Facultad de Estudios Superiores
Acatlán

# PLSQL

José Antonio Coria Fernández
Email: diplomadobd@gmail.com

# Contenido



- ¿Qué es una transacción?
- Estados de una transacción
- ¿Cuándo inicia y finaliza una transacción?
- Ventajas de las sentencias COMMIT y ROLLBACK
- Control explicito del control de transacciones
- Consistencia en la lectura
- ¿Qué son los bloqueos?
- Acciones en una transacción
- Problemas en la intercalación de transacciones.
- Planes
- Control de concurrencia

# Transacciones



- Acción o serie de acciones llevadas a cabo por un único usuario o por un programa de aplicación y que lee y actualiza el contenido de la base de datos.
- Para asegurar la integridad de los datos se necesita que el sistema de base de datos mantenga las siguientes propiedades de las transacciones **(ACID)** :
  * **Atomicidad.** Todas las operaciones de la transacción se realizan adecuadamente en la base de datos o ninguna de ellas se ejecuta, propiedad del todo o nada.
  * **Consistencia.** Una transacción conserva la consistencia si partiendo de un estado consistente de la base de datos la deja en otro estado consistente.
  * **Aislamiento.** Una transacción debe parecer que se está ejecutando de forma aislada de las demás transacciones. Es decir, las transacciones no deben interferir entre sí.
  * **Durabilidad.** Los efectos de una transacción completada con éxito (confirmada) se registran en la base de datos y no deben perderse debido a una falla posterior.

# Transacciones



**Estados de una transacción**

**Activa**, el estado inicial; la transacción permanece en este estado durante su ejecución.

- **Parcialmente comprometida**, después de ejecutarse la última instrucción.
- **Fallida**, tras descubrir que no puede continuar la ejecución normal.
- **Abortada**, después de haber retrocedido la transacción y restablecido la base de datos a su estado anterior al comienzo de la transacción.
- **Comprometida**, tras completarse con éxito

