B4

Sistemas de Bases de Datos Relacionales

Tema 11. Aspectos básicos del Procesamiento de Transacciones

Tema 11. Aspectos básicos del procesamiento de transacciones

Objetivos

- Conocer el concepto de **transacción** en un sistema de bases de datos (SBD), identificar sus **estados** y **propiedades** y comprender cómo el SGBD **establece** y **finaliza** las transacciones
- Comprender los problemas que puede acarrear la **concurrencia** de transacciones y qué mecanismos emplea un SGBD para su control
- Entender los efectos que puede tener un **fallo** en un sistema de bases de datos y qué mecanismos utiliza un SGBD para **recuperar** la consistencia de los datos
- Conocer cómo el SGBD consigue una ejecución de consultas eficiente

Tema 11. Aspectos básicos del procesamiento de transacciones

Contenidos

- 11.1 Soporte de transacciones
- 11.2 Concurrencia de transacciones
- 11.3 Recuperación tras fallos
- 11.4 Procesamiento y optimización de consultas

Introducción a...

- Anexo: Más sobre Concurrencia en Oracle

Tema 11. Aspectos básicos del procesamiento de transacciones

Bibliografía

- [CB 2015] Connolly, T.M.; Begg C.E.: ***Database Systems: A Practical Approach to Design, Implementation, and Management***, 6th Edition. Pearson. *Capítulos 1, 22 y 23.*

- [EN 2016] Elmasri, R.; Navathe, S.B.: ***Fundamentals of Database Systems***, 7th Edition. Pearson. *Capítulos 1, 18, 19, 20, 21 y 22.*

Soporte de Transacciones

- Una **transacción** es una acción, o una **secuencia de acciones, que lee y/o actualiza el contenido de la base de datos…**
  - Puede ser un programa completo, una parte de un programa, una serie de sentencias SQL, o una única sentencia
  - Puede implicar cualquier número de operaciones de acceso a la BD
- … que se ejecuta como una unidad: es una **unidad lógica de procesamiento**
  - *Ejemplo*: transferencia de dinero entre dos cuentas bancarias →

Transferir 200€ de una cuenta X a otra cuenta Y

1. Leer el saldo de X
2. Comprobar que el saldo de X es superior a 200
   Si no, indicar que no se puede realizar la transferencia y finalizar
3. Restar 200 del saldo de X
4. Sumar 200 al saldo de Y
5. …

¿Por qué definir transacciones?

Transferir 200€ de una cuenta X a otra cuenta Y

1. SELECT saldo
   FROM cuenta
   WHERE ccc = X;
2. SI (saldo<200) ENTONCES
   ko_saldo_insuficiente(saldo);
3. UPDATE cuenta
   SET saldo=saldo – 200
   WHERE ccc = X;
4. UPDATE cuenta
   SET saldo=saldo + 200
   WHERE ccc = Y;
5. …

- Saldos iniciales:
  1000€ en X y 500€ en Y
- ¿Qué pasa si ocurre un fallo de la transacción justo tras el paso 3 y nunca llega a ejecutarse el paso 4?
  - La cuenta X queda con 800€
  - El saldo de Y aún es 500€
  - ***¿Dónde están los 200€?***
  - ¡¡Datos incoherentes!!
- Esto es porque algunos cambios se han hecho, y otros no: ***la transferencia se ha ejecutado parcialmente***

¿Por qué definir transacciones?

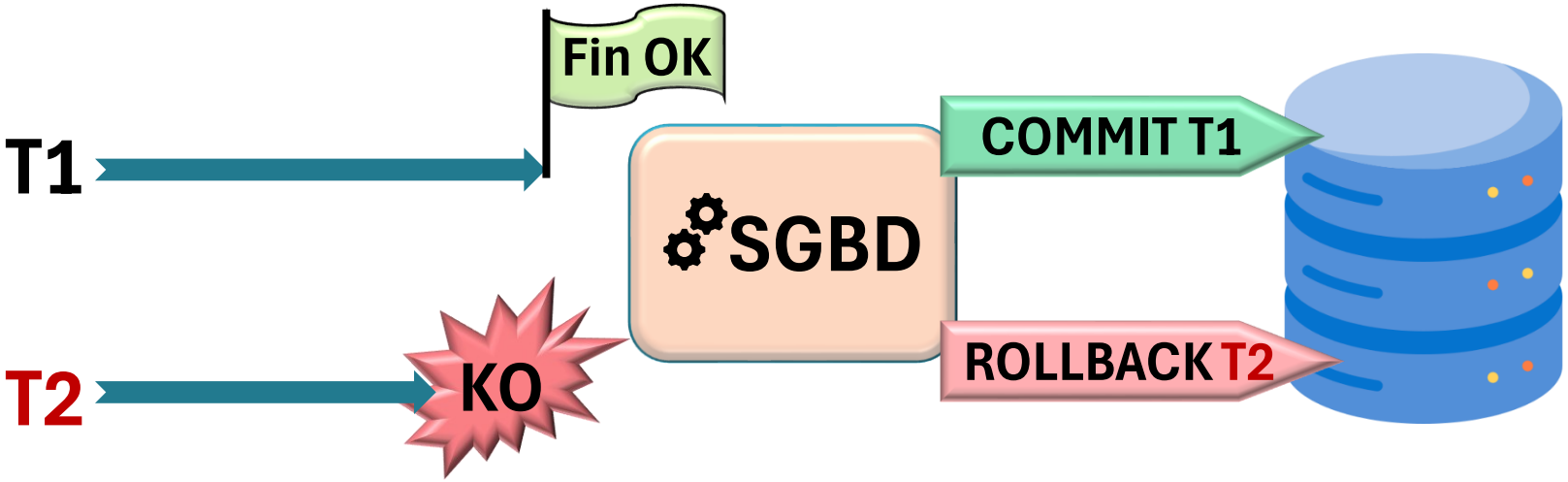
- Hay conjuntos de operaciones que **sólo tienen sentido** si se ejecutan **de forma completa**
- Por esto el concepto de **transacción** se define con esta **idea clave:**

