disco que requiere.

- Por ejemplo, para que el ODBMS pueda completar las transacciones en curso de validación (último punto del apartado anterior), debe escribir las imágenes finales en el registro de incidencias a medida que la aplicación se ejecuta.
- Si el rendimiento es un factor crítico e implica decidir entre mantener copia de las imágenes finales o de las previas, la elección suele decantarse sobre estas últimas. Entonces, el ODBMS sólo puede deshacer las transacciones imcompletas.

El rendimiento de un ODBMS viene afectado por muchos factores. Uno de los más importantes esta asociado a la carga que supone la gestión de transacciones y del registro de incidencias.

- La utilización de *after-images* dobla el número de operaciones de escritura en disco que debe realizar el ODBMS.
- Algunos ODBMSs proporcionan opciones para minimizar la carga asociada al registro de incidencias. Por ejemplo, si sólo se modifica una pequeña parte de un objeto muy grande, en el registro de incidencias sólo se refleja esta parte modificada. De forma análoga, si se está modificando conjuntamente un gran número de pequeños objetos, puede escribirse un registro global en lugar de registros individuales para cada objeto.
- Muchos ODBMSs permiten al usuario o al administrador de la base de datos la opción de activar o desactivar selectivamente el mecanismo de registro de *before-* y *after-images*. Desactivar esta opción puede mejorar el rendimiento de forma significativa.