# Transacciones



## Estados de una transacción

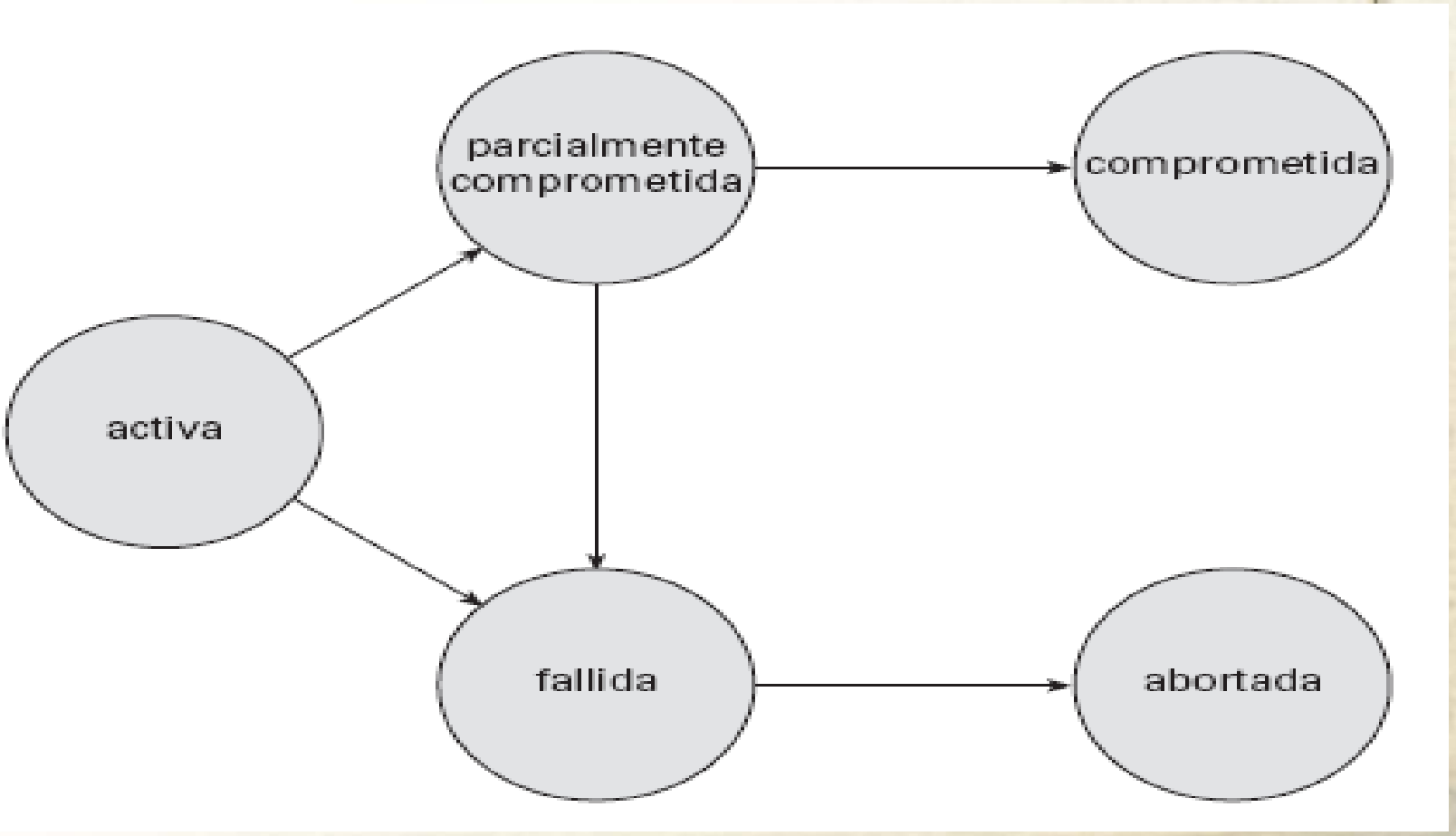
El sistema al encontrarse con una transacción que es fallida y después de pasar a abortada, tiene dos opciones:

- **Reiniciar** la transacción, pero sólo si la transacción se ha abortado a causa de algún error hardware o software que no lo haya provocado la lógica interna de la transacción. Una transacción reiniciada se considera una nueva transacción.
- **Cancelar** la transacción. Normalmente se hace esto si hay algún error interno lógico que sólo se puede corregir escribiendo de nuevo el programa de aplicación, o debido a una entrada incorrecta o debido a que no se han encontrado los datos deseados en la base de datos.

# Transacciones



## Estados de una transacción



# Transacciones



Las transacciones proporcionan mayor flexibilidad y control cuando los datos cambian y ello asegura la consistencia de los datos en el caso de un fallo en el proceso del usuario o del sistema.

Las transacciones consisten de sentencias DML que componen un cambio consistente en los datos.
Por ejemplo:  una transferencia de fondos entre dos cuentas debe incluir la resta de una cuenta y  ingreso a otra cuenta por la misma cantidad. En su conjunto ambas acciones deben fallar o triunfar; la resta no se debe cometer sin la suma.

# Transacciones



***¿Cuando inicia y finaliza una transacción?***

Una transacción inicia cuando la primera sentencia DML es encontrada y finaliza cuando ocurre alguno de los siguientes puntos:

- Una sentencia COMMIT o ROLLBACK es usada
- Una sentencia DDL, como CREATE es utilizada
- Una sentencia DCL es usada
- El usuario sale de *i*SQL*Plus
- Una computadora falla o el sistema falla

Después de que una transacción finaliza, la siguiente sentencia SQL ejecutada automáticamente inicia la siguiente transacción.