Una transacción es **ATÓMICA**
O se ejecutan todas las operaciones que componen la transacción, o no se realiza ninguna

- Y eso es el "Soporte de Transacciones": garantizar que se llevan a cabo **todas las operaciones de actualización** de datos correspondientes a una determinada **transacción,**
  o bien, que no se ejecute **ninguna** de ellas

Final de una transacción e implicaciones

- Toda transacción termina con éxito o con fracaso
  - Si finaliza con **éxito** (COMMIT), se garantiza que todo cambio realizado por la transacción está consolidado en la BD en disco
  - Si termina con **fracaso** (ROLLBACK), en realidad NO SE HA EJECUTADO
    - Todas sus operaciones de actualización de datos han sido anuladas



Propiedades ACID de una transacción

Responsable

A Atomicidad

Subsistema de **Recuperación** del SGBD

C Consistencia

Programadores +
Subsistema de **Integridad** del SGBD

I Isolation (aislamiento)

Subsistema de **Concurrencia** del SGBD

D Durabilidad

Subsistema de **Recuperación** del SGBD

Propiedades ACID de una transacción

- **Atomicidad**
  - Ejecución 'Todo o Nada'
- **Consistencia**
  - Una transacción T lleva la BD de un estado de consistencia (integridad) a otro estado de consistencia
  - Al inicio de una transacción y justo al terminar, la BD está en un estado de consistencia: se cumplen todas las restricciones de integridad
  - Durante la ejecución de una transacción es posible que alguna restricción no se cumpla

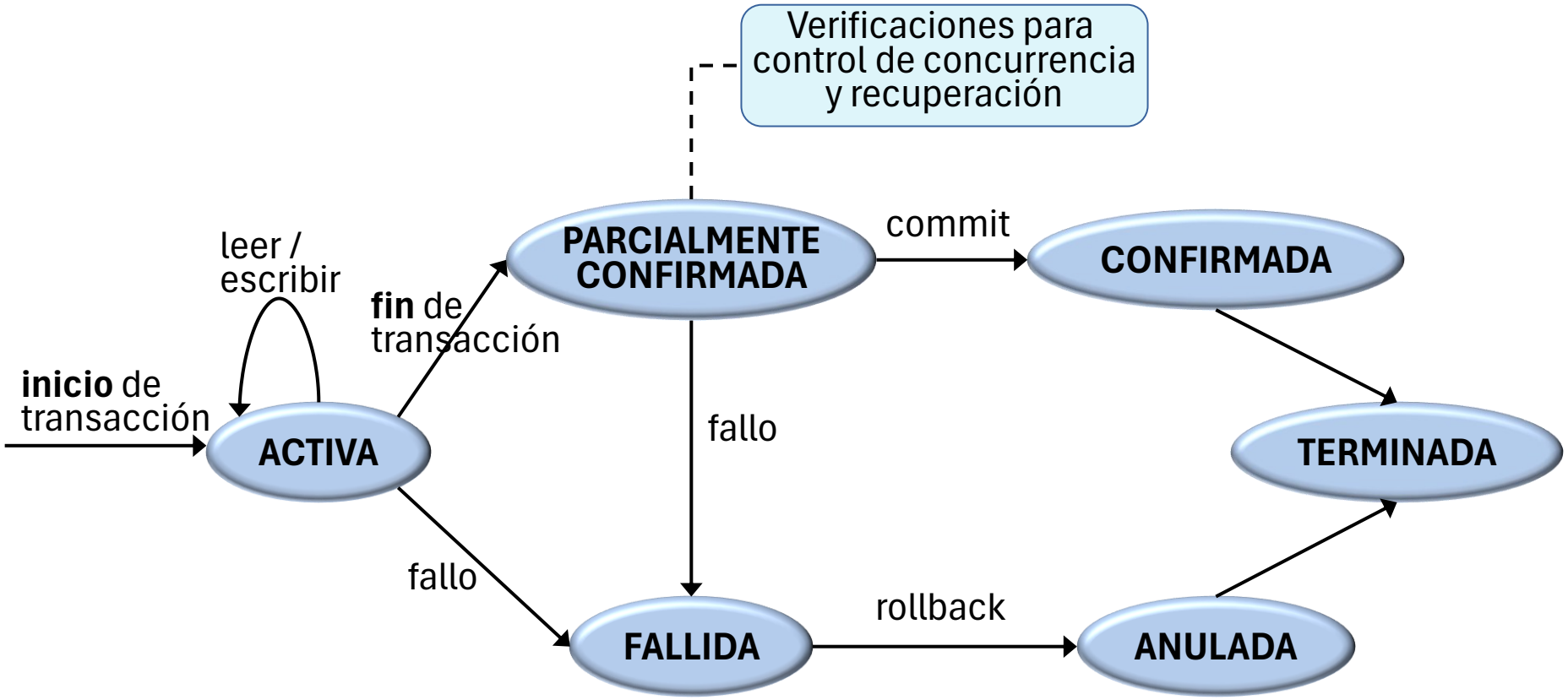
Propiedades ACID de una transacción

- **Aislamiento** (**I**solation)

  Las transacciones son procesos o *threads* (programas en ejecución) concurrentes: puede haber conflictos entre ellos si acceden a los mismos datos

  - La ejecución de T es independiente de la del resto de transacciones
  - Los cambios producidos por T durante su ejecución no deberían ser visibles para otras transacciones hasta que T finalice
  - Puede no imponerse de forma estricta
    → niveles de aislamiento
- **Durabilidad**
  - Una vez que T finaliza con éxito y es confirmada, sus cambios están almacenados en la BD de forma permanente y perduran, aunque el sistema falle justo después

