**Transacciones y Control de Concurrencia**

**Transacción**

Una transacción es una colección de operaciones que realiza una única función lógica en una aplicación de base de datos.

**Requerimientos de un Sistema de Manejo de Base de Datos**

* Atomicidad: las operaciones de una transacción se ejecutan todas o ninguna.
* Consistencia: la ejecución de una transacción en aislación (esto es, sin otras transacciones ejecutándose concurrentemente) preserva la consistencia de la BD.
* Aislación: Aunque las transacciones se ejecuten concurrentemente, su ejecución debe ser equivalente a alguna secuencia de ejecuciones en serie.
* Durabilidad: Después de que una transacción se ejecutó con éxito, los cambios en la BD persisten, más allá de las fallas del sistema.

**Tratamiento de Deadlocks**

Un sistema está en deadlock (abrazo mortal) si existe un conjunto de 2 o más transacciones {T1, T2, …, Tn} (n>=2) en espera tales que T1 espera un ítem retenido por T2, T2 espera un ítem retenido por T3, …, Tn-1 espera un ítem retenido por Tn, y Tn espera un ítem retenido por T1.

**Prevención de Deadlocks**

Ordenamiento de Datos: Con esta técnica, los bloqueos se pueden pedir en un único orden y no es posible entrar en deadlock. Desventajas:

* Antes de que comience cada transacción, es difícil predecir que ítems va a requerir.
* Baja utilización de recursos.

Por Retroceso: Aquellas transacciones que solicitan un bloqueo que puede generar un deadlock son retrocedidas.

* Wait-Die (Esperar-Morir): Si Ti requiere un dato que tiene Tj, Ti espera si ts(Ti) < ts(Tj) (Ti es más antigua que Tj). De lo contrario, Ti es retrocedida.
* Wound-Wait (Herir-Esperar): Si Ti requiere un dato que tiene Tj, Ti espera si ts(Ti) > ts(Tj) (Ti es más joven que Tj). De lo contrario, Tj es retrocedida y sus recursos apropiados por Ti.

**Detección y Recuperación de Deadlocks**

No se utiliza un protocolo para evadir deadlocks. En determinados momentos, el sistema busca situaciones de deadlock y si se presentan inicia un proceso de recuperación. Para ello el sistema necesita:

* Mantener información acerca de la asignación de ítems de datos a las transacciones, así como de las solicitudes.
* Proveer un algoritmo de detección para determinar si el sistema está en deadlock.
* Recuperarse de un deadlock cuando el sistema detecta que existe tal situación.

**Recuperación de Deadlocks**

Se deben llevar a cabo tres acciones:

* Seleccionar una víctima: Dado un conjunto de transacciones en deadlock, determinar cuáles transacciones deben retroceder.
* Retroceder transacción: Determinar si una transacción debe ser retrocedida por completo (deshaciendo todos los cambios que hizo) o parcialmente (esto requiere información adicional del sistema).
* Evitar inanición: El sistema debe elegir como víctima a una transacción un numero finito de veces para evitar que la misma entre en estado de inanición (cuando nunca se completa su ejecución).

**Recuperación ante Fallos**

**Puntos de Verificación Difusos**

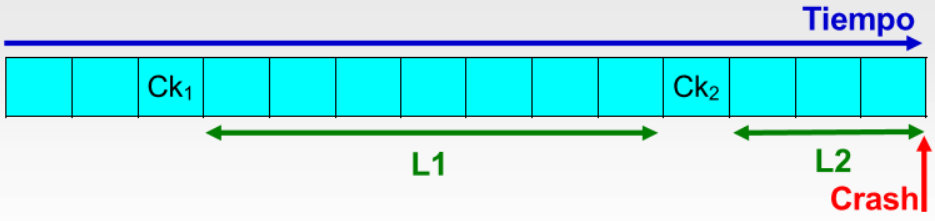
Los puntos de verificación (checkpoints) requieren que todas las actualizaciones a la base de datos se suspendan temporariamente. La técnica de *puntos de verificación difusos* permite que se reinicien las actualizaciones después de grabar el registro de checkpoint en bitácora, pero antes de que los bloques modificados se escriban en disco.

En lugar de recorrer hacia atrás la bitácora hasta encontrar un registro de checkpoint (donde se hizo la última escritura segura), se almacena en una posición fija del disco la ubicación del ultimo checkpoint (last-checkpoint). Antes de escribir el registro de checkpoint, se crea una lista de todos los buffers modificados.

La información de last-checkpoint se actualiza después de que todos los bloques de buffer en la lista de modificados han sido grabados en disco. Obviamente, los bloques de buffer no deben modificarse mientras están siendo grabados en disco. Se debe contar con un protocolo de escritura por adelantado (write-ahead) para que se graben antes los registros de la bitácora (undo) pertenecientes a los bloques que están siendo grabados en memoria estable.

Este esquema usa un *registro de bitácora lógico* (mediante una operación de modificación sobre los datos en sí) para propósitos de deshacer (undo), y un *registro de bitácora físico* (guardando los valores viejos y nuevos) para propósitos de rehacer (redo) y deshacer (undo) transacciones.

*Ejemplo:*



En el momento en que el Checkpoint 2 se graba en la bitácora (Ck2), todos los bloques modificados en el buffer de la base de datos antes del Checkpoint 1 (Ck1, que en este caso es el last-checkpoint) fueron volcados a memoria estable.

Las actualizaciones a la base de datos en el periodo de tiempo L1 se graban en el buffer, pero no necesariamente en memoria estable. Si un bloque de la base de datos está en el buffer cuando se graba el registro Ck2, no necesariamente el mismo estará en memoria estable.

Para garantizar que un bloque de la base de datos modificado en L1 este en memoria estable necesitamos que se agregue un nuevo registro de Checkpoint (en este ejemplo, lo sería un hipotético Ck3). Por lo tanto, se asume que los cambios hechos en la base de datos en el periodo L1 NO están en memoria estable.

El proceso de recuperación en este ejemplo comienza desde Ck1 (last-checkpoint) hasta Ck2. De este modo, el proceso de recuperación es más lento, pero permite que las transacciones se reanuden de inmediato cuando se inicia un proceso de checkpointing.

**Comparación de Métodos**

* Bitácora Tradicional:
  + Permite recuperación ante fallos (+)
  + Consume excesivo tiempo de operaciones Redo en caso de un fallo (-)
* Bitácora con Checkpointing:
  + Se ahorran operaciones Redo en caso de fallos (+)
  + Se paralizan las transacciones durante el proceso de checkpointing (-)
* Bitácora con Checkpointing Difuso:
  + No se paralizan las transacciones durante el proceso de checkpointing (+)
  + Se consume más tiempo en la recuperación, pero esto se compensa con la reanudación inmediata de las transacciones después del checkpointing (+)

**Perdida de información NO volátil**

Aunque rara vez se producen fallos en la memoria no volátil, debemos estar preparados para este tipo de fallos. Una técnica es copiar periódicamente el contenido entero (dump) de la base de datos a un almacenamiento estable (cintas magnéticas). Si falla el almacenamiento no volátil, se restaura el contenido de la última copia. Una vez hecho esto, el sistema usa la bitácora para llevar a la base de datos al más reciente estado consistente posible.

Ninguna transacción puede estar activa durante el proceso de dump y se ejecuta un proceso similar al checkpointing:

1. Se vuelcan los registros de bitácora residiendo actualmente en memoria principal al almacenamiento estable.
2. Se vuelcan todos los bloques de buffer al disco.
3. Se copan los contenidos de la base de datos al almacenamiento estable.
4. Se vuelca un registro de bitácora <dump> al almacenamiento estable.

**Implementación de Memoria Estable**

La información que reside en memoria estable nunca se pierde. Para alcanzar ello, debe repetirse la información (de manera controlada) en varios medios de almacenamiento. La transferencia de bloques entre la memoria y el disco puede resultar en:

* Terminación con Éxito: La información transferida llego integra a su destino.
* Fallo Parcial: Ocurrió un fallo durante la transferencia y el bloque destino tiene información incorrecta.
* Fallo Tota: Ocurrió un fallo al inicio de la transferencia y el bloque destino queda intacto.

Si ocurre un fallo en la transferencia de datos, el sistema debe detectarlo e invocar a un sistema de recuperación que restaure el bloque a un estado consistente. Para hacerlo, el sistema debe mantener, al menos, dos bloques físicos por cada bloque lógico de la base de datos. Una operación de salida se ejecuta como sigue:

1. Escribir la información en el primer bloque físico.
2. Una vez que se completa con éxito la primera escritura, escribir la misma información en el segundo bloque físico.
3. La salida está completa solo después de terminar con éxito la segunda escritura.

Durante la recuperación, se examina cada par de bloques físicos:

* Si los dos son iguales y no existen errores detectables, no es necesario tomar acciones.
* Si un bloque contiene un error detectable, se sustituye su contenido por el valor del otro bloque.
* Si ambos bloques contienen errores no detectables, pero el contenido es diferente, entonces se vuelve a repetir el proceso de resguardo.

Este procedimiento garantiza que una escritura en almacenamiento estable termine con éxito o no produzca un cambio alguno. El intento de escribir en almacenamiento estable tiene éxito solo si se escriben todas las copias correctamente. Al igual que en checkpointing, se recurre a métodos difusos para no paralizar el sistema.

**ARIES**

Es una técnica avanzada de recuperación ante fallos:

* Incorpora numerosas optimizaciones para reducir la sobrecarga durante el procesamiento de transacciones y acelerar las recuperaciones.
* Utiliza Algoritmos de Recuperación Avanzados, pero remueve optimizaciones para lograr una mejor performance promedio.

AIRES utiliza:

1. Un numero de secuencia de bitácora (LSN: log sequence number) para identificar registros de bitácora.
   1. Almacena LSN’s en páginas para identificar que actualizaciones se aplicaron a la página de base de datos.
2. Redo’s “Filosoficos”
3. Una tabla de páginas modificadas para evitar Redos innecesarios durante la recuperación.
4. Checkpointing Difuso que registra información de páginas modificadas y no requiere que esas páginas sean escritas en almacenamiento estable en el momento de checkpointing.

**Sistemas de Bases de Datos**

**Sistemas Centralizados**

*Características:*

* No interactúan con otros sistemas.
* Abarcan desde sistemas para un único usuario a mainframes de alta performance.
* Contienen una CPU, una memoria principal y una serie de controladores encargados de funciones específicas.
* La CPU y los controladores pueden ejecutar acciones de manera concurrente, compitiendo por el acceso a memoria.
* El uso de memorias cache recude el número de accesos de la CPU a la memoria principal

*Pueden ser:*

* Sistemas Monousuario: Son principalmente usadas o administradas por una persona, cuentan con una CPU, uno o dos discos rígidos y un sistema operativo monousuario. Ejemplos: PC’s, Workstations, etc.
* Sistemas Multiusuario: Generalmente cuentan con varios usuarios, varias CPU’s, más de una memoria y un sistema operativo multiusuario. Puede atender a varios usuarios que están conectados al sistema por medio de terminales.