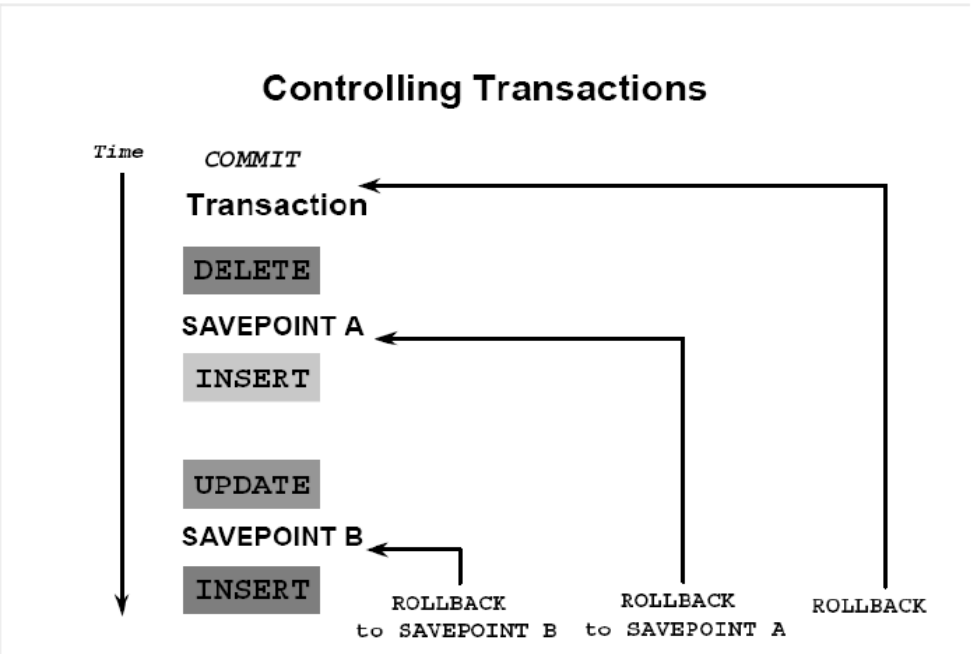
Una sentencia DDL o DCL es automáticamente completada y por consiguiente implícitamente finaliza una transacción.

# Transacciones



***Ventajas de las sentencias COMMIT y ROLLBACK***

- Asegurar la consistencia de datos
- Ver previamente los cambios de los datos antes de hacerlos permanentes
- Agrupar operaciones lógicamente relacionadas



# Transacciones



***Control explicito del control de transacciones***

Se puede controlar la lógica de transacciones con el uso de las sentencias COMMIT, SAVEPOINT y ROLLBACK.

| Sentencia | Descripción |
|---|---|
| COMMIT | Finaliza la transacción actual haciendo permanentes todos los cambios pendientes |
| SAVEPOINT name | Marca un punto de la transacción actual |
| ROLLBACK | Finaliza la transacción actual descartando todos los cambios pendientes |
| ROLLBACK to SAVEPOINT name | ROLLBACK TO SAVEPOINT retorna la transacción actual a la marca especificada, en consecuencia deshace todos los cambios y/o SAVEPOINTS creados después de la marca especificada a la cual retorna.<br>Si se omite la cláusula TO SAVEPOINT, la sentencia ROLLBACK deshace toda la transacción. |

# Transacciones



***Deshaciendo cambios a un SAVEPOINT***

Se pueden crear marcas en una transacción con la utilización de la sentencia SAVEPOINT la cual divide la transacción en secciones más pequeñas. Por lo que se pueden descartar los cambios pendientes hasta la marca con el uso de la sentencia ROLLBACK TO SAVEPOINT. Si se crea una segunda marca con el mismo nombre de un savepoint anterior, el savepoint anterior es eliminado.