Estados de una transacción



Inicio de una transacción

- Al **ejecutar** una (primera) **sentencia SQL** (LDD o LMD)…
  - **interactiva** (vía *SQLDeveloper*, por ejemplo), o
  - incluida en un **programa** que no tiene ya una transacción en progreso
    - Si el programa ya tiene una transacción en curso, la sentencia SQL se integra dentro de dicha transacción
- Ejemplos
  - *Introducción* de una tarjeta bancaria en un cajero automático
  - Solicitar *realizar otra operación* (enviar dinero vía Bizum) tras haber concluido una operación bancaria (consultar saldo)
  - Escritura de la *primera sentencia* SQL en una sesión de *SQLDeveloper*
  - Seleccionar *Tramitar Pedido* en una web de venta online, de modo que se inicia el procedimiento de pago de los productos en el carrito de la compra
  - …

Finalización de una transacción

- Al **ejecutar explícitamente** la operación **COMMIT** (**confirmar**) xor **ROLLBACK** (**anular**)
- Al **terminar el programa en ejecución**, ya sea con **éxito** (COMMIT implícito) o con un **error** (ROLLBACK implícito)
- *Ejemplos*
  - Concluye con éxito una operación bancaria (reintegro), y el usuario confirma que no desea realizar ninguna operación más
  - Un/a estudiante escribe y ejecuta ROLLBACK durante su sesión de trabajo con *SQLDeveloper*
  - El cajero automático cancela una operación de reintegro porque no hay saldo suficiente en la cuenta de origen
  - El/la cliente del banco decide anular la operación bancaria en curso, pulsando el botón de "cancelar" en el cajero automático
  - El cajero automático deja de funcionar a mitad de operación por un fallo eléctrico…

Soporte de transacciones en Oracle

Inicio de transacción

ⓘ No existe sentencia BEGIN TRANSACTION

- Cuando **no** hay ya una **transacción en progreso**, y se **ejecuta una sentencia LMD**
  - Sentencias LMD: INSERT, UPDATE, DELETE, SELECT
- Cuando **se ejecuta una sentencia LDD**
  - Sentencias LDD: CREATE, ALTER, DROP, RENAME, COMMENT
  - Si ya existe una transacción en ejecución, Oracle la finaliza con COMMIT y comienza una nueva para esta sentencia LDD

Soporte de transacciones en Oracle

Fin de transacción con éxito: COMMIT

Finaliza la transacción actual y **confirma** (hace permanentes) los cambios realizados

- COMMIT **explícito** (por parte del programa o usuario)
  - Ejecución de la sentencia **COMMIT;**
- COMMIT **implícito** (por parte del SGBD) cuando...
  - El programa finaliza de forma normal y sin errores
  - Se sale de la herramienta (*SQLDeveloper*, ...) correctamente
  - Se ejecuta una sentencia LDD

    Cada sentencia LDD es tratada como una transacción en sí misma

    - Oracle realiza un COMMIT antes de ejecutarla, y si tiene éxito, otro COMMIT después

Soporte de transacciones en Oracle

Fin de transacción con fracaso: ROLLBACK

Finaliza la transacción actual y **deshace** (anula) los cambios realizados

- ROLLBACK **explícito** (por parte del programa o usuario)
  - Ejecución de la sentencia **ROLLBACK;**
- ROLLBACK **implícito** (por parte del SGBD) cuando…
  - El programa finaliza de forma anormal, inesperada o errónea
  - Se sale de la herramienta (*SQLDeveloper*, …) apagando el PC, o **cerrando la ventana directamente**, sin desconectar de la BD

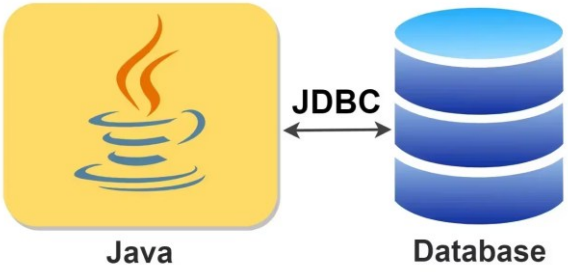
Transacciones en la práctica

Ejemplos de código

- Sesiones interactivas con SGBD Oracle

- Acceso a BD desde programas PL/SQL Oracle

- Acceso a BD a través de JDBC



Sesión interactiva y transacciones Oracle

Conexión a *SQL Developer:*	**(inicio de sesión interactiva)**
SELECT ...;	Inicia **T1**
SELECT ...;	
UPDATE...;	
ALTER TABLE...;	Finaliza T1 (COMMIT) Inicia **T2** y finaliza T2 (COMMIT)
INSERT INTO...;	Inicia **T3**
INSERT INTO...;	
CREATE VIEW ...;	Finaliza T3 (COMMIT) Inicia **T4** y finaliza T4 (COMMIT)
SELECT ...;	Inicia **T5**
DELETE ...;	
SELECT...;	
INSERT INTO...;	
INSERT INTO...;	
Desconexión del *SQLDeveloper*	Solicita COMMIT o ROLLBACK al usuario y así finaliza T5

Sesión interactiva y transacciones Oracle

Conexión a *SQL Developer*	**(inicio de sesión interactiva)**
SELECT ...;	Inicia **T1**
SELECT ...;	
UPDATE...;	
DROP TABLE...;	Finaliza T1 (COMMIT) Inicia **T2** y finaliza T2 (COMMIT)
INSERT INTO...;	Inicia **T3**
UPDATE...;	
SELECT...;	
INSERT INTO...;	
ROLLBACK;	Finaliza T3 (KO) Deshace TODA modificación de T3
SELECT ...;	Inicia **T4**
DELETE ...;	
INSERT INTO...;	
INSERT INTO...;	
COMMIT;	Finaliza T4 (COMMIT)
Desconexión del *SQL Developer*	