**Sistemas Cliente-Servidor**

Actualmente, los sistemas centralizados actúan como sistemas servidores que satisfacen los requerimientos generados por sistemas clientes. Las aplicaciones de bases de datos se dividen en:

* Aplicaciones Back-End: Manejan estructuras de acceso, evaluación y optimización de consultas, control de concurrencia y recuperación antes fallos.
* Aplicaciones Front-End: Abarcan generadores de reportes y facilidades para el diseño de interfaces gráficas para el usuario.

*Categorías:*

* Servidores de Transacciones (de Consultas): Proveen una interface mediante la cual los clientes pueden enviar requerimientos para ejecutar una acción, en respuesta a lo cual ellos ejecutan la acción y envían el resultado al cliente.
* Servidores de Datos: Interactúan con los servidores haciendo requerimientos para lectura o actualización de datos. Permiten diferentes unidades de datos (paginas, tuplas, objetos) y protegen a los datos de posibles inconsistencias.

**Sistemas Paralelos**

Mejoran el procesamiento y las velocidades de E/S usando más de una CPU y más de un medio de almacenamiento. En estos sistemas, muchas operaciones se desarrollan simultáneamente, en contraposición a los sistemas seriales y concurrentes.

Categorías:

* Coarse-Grain (Granularidad gruesa): Maquinas con un pequeño número de procesadores.
* Massively parallel o Fine-Grain (Granularidad fina): Maquinas con cientos de procesadores pequeños.

*Medidas de Performance*

Un SMBD considera dos parámetros:

* Rendimientos (throughput): El número de tareas que pueden terminarse en un periodo de tiempo.
* Tiempo de retorno: El tiempo que toma para completar un proceso.

En cambio, un SO también considera:

* Utilización del CPU: Se busca maximizar el uso del/los procesador/es central/es.
* Tiempo de espera: El tiempo que espera un proceso en realizar una operación de E/S.
* Tiempo de respuesta: El tiempo que toma en dar una primera respuesta a un proceso.

Los sistemas paralelos aumentan el rendimiento procesando muchas transacciones en paralelo. El tiempo de retorno también es mejor ya que pueden desarrollar subtareas de una transacción en paralelo. Cuestiones de estudio en sistemas paralelos:

* Speedup: Se refiere a ejecutar una tarea en menos tiempo incrementando el grado de paralelismo.
* Scaleup: Se refiere a tratar tareas más complejas incrementando el grado de paralelismo.

*Arquitecturas:*

* Memoria Compartida: Todos los procesadores comparten una memoria en común.
* Disco Compartido: Todos los procesadores comparten un medio de almacenamiento.
* Nada Compartido: Cada procesador tiene su propia memoria y su propio disco.
* Jerárquico: Es un sistema hibrido entre memoria compartida, disco compartido y nada compartido.

**Sistemas Distribuidos**

La base de datos se almacena en varias computadoras en un sistema distribuido comunicado mediante diferentes dispositivos de comunicación. En estos sistemas, las computadoras pueden variar en tamaño y función, desde workstations hasta mainframes. Cada computadora en un sistema distribuido se denomina sitio o nodo.

Los sistemas distribuidos se asemejan a los sistemas paralelos con estructura nada compartido. Sin embargo, generalmente están en lugares diferentes (geográficamente separados), son administrados independientemente y tienen una interconexión más lenta.

Otra diferencia es el tipo de transacciones que admiten. Las transacciones locales acceden a datos de un único sitio mientras que las transacciones globales pueden acceder a datos que están en sitios diferentes a donde se inicia la transacción.

*Características:*

* Cada sitio conoce la existencia de los otros.
* Los sitios compartes un esquema global, aunque algunas relaciones (conjuntos de tuplas) pueden ser almacenados entre otros sitios.
* Cada sitio provee un entorno para ejecutar tanto transacciones locales como globales.
* Cada sitio ejecuta el mismo software de manejo de bases de datos distribuidas.
* Comparte muchas características de los sistemas cliente-servidor y de los sistemas paralelos.

|  |  |
| --- | --- |
| Ventajas | Desventajas |
| Compartimiento de Datos: Los usuarios de un sitio pueden acceder a datos de otros sitios. | Costo del Desarrollo de Software: Es más difícil y costoso implementas un sistemas de base de datos distribuido. |
| Autonomía: Cada sitio es capaz de mantener el control de los datos que están almacenados localmente. Existe un administrador global del sistema y cada sitio cuenta con un administrador local. | Mayor Probabilidad de Errores: Puesto que los sitios que constituyen el sistema distribuido operan en paralelo, es más difícil asegurar la correctitud de los algoritmos. |
| Disponibilidad: Si un sitio falla, el resto de los sitios puede continuar operando. Si la información se replica, el fallo de un sitio puede no afectar a todo el sistema. | Incremento en la Sobrecarga de Procesamiento: El intercambio de mensajes y los cómputos adicionales para coordinar los sitios provocan una gran sobrecarga. |

Cada sitio de un sistema contiene dos subsistemas:

* El Gestor de Transacciones: Es el encargado de aquellas transacciones (o subtransacciones) que acceden a datos almacenados en el sitio local.
* El Coordinador de Transacciones: Es el encargado de la ejecución de las transacciones (locales o globales) iniciadas en el sistema.

*Tipos de Redes*

* LANs (Local Area Networks): Cubren áreas geográficas pequeñas (un edificio o varios edificios cercanos).
* WANs (Wide Area Networks): Cubren áreas geográficas extensas (ciudades, países, continentes).

*Topologías de Redes*

* Red Totalmente Conectada: Existe un vínculo entre cada par de nodos.
  + Gran tolerancia a fallas de comunicación (+)
  + Muy alto costo de instalación (-)
* Red Parcialmente Conectada: Algunos pares de nodos no están conectados.
  + Aceptable costo de instalación (+)
  + Si un mensaje falla por un camino, puede ser redirigido por otro camino (-)
* Red Tipo Árbol: Existe una estructura arbórea con un nodo raíz y sus hijos. Cada nodo puede tener 0 o más nodos hijos, pero un solo nodo padre.
  + Bajos costos de instalación y comunicación (+)
  + Si se cae un nodo, sus hijos quedan aislados (-)
* Red Estrella: Existe un nodo al cual todos están conectados.
  + El costo de comunicación es bajo (+)
  + Si falla un nodo lateral queda aislado pero el resto continua de manera normal (+)
  + Si falla el nodo central, el sistema queda totalmente desconectado (-)
* Red Anillo: Los nodos están conectados circularmente.
  + El grado de disponibilidad es alto (+)
  + El costo de comunicación suele ser alto porque debe atravesar muchos vínculos. (-)

**Transacciones Distribuidas**

Para garantizar la atomicidad, es preciso que todas las localidades en las que haya ejecutado la transacción T coincidan en el resultado final de la operación. Las subtransacciones de T deben quedar cometidas o abortadas en todas las localidades.

**Protocolo de Compromiso Distribuido**

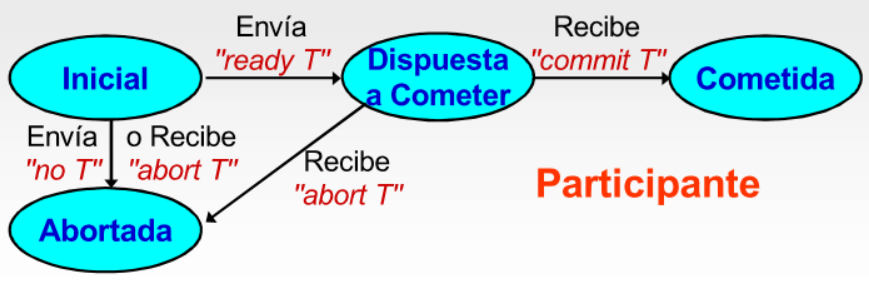
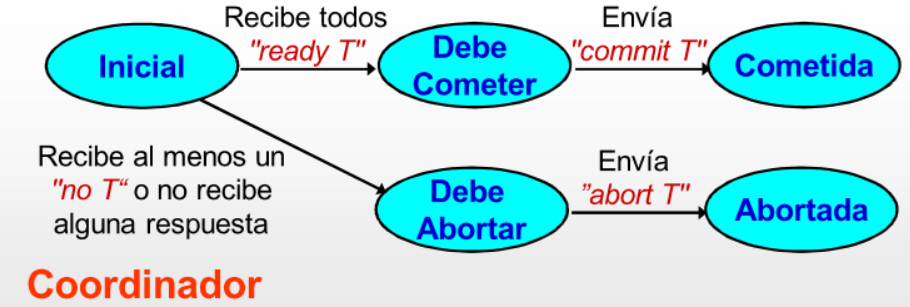
El protocolo de bloqueo de 2 fases en entornos distribuidos es difícil de implementar ya que las subtransacciones en sitios diferentes complica el proceso considerablemente. Supongamos que una transacción T fue iniciada en un sitio S y fue dividida en varias subtransacciones en diferentes sitios, incluida el mismo sitio. La subtransaccion perteneciente al sitio S se denomina *coordinador* mientras que las otras subtransacciones se denominan *participantes*.

Cada subtransaccion Ti decide cometer o abortar. Cuando Ti decide que hacer, envía un “ready T” o un “no T”, que representa un voto para cometer o abortar respectivamente. Si Ti envía un “no T” entonces sabe que T debe abortar. En cambio, si envía un “ready T” no puede asegurar que T sea cometida, ya que eso depende de lo que voten los otros participantes. Después de votar “ready T”, Ti debe esperar la decisión del coordinador.

El coordinador realizara un proceso de decisión:

* Si recibe todos “ready T” de los participantes, decide cometer y envía un “commit T” a todos los participantes.
* Si recibe al menos un “no T” de un participante, decide abortar y envía un “abort T” a todos los participantes.

Cuando los participantes reciben la orden de cometer, realizar el compromiso, escribiendo los registros de bitácora, volcando los cambios al almacenamiento no volátil y liberando los bloqueos. Notemos que en este esquema existen mensajes de voto y mensajes de decisión, y que, como el coordinador también es un participante, los mensajes que se envía a sí mismo no requieren tráfico de red.

Cuando se presentas fallas en la red, este protocolo puede llevar a estados de bloqueo, esto es, una subtransaccion en un sitio que no fallo no puede cometer ni abortar hasta que se repare la falla en el sitio origen.

Ejemplo: Una subtransaccion Ti tiene un bloque sobre una copa de un ítem de dato A y alcanza su punto de compromiso (dispuesta a cometer).

* Después de un tiempo prolongado de espera, no recibe el mensaje “commit T” ni “abort T” del coordinador.
* Luego, debe permanecer en ese estado, reteniendo el bloqueo sobre la copia local de A.
* Ti queda bloqueada (no en deadlock, ya que no hay espera circular) y no puede decidir por si sola.

Si Ti decide cometer sin esperar la decisión del coordinador, puede que otra subtransaccion con una copia local de A haya decidido abortar, sin que el coordinador le haya comunicado a Ti que aborte.

* Si Ti comete, otra subtransaccion puede leer la copia local de A, que no debería haber sido cambiada.

Si Ti decide abortar sin esperar la decisión del coordinador, podría ser el caso en que el coordinador recibió un “ready T” de todos los participantes, pero la red fallo, aislando a Ti del coordinador.

* Si los participantes reciben el mensaje “commit T” del coordinador, pueden escribir nuevos valores para las diferentes copias de A.

**Protocolo de Compromiso de 2 Fases**

Sea T una transacción que se inició en el sitio Si, y sea Ci el coordinador de transacciones de esa localidad. Cuando T termina de ejecutarse, esto es, cuando todos los sitios en las que se ejecutó T informan a Ci que T llego a su término, Ci inicia el protocolo de compromiso de 2 fases:

*Fase 1 (de Votación)*: Se recopila información de los diferentes sitios para saber si las diferentes subtransacciones pueden completarse.

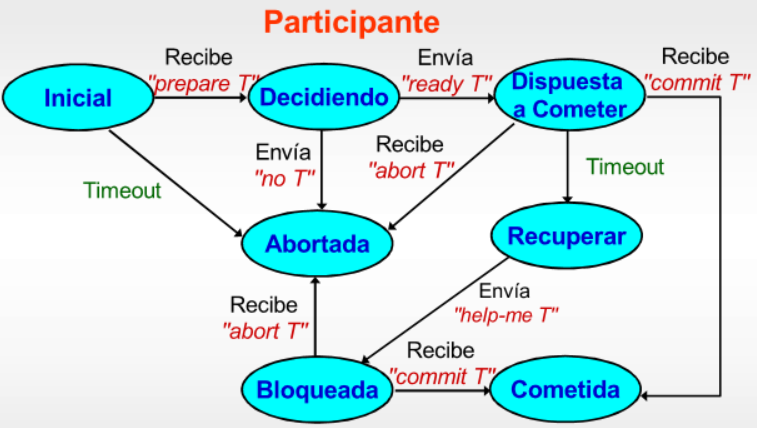
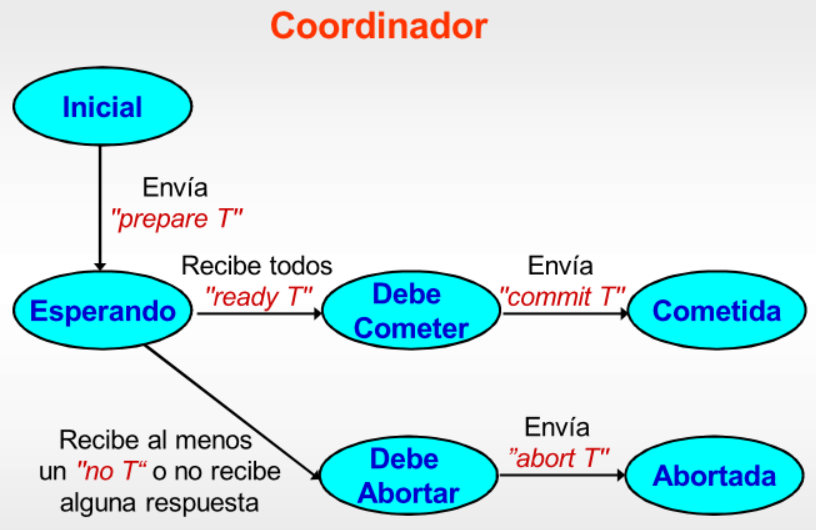
* Ci agrega el registro <prepare T> a la bitácora y lo graba en memoria estable.
* Ci envía un mensaje “prepare T” a todos los sitios donde se ejecutó T.
* Cada gestor que recibe el mensaje determina si puede o no cometer su porción de T:
  + Respuesta Si: agrega <ready T> a la bitácora, la graba en memoria estable y envía el mensaje “ready T” a Ci.
  + Respuesta No: agrega <no T> a la bitácora, la graba en memoria estable y envía el mensaje “abort T” a Ci.

*Fase 2 (de Decisión)*: Se determina si la transacción global puede cometerse o no.

* Ci recibe respuesta de todos los sitios o transcurre un periodo determinado de tiempo.
* Si recibe un mensaje “ready T” de cada sitio que ejecuto una subtransaccion de T, entonces graba el registro <commit T> a la bitácora, se graba en memoria estable y se envía el mensaje “commit T” a todos los sitios participantes.
* En otro caso, se graba el registro <abort T> en la bitácora, se graba en memoria estable y se envía el mensaje “abort T” a todos los sitios participantes.

Un sitio en el cual T se ejecuto puede abortar T de manera incondicional en cualquier momento anterior al envío del mensaje “ready T”. El mensaje “ready T” es una promesa de que se cometerá T o abortará cuando lo indique el coordinador Ci. La única manera que tiene un sitio de hacer tal promesa es almacenando tal información en memoria estable. Si el sitio se cae después de enviar el “ready T”, puede estar inhabilitado de cumplir su promesa.

Puesto que se requiere unanimidad para cometer una transacción, la suerte de T se sella tan pronto como un sitio responde “abort T”. Puesto que el sitio coordinador Si es uno de los sitios en donde se ejecuta T, su coordinador (Ci) puede unilateralmente decidir abortar T. El veredicto sobre T lo determina el coordinador mediante un <commit T> o <abort T> en la bitácora y luego en memoria estable.

*Fallos en PC de 2 Fases*

Los diferentes tipos de errores son:

* Falla de un sitio participante: En ese caso el coordinador detecta el fallo y decide si puede continuar o no la ejecución de la transacción.
* Falla del coordinador: Los participantes deben decidir la suerte del coordinador o esperar que el mismo se recupere.
* Partición de la Red: El coordinador y los participantes pueden quedar:
  + En una misma partición
  + En varias particiones diferentes

*Fallo de un Participante*

El coordinador Ci detecta que un sitio ha fallado:

* Si el sitio falla antes de responder “ready T”, asume que fue un “abort T”.
* Si el sitio falla después de responder “ready T”, se ignora la falla.

Cuando un participante se recupera de un fallo se mira la bitácora:

* Si contiene un <commit T> ejecuta un Redo(T).
* Si contiene un <abort T> ejecuta un Undo (T).
* Si contiene un <ready T> debe consultar a Ci (enviando un mensaje “help-me T”) para decidir qué hacer con T:
  + Si Ci contesta “commit T”, ejecutar Redo(T).
  + Si Ci contesta “abort T”, ejecutar Undo(T).
  + Si Ci no contesta debe comunicarse con los otros sitios.
* En cualquier otro caso, se ejecuta Undo(T).

*Fallo del Coordinador*

Existen dos opciones:

* Los participantes se comuniquen entre si y decidan qué hacer.
* Los participantes esperan que se recupere el coordinador.

Nuevamente, es necesario inspeccionar la bitácora:

* Si un sitio activo contiene un <commit T> en su bitácora, entonces T debe ser cometida.
* Si un sitio activo contiene un <abort T> en su bitácora, entonces T debe ser abortada.
* Si algún sitio no contiene un <ready T> entonces el coordinador fallado Ci no pudo haber decidido cometer T ya que no recibió un “ready T” de todos los participantes. En general, en este caso se decide abortar T.
* En otro caso, cada participante debe tener un <ready T> en su bitácora y los participantes deben esperar que se recupere Ci.

*Recuperación y Control de Concurrencia*

Cuando un sitio que fallo se reinicia, se pueden utilizar los algoritmos de recuperación vistos. En los protocolos de compromiso distribuido, el procedimiento de recuperación debe tratar con las transacciones dudosas. Las transacciones dudosas son aquellas para las cuales se encuentra un registro <ready T> en la bitácora, pero nunca un registro <commit T> ni <abort T>. Cuando se inicia un proceso de recuperación, la ejecución normal de un sitio no puede continuar hasta tanto las transacciones dudosas no hayan sido cometidas o abortadas. Detectar el estado (de bloqueos) de todas las transacciones dudosas puede ser lento pues necesita conectar varios sitios.

Para evitar este problema, se modifica el esquema de bitácora, grabando un registro <ready T, L> (L es la lista de los bloqueos retenidos) en lugar de un registro <ready T>. Después de que cada transacción dudosa readquirió sus bloqueos, el procesamiento de una transacción puede comenzar en el sitio, aun antes de que las transacciones dudosas hayan alcanzado el estado de cometidas/abortadas. Esto implica que la ejecución completa o el retroceso de las transacciones dudosas se puede desarrollar de manera concurrente con otras nuevas transacciones. Las nuevas transacciones en conflicto con las transacciones dudosas no podrán continuar hasta que las transacciones dudosas en conflicto sean cometidas o abortadas.

*Partición de la Red*

* Si el coordinador y todos los participantes permanecen en una partición, la falla no tiene efecto.
* Si el coordinador y los participantes quedan en particiones diferentes:
  + Los sitios en la partición que NO contiene al coordinador creen que el coordinador fallo y ejecutan el protocolo para tratar con la falla del coordinador.
    - No hay peligro, aunque algunos sitios pueden esperar por decisiones del coordinador.
  + El coordinador y los sitios en la misma partición del coordinador creen que los otros sitios han fallado y siguen el protocolo normalmente.
    - No hay riesgos a la vista.

El protocolo de Compromiso de 2 Fases evita ciertas situaciones de bloqueo que se producen en el Protocolo de Compromiso Distribuido, pero aun así se producen bloqueos. Más aun, ningún esquema puede evitar bloqueos cuando la red se desconecta, pero puede reducirse el número de posibles bloqueos. Intuitivamente, el PC de 2 Fases permite a un participante cometer tan pronto como todos los participantes hayan votado “ready T”.

El PC de 3 Fases permitirá a un participante cometer no solo cuando sepa que todos los participantes han votado cometer, sino cuando todos lo sepan también (y si fallan, podrán saberlo cuando se recuperen).

**Protocolo de Compromiso de 3 Fases**

En el PC de 3 Fases existe una tercera ronda de mensajes. El segundo mensaje desde el coordinador es “precommit T” y notifica a todos los participantes que todos están dispuestos a cometer, como antes. Sin embargo, un participante no comete hasta no recibir este mensaje. Cada participante determina que recibió este mensaje enviando un mensaje “acknowledge T” al coordinador. La tercera fase exige que el coordinador recopile todos estos mensajes y, una vez que recibe cierto número, envía un mensaje “commit T” a cada uno de los participantes.

Este protocolo evita la posibilidad de bloqueos siempre que se restrinja el número de fallas posibles (algo muy difícil de garantizar). El PC de 3 fases exige que:

* No ocurran particiones de la red (para que, en caso de fallos, se pueda elegir un nuevo coordinador).
* A lo sumo k sitios participantes pueden fallar mientras se ejecuta el protocolo (k es un parámetro que mide la tolerancia a los fallos).
* En cualquier momento, debe haber al menos k+1 sitios funcionando correctamente.

Para evitar bloqueos, se agrega una fase en la cual se alcanza una decisión preliminar sobre el destino de T:

Fase 1: igual al PC de 2 Fases.

Fase 2:

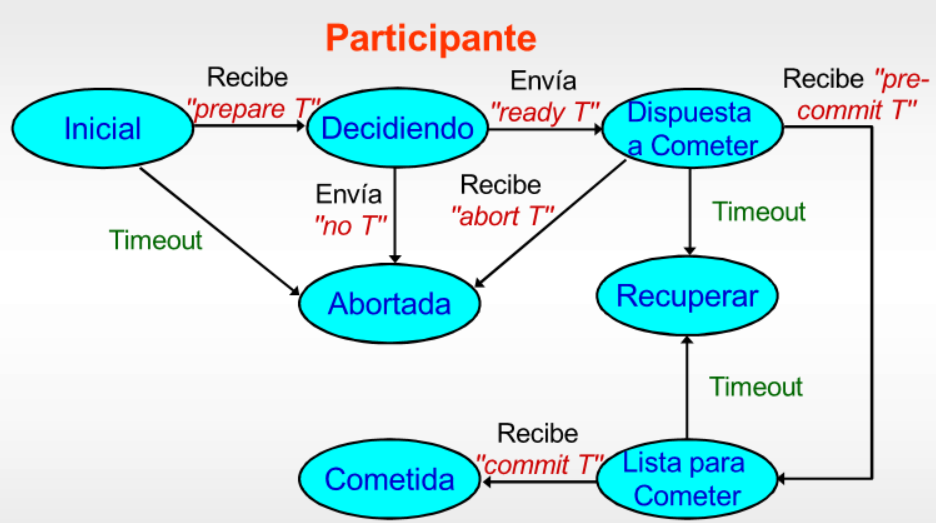
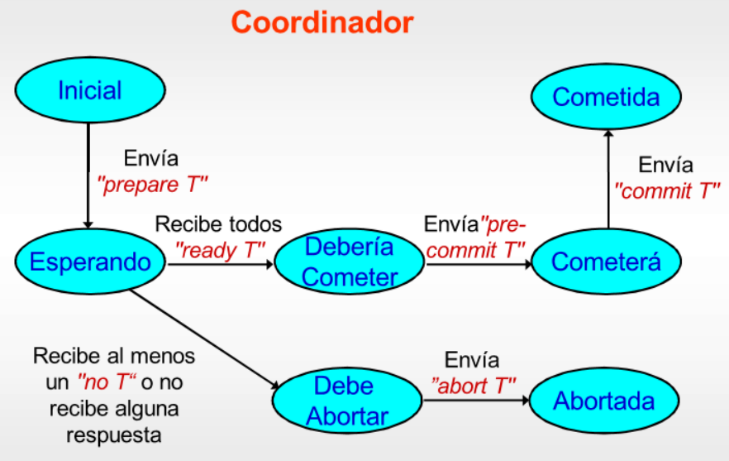
* Si Ci recibe un mensaje “no T” de un participante o si no recibe respuesta (en cierto periodo), decide abortar T, lo graba en bitácora y envía el mensaje “abort T” a todos los participantes.
* Si Ci recibe un mensaje “ready T” de cada sitio participante, toma la decisión preliminar de pre-compromiso, grabando en bitácora <precommit T> y envía el mensaje “precommit T” a todos los participantes.
* Cuando un participante recibe el mensaje de aborto o pre-compromiso graba en bitácora el mensaje respectivo (<abort T> o <precommit T>) y envía el mensaje <acknowledge T> al coordinador.

Fase 3: Esta fase se ejecuta solamente si la decisión de la Fase 2 fue de pre-compromiso.

* Después que se enviaron los respectivos mensajes “precommit T” a todos los participantes, debe esperar que al menos k sitios envíen el mensaje “acknowledge T”, En la práctica, este mensaje de reconocimiento no siempre es necesario, aunque puede ayudar a detectar ciertos errores.
* Recién en este momento, el coordinador decide cometer grabando en bitácora <commit T> y enviando un mensaje “commit T” a cada uno de los participantes.
* Cuando cada participante recibe ese mensaje, lo graba en su respectiva bitácora.

Como en el PC de 2 Fases, un sitio puede decidir abortar T enviando un mensaje “abort T” antes del “ready T”. Mientras que en el PC de 2 Fases el coordinador puede incondicionalmente abortar T antes de enviar el mensaje “commit T”, el mensaje “procommit T” es una promesa del coordinador de que eventualmente se cometerá T por voto de los participantes.

La fase 3 de este protocolo lleva a una decisión de commit por lo que parece de poca utilidad práctica. El rol de la fase 3 se justifica en caso de fallos

*Fallo de un participante*

* Si contiene un <commit T> en su bitácora, ejecutar Redo(T).
* Si contiene un <abort T> en su bitácora, ejecutar Undo(T).
* Si contiene un <ready T> pero no un <abort T> ni un <precommit T> consulta con Ci:
  + Si Ci responde “abort T”, ejecuta Undo (T).
  + Si Ci responde “precommit T”, graba el registro <precommit T> en su bitácora y envía un mensaje “acknowledge T” al Ci.
  + Si Ci responde “commit T”, ejecuta Redo(T).
  + Si Ci no responde se ejecuta el protocolo de fallo del coordinador.
* Si contiene un <precommit T> pero no un <abort T> ni un <commit T> consulta con Ci y ejecuta Undo(T) si recibe el mensaje “abort T” y Redo(T) si recibe un “commit T”.

*Fallo del coordinador*

1. Los sitios participantes activos seleccionan un nuevo coordinador.
2. El nuevo coordinador Cnew envía un mensaje a cada sitio participante requiriendo el estado local de T.
3. Cada sitio participante (incluyendo Cnew) determina el estado local de T (cometida, abortada, lista, precometida, no lista).
4. Dependiendo de las respuestas recibidas, Cnew decide si cometerá o abortará T.

Esta secuencia de pasos se conoce como Protocolo de Fallo del Coordinador.

*¿Cómo detectar el estado de T en cada sitio?*

* Cometida: Cuando la bitácora contiene un registro <commit T>.
* Abortada: Cuando la bitácora contiene un registro <abort T>.
* Lista: La bitácora contiene un <ready T> pero no contiene un <abort T> ni un <precommit T>.
* Precometida: La bitácora contiene un registro <precommit T> pero no contiene un registro <abort T> ni un registro <commit T>
* No Lista: La bitácora no contiene un <ready T> ni un <abort T>

*¿Qué decide hacer el nuevo coordinador Cnew?*

* Si al menos un sitio tiene el estado cometido, entonces Cnew decide cometer T.
* Si al menos un sitio tiene el estado abortado, entonces Cnew decide abortar T (no es posible que exista un sitio con estado cometido y otro con estado abortado).
* Si ningún sitio esta cometido o abortado, pero existe al menos un sitio en estado precometido, Cnew reanuda el protocolo enviando un nuevo mensaje de pre-compromiso.
* En otro caso, Cnew aborta T.

El nuevo coordinar Cnew puede llegar a conocer el estado del coordinador fallado Ci:

* Si un sitio activo tiene un <commit T> en su bitácora, entonces Ci tiene que haber decidido cometer T.
* Si un sitio activo tiene un <precommit T> en su bitácora, entonces Ci alcanzo un estado de pre-compromiso, lo que implica que todos los sitios alcanzaron el estado de listos.

A diferencia del PC de 2 Fases, Cnew no comete T unilateralmente ya que puede producir un bloqueo si Cnew falla.

*¿Qué pasa si ningún sitio activo recibió el mensaje de pre-compromiso de Ci?*

* Ci ha decidido cometer antes de fallar
* Ci ha decidido abortar antes de fallar
* Ci no ha decidido sobre la suerte de T

La primera alternativa no es posible y, por lo tanto, veremos que es seguro abortar.

Si Ci decidió cometer, entonces al menos k sitios decidieron pre-cometer T y enviaron un reconocimiento (acknowledge) a Ci. Por lo tanto, existen al menos k sitios activos y al menos uno de ellos debe informar a Cnew que recibió un mensaje de pre-compromiso. Si ningún sitio activo recibió un mensaje de pre-compromiso, no es posible que Ci haya decidido cometer.

*PC de 2 Fases vs PC de 3 Fases*

Ambos protocolos buscan satisfacer el principio de atomicidad y también asumen ciertas condiciones en la red (que no se pierden mensajes y que los mensajes llegan en el mismo orden en que fueron enviados).

El PC de 3 Fases exige condiciones adicionales como, por ejemplo, que la red no pueda particionarse en dos o más grupos que no se puedan comunicar entre sí.

El PC de 2 Fases puede dejar bloqueada a una transacción, ya que el sitio destino de una transacción no se conoce hasta que el sitio fallado (coordinador) no se recupera. En cambio, El PC de 3 Fases evita ciertas situaciones de bloqueo, pero requiere un mayor tráfico de mensajes.

Intuitivamente, el PC de 2 Fases permite que un participante cometa T ni bien descubre que los participantes votaron cometer T. En cambio, en el PC de 3 Fases un participante comete T no solo cuando sabe que los participantes han votado cometer T, sino también cuando conozca que cierto número de participantes saben eso (si alguno falla, lo deberá saber cuándo se recupere).

Más allá de eso, el PC de 2 Fases es más comúnmente usado a pesar de sus bloqueos potenciales ya que la posibilidad de que estos ocurran es suficientemente baja con respecto al costo extra que insume la tercera fase en el Pc de 3 Fases.

**Seguridad**

La seguridad en BD’s involucra

* Cuestiones éticas y legales referidas a los derechos para acceder a cierta información
* Políticas a nivel gubernamental, institucional o corporativo acerca de que información no debería ser publica
* Cuestiones de seguridad sobre qué aspectos deben ser tratados a nivel de hardware, que aspectos a nivel de SO y que aspectos a nivel de DBMS.
* La necesidad de identificar niveles de seguridad y categorizar los datos y los usuarios según esas clasificaciones.

Los principales daños a una BD pueden ser:

* Perdida de Integridad: La integridad de BD’s exige que la información quede resguardada de modificaciones inadecuadas. La integridad se pierde cuando se realizan cambios desautorizados de manera accidental o intencional.
* Perdida de Disponibilidad: La disponibilidad se refiere a hacer objetos disponibles a humanos o programas que tienen un derecho legítimo de accederlos.
* Perdida de Confidencialidad: La confidencialidad se refiere a la protección de datos de una exposición incorrecta. Esto puede resultar en perdida de confidencia publica, bochorno o acciones legales contra la organización.

UN DBMS típicamente incluye un subsistema de seguridad de la base de datos y autorización, el cual es responsable de garantizar la seguridad de porciones de una base de datos contra acceso desautorizado.

Utiliza dos mecanismos de seguridad:

* Discrecionales: Conceden privilegios a usuarios, incluyendo las capacidades de acceder a archivos de datos específicos, registros o campos en un modo determinado (lectura, inserción, borrado, modificación).
* Obligatorios: Fuerzan una seguridad multi-nivel, clasificando a los datos y a los usuarios en diferentes clases. Por ejemplo, una política de seguridad es permitir a los usuarios de una clase ver solamente los datos propios o de otros usuarios de clases más bajas.

Medidas de control

Existen cuatro principales medidas para proveer seguridad de datos en bases de datos:

* Control de acceso
* Control de inferencia
* Control de flujo
* Encriptación de datos

Un problema común de seguridad en sistemas de computación es prevenir que personas no autorizadas accedan al sistema para obtener información o hacer cambios maliciosos a los datos.

Cuestiones de seguridad

Típicamente, una DBMS incluye un subsistema de seguridad y autorización de la base de datos, que es responsable de asegurar a distintas partes de la base de datos contra accesos desautorizados (mecanismos de seguridad discrecionales y obligatorios).

Los mecanismos de seguridad de un DBMS deben incluir herramientas para restringir el acceso a la base de datos como un todo: esta función se conoce como función de control de acceso y es implementada mediante la creación de cuentas de usuario y contraseñas para controlar el proceso de ingreso (login) mediante el DBMS.

El problema de seguridad asociado con bases de datos consiste en controlar el acceso a la base de datos estadística, la cual es usada para proveer información estadística o totales de valores basados en diferentes criterios. Las contramedidas al problema de seguridad de bases de datos estadísticas se denominan medidas de control de inferencia.

Otra medida de seguridad es el control de flujo, que previene que la información fluya de modo tal que alcance a usuarios desautorizados. Los canales que son rutas para que la información circule de manera que viole las políticas de seguridad de una organización se denominan canales furtivos (covert channels).

Un aspecto más de la seguridad es la encriptación de datos, la cual se utiliza para proteger datos importantes (por ejemplo, números de tarjeta de crédito) que se transmiten a través de una red de comunicación. Los datos se codifican utilizando algún algoritmo de codificación. Un usuario desautorizado que accede a datos codificados deberá tener dificultades para descifrarlos, aunque usuarios autorizados deben contar con claves o algoritmos de decodificación o desencriptado para descifrarlos.

Administrador de una BD

El administrador de una base de datos (database administrator o DBA) es la autoridad principal encargada del manejo de un sistema de BD. Las responsabilidades de un administrador incluyen:

* Conceder privilegios a usuarios que necesitan usar el sistema
* Clasificar a los usuarios y a los datos, de acuerdo con las políticas de la organización

El administrador es responsable de toda la seguridad del sistema de BD. Tiene una cuenta DBA en el DBMS, tal que a veces se denominan cuentas del sistema o superusuario. Estas cuentas proveen poderosas capacidades como:

1. Creación de cuentas

2. Concesión de privilegios

3. Revocación de privilegios

4. Asignación de niveles de seguridad

La acción 1 permite controlar el acceso, las acciones 2 y 3 son discrecionales y la acción 4 es usada para controlar la autorización obligatoria.

Protección de Acceso, Cuentas de Usuario y Auditoría de BD

Si una persona o grupo de personas necesitan acceder a un sistema de BD, deben solicitar una cuenta de usuario. El administrador creará un nuevo identificador de cuenta y contraseña para el usuario si el/ella considera que existe una legítima necesidad de acceder a la BD. El usuario debe ingresar (loguearse) en el DBMS ingresando su identificador de cuenta y su contraseña cada vez que requiera ingresar al sistema.

El sistema de BD debe mantener rastro de todas las operaciones en la BD que realiza cada usuario en cada sesión que ingresa al sistema. Para mantener un registro de todas las actualizaciones realizadas en la BD y del usuario en particular que realizó cada actualización, se mantiene una bitácora (system log) que incluye una entrada por cada operación realizada en la BD, y que puede requerir una recuperación en caso de que falle una transacción o el sistema se caiga.

Si se sospecha cualquier intento de intromisión en la BD, se realiza una auditoría de BD. Consiste en revisar la bitácora para examinar todos los accesos y todas las operaciones realizadas durante un cierto periodo de tiempo. Un registro de bitácora esencial de la BD se utiliza principalmente para propósitos de seguridad y a veces se lo conoce como rastro de auditoría (audit trail).

**Control de acceso discrecional**

El método típico de control de acceso discrecional en una BD consiste en conceder y revocar privilegios. Hay dos tipos de privilegios: el primero consiste en tipos de privilegios a nivel cuenta, donde el administrador especifica privilegios particulares para cada cuenta, independientemente de las relaciones en la BD. El segundo consiste en tipos de privilegios a nivel relación (o a nivel tabla), donde el administrador puede controlar los privilegios de acceso a cada tabla o vista en la BD.

*1) Privilegios a nivel cuenta*

Habilitan capacidades propias de la cuente y pueden incluir:

- El privilegio CREATE SCHEMA o CREATE TABLE para crea un esquema o una relación

- El privilegio CREATE VIEW para crear una vista

- El privilegio ALTER, para aplicar cambios a los esquemas tales como agregar o quitar atributos de una relación

- El privilegio DROP, para borrar relaciones o vistas

- El privilegio MODIFY, para insertar, borrar, o modificar tuplas

- El privilegio SELECT, para obtener información de la BD usando una consulta de tipo SELECT

*2) Privilegios a nivel relación*

Incluye relaciones base y relaciones virtuales (vistas). La concesión y el revocamiento de privilegios generalmente sigue un modelo de autorización para privilegios discrecionales conocido como modelo de matriz de acceso donde:

- Las filas de una matriz M representan sujetos (usuarios, cuentas, programas)

- Las columnas representan objetos (relaciones, registros, columnas, vistas, operaciones)

- Cada posición M (i, j) en la matriz representa los tipos de privilegios (read, write, update) que el sujeto i tiene sobre j

Para controlar la concesión y el revocamiento de privilegios de relación, cada relación R en una BD tiene asignado una cuenta propietario (owner account), que generalmente se usa cuando la cuenta se crea por primera vez. El propietario de una relación tiene todos los privilegios sobres esa relación. En SQL2, el administrador puede asignar un propietario a todo un esquema, creando el esquema y asociando un identificador de autorización apropiado con el esquema usando el comando CREATE SCHEMA. Aquel que disponga de una cuenta propietario puede pasar privilegios a otros usuarios de cualquiera de sus relaciones, concediendo privilegios a sus respectivas cuentas.

En SQL, pueden concederse los siguientes tipos de privilegios para cada relación R:

- SELECT (obtener o leer) privilegios sobre R:

* Da a la cuenta privilegios de obtención de datos
* En SQL, esto da a la cuenta el privilegio de usar la sentencia SELECT para obtener tuplas de R

- MODIFY (modifica) privilegios sobre R:

* Esto da a la cuenta la capacidad de modificar tuplas de R
* En SQL, esto se divide en los privilegios UPDATE, DELETE e INSERT para aplicar los correspondientes comandos SQL sobre R
* Además, los privilegios INSERT y UPDATE pueden especificar que solamente ciertos atributos pueden modificarse mediante la cuenta

- REFERENCES sobre R:

* Este privilegio da a la cuenta la capacidad de referenciar a la relación R cuando se especifican restricciones de integridad
* El privilegio también puede estar acotado a atributos específicos de R

Notar que, al crear una vista, la cuenta debe tener el privilegio SELECT sobre todas las relaciones involucradas en la definición de la vista.

El mecanismo de vistas es un importante mecanismo de autorización discrecional sobre sus propios privilegios. Por ejemplo: si el dueño A de una relación R desea que otra cuenta B pueda obtener solamente algunos atributos de R, entonces A puede crear una vista V de R que solamente incluya esos atributos y luego concederlo con la sentencia SELECT on V to B. Esto limita a B a retribuir solamente ciertas tuplas de R: se puede crear una vista V’ por medio de una consulta que selecciona solamente aquellas tuplas R que A desea permitir acceder a B.

Revocación de privilegios

En algunos casos, es deseable conceder privilegios a un usuario temporalmente. Por ejemplo: un propietario de una relación puede desear conceder el privilegio SELECT a un usuario para una tarea específica y luego revocar ese privilegio una vez que la tarea fue completada. Por lo tanto, requiere un mecanismo para revocar privilegios. En SQL, se incluye un comando REVOKE con el propósito de cancelar privilegios

Propagación de privilegios

Si el propietario A de una relación R concede un privilegio sobre R a otra cuenta B, el mismo puede darse a B con o sin el privilegio GRAN OPTION.

Si se concede el privilegio GRANT OPTION, entonces B también puede conceder privilegios sobre R a otras cuentas.

Supongamos que B recibe el privilegio GRANT OPTION de A y B concede también este privilegio sobre R a una tercera cuenta C (con GRAN OPTION), entonces se pueden propagar los privilegios sobre R a otras cuentas desconocidas para el propietario de R.

Si el propietario de una cuenta A revoca el privilegio concedido a B, todos los privilegios que B propagó basado en ese privilegio, deberían ser automáticamente revocados por el sistema.

*Ejemplo*

Supongamos que el administrador crea cuatro cuentas A1, A2, A3, A4 y desea que solamente A1 pueda crear relaciones. Entonces, el administrador debe escribir en SQL: **GRANT CREATETAB TO A1;**

En SQL2, puede obtenerse el mismo efecto si el administrador escribe el siguiente comando: **CREATE SCHEMA Ejemplo AUTHORIZATION A1;**

La cuenta de usuario A1 puede crear tablas usando el esquema llamado Ejemplo. Supongamos que A1 crea dos relaciones base Empleado y Departamento: A1 es el propietario de estas dos relaciones y, por lo tanto, de todos los privilegios sobre ellas. Supongamos que A1 desea conceder a A2 el privilegio para insertar y borrar tuplas en ambas relaciones, pero A1 no desea que A2 pueda propagar estos privilegios a otras cuentas: **GRANT INSERT, DELETE ON Empleado, Departamento TO A2.**

Supongamos que A1 desea permitir que A3 obtenga información de ambas tablas y también pueda propagar el privilegio SELECT a otras cuentas: **GRANT SELECT ON Empleado, Departamento TO A3 WITH GRANT OPTION;**

A3 puede conceder a A4 el privilegio SELECT sobre la relación empleado: **GRANT SELECT ON Empleado TO A4;**

Notar que A4 no puede propagar el privilegio SELECT porque la opción GRANT OPTION no fue concedida.

Supongamos que A1 desea revocarle a A3 el privilegio SELECT sobre la relación Empleado: **REVOKE SELECT ON Empleado FROM A3;**

Ahora el DBMS automáticamente revocará el privilegio SELECT sobre Empleado de A4 también, puesto que A3 había concedido el privilegio a A4 y A3 ya no tiene más ese privilegio

Supongamos que A1 desea retornar a A3 una capacidad limitada para utilizar SELECT sobre la relación Empleado y desea que A3 tenga la capacidad de propagar ese privilegio. La limitación consiste en que A3 pueda obtener los atributos Nombre, Nacimiento y Dirección para las tuplas con Cat=5:

CREATE VIEW A3Empleado AS

SELECT Nombre, Nacimiento, Direccion

FROM Empleado

WHERE Cat=5;

Después de que se crea la vista, A1 puede conceder A3 el privilegio SELECT sobre la vista A3Empleado: **GRANT SELECT ON A3Empleado TO A3 WITH GRANT OPTION.**

Finalmente, supongamos que A1 desea permitir que A4 actualice solamente el atributo Salario de la relación Empleado: **GRANT UPDATE ON Empleado (Salario) TO A4;**

Los privilegios UPDATE e INSERT pueden especificar que ciertos atributos pueden ser actualizados o insertados en la relación. Otros atributos tales como SELECT y DELETE no especifican ningún tipo de atributo

Límites en la propagación de privilegios

Se han desarrollado varias técnicas para limitar la propagación de privilegios, aunque las mismas no han sido implementadas en la mayoría de los DBMS’s y en general, tampoco son parte de SQL.

Limitar la propagación horizontal a un número i significa que una cuenta B a la que se le concedió el privilegio GRANT OPTION, puede propagar dicho privilegio a lo sumo a otras i cuentas.

Limitar la propagación vertical es más complicado; esto limita la profundidad de la concesión de privilegios

**Control de acceso obligatorio y basado en roles para seguridad multinivel**

Las técnicas de control de acceso discrecionales de conceder y revocar privilegios en relaciones ha sido el principal mecanismo de seguridad en sistemas de BD’s.

Este es un método todo o nada, un usuario tiene o no cierto privilegio. En muchas aplicaciones, es necesario contar con políticas adicionales de seguridad que clasifican a los datos y a los usuarios en clases de seguridad: esta propuesta es un control de acceso obligatorio que típicamente se lo puede combinar con mecanismos de control discrecionales.

Típicamente las clases de seguridad se clasifican en supersecreto (T, top secret), secreto (S, secret), confidenciales (C, confidential) y no clasificadas (U: unclassified), donde T es el nivel más alto y U el más bajo.

El modelo comúnmente usado para seguridad multinivel, conocido como Modelo Bell-LaPadula, clasifica cada sujeto (usuario, cuenta, programa) y cada objeto (relación, tupla, columna, vista, operación) en una de las clasificaciones de seguridad: T, S, C o U:

Clearance (clasificación) de un sujeto S como class(S) y Classification (clasificación) de un objeto O como class(O)

Deben respetarse dos restricciones en el acceso a los datos basado en clasificaciones sujeto/objeto:

- Propiedad de Seguridad Simple: un sujeto S no puede leer determinado objeto O a menos que class(S) > class(O)

- Propiedad Estrella: un sujeto S no puede escribir determinado objeto O a menos que class(S) < class(O)

Para incorporar nociones de seguridad multinivel en un modelo de base de datos relacional, es común considerar a los valores de atributos de las tuplas como objetos de datos. Por lo tanto, cada atributo está asociado con un atributo de clasificación C en el esquema, y cada valor de un atributo en una tupla está asociado con la correspondiente clasificación de seguridad.

Además, en algunos modelos, se agrega un atributo de clasificación de tuplas TC a los atributos de la relación para proveer una clasificación para cada tupla como un todo. Por lo tanto, un esquema de relación multinivel R con n atributos podría representarse como: R (A1, C1, A2, C2, …, An, Cn, TC) donde cada Ci representa el atributo de clasificación asociado con el atributo Ai.

El valor del atributo TC en cada tupla t (que es el más alto de todos los valores de clasificación en t) provee una clasificación general para la tupla misma, donde cada Ci provee una clasificación de seguridad más fina para cada valor de atributo en la tupla.

La clave aparente de una relación multinivel es el conjunto de atributos que debería haber formado la llave primaria en relación regular (mononivel no multinivel)

Una relación multinivel podrá contener diferentes datos para sujetos (usuarios) con diferentes niveles de clasificación (clearance). En algunos casos, es posible almacenar una única tupla en la relación en un nivel de clasificación más alto y producir las correspondientes tuplas en una clasificación de más bajo nivel mediante un proceso conocido como filtrado. En otros casos, es necesario almacenar dos o más tuplas en diferentes niveles de clasificación con los mismos valores para la clave aparente. Esto lleva a un concepto de poli-instanciación donde varias tuplas pueden tener el mismo valor para una clave aparente, pero tienen diferentes valores de atributos para usuarios en diferentes niveles de clasificación.

En general, la regla de integridad de entidad para relaciones multinivel establece que el conjunto de los atributos que son miembros de la clave aparente no debe ser nulo y debe tener la misma clasificación de seguridad dentro de cada tupla individual. Además, todos los otros valores de atributos en la tupla deben tener una clasificación de seguridad mayor o igual que el de la clave aparente. Esta restricción asegura que un usuario pueda ver la clave si el usuario tiene permiso de ver cualquier parte de la tupla completa.

Otras reglas de integridad son las denominadas integridad nula e integridad interinstancia. Informalmente, estas reglas aseguran que si un valor de tupla en algún nivel de seguridad puede ser filtrado (derivado) de una tupla de clasificación más alta, entonces es suficiente almacenar la tupla clasificada más alta en la relación multinivel.

*Control discrecional vs Control obligatorio*

Las políticas de control de acceso discrecional están caracterizadas por un alto grado de flexibilidad, que los hace apropiados para una gran variedad de dominios de aplicación. Su principal desventaja es su vulnerabilidad ante ataques maliciosos, tales como troyanos embebidos en programas de aplicación.

En contraste, las políticas obligatorias aseguran un alto grado de protección de manera tal de evitar cualquier flujo ilegal de información. Las políticas obligatorias, tienen la desventaja de ser demasiado rígidas y son solamente aplicables en entornos limitados. En muchas situaciones prácticas, las políticas discrecionales son preferidas porque ofrecen una mejor relación costo-beneficio entre seguridad y aplicabilidad

*Control de acceso basado en roles*

La técnica de control de acceso basado en roles emergió rápidamente en 1990 como una tecnología para manejar e imponer seguridad en sistemas de empresas en gran escala. Su idea básica es que los permisos están asociados con roles, y que los usuarios tienen asignados roles apropiados. Los roles pueden crearse y eliminarse usando CREATE ROLE y DESTROY ROLE. La conexión y el revocamiento de privilegios de los roles puede realizarse con los comandos GRANT y REVOKE.

Estas técnicas parecen una alternativa viable a los métodos de control de acceso discrecionales y obligatorios: asegura que solamente los usuarios autorizados puedan acceder a ciertos datos y recursos. Muchos DBMS’s admiten el concepto de roles, donde se pueden asignar privilegios a los roles. El uso de jerarquías de roles es una manera natural de organizar roles para reflejar las líneas de organización de autoridad y responsabilidad.

Otra importante consideración en sistemas de control de acceso basado en roles es que pueden definirse restricciones temporales sobre los roles, tales como el tiempo y la duración de las activaciones de los roles, el tiempo de disparo de un rol por una activación de alguna otra regla, etc.

En general, los sistemas de control de acceso basados en roles son uno de los objetivos de alcanzar por aplicaciones web. En contrapartida, los métodos discrecionales y obligatorios carecen de capacidades para soportar requerimientos de seguridad emergentes en empresas y aplicaciones web.

*Políticas de control para e-commerce y web*

Los entornos para comercio electrónico (e-commerce) requieren elaborar políticas que van más allá de los DBMS’s tradicionales. En un comercio electrónico, los recursos a ser protegidos no son solamente datos tradicionales sino también conocimiento y experiencia. Los mecanismos de control de acceso deberían ser lo suficientemente flexibles para soportar un amplio espectro de protección de objetos. Un requerimiento relacionado es el soporte para control de acceso basado en contenido.

Otro requerimiento es la heterogeneidad de los sujetos, que requiere políticas de control de acceso basadas en las características y clasificaciones de los usuarios. Una posible solución, teniendo en cuenta los perfiles de usuarios en la formulación de políticas de control de acceso, es soportar la noción de credenciales. Una credencial es un conjunto de propiedades referidas a un usuario que son relevantes a los propósitos de seguridad (por ejemplo, edad y posición en la organización). Se cree que el lenguaje XML puede jugar un rol clave en el control de acceso para aplicaciones e-commerce.

**Seguridad en BD’s Estadísticas**

Las BD’s Estadísticas son principalmente usadas para obtener estadísticas en diferentes poblaciones. La BD puede contener datos confidenciales sobre individuos, que deberían quedar protegidos del acceso de usuarios. Los usuarios tienen la posibilidad de obtener información estadística sobre poblaciones tales como promedios, sumas, cantidades, máximos, mínimos y desviaciones estándar.

Una población es un conjunto de tuplas de una relación (tabla) que satisfacen cierta condición de selección. Las consultas estadísticas involucran aplicar funciones estadísticas a una población de tuplas.

Por ejemplo, podemos desear obtener el número total de individuos en una población o el ingreso promedio de la población. Sin embargo, no está permitido obtener datos individuales, tales como el ingreso de una persona específica.

Las técnicas de seguridad en BD’s Estadísticas deben prohibir la obtención de datos individuales. Esto puede lograrse prohibiendo consultas que obtengan valores de los atributos, pero permitiendo consultas que involucren funciones estadísticas agregadas tales como COUNT, SUM, MIN, MAX, AVERAGE, SATANDAR DESVIATION, etc. Estas son consultas estadísticas

Es responsabilidad del DBMS asegurar la confidencialidad de la información acerca de los individuos y a la vez proveer (a los usuarios) resúmenes estadísticos de los datos sobre los individuos de la BD. En algunos casos es posible inferir los valores de tuplas individuales a partir de secuencias de consultas estadísticas. Esto es particularmente verdadero cuando las condiciones resultan en una población compuesta por pocas tuplas.

**Control de flujo**

El control de flujo regula la distribución o el flujo de información entre los objetos accesibles. Un flujo entre un objeto X y un objeto Y sucede cuando un programa lee valores de X y escribe valores en Y. El control de flujo chequea que la información contenida en algunos objetos no fluya explícita o implícitamente en objetos menos protegidos.

Una política de flujo especifica los canales en los cuales la información puede circular. La política de flujo más simple especifica solamente dos clases de información: confidencial y no confidencial. Dicha política permite todos los flujos excepto aquellos de clase confidencial a clase no confidencial

Canales furtivos

Un canal furtivo permite transferir información que viole la seguridad y la política de seguridad. Permite que la información circule de altos niveles de clasificación a bajos niveles de clasificación de manera inadecuada. Pueden ser clasificados en dos grandes categorías:

- Canales de Almacenamiento, los cuales no requieren ninguna sincronización temporal, y la información es transportada accediendo a la información del sistema o, en caso contrario, siendo inaccesible al usuario.

- Canales de Timing, los cuales permiten que la información se transporte por la ocurrencia de eventos o la activación de procesos

Algunos expertos en seguridad creen que una forma de evitar canales furtivos es que los programadores no tengan realmente acceso a aquellos datos sensitivos que un programa se supone procesa después de que el mismo fue puesto en funcionamiento

**Encriptación**

La encriptación es un medio de mantener datos seguros en un entorno inseguro. Consiste en aplicar un algoritmo de encriptación de datos usando alguna clave de encriptación pre-especificada. Los datos resultantes tienen que ser desencriptados usando una clave de desencriptado para recuperar los datos originales.

Encriptación estándar y avanzada

La encriptación de datos estándar (DES: Data Encryption Standard) es un sistema desarrollado por el gobierno de Estados Unidos para el uso del público en general. Ha sido ampliamente aceptado como un estándar criptográfico tanto en Estados Unidos como en otros países. DES puede proveer encriptación end-to-end en el canal entre el emisor A y el receptor B.

El algoritmo DES es una cuidadosa y compleja combinación de dos de los bloques fundamentales de la encriptación: substitución y permutación (transposición). Se basa en la aplicación repetida de estas dos técnicas en un total de 16 ciclos. El texto original del mensaje se encripta en bloques de 64 bits. Después de cuestionar la seguridad del DES, el Instituto Nacional de Estándares (NIST), introdujo los estándares avanzados de encriptación (AES: Advanced Encryption Standars), donde este algoritmo tiene un bloque de 128 bits y así es más difícil de vulnerar.

Clave pública

La encriptación de clave pública está basada en funciones matemáticas y no en operaciones sobre patrones de bits. Se usan dos claves separadas (la tradicional utiliza una única clave). El uso de dos claves puede tener profundas consecuencias en áreas de confidencialidad, distribución de claves y autentificación.

Las dos claves usadas se conocen como clave pública y clave privada. Un esquema de clave pública tiene las siguientes componentes: texto original (plaintext), algoritmo de encriptado, claves pública y privada, texto codificado, algoritmo de desencriptado.

La clave pública es conocida por todos y la clave privada es solamente conocida por su dueño. Básicamente el método de encriptación consiste en una clave de encriptado y otra clave diferente de desencriptado.

Los pasos del algoritmo son:

1) Cada usuario genera un par de claves a ser usadas para el encriptado y desencriptado del mensaje

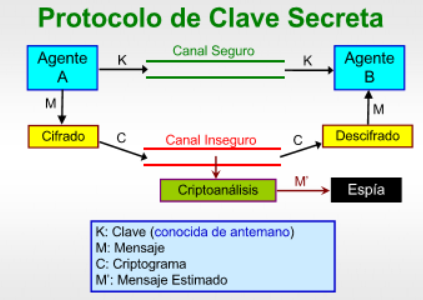
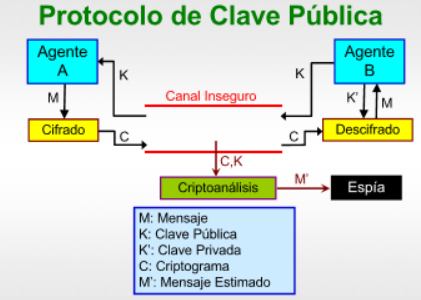
2) Cada usuario coloca una de las dos claves en el registro público y otra en el archivo accesible. La clave privada se mantiene en secreto al resto

3) Si el emisor desea enviar un mensaje privado a un receptor, el emisor encripta el mensaje usando la clave pública del receptor

4) Cuando el receptor recibe el mensaje, lo desencripta usando su clave privada

5) Ningún otro receptor puede desencriptar el mensaje porque solamente el receptor conoce su clave privada

El algoritmo de Encriptado RSA fue uno de los primeros esquemas de clave pública y fue introducido en 1978 por Ron Rivest (R), Adi Shamir (S) y Len Adleman (A) en el MIT. Incorpora resultados de la teoría de números, tales como la dificultad de encontrar factores primos grandes de un número grande. Opera con aritmética modular (fuera del alcance del curso).

Protocolo de clave pública

Los protocolos de clave pública fueron posibles a partir del estudio sistemático de la algoritmia, a partir de 1970. Rivest, Shamir y Adleman presentaron en 1978 el sistema criptográfico RSA.

- A es el emisor del mensaje

- El receptor del mensaje (B) elige dos números primos p y q de 100 dígitos (cuanto más grandes, más difícil de quebrar el cifrado) y calcula z = pq asumiendo que puede hallar p y q de manera eficiente

- Una vez obtenido z, es muy costoso factorearlo

- B también elige aleatoriamente un número n tal que 1 < n < z-1 que no tenga factores comunes con el producto (p-1) (q-1)

- Si n no cumple tal propiedad, debe elegir un nuevo n, aunque cada intento tiene una alta probabilidad de éxito

Existe un algoritmo eficiente (basado en el Algoritmo de Euclides) que, dado cualquier n, comprueba que verifique tal propiedad y al mismo tiempo calcula el único s, 1 < s < z-1 tal que se verifica que ns mod (p-1) (q-1) = 1.

Lo interesante de los números p y q es que se puede probar que si 1 < a < z-1 entonces vale que ax mod z = a si x mod (p-1) (q-1) =1.

El agente A envía el mensaje m por ejemplo codificándolo en formato ASCII y asegurando que 1 < m < z-1. Si no se cumple esa restricción, divide el mensaje en pates de tamaño apropiado. La clave pública está formada por z y n y la clave secreta (solo conocida por B) por s.

El remitente A del mensaje m, con 1 < m < z-1 debe calcular c = mn mod z para encriptarlo

Function ExpoMod (a, n, z) {Calcula an mod z)

r 🡨 a mod z

if n = 1

then return r

else if n es par

then

p 🡨 a\*a mod z

ExpoMod (p, n/2, z)

else

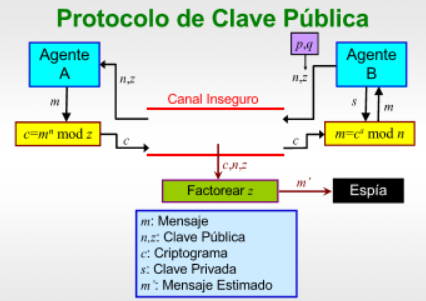
return (a\*ExpoMod (a, n-1, z)) mod z

Cuando B recibe m, a partir de la clave secreta debe desencriptarlo calculando:

cs mod z = (mn mod z) s mod z

cs mod z = (mn)s mod z = mns mod z = m

Es necesario una implementación eficiente de la exponenciación modular xy mod z, y para ello existe el algoritmo ExpoMod. El espía, con conocimiento de c, n y z, solo puede factorear z en pq para hallar s y luego calcular m. El problema es que no se puede factorear de manera eficiente números tan grandes como z.



**Firmas digitales**

Una firma digital es un ejemplo de usar técnicas de encriptación que proveen servicios de autentificación en aplicaciones de comercio electrónico. Es una forma de asociar una marca única a un individuo dentro de un cuerpo de texto. La marca debería ser inolvidable, de modo que otros puedan chequear que la firma viene del origen esperado.

Una firma digital consiste en usar cadenas de símbolos. Las firmas digitales deben ser diferentes para cada uno y cada firma digital es una función del mensaje que está siendo firmado, junto con la estampilla de tiempo.

Las técnicas de clave pública se utilizan para crear firmas digitales

**Sistemas de Soporte de Decisión (DSS)**

Los Sistemas de Soporte de Decisión (DSS’s) se usan para tomar decisiones de negocios, muchas veces basados en información recolectada en línea por sistemas de procesamiento de transacciones. Ejemplos de decisiones de negocios, como qué productos comprar, qué productos ofertar, a quien enviar promociones, etc. Ejemplos de datos usados para tomar decisiones, como detalles de transacciones de ventas de comercio, perfiles de clientes (ingresos, edad, sexo, etc.).

Análisis de Datos: las tareas se pueden realizar con herramientas especializadas y extensiones SQL. Por ejemplo, para cada categoría de producto y cada región, encontrar las ventas totales en el último trimestre y compararlas con las mismas del trimestre anterior; o puede ser para la categoría del producto y cada categoría de cliente

Análisis Estadístico: se usan paquetes (por ejemplo, S++) que son “interfaceados” con bases de datos: el análisis estadístico es una disciplina en sí, no cubierta en esta presentación.

Minería de Datos (Data Mining): busca descubrir conocimiento automáticamente en la forma de reglas estadísticas y patrones utilizando grandes bases de datos. Por ejemplo, los clientes que compran smartphones o celulares de última generación son candidatos a tentarse con lo último en tecnología

Data Warehouse: almacena en un único sitio usando un esquema unificado, información recopilada de múltiples fuentes. Esto es importante para grandes comercios que generan datos de múltiples secciones y, más aún, de diferentes ciudades. Los datos pueden ser comprados externamente.

**Procesamiento Analítico en Línea (OLAP)**

Es un análisis interactivo de datos, permitiendo que los datos sean resumidos y vistos de diferentes maneras casi inmediatamente (con una demora insignificante). Los datos pueden modelarse como atributos de dimensión o atributos de medidas, se denominan datos multidimensionales.

Atributos de medidas: miden algún valor, pueden ser valores agregados (totales, porcentajes)

Atributos de dimensión: definir las dimensiones en las cuales se miden los atributos (nombre, color, tamaño, etc.)

Naturaleza del Análisis OLAP

-Agregación: se calculan las ventas totales, porcentajes, etc.

-Comparación: se comparan presupuestos con gastos

-Ranking: se obtienen los productos mejores (top ten), se calculan movimientos trimestrales, etc.

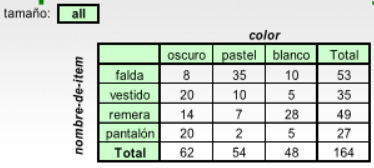
Se accede a datos detallados y datos agregados

Existen criterios de especificación complejos

Se utiliza visualización

Se necesita respuesta interactiva para consultas agregadas

*Tabla cruzada*

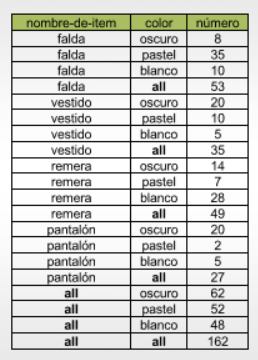
 También conocida como tabla-pivote.

Los valores para un atributo de dimensión forman los encabezados de la fila

Los valores para el otro atributo de dimensión forman los encabezados de columna

Los otros atributos de dimensión se listan arriba

Los valores en celdas individuales son (agregadas de) los valores de los atributos de dimensión que especifican la celda

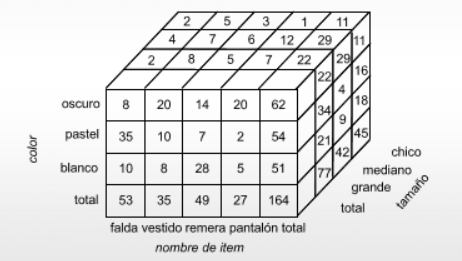


*Representación relacional*

Las tablas cruzadas pueden representarse como relaciones. Usamos all para representar agregados

*Cubo de Datos*

Un cubo de datos es una generalización multidimensional de una tabla cruzada. Puede tener n dimensiones y las tablas cruzadas pueden ser usadas como vistas de cubo de datos.



*OLAP: Procesamiento Analítico Online*

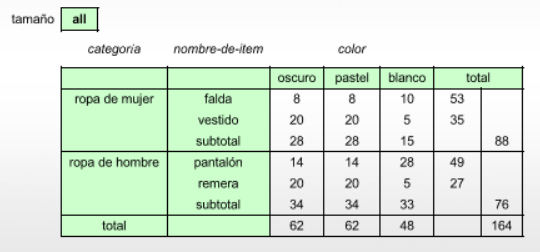
- Pivoting: cambiar las dimensiones usadas en una tabla cruzada

- Slicing: crear una tabla cruzada solamente para valores fijos (selección en algebra relacional). A veces se denomina dicing, particularmente cuando se fijan valores para múltiples dimensiones (proyección en álgebra relacional)

- Rollup: moverse desde una granularidad más fina a una granularidad más gruesa por medio de agregación (se resumen datos)

- Drill down: moverse desde una granularidad más gruesa a una granularidad más fina (se obtiene más detalle)

*Tabla cruzada con jerarquía*

 La jerarquía en atributo de dimensiones permite que las dimensiones se vean en diferente nivel de detalle. Por ejemplo, la dimensión Fecha-Hora, puede usarse para agregar por hora del día, día, día de semana, mes, trimestre, año (Jerarquía de tiempo); o una Ciudad, en Partido, Provincia, País, Continente (Jerarquía de Lugar).

Entonces la tabla cruzada puede extenderse fácilmente con jerarquías, descendiendo o ascendiendo en una jerarquía

Puntos fuertes de la OLAP

- Es una herramienta de visualización potente

- Provee tiempos de respuesta rápidos e interactivos

- Es bueno para analizar series de tiempo

- Puede ser útil para encontrar clusters

- Muchos productos ofrecen herramientas OLAP

Implementación OLAP

* Los primeros sistemas OLAP usaban arreglos multidimensionales en memoria para almacenar cubos de datos y eran conocidos como Sistemas Multidimentional OLAP (MOLAP)
* Las implementaciones OLAP que usaban solamente características de bases de datos relacionales se denominaban Sistemas Relational Olap (ROLAP)
* Los sistemas híbridos, que almacenan cierta información resumida en memoria, así como datos y otra información en una base de datos relacional, se denominan Sistemas Hybrid OLAP (HOLAP)

Los primeros sistemas OLAP precomputaban todos los posibles agregados para proveer respuestas online. En dichos sistemas, los requerimientos de espacio y de tiempo podían ser casi prohibitivos (2^n combinaciones de group by). Basta con precomputar algunos agregados y computar otro por demanda a partir de agregados precomputados. Por ejemplo, podemos computar un agregado sobre (nombre-item, color) a partir del agregado sobre (nombre-item, color, tamaño). Sirve para casi todos los agregados excepto algunos que no admiten descomposición como el elemento mediano. Es más económico computar agregados de agregados que sobre la relación original.

Existen varias optimizaciones para computar múltiples agregados:

* Se puede computar un agregado sobre (nombre-item, color) de un agregado sobre (nombre-item, color, tamaño)
* Se pueden computar agregados sobre (nombre-item, color, tamaño), (nombre-item, color) y (nombre-item) usando una única sentencia de ordenamiento de la base de datos

Los datos en un cubo de datos no pueden generarse mediante una simple consulta SQL usando el constructor group by ya que los agregados se computan mediante diferentes agrupamientos de los atributos de dimensión.

**Operaciones en SQL**

1. Agregación extendida en SQL

El constructor cube computa la unión de diferentes group by en cada subconjunto de los atributos especificados. Por ejemplo, la consulta **select nombre-item, color, tamaño, sum(numero) from ventas group by cube (nombre-item, color tamaño)** computa la unión de 8 diferentes grupos de la relación ventas: {(nombre-item, color, tamaño), (nombre-item, color), (nombre-item, tamaño), (color, tamaño), (nombre-item), (color), (tamaño), ()} donde () denota una lista group by vacía. Para cada grupo el resultado contiene el valor nulo para atributos no presentes en el grupo.

La representación relacional de la tabla cruzada mostrada anteriormente, pero con el valor null en lugar de all, se computa con la consulta: **select nombre-item, color, sum(numero) from ventas group by cube (nombre-item, color).**

La función grouping () puede aplicarse sobre un atributo, donde retorna 1 si el valor es un valor null representado a all y 0 en otro caso:

**Select nombre-item, color, tamaño, sum(numero),**

**Grouping(nombre-item) as flag-nombre-item**

**Grouping(color) as flag-color**

**Grouping(tamaño) as flag tamaño**

**From ventas**

**Group by cube (nombre-item, color, tamaño)**

La salida es la misma que la versión sin el grouping (), pero con 3 columnas extras para los flags (que contienen un 1 si el campo correspondiente es un null representado en all). En lugar de usar etiquetas para indicar que los valores null representan all, podemos reemplazar el valor null por un valor a nuestra elección: **decode(grouping(nombre-item), 1, ‘all’, nombre-item).** Esta expresión retorna el valor de ‘all’ si el valor de nombre-item es un valor null representado a all, y retorna el valor efectivo de nombre-item en otro caso

El constructor rollup permite unir cada prefijo de una lista de atributos. Por ejemplo:

**Select nombre-item, color, tamaño, sum(numero)**

**From ventas**

**Group by rollup (nombre-item, color, tamaño)**

Esta operación une 4 grupos: (nombre-item, color, tamaño), (nombre-item, color), (nombre-item), (). Puede usarse para generar agregados en varios niveles de jerarquía. Supongamos que la tabla categoríaitem (nombre-item, categoría) da la categoría de cada ítem

**Select nombre-item, categoría, sum(numero)**

**From ventas, categoríaitem**

**Where ventas.nombre-item=categoriaitem.nombre-item**

**Group by rollup (categoría, nombre-item)**

Esta consulta generaría un resumen jerárquico ordenado por nombre de ítem y por categoría

Se pueden utilizar múltiples rollups y cubes en una única sentencia group by. Cada uno de ellos genera un conjunto de grupos por listas, y realiza un producto cartesiano entre ellos para generar un grupo de listas más completos:

**Select nombre-item, color, tamaño, sum(numero)**

**From ventas**

**Group by rollup(nombre-item), rollup (color, tamaño)**

Esta consulta genera los grupos:

{nombre-item, ()} x [ (color, tamaño), (color), ()} = {(nombre-item, color, tamaño), (nombre-item, color), (nombre-item), (color, tamaño), (color), ()}

1. Ranking

Se hace en conjunción con una especificación de orden. Por ejemplo, encontrar el estudiante con promedio más alto, encontrar los clientes que más consumen, etc. Dada una relación notas-estudiante (estudiante-id, notas) se puede encontrar el ranking (según sus notas) de cada estudiante: **select estudiante-id, rank () over (order by notas desc) as e-rank from notas-estudiante**

Si queremos ordenarlos por nota, es necesario usar una clausula adicional order by: **select estudiante-id, rank () over (order by notas desc) as e-rank from notas-estudiante order by e-rank**

El ranking puede dejar espacios o gaps: por ejemplo, si 2 estudiantes tienen el mismo ranking tope(digamos1), el próximo ranking es 3. La cláusula dense\_rank evita que existan espacios, generando rankings subsecuentes.

El ranking puede hacerse con datos particionados. Por ejemplo, encontrar el ranking de los estudiantes de cada sección:

**Select estudiante-id, sección,**

**Rank () over (partition by selection order by notas desc) as sec-rank**

**From notas-estudiante, sección-estudiante**

**Where notas-estudiante.estudiante-id=sección-estudiante.estudiante-id**

**Order by sección, sec-rank**

Una única sentencia select puede especificar más de un ranking: un ranking dentro de un ranking más general. Esto ocurre normalmente cuando el ranking se aplica después de aplicar la sentencia group-by

* percent\_rank de una tupla da el ranking de la tupla como una fracción. Si el ranking de la tupla es r (entre n tuplas), su percent rank es: (r-1) (n-1)
* cume\_dist de una tupla da una distribución acumulativa. Se define como p/n donde p es el número de tuplas en la partición con ordenamiento precedente o igual al valor de orden de una tupla, y n es el número de tuplas en partición
* row\_number ordena las filas y da a cada una de ellas un número único correspondiente a su posición ordenada; filas diferentes con el mismo valor de ordenamiento podrían tener row numbers diferentes
* nitle(n) toma las tuplas en cada partición en el orden especificado y las divide en n buckets con igual número de tuplas. Si el número de tuplas no es divisible por n, cada bucket puede diferir en, a lo sumo, un elemento. Tuplas con el mismo ranking pueden caer en buckets diferentes.

La presencia de valores nulos puede complicar la definición de rankings ya que no queda claro que lugar ocupan en un orden. SQL:1999 permite al usuario especificar donde son rankeados los valores nulos con las cláusulas nulls first o nulls last.

1. Windowing

Un ejemplo de una consulta ventana (window query) es, dadas las ventas de cada día, encontrar el promedio de ventas de determinada semana; otro ejemplo puede ser determinar el promedio de dinero que existió en una caja de ahorros durante un mes para calcular el interés que se le acreditará a la misma.

Dada la relación ventas (fecha, valor):

**Select fecha, sum(valor) over**

**(order by fecha between rows 1 preceding and 1 following)**

**From ventas**

Esta consulta genera las ventas de un día, así como también del día anterior y del día siguiente. También se puede especificar otra ventana dando, por ejemplo, 10 días previos y 10 días posteriores.

* Between rows unbounded preceding and current: obtener datos de todas las filas anteriores y la actual
* Rows unbounded preceding: obtener datos de todas las filas anteriores excluyendo la actual
* Range between 10 preceding and current row: obtener datos de 10 filas anteriores y la actual
* Range Interval 10 day preceding: obtener datos de 10 filas anteriores excluyendo la actual

**Data Warehousing**

Las fuentes de datos a veces almacenan solamente datos actuales, no datos históricos. La toma de decisiones en grandes compañías requiere una visión global de los datos organizacionales, incluyendo los datos históricos. Un data warehouse es un repositorio de información recopilada de diferentes fuentes, almacenada bajo un esquema unificado en un sitio. Esto simplifica las consultas, permitiendo el estudio de tendencias históricas y separa la toma de decisiones de los sistemas de procesamiento de transacciones.

Definido de muchas formas, pero no de una forma rigurosa, es una colección de datos orientada a temas, integrada, variante en el tiempo y no volátil, destinada al proceso de toma de decisiones. Data Warehousing es el proceso de construir y usar “data warehouses”. Un depósito de datos único, completo y consistente, obtenido de una variedad de fuentes diferentes que fueron dejados disponibles por usuarios finales de modo tal que puedan ser extendidos y usados en un contexto comercial.

1. Data Warehouse “orientado a temas”

- Organizada en torno a temas centrales, tales como cliente, producto, ventas.

- Enfocada en el modelamiento y análisis de datos para la toma de decisiones, no en operaciones diarias o procesamiento de transacciones

- Proveen una visión simple y concisa en relación a cuestiones particulares excluyendo datos que no son útiles en el proceso de apoyo de decisiones

1. Data Warehouse “integrado”

- Construidas integrando múltiples fuentes de datos heterogéneas. Por ejemplo: BD relacionales, archivos “flat”, registros de transacciones on-line, etc.

- Se aplican técnicas de “data cleaning” y “data integration”

- Aseguran la consistencia en convenciones de nombres, codificación de estructuras, atributos de medidas, etc. Entre diferentes fuentes de datos. Por ejemplo, para el precio de un hotel unificar la moneda, el impuesto, si cubre el desayuno o no. Cuando los datos se mueven al “warehouse”, se los convierte apropiadamente

1. Data Warehouse “variante en tiempo”

- El horizonte de tiempo para data warehouses es significativamente mayor que el de los sistemas operacionales convencionales

- Cada estructura “clave” en el data warehouse

1. Data Warehouse “no volátil”

- Un almacenamiento físicamente separado de los datos, transformado a partir del entorno operacional

- La actualización operacional de datos no tiene lugar en un entorno de data warehouse. Es por ello que no se requieren mecanismos para procesamiento de transacciones, recuperación ante fallos ni control de concurrencia. Requiere solo dos operaciones de acceso a los datos: carga inicial y acceso.

Como se construye un data warehouse

1. Los datos deben extraerse de múltiples fuentes heterogéneas (bdd, hojas de cálculos, datos de entorno, etc.)
2. Los datos deben ser uniformizados y deben eliminarse inconsistencias (si las hubiera) de las múltiples fuentes
3. Los datos deben ser “limpiados” para asegurar validez. Data cleaning es un proceso complejo y el más demandante en este proceso
4. Los datos deben ajustarse al modelo de datos del Data Warehouse
5. Los datos deben ser cargados en el Data Warehouse. Una vez cargados, deben ser actualizados en periodos de tiempo que difieren de los tiempos de actualización de una base de datos tradicional