# Transacciones



***Ejemplo Deshaciendo cambios a un SAVEPOINT***

```
declare
v_errors_yn VARCHAR2(1) := 'N';
begin
update emp
set sal = sal * 1.1 →5
where deptNo = 10;
savepoint  SaveDept10Update;
update emp
set sal = sal * .9
where deptNo = 20;
……
ROLLBACT TO SaveDept10Update:
```

# Transacciones



## Procesando transacciones implícitas

| Estatus | Circunstancias |
|---|---|
| COMMIT automático | Sentencias DDL o DCL son usadas.<br>Abandonas *i*SQL*Plus de forma normal, sin el uso explicito de comandos COMMIT o ROLLBACK |
| ROLLBACK automático | Terminación anormal de *i*SQL*Plus o falla del sistema |

**Nota:** Un tercer comando está disponible en SQL*Plus. El comando AUTOCOMMIT puede ser habilitado o deshabilitado. Si se habilita, cada sentencia DML individual es cometida e inmediatamente es ejecutada. No se pueden deshacer estos cambios. Si se deshabilita, la sentencia COMMIT debe ser usada explícitamente. También, la sentencia COMMIT es usada cuando una sentencia DDL es usada o cuando se sales de SQL*Plus.

# Transacciones



## Realizando Cambios (commiting)

El estado de los datos antes de usar las sentencias COMMIT o ROLLBACK es:

- Las operaciones de manipulación de datos afectan en primera instancia el buffer de la base de datos; por lo tanto, el estado previo de los datos se puede recuperar.
- El usuario actual puede revisar los resultados de las operaciones de manipulación de datos para consultar las tablas con el uso de la sentencia SELECT.
- Otros usuarios no pueden ver los resultados de las operaciones de manipulación de datos hechas por el usuario actual. El servidor de Oracle establece la consistencia para asegurar que cada usuario vea los datos como existieron desde el último commit.
- Las filas afectadas son bloqueadas; otros usuarios no pueden cambiar los datos de las filas afectadas.

# Transacciones



## Realizando Cambios (commiting)

Después de una sentencia COMMIT:

- Los cambios de datos son escritos en la base de datos
- El estado previo de los datos es perdido permanentemente
- Todos los usuarios pueden ver los resultados de la transacción
- Los bloqueos en las filas afectadas son liberados; las filas son ahora disponibles para que otros usuarios realicen nuevos cambios a los datos
- Todos los savepoints son borrados.

# Transacciones



## Ejemplo Realizando Cambios (commiting)

Elimine los departamentos 290 y 300 de la tabla DEPARTMENTS, y actualice una fila en la tabla COPY_EMP. Haga todos los cambios permanentes.

```
DELETE FROM departments
WHERE  department_id IN (290, 300);

2 rows deleted.

UPDATE  copy_emp
  SET   department_id = 80
  WHERE employee_id = 206;

1 row updated.

COMMIT;

Commit Complete.
```

# Transacciones



***Deshaciendo cambios (rolling back)***

Después de una sentencia ROLLBACK:

- Los cambios a los datos son deshechos
- El estado previo de los datos es restablecido
- Los bloqueos de las filas afectadas son liberados

**Ejemplo**

Mientras se intenta eliminar un registro de la tabla TEST, se puede accidentalmente vaciar la tabla. Se puede corregir el error, utilizando la sentencia apropiada, y haciendo permanente el cambio a los datos

# Transacciones



***Ejemplo deshaciendo cambios (rolling back)***

```
DELETE FROM test;
25,000 rows deleted.

ROLLBACK;
Rollback complete.

DELETE FROM test
WHERE  id = 100;
1 row deleted.

SELECT *
FROM   test
WHERE  id = 100;
No rows selected.

COMMIT;
Commit complete.
```

# Transacciones



***Consistencia en la lectura***

Los usuarios de la base de datos acceden a la base de datos de dos maneras:

- Con operaciones de lectura (Sentencia SELECT)
- Con operaciones de escritura (Sentencias INSERT, UPDATE y DELETE)

Se necesitan lecturas consistentes para que ocurra lo siguiente:

- Las operaciones de lectura y escritura a la base de datos aseguran una vista consistente de los datos

# Transacciones



***Consistencia en la lectura***

- Las operaciones de lectura no ven datos que estén en un proceso de cambio
- Las operaciones de escritura aseguran que los cambios a la base de datos sean hechos en un camino consistente.
- Los cambios hechos por una operación de escritura no interrumpen ni entran en conflicto con los cambios hechos por otra operación de escritura.

*** Las lecturas consistentes son una implementación automática

# Transacciones



***¿Qué son los bloqueos?***

Los bloqueos son mecanismos que impiden destruir interacciones entre transacciones acensando al mismo recurso, cualquier objeto de usuario (como tablas o filas) u objetos del sistema no visibles a los usuarios (como estructuras de datos compartidas y filas del diccionario de datos).

Los bloqueos de Oracle son ejecutados automáticamente y no requiere la acción del usuario

# Transacciones



**Acciones de una Transacción**

Las acciones que puede ejecutar una transacción son: lectura y escritura de objetos de la BD.

Rt(O): denota lectura del objeto O por transacción t
Wt(O) : denota escritura de O por t

# Transacciones



**Problemas al Intercalar Transacciones**

Dos acciones sobre el mismo objeto de la BD entran en conflicto si por lo menos una de las acciones es una escritura.

Existen tres tipos de conflictos entre dos transacciones *T*1 y *T*2:

1. **Conflicto de escritura-lectura (WR),** o lectura de datos no comprometidos. El principal problema es la lectura de datos no comprometidos, es decir, T2 lee un objeto que fue modificado por T1, pero aún no ha sido comprometido (commited).

Ejemplo: Considere las transacciones *T*1 y *T*2:

*T*1 transfiere 100 desde la cuenta *A* a la *B*
*T*2 incrementa *A* y *B* por el 10%
Supongamos los valores iniciales *A* = 1000 y *B* = 500

# Transacciones



## Problemas al Intercalar Transacciones

2. **Conflicto de lectura-escritura (RW),** o lecturas no repetidas. El principal problema es la lectura no repetida de datos, es decir, modifica el valor de un objeto A que es leído por una T1, mientras T1 aún esta en progreso.

Ejemplo: Considere las transacciones *T*1 y *T*2, y sea *A* el numero de copias disponibles de un libro

*T*1 lee *A* = 1
*T*2 lee *A* = 1, resta 1 y compromete (A=0)
*T*1 trata de restar 1 a *A* y recibe un error, *A* ya no tiene el valor 1!

# Transacciones



## Problemas al Intercalar Transacciones

3. **Conflicto de escritura-escritura (WW),** o sobreescritura de datos no comprometidos. T2 sobreescribe el valor de un objeto A que ya ha sido modificado por una transacción T1, mientras T1 todavía está en progreso.

Ejemplo: Suponga que los salario de Pedro y Juan deben ser iguales

*T*1 asigna salarios iguales a 2000
*T*2 asigna salarios iguales a 1000

El plan *T*1 - *T*2 asigna salarios iguales a 1000,
El plan T2 - *T*1 asigna salarios iguales a 2000

# Transacciones



## Planes

Un plan de ejecución (planicación) es una lista de acciones (lectura, escritura, comprometer o abortar) de un conjunto de transacciones

El orden en el cual dos acciones de una transacción *T* aparecen en un plan debe ser el mismo orden que el orden en el que aparecen en *T*

*Ejemplo:*

| Plan no serial | |
|---|---|
| $T_1$ | $T_2$ |
| $R(A)$ | |
| $W(A)$ | |
| | $R(B)$ |
| | $W(B)$ |
| $R(C)$ | |
| $W(C)$ | |
| $commit_{T_1}$ | |
| | $commit_{T_2}$ |

| Serial: $T_1 - T_2$ | |
|---|---|
| $T_1$ | $T_2$ |
| $R(A)$ | |
| $W(A)$ | |
| $R(C)$ | |
| $W(C)$ | |
| $commit_{T_1}$ | |
| | $R(B)$ |
| | $W(B)$ |
| | $commit_{T_2}$ |

*Si las acciones de diferentes transacciones no son intercaladas, i.e. las transacciones son ejecutadas de inicio a fin, una por una, el plan se denomina plan serial. Para el ejemplo existen dos posibles planes seriales: T1-T2 o T2- T1*

# Transacciones



**Control de concurrencia**

El proceso de gestionar una serie de operaciones simultáneamente en la base de datos de modo que estas no interfieran unas con otras.

Sin este tipo de mecanismos, se pueden producir problemas de:

**Actualización perdida**. Una operación de actualización que en apariencia haya sido completada con éxito por un usuario puede ser sobrescrita por otro usuario.

**Dependencia no confirmada**. Se produce cuando se permite una transacción ver los resultados intermedios de otra transacción antes de que esta haya sido confirmada.

**Análisis coherente**. Surge cuando una transacción lee varios valores de la base de datos mientras que una segunda transacción actualiza uno de ellos durante la ejecución de la primera.

# Transacciones



## Control de concurrencia basado en bloqueos

Los bloqueos puedes ser compartidos (lectura) o exclusivos (escritura).

**Bloqueo de dos fases**. En el bloqueo de dos fases, una transacción adquiere todos sus bloqueos antes de liberar alguno de ellos.

El protocolo 2PL exclusivo (estricto) tiene dos reglas:

1. Si una transacción *T* quiere leer (respectivamente, modificar) un objeto, primero solicita un bloqueo compartido (respectivamente, exclusivo) sobre el objeto
2. Todos los bloqueos concedidos a una transacción se liberan cuando la transacción se completa.

Obviamente, si una transacción tiene un bloqueo exclusivo sobre un objeto, también puede leer el objeto. Una transacción que requiere un bloqueo es suspendida hasta que el SGBD pueda otorgar el candado

# Transacciones



## Control de concurrencia basado en bloqueos

Ejemplo: Considere las transacciones *T*1 y *T*2:

*T*1 transfiere 100 desde la cuenta *A* a la *B*

*T*2 incrementa *A* y *B* por el 10% con valores iniciales de *A* = 1000 y *B* = 500, el siguiente plan no está permitido por el protocolo 2PL estricto:

| $T_1$ | $T_2$ |
|---|---|
| $R(A)$, A=1000 | |
| $W(A)$, A=900 | |
| | $R(A)$, A=900 |
| | $W(A)$, A=990 |
| | $R(B)$, B=500 |
| | $W(B)$, B=550 |
| | $commit_{T_2}$ |
| $R(B)$= 550 | |
| $W(B)$= 650 | |
| $commit_{T_1}$ | |

# Transacciones



## Control de concurrencia basado en bloqueos

Ejemplo:

| $T_1$ | $T_2$ |
|---|---|
| $X(A)$ | |
| $R(A)$, A=1000 | |
| $W(A)$, A=900 | |
| $X(B)$ | |
| $R(B)$= 500 | |
| $W(B)$= 600 | |
| $commit_{T_1}$ | |
| | $X(A)$ |
| | $R(A)$, A=900 |
| | $W(A)$, A=990 |
| | $X(B)$ |
| | $R(B)$, B=600 |
| | $W(B)$, B=660 |
| | $commit_{T_2}$ |

# Transacciones



**Marcado temporal.**

Las transacciones se ordenan de tal forma que las transacciones más antiguas tienen prioridad en caso de conflicto.

**Interbloqueo**

Una forma usual de resolver los interbloqueos es usando un mecanismo de tiempos, i.e., si una transacción ha esperado un candado por mucho tiempo, se asume que ha ocurrido un interbloqueo y la transacción se aborta

Los interbloqueos pueden prevenirse:

1. Bloqueando partes de los objetos, e.g. tuplas en vez de tablas completas
2. Reduciendo el tiempo en que una transacción puede mantener un candado
3. Reduciendo la cantidad de objetos que leídos y modificados