Anexo

Bloque de código PL/SQL de Oracle


```
PROCEDURE transferencia (ctaOrigen IN VARCHAR(23),
          ctaDestino IN VARCHAR(23), importe IN NUMBER(11,2)
IS
  saldo_origen NUMBER(11,2);
BEGIN
  SELECT saldo INTO saldo_origen FROM cuenta WHERE ccc = ctaOrigen;
  IF (saldo_origen < importe) THEN
    RAISE_APPLICATION_ERROR (-20099, 'Saldo insuficiente');
  ENDIF;
  UPDATE cuenta SET saldo = saldo - importe
    WHERE ccc = ctaOrigen;
  UPDATE cuenta SET saldo = saldo + importe
    WHERE ccc = ctaDestino;
  INSERT INTO movimiento
    VALUES ('transf',ctaOrigen,ctaDestino,importe*(-1),SYSDATE);
  INSERT INTO movimiento
    VALUES ('transf',ctaDestino,ctaOrigen,importe,SYSDATE);
  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
  dbms_output.put_line('Error en la transaccion:'||SQLERRM);
  dbms_output.put_line('Se deshacen las modificaciones);
  ROLLBACK;
END transferencia;
```

El/la programador/a debe escribir el código siendo consciente de **qué debe ejecutarse como una transacción**

Y capturar **excepciones** (errores en tiempo de ejecución) y **decidir** qué hacer (rollback o no)

Anexo

Acceso a Bases de Datos vía JDBC

- **J**ava **D**ata**B**ase **C**onnectivity
  - API Java que incluye la declaración de interfaces para ejecutar consultas SQL contra Bases de Datos Relacionales
- Para acceder a una base de datos es necesario un ***driver*** que implementa todas las interfaces del API
  - Cada base de datos (MySQL, Oracle, …) proporciona un fichero `.jar` (con todas las clases) que se tiene que añadir al *classpath* del programa.

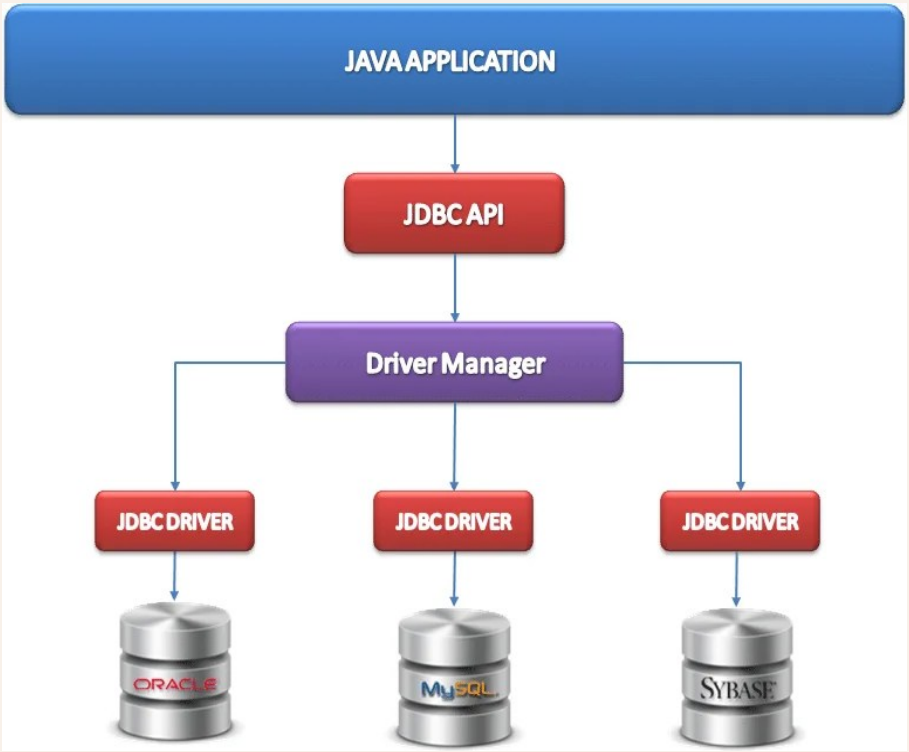


Imagen: https://waytoeasylearn.com/learn/jdbc-architecture/

Anexo

Acceso a BD vía JDBC

- Las interfaces se encuentran en el paquete **`java.sql`** y permiten:
  - Establecer una conexión con una base de datos
  - Ejecutar una consulta (de cualquier tipo)
  - Procesar los resultados

```
// Establecer la conexión
Connection conn = DriverManager.getConnection (
        "jdbc:oracle:thin:@miBD", "miLogin", "miPassword");
// Ejecutar una consulta
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT nss, nombre FROM
EMPLEADO");
// Procesar los resultados
while (rs.next()) {       //mientras haya filas que recorrer
   int nss = rs.getInt("nss");
   String nombre = rs.getString("nombre");
   …
}
```

Anexo

Acceso a BD vía JDBC: **excepciones**

- Cuando en la ejecución del programa se encuentra un problema en la conexión o acceso a la BD se lanza la excepción *comprobada* **`SQLException`**
  - Cada `SQLException` contiene información que ayuda a determinar la causa del error (descripción, código, etc.)
  - Con esto, el programador sabe **decidir si anular la transacción** o no

```
...} catch( SQLException ex ) {
  System.out.println("SQLException:"+ex.getMessage());
  System.out.println("SQLState: "+ex.getSQLState());
  System.out.println("ErrorCode: "+ex.getErrorCode());

  conn.rollback();
}
```

Anexo

Acceso a BD vía JDBC: **transacciones**

- Ejecución de bloques de consultas SQL manteniendo las propiedades ACID
- Una conexión funciona en **modo autocommit** por defecto:
  - Cada sentencia representa una sola transacción
  - Método `void setAutoCommit(boolean b)`
- Para definir un bloque de consultas/sentencias (transacción):
  - **Desactivar** el **modo autocommit** de la conexión.

```
conexion.setAutoCommit(false);
```

  - **Finalizar** cada transacción ejecutando `commit()` o `rollback()` sobre la conexión

Anexo

Acceso a BD vía JDBC: **transacciones**


```
public void insertar_departamento(String cod, String nom, int dir)
   { try {
        //inicia transacción y desactiva el AutoCommit
        conn.setAutoCommit(false);
        String sql = "INSERT INTO departamento VALUES (?,?,?)";
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, cod);
        ps.setString(2, nom);
        ps.setInt(3, dir);
        ps.executeUpdate();
        System.out.println("Departamento insertado");

                        //más cosas…
        conn.commit();
    } catch(SQLException e) {
        System.out.println("Error al insertar");
        conn.rollback()
    }
}
```

11.2. Concurrencia de Transacciones

- Los **sistemas de procesamiento de transacciones** son sistemas con grandes bases de datos y cientos de usuarios concurrentes que ejecutan transacciones
  - Sistemas de reservas en aerolíneas, sistemas bancarios, procesamiento de tarjetas de crédito, supermercados, compra de entradas para conciertos, venta de productos online, etc.
- Requieren alta disponibilidad (24/7/365), y respuesta rápida para sus cientos de usuarios
- Las *transacciones* emitidas por diferentes usuarios *suelen ejecutarse concurrentemente, y pueden acceder a y actualizar los mismos elementos* de BD
  - Se consigue la disponibilidad de datos compartidos actualizados
  - Pero el acceso simultáneo a los datos debe hacerse sin interferencias ni inconsistencias

Control de la Concurrencia

- Garantiza la **actualización correcta** de la base de datos cuando hay múltiples usuarios modificando de manera concurrente la base de datos

T1: ***Transferir 200€ de la cuenta X a otra cuenta Y***

1. SELECT saldo
   FROM cuenta
   WHERE ccc = X;
2. SI (saldo<200) ENTONCES
   ko_saldo_insuficiente(saldo);
3. UPDATE cuenta
   SET saldo=saldo – 200
   WHERE ccc = X;
4. UPDATE cuenta
   SET saldo=saldo + 200
   WHERE ccc = Y;

T2: ***Ingresar 100€ en la cuenta X***

1. SELECT saldo
   FROM cuenta
   WHERE ccc = X;
2. solicitar_confirmacion() ;
3. UPDATE cuenta
   SET saldo=saldo + 100
   WHERE ccc = X;

Control de la Concurrencia

¿Qué pasa si la concurrencia no se controla?

ⓘLa **ejecución** de las transacciones es **intercalada**

```
T1.1 SELECT saldo FROM cuenta
        WHERE ccc = X;
T1.2 SI (saldo<200) ENTONCES
        ko_saldo_insuficiente(saldo);
T2.1 SELECT saldo FROM cuenta
        WHERE ccc = X;
T1.3 UPDATE cuenta SET saldo=saldo – 200
        WHERE ccc = X;
T2.2 solicitar_confirmacion() ;
T1.4 UPDATE cuenta SET saldo=saldo + 200
        WHERE ccc = Y;
T2.3 UPDATE cuenta SET saldo=saldo + 100
        WHERE ccc = X;
```

Saldos iniciales X: 1000€
Y: 500€

T1 **lee** X.saldo=1000

T2 **lee** X.saldo=1000

T1 calcula X.saldo=1000-200
y **escribe X.saldo=800**

T1 **lee** Y.saldo=500,
calcula Y.saldo=500+200
y **escribe** Y.saldo=700

T2 calcula X.saldo=1000+100
y **escribe X.saldo=1100** **¡¡ERROR!!**

- T2 *lee* X.saldo de *antes* de que T1 lo modifique, y lo *actualiza después* de que lo haga T1. Consecuencia: ***Actualización Perdida***

Control de la Concurrencia

- La **ejecución intercalada** (concurrente) de las transacciones puede provocar **conflictos** e **interferencias** destructivas
  - Problema de la actualización perdida, lectura sucia, etc.
- Objetivo del SGBD: **planificar las transacciones** de forma tal que su **ejecución** sea **concurrente** **y** al mismo tiempo se **evite todo conflicto** entre ellas…
- … por lo que su **efecto** es **equivalente a la *ejecución en serie*** de las mismas transacciones
- Se consigue ***planificaciones serializables***

La **planificación serie equivalente** es **T3→T1→T2**

T1	*T2*	*T3*
		1.leer(Y);
		2.leer(Z);
3.leer(X);		
4.escribir(X);		
		5.escribir(Y);
		6.escribir(Z);
	7.leer(Z);	
8.leer(Y);		
9.escribir(Y);		
	10.leer(Y);	
	11.escribir(Y);	
	12.leer(X);	
	13.escribir(X);	

Control de la Concurrencia

- ¿Y cómo lo consigue el SGBD?
  Utilizando **técnicas de control de la concurrencia**
- El **bloqueo** (***locking***) es una de las más empleadas en los SGBD comerciales
- Otras que no veremos, pero están en la bibliografía, son ...
  - Métodos de Marcas de Tiempo (*timestamping*)
  - Técnicas de Multiversión
  - Técnicas o protocolos Optimistas

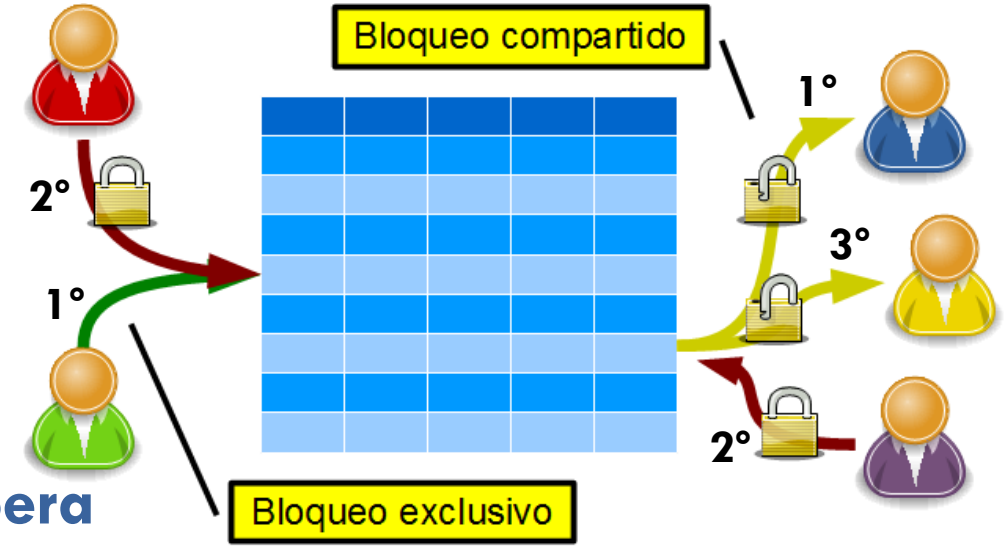
Bloqueo

- Uso de candados/cerrojos/**bloqueos** para controlar el acceso concurrente o simultáneo a los datos de la BD

- **Tipos** de bloqueos
  - **Bloqueo Compartido** (de lectura)
    - Si T tiene un bloqueo compartido sobre un elemento de datos, T puede **leer** el elemento
    - Otras transacciones pueden **leer** el mismo elemento al mismo tiempo
    - Ni la transacción que lee, ni ninguna otra, pueden escribir el elemento
  - **Bloqueo Exclusivo** (de escritura)
    - Si T tiene un bloqueo exclusivo sobre un elemento de datos, **sólo** dicha transacción puede **leer y escribir** el elemento
    - El resto de las transacciones no pueden leer ni escribir ese elemento

Bloqueos: reglas de uso

- Si una transacción necesita **leer un elemento** **X**, antes debe obtener un **bloqueo compartido** sobre **X**
  *(si no lo consigue, **espera**)*
- Si una transacción desea **escribir un elemento** **Y**, antes debe conseguir un **bloqueo exclusivo** sobre **Y**
  *(si no lo consigue, **espera**)*
- Una vez realizada la operación, la transacción debe **desbloquear** el elemento de datos
- Una operación de **COMMIT** o **ROLLBACK libera todos los bloqueos** adquiridos por la transacción



Accesos no permitidos por no conseguir el bloqueo correspondiente

Bloqueos y Niveles de Aislamiento

- Los SGBD utilizan los **bloqueos** y otros protocolos de control de concurrencia de forma **implícita**
  - Automática. Transparente
  - Los programadores **no han de añadirlos a su código**
- Y permiten que el usuario/programador pueda **elegir** el **nivel de aislamiento para cada transacción** T
  - El nivel de aislamiento de T define cuál es el **grado de interacción de T con el resto de las transacciones**

▶ Los SGBD **implementan** los niveles de **aislamiento utilizando bloqueos**

Esta es una propiedad ACID de las transacciones, que el Subsistema de Concurrencia del SGBD debe garantizar

Niveles de Aislamiento en ANSI SQL

- Sentencia del ANSI SQL que permite definir el nivel de aislamiento para una transacción:

  SET TRANSACTION ISOLATION LEVEL nivel;

- Debe ser la 1ª sentencia de la transacción
- Valores posibles para "nivel":
  - **READ UNCOMMITED**
  - **READ COMMITTED**
  - **REPEATABLE READ**
  - **SERIALIZABLE** ←Aislamiento absoluto
    - Garantizar el aislamiento absoluto de las transacciones no siempre es necesario, y puede disminuir el rendimiento. Se suele <u>relajar el aislamiento</u> eligiendo el resto de niveles

Niveles de Aislamiento en ANSI SQL

- Los niveles de aislamiento se definen en función de los **problemas** –provocados por la interacción entre transacciones concurrentes – **que permiten y evitan**

Nivel de Aislamiento	Lectura sucia	Lectura no repetible	Lectura Fantasma
READ UNCOMMITED	Posible	Posible	Posible
READ COMMITED	No	Posible	Posible
REPEATABLE READ	No	No	Posible
SERIALIZABLE	No	No	No

- El nivel activo por defecto suele ser el SERIALIZABLE, aunque para algunos SGBD (como Oracle) es el nivel READ COMMITTED

Problemas según el nivel de aislamiento

- **Lectura sucia** (Dirty Read)
  - Una transacción **T1 actualiza** ciertos **datos**, después otra transacción **T2 lee** dichos datos (ya modificados por T1 que aún no se ha confirmado)
  - **Si T1 ejecuta ROLLBACK** y los datos vuelven al estado original
  - En ese momento, **los datos leídos por T2 no son válidos:** son fruto de una lectura sucia
- **Lectura no repetible** (Non-repeatable Read)
  - Después de que una transacción **T1 lea** ciertos datos de la BD, otra transacción **T2 los actualiza** y se **confirma**
  - Cuando **T1 lee de nuevo** los mismos datos encontrará que **han cambiado:** es una lectura no repetible, los mismos datos leídos 2 veces son diferentes
- **Lectura fantasma** (Phantom Read)
  - Una transacción **T1** ejecuta una **consulta, otras** transacciones **insertan nuevas** filas y se **confirman**
  - La transacción **T1** ejecuta de nuevo **la misma consulta** y el resultado muestra las **nuevas filas** (fantasmas)

Niveles de Aislamiento en ANSI SQL

- **READ UNCOMMITED**
  - Es posible que se **lean datos modificados por transacciones no confirmadas**
  - Los **bloqueos de lectura y de escritura** son **liberados** en cuanto **finaliza la sentencia**

T1	T2 (READ UNCOMMITED)
	SELECT SUM(salario) -- *suma 1* FROM empleado;
UPDATE empleado SET salario = salario *1.05 WHERE nss IN (SELECT nssemp FROM familiar);	
	SELECT AVG(salario) FROM empleado; T2 ve el UPDATE de T1 (*Dirty Read*) y calcula la media con los **nuevos** datos
COMMIT;	
	SELECT SUM(salario) FROM empleado; El resultado de la suma es diferente a la suma 1 (*Nonrepeatable Read*)

Niveles de Aislamiento en ANSI SQL

READ COMMITTED

- Siempre se **leen datos modificados por transacciones confirmadas**
- **Cada sentencia** dentro de la transacción **sólo ve los datos confirmados antes del inicio de la sentencia** (no de la transacción)
- Se consigue haciendo que los **bloqueos de lectura** sean **liberados** en cuanto **finaliza la SELECT** y los **bloqueos de escritura no** sean **liberados hasta el final de la transacción**

T1	T2 (READ COMMITED)
	SELECT SUM(salario) -- *suma 1* FROM empleado;
UPDATE empleado SET salario = salario*1.05 WHERE nss IN (SELECT nssemp FROM familiar);	T2 NO ve el UPDATE de T1 porque T1 aún no se ha confirmado. Calcula la media con los mismos datos que la **suma1**
	SELECT AVG(salario)FROM empleado;
COMMIT;	
T2 ya ve el UPDATE de T1 confirmada y calcula la suma con los nuevos datos	SELECT SUM(salario) FROM empleado; El resultado de esta suma es DIFERENTE a la **suma1** (*Nonrepeatable Read*)

Niveles de Aislamiento en ANSI SQL

REPEATABLE READ

- Se realizan **siempre lecturas 'repetibles'**: dentro de una transacción **siempre se obtienen los mismos valores en diferentes lecturas sucesivas de los mismos datos** ▶Pero pueden ocurrir lecturas 'fantasma'
- Se consigue haciendo que los **bloqueos de lectura y escritura no** sean **liberados hasta el final de la transacción**

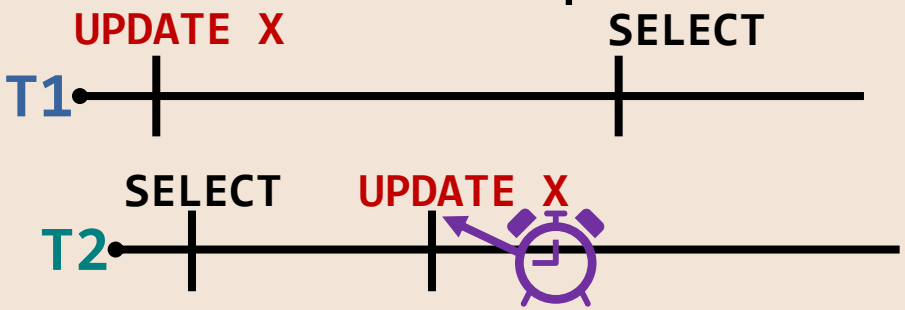
T1 (READ UNCOMMITED)	T2 (REPEATABLE READ)
	SELECT SUM(salario) -- *suma 1* FROM empleado;
UPDATE empleado SET salario = salario*1.05 WHERE nss IN (SELECT nssemp FROM familiar);	T2 calcula la media con los mismos datos que la **suma1**
T1 NO puede completar el UPDATE porque T2 tiene bloqueadas las filas de EMPLEADO	SELECT AVG(salario)FROM empleado;
el prompt no aparece... (T1 espera)	SELECT SUM(salario) FROM empleado;
	COMMIT; El resultado de esta suma es idéntico a la **suma1** (*Repeatable Read*)
T1 completa el UPDATE con éxito, porque T2 se ha confirmado	

Niveles de Aislamiento en ANSI SQL

- **SERIALIZABLE - Aislamiento absoluto**
  - Cada sentencia dentro de la transacción sólo **ve los datos confirmados antes del inicio de la transacción** y sus propios cambios
  - Se consigue haciendo que los **bloqueos de lectura** y **escritura no** sean **liberados hasta el final de la transacción**
  - **Y**, además, realizando '**bloqueos de rango**': se bloquean los datos seleccionados con cada SELECT...FROM...WHERE
    - Estos bloqueos de rango evitan las lecturas 'fantasma'

Control de Concurrencia en Oracle

- Para conseguir los niveles de aislamiento *Oracle* impone **bloqueos a nivel de fila**
- Y obliga a que una transacción T2 espere si intenta modificar una fila que ha sido actualizada por otra T1 no confirmada



  - Si **T1 falla**, deshace sus cambios (ROLLBACK) y libera sus bloqueos, así que **T2** puede modificar la fila y continuar
  - Si **T1 finaliza con COMMIT** y libera sus bloqueos, entonces…
    - Si **T2** está en modo READ COMMITED, T2 puede modificar la fila
    - Si **T2** está en modo SERIALIZABLE, T2 falla: devuelve un ERROR indicando que se ha infringido la serialización (ver ejemplo 2)

Control de Concurrencia en Oracle

- *Resumen de las reglas de comportamiento de los bloqueos para lectores y escritores:*

  - ***Una fila es bloqueada sólo cuando es modificada por un escritor***
    - Cuando una sentencia actualiza una fila, la transacción adquiere un bloqueo sólo para esa fila
    - Así se maximiza la concurrencia: otras transacciones pueden acceder a <u>otras</u> filas de la tabla

  - ***Un escritor de una fila bloquea a otros escritores de la misma fila concurrentes***
    - Si una transacción está modificando una fila, un bloqueo de fila impide que otra transacción modifique la misma fila simultáneamente

Control de Concurrencia en Oracle

- *Resumen de las reglas de comportamiento de los bloqueos para lectores y escritores (cont.)*

  - ***Un lector nunca bloquea a un escritor***
    - Oracle no bloquea filas para lectura *(gracias al modelo de consistencia multiversión, explicado en el Anexo 1)*, por lo que un escritor puede modificarlas
    - La excepción es la sentencia `SELECT ... FOR UPDATE`, que bloquea las filas que se están leyendo *(se ve en una diapositiva posterior)*

  - ***Un escritor nunca bloquea a un lector***
    - *(Gracias al modelo de consistencia multiversión, explicado en el Anexo 1)*

Concurrencia y Niveles de Aislamiento en la práctica: sesiones interactivas Oracle

Ejemplo 1

Session 1	Session 2	Explanation
SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 6200 Greene 9500		Session 1 queries the salaries for Banda, Greene, and Hintz. No employee named Hintz is found.
SQL> UPDATE employees SET salary = 7000 WHERE last_name = 'Banda';		Session 1 begins a transaction by updating the Banda salary. The default isolation level for transaction 1 is READ COMMITTED.
	SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;	Session 2 begins transaction 2 and sets the isolation level explicitly to READ COMMITTED.
	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 6200 Greene 9500	Transaction 2 queries the salaries for Banda, Greene, and Hintz. Oracle Database uses read consistency to show the salary for Banda before the uncommitted update made by transaction 1.
	SQL> UPDATE employees SET salary = 9900 WHERE last_name = 'Greene';	Transaction 2 updates the salary for Greene successfully because transaction 1 locked only the Banda row
SQL> INSERT INTO employees (employee_id, last_name, email, hire_date, job_id) VALUES (210, 'Hintz', 'JHINTZ', SYSDATE, 'SH_CLERK');		Transaction 1 inserts a row for employee Hintz, but does not commit.

T2 (**READ COMMITED**): cada sentencia verá los datos modificados por transacciones confirmadas ANTES de dicha **sentencia**

S2 (T2) no ve el UPDATE de S1 (T1) porque **no** está confirmado

Session 1	Session 2	Explanation
	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 6200 Greene 9900	Transaction 2 queries the salaries for employees Banda, Greene, and Hintz. Transaction 2 sees its own update to the salary for Greene. Transaction 2 does not see the uncommitted update to the salary for Banda or the insertion for Hintz made by transaction 1.
	SQL> UPDATE employees SET salary = 6300 WHERE last_name = 'Banda'; -- prompt does not return	Transaction 2 attempts to update the row for Banda, which is currently locked by transaction 1, creating a conflicting write. Transaction 2 waits until transaction 1 ends.
SQL> COMMIT; T1		Transaction 1 commits its work, ending the transaction.
	1 row updated. SQL>	The lock on the Banda row is now released, so transaction 2 proceeds with its update to the salary for Banda.
	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 6300 Greene 9900 Hintz	Transaction 2 queries the salaries for employees Banda, Greene, and Hintz. The Hintz insert committed by transaction 1 is now visible to transaction 2. Transaction 2 sees its own update to the Banda salary.
	COMMIT; T2	Transaction 2 commits its work, ending the transaction.
SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 6300 Greene 9900 Hintz		Session 1 queries the rows for Banda, Greene, and Hintz. The salary for Banda is 6300, which is the update made by transaction 2. The update of Banda's salary to 7000 made by transaction 1 is now "lost."

S2 (T2) ve su propio UPDATE del salario de Greene, y NO VE el de Banda hecho por T1, ni la inserción de Hintz

S2 (T2) ve el INSERT hecho por S1 porque T1 se confirmó antes de esta SELECT

S2 (T2) ve su propio UPDATE del salario de Banda, y NO el de S1 (T1)

S1 ve el UPDATE sobre Banda y Green de S2 porque T2 se confirmó antes de esta SELECT

S1 "ha perdido" su propio UPDATE sobre Banda, porque se reescribió con el de T2: el 7000 SE PERDIÓ

Concurrencia y Niveles de Aislamiento en la práctica: sesiones interactivas Oracle

Ejemplo 2

Session 1	Session 2	Explanation
SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 6200 Greene 9500		Session 1 queries the salaries for Banda, Greene, and Hintz. No employee named Hintz is found.
SQL> UPDATE employees SET salary = 7000 WHERE last_name = 'Banda';	T2 (**SERIALIZABLE**) sólo verá los datos modificados por transacciones confirmadas ANTES de su comienzo	Session 1 begins transaction 1 by updating the Banda salary. The default isolation level for is READ COMMITTED.
	SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	Session 2 begins transaction 2 and sets it to the SERIALIZABLE isolation level.
S2 (T2) no ve el UPDATE de S1 (T1) sobre Banda porque T1 **no** está confirmada	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 6200 Greene 9500	Transaction 2 queries the salaries for Banda, Greene, and Hintz. Oracle Database uses read consistency to show the salary for Banda *before* the uncommitted update made by transaction 1.
	SQL> UPDATE employees SET salary = 9900 WHERE last_name = 'Greene';	Transaction 2 updates the Greene salary successfully because only the Banda row is locked.
SQL> INSERT INTO employees (employee_id, last_name, email, hire_date, job_id) VALUES (210, 'Hintz', 'JHINTZ', SYSDATE, 'SH_CLERK');		Transaction 1 inserts a row for employee Hintz.
SQL> COMMIT; T1		Transaction 1 commits its work, ending the transaction.

Session 1	Session 2	Explanation
SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 7000 Greene 9500 Hintz	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 6200 Greene 9900	Session 1 queries the salaries for employees Banda, Greene, and Hintz and sees changes committed by transaction 1. Session 1 does not see the uncommitted Greene update made by transaction 2. Database read consistency ensures that the Hintz insert and Banda update committed by transaction 1 are *not* visible to transaction 2. Transaction 2 sees its own update to the Greene salary.
	COMMIT; ← T2	Transaction 2 commits its work, ending the transaction.
SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 7000 Greene 9900 Hintz	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 7000 Greene 9900 Hintz	Both sessions query the salaries for Banda, Greene, and Hintz. Each session sees all committed changes made by transaction 1 and transaction 2.
SQL> UPDATE employees SET salary = 7100 WHERE last_name = 'Hintz';		Session 1 begins transaction 3 by updating the Hintz salary. The default isolation level for transaction 3 is READ COMMITTED.
	SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	Session 2 begins transaction 4 and sets it to the SERIALIZABLE isolation level.
	SQL> UPDATE employees SET salary = 7200 WHERE last_name = 'Hintz'; -- prompt does not return	Transaction 4 attempts to update the salary for Hintz, but is blocked because transaction 3 locked the Hintz row Transaction 4 queues behind transaction 3.
SQL> COMMIT; ← T3		Transaction 3 commits its update of the Hintz salary, ending the transaction.

S1 ve sus propios UPDATE e INSERT pues T1 está confirmada
S1 NO ve el UPDATE de S2 sobre Greene porque T2 no se ha confirmado

El UPDATE de S2 sobre Greene (T2) tuvo éxito porque la fila no estaba bloqