# Transacciones

M. Andrea Rodríguez-Tastets

Universidad de Concepción,Chile
www.inf.udec.cl\ ∼andrea
andrea@udec.cl

II Semestre - 2014

## Objetivos de la Unidad

Entender el concepto de transacciones.

## Transacciones

- Una transacción es una unidad lógica de procesamiento de la base de datos que incluye una o más operaciones de acceso a la base de datos
- Se distinguien transacciones solo de lectura a la base de datos o transacciones de actualización, es decir, leer, cambiar y escribir un elementos a la base de datos.
- Las transacciones pueden delimitarse con sentencias explícitas begin transaction and end transaction

## Transacciones (cont.)

- Asumimos que una base de datos es modelada en forma simplificada como una colección de elementos de datos con nombre (registro, bloque en disco, páginas..)
- Las operaciones básicas son: Leer_elemento(X) y Escribir_elemento(X)

## Transacciones (cont.)

Leer_elemento(X):

- Encontrar la dirección del bloque de disco que contiene el elemento X.
- Copiar ese bloque de disco en un almacenamiento intermedio (buffer) dentro de la memoria principal (si es que ese bloque no está ya en algún buffer de la memoria principal).
- Copiar el elemento X del almacenamiento intermedio en la variable de programa X.

## Transacciones (cont.)

Escribir_elemento(X):

- Encontrar la dirección del bloque de disco que contiene el elemento X.
- Copiar ese bloque de disco en un almacenamiento intermedio (buffer) dentro de la memoria principal (si es que ese bloque no está ya en algún buffer de la memoria principal).
- Copiar el elemento X desde la variable del programa X en el lugar correcto dentro del búfer.
- Almacenar el bloque actualizado desde el búfer al disco.

## Concurrencia

- En sistemas multiusuario, es necesario un mecanismo para controlar la concurrencia. Se pueden producir inconsistencias importantes derivadas del acceso concurrente (3 problemas).
- Las transacciones de los usuarios se podrían ejecutar de manera concurrente y podrían acceder y actualizar los mismos elementos de la BD.

## Control de concurrencia

- Problemas: (i) actualización perdida, (ii) actualización temporal (lectura sucia) and (iii) resumen incorrecto (iv) lectura no repetible (v) fantasmas.
- Ejemplo: Sistema de BD de reservas en una línea área. Transacción T1: transfiere N reservas de un vuelo, cuyo número de asientos reservados está almacenado en el elemento de la BD llamado X, a otro vuelo, cuyo número de asientos reservados está almacenado en el elemento de la BD llamado Y. Transacción T2: agrega un número de asientos al vuelo con asientos reservados guardados en elemento X.

## Actualización perdida

- Esto ocurre cuando las transacciones que tienen acceso a los mismos elementos de la BD tienen sus operaciones intercaladas de modo que hacen incorrecto el valor de algún elemento.
- T1 y T2 se introducen al mismo tiempo y sus operaciones se intercalan.

| T1 | T2 |
|---|---|
| R(X) | |
| X = X - N | |
| | R(X) |
| | X = X + N |
| W(X) | |
| R(Y) | |
| | W(X) |
| Y=Y+N | |
| W(Y) | |

## Actualización perdida (cont.)

- El valor final del elemento X es incorrecto, porque T2 lee el valor de X ANTES de que T1 lo modifique en la BD, con lo que se pierde el valor actualizado que resulta de T1.
- Si X=80 al principio, N=5 (T1 transfiere 5 reservas de asientos del vuelo que corresponde a X al vuelo que corresponde a Y) y M=4 (T2 reserva 4 asientos en X), el resultado final debera ser X=79, pero es X=84, porque la actualizacin de T1 que eliminó 5 asientos de X se ha perdido.

## Actualización temporal

- Esto ocurre cuando una transacción actualiza un elemento de la BD y luego la transacción falla por alguna razón. Otra transacción tiene acceso al elemento actualizado antes de que se restaure a su valor original.
- T1 actualiza el elemento X y después falla antes de completarse, así que el sistema debe cambiar X otra vez a su valor original.
- Antes de que pueda hacerlo, la transacción T2 lee el valor "temporal" de X, que no se grabará permanentemente en la BD debido al fallo de T1.
- El valor que T2 lee de X se llama dato sucio, porque fue creado por una transacción que no se ha completado ni confirmado todaví a.

## Actualización temporal (cont.)

| T1 | T2 |
|---|---|
| R(X) | |
| X = X - N | |
| W(X) | |
| | R(X) |
| | X = X + N |
| W(X) | |
| R(Y) | |

T1 falla y debe restaurar X a su antiguo valor, mientras tanto T2 ha leído el valor temporal incorrecto de X.

## Resumen incorrecto

Si una transacción está calculando una función agregada de resumen sobre varios registros mientras otras transacciones están actualizando algunos de ellos, puede ser que la función agregada calcule algunos valores antes de que se actualicen y otros después de actualizarse.

| T1 | T2 |
|---|---|
| | Sum = 0 |
| | R(A) |
| | Sum = Sum+A |
| R(X) | |
| X = X-N | |
| W(X) | |
| | R(X) |
| | Sum= Sum+X |
| | R(Y) |
| | Sum = Sum+Y |
| R(Y) | |
| Y=Y+N | |
| W(Y) | |

T2 lee X después de restarle N a X y lee Y antes de sumárle N, así que el resultado es un resumen incorrecto (discrepancia de N)

## Lectura no repetible

| T1 | T2 |
|---|---|
| R(X) | |
| | R(X) |
| | W(X) |
| R(X) | |

Fantasmas: una lectura de una tupla que no existía antes.
Muchas más problemas...

## Manejador de Transacciones

Siempre que se introduce una transacción a un SGBD para ejecutarla, el sistema tiene que asegurarse de que:

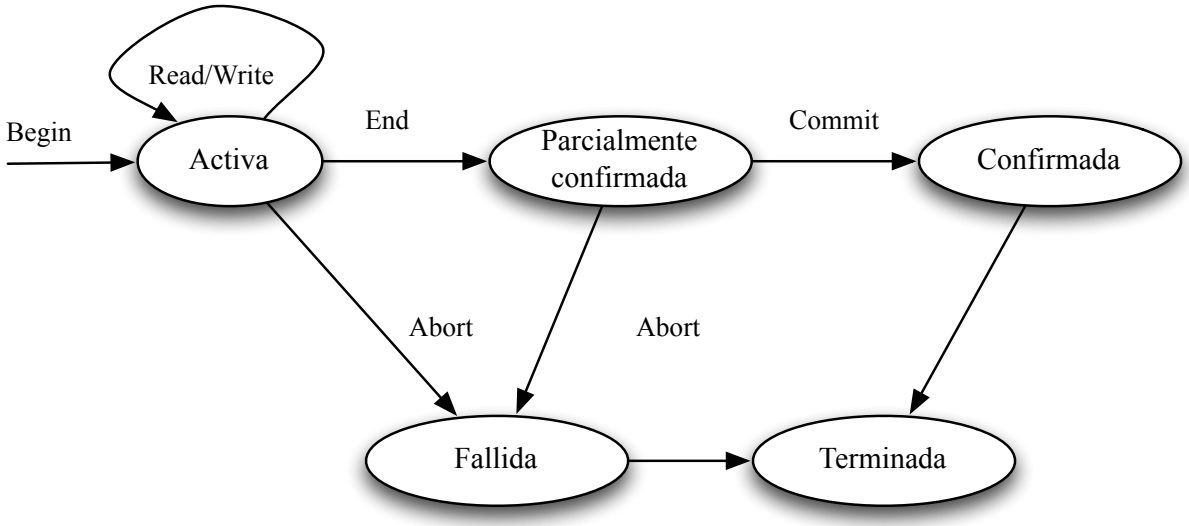
- Todas las operaciones de la transacción se realicen con éxito y su efecto quede registrado permanentemente en la BD.
- Que la transacción no tenga efecto alguno sobre la BD ni sobre otra transacción, si es que no se completa o falla.

## Tipos de Fallas

- Caída del Sistema: durante la ejecución de la transacción se produce un error de hardware, software o red.
- Error de transacción o del sistema: alguna operación de la transacción puede hacer que ésta falle (overflow) o errores de lógica.
- Errores locales o condiciones de excepción, por ejemplo saldo insuficiente. Esta podría programarse y no ser un tipo de fallo.
- Imposición de control de concurrencia: puede suceder que método de control de concurrencia aborte una transacción o deadlock (varias transacciones pelean por el mismo recurso).
- Fallo del disco
- Catástrofes

## Estado de transacciones

Una transacción es una unidad atómica de trabajo que se realiza por completo o bien no se efectúa en absoluto



## Estado de transacciones (cont.)

- El componente del sistema encargado de lograr la atomicidad se conoce como administrador de transacciones y las operaciones COMMIT (comprometer) y ROLLBACK (retroceder) son la clave de su funcionamiento.
- La operación COMMIT señala el término exitoso de la transacción: le dice al administrador de transacciones que se ha finalizado con éxito una unidad lógica de trabajo, que la base de datos está o debería estar de nuevo en un estado consistente y que se pueden comprometer, o hacer permanentes todas las modificaciones efectuadas por esa unidad de trabajo.
- La operación ROLLBACK, en cambio, señala el término no exitoso de la transacción: le dice al administrador de transacciones que algo salió mal, que la base de datos podría estar en un estado inconsistente y que todas las modificaciones efectuadas hasta el momento por la unidad lógica de trabajo deben retroceder o anularse.

## Características de las transacciones

- Atomicidad, en el sentido que hemos especificado anteriormente: se ejecutan todas las sentencias o ninguna.
- Preservación de la consistencia: la ejecución de una transacción deja la BD en un estado consistente.
- Aislamiento, ya que una transacción no muestra los cambios que produce hasta que finaliza.
- Persistencia, ya que una vez que finaliza la transacción con éxito, sus efectos perduran en la BD.
- Seriabilidad, en el sentido de que el efecto de ejecutar transacciones concurrentemente debe ser el mismo que se produciría al ejecutarlas por separado en un orden secuencial según van entrando en el sistema.

## Planes

- Un plan/planificación $P$ de $n$ transacciones $T_i$ , es la descripción del orden en que se ejecutan las operaciones de dichas $T_i$ , manteniendo, para cada $T_i$ , el orden original de sus operaciones.
- Un plan secuencial (o en serie) de un conjunto de T es aquel en el que todas las operaciones de cada $T_i$ se ejecutan en secuencia sin que se intercale ninguna operación de otra $T_j$ .
- Todos los planes secuenciales son correctos porque garantizan las propiedades de una transacción.
- Los planes secuenciales pueden ser ineficientes: conviene intercalar operaciones siempre que mantengan las propiedades de las transacciones.
- Un plan equivalente: aquel que produce los mismos efectos en la BD.
- Plan secuenciable o serializable: aquel que es equivalente a otro secuencial (también es correcto)

## Planes (cont.)

Dos operaciones de un plan están en conflicto si:

- Pertenecen a distintas transacciones.
- Tienen acceso al mismo elemento X.
- Al menos una de las operaciones es W(X).
- Para el $P_1$, $R_1(X)$ y $W_2(X)$ están en conflicto.

## Equivalencia por conflictos

Dos planes P1 y P2 son equivalentes por conflictos si, para todo par de operaciones en conflicto, ambas operaciones se ejecutan en el mismo orden en P1 y P2. Si dos operaciones (de dos Ts) no están en conflicto dentro de un plan, se pueden cambiar de orden. Esto permite conseguir planes equivalentes más eficientes (intercalando todo lo posible). Un plan P es secuenciable en cuanto a conflictos si se puede encontrar un plan secuencial P que sea equivalente por conflictos al P.

## Equivalencia por vista

Dos planes P y P' son equivalentes por vista si ambos tienen las mismas Ts y en los dos se cumplen las siguientes condiciones para los mismos valores, operaciones y transacciones:

- Si el valor leído por cada $R_i(X)$ de Ti cumple que fue producido por $W_j(X)$, o $X$ era el valor inicial del plan P, entones se ha de cumplir lo mismo en P' (asegurando que ambos leen los mismos valores).
- La operacin $W_k(Y)$ es la última que escribe $Y$ en el plan P también lo es en P (asegura que ambos Ps dejan el mismo estado final en la BD).

Un plan P es secuenciable en cuanto a vistas si se puede encontrar un plan secuencial P que sea equivalente por vistas al P.

# Ejemplo Plan en Serie (Plan A)

| T1 | T2 |
|---|---|
| R(X) | |
| X = X - N | |
| W(X) | |
| R(Y) | |
| Y = Y+N | |
| W(Y) | |
| | R(X) |
| | X=X+M |
| | W(X) |

# Ejemplo Plan NO en Serie (Plan B)

| T1 | T2 |
|---|---|
| R(X) | |
| X = X - N | |
| W(X) | |
| | R(X) |
| | X=X+M |
| | W(X) |
| R(Y) | |
| Y = Y+N | |
| W(Y) | |

## Equivalencia de planes

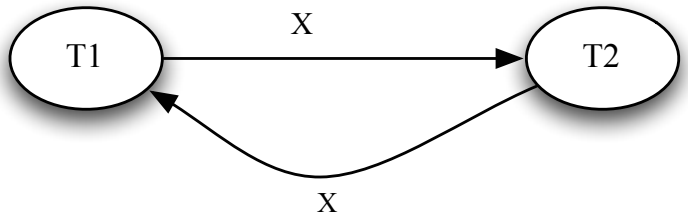
- El plan $B$ es equivalente por conflicto al plan $A$. El plan $B$ es serializable.
- Para probar si un plan es serializable por conflictos se usa un grafo de precedencia o grafo de serialización, que es un grafo dirigido $G = (N, A)$, con $N = \{T_1, T_2, \ldots, T : n\}$ y $A = \{a_1, a_2, \ldots, a_n\}$ arcos. Cada arco $a_i$ tiene la forma $(T_j \rightarrow T_k)$, con $j, k = 1, n$, donde $T_j$ es el nodo inicial de $a_i$ y $T_k$ es el nodo final. El arco se crea si una de las operaciones de $T_j$ aparece en el plan antes que alguna operación en conflicto de $T_k$.

## Equivalencia de planes (cont.)

- Para cada transacción $T_i$ que participa en el plan, crear un nodo etiquetado $T_i$ en el grafo.
- Para cada paso en $P$ en el que $T_j$ ejecuta una orden $R(X)$ después de que $T_i$ ejecute una orden $W(X)$, crear un arco $(T_i \rightarrow T_j)$ en el grafo.
- Para cada paso en $P$ en que $T_j$ ejecuta una operació $W(X)$ después de que $T_i$ ejecute $R(X)$, crear un arco $(T_i \rightarrow T_j)$ en el grafo.
- Para cada paso en $P$ en que $T_j$ ejecuta $W(X)$ después de que $T_i$ ejecute $W(X)$, crear un arco $(T_i \rightarrow T_j)$ en el grafo.
- El Plan $P$ es serializable si y solo si el grafo no tiene ciclos.

## Grafo de precedencia (ejemplo)

Asuma el siguiente plan con dos transacciones:
$P : R_1(X)R_2(X)W_1(X)R_1(Y)W_2(X)W_1(Y)$.
El grafo de precedencia es entonces:




Ya que existe un ciclo, el plan no es serializable.

Ejercicio: Considere el siguiente plan:
$P = R_2(Z)R_2(Y)W_2(Y)R_3(Y)R_3(Z)R_1(X)W_1(X)$
$W_3(Y)W_2(Z)R_2(X)R_1(Y)W_1(Y)W_2(X)$

## Archivo de log

Para conseguir anular y recuperar transacciones, el método más usado consiste en utilizar un archivo de diario o log en el que va guardando toda la información necesaria para deshacer (en caso de fracasar) o rehacer ( en caso de recuperar) las transacciones. Este archivo consta de:

- identificador de la transacción
- hora de modificación
- identificador del registro afectado
- tipo de acción
- valor anterior del registro
- nuevo valor del registro
- información adicional

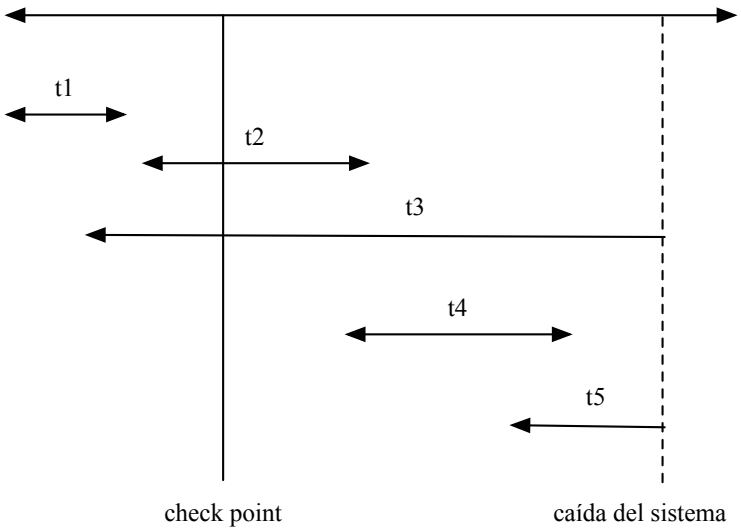
## Puntos de revisión (check points)

Un concepto relacionado con los archivos de log es el CHECKPOINT, que permite manejar en forma eficiente el contenido de los archivos log, ya que permiten no tener que recorrer todo el archivo de log, ante fallas. El establecimiento de puntos de revisión implica:

- Grabar físicamente el contenido de los búfers de datos a la base de datos física.
- Grabar físicamente un registro de punto de revisión especial dentro del archivo de log o bitácora.

Los puntos marcados como checkpoint, permiten la recuperación de la base de datos en caliente, es decir, después de la caída del sistema se obtiene la dirección del registro de recuperación más reciente y se recorre el archivo de log desde el punto marcado como checkpoint

## Puntos de revisión (cont)




La transacción t1 no se ve afectada por la falla del sistema, ni por el proceso de recuperación, por haberse completado antes del último punto de recuperación. Las transacciones t2 y t4, a pesar de haber terminado no han sido grabadas en la base de datos, ya que éstas serían comprometidas en un checkpoint. Las transacciones t3 y t5 deberán rehacerse ya que no han concluido.

## Transacciones en SQL

- La definición de transacción en SQL es una unidad lógica de trabajo que garantiza ser atómica.
- No tiene un sentencia de iniciación explícita, pero si terminan con COMMIT o ROLLBACK
- Las característica de las transacciones en SQL se definen con la sentencia SET TRANSACTION, la que especifica modo de acceso, tamaño del área de diagnóstico y nivel de aislamiento.

## Transacciones en SQL: modo de acceso

Puede especificar un READ ONLY o READ WRITE. Un modo READ WRITE permite que se ejecuten operaciones de insertar, modificar, crear o borrar. En cambio un modo READ ONLY solo sirve para leer.

## Transacciones en SQL: área de diagnóstico

DIAGNOSTICS SIZE *n*, especifica un valor entero que indica el número de condiciones que se pueden tomar simultáneamente en el área de diagnóstico. Estas condiciones suministan información al usuario de errores o excepciones de sentencias SQL ejecutadas más recientemente.

## Transacciones en SQL: nivel de aislamiento

ISOLATION LEVEL <aislamiento>, donde aislamiento puede tomar los sgtes valores:

- READ UNCOMMITED (lectura no confirmada)
- READ COMMITED (lectura confirmada)
- REPEATABLE READ (lectura repetible)
- SERIALIZABLE

## Transacciones en SQL: nivel de aislamiento (cont.)

Si se usa algo menos restrictivo que serializable, pueden ocurrir una de las siguientes violaciones:

- Lectura sucia: Ocurre cuando una transacción usa los valores actualizados por otra transacción aún no confirmada y que aborta.
- Lectura no repetible: Una trasacción T1 puede leer un valor de una tabla. Si una trasacción T2 actualiza luego ese valor y T1 vuelve a leer el valor, T1 verá un valor diferente.
- Fantasmas: Una transacción T1 puede leer un conjunto de tuplas, quizás dependiendo de la condición WHERE de una consulta. Si una transacción T2 inserta una nueva tupla que también satisface la condición del WHERE usada en T1, entonces cuando $T1$ se repita verá una tupla que no existía previamente.

## Transacciones en SQL: violaciones posibles

| Nivel de aislamiento | Lectura sucia | Lectura repetible | Fantasmas |
|---|---|---|---|
| Lectura no confirmada | Si | Si | Si |
| Lectura confirmada | No | Si | Si |
| Lectura repetible | No | No | Si |
| Serializable | No | No | No |

## Ejemplo SQL

```
EXEC SQL WHENEVER SQLERROR GOTO UNDO;
EXEC SQL SET TRANSACTION
     READ WRITE
     DIAGNOSTICS SIZE 5
     ISOLATION LEVEL SERIALIZABLE;
EXEC INSERT...;
EXEC UPDATE...;
EXEC COMMIT;
GOTO: THE_END;
UNDO: EXEC SQL ROLLBACK;
THE_END:...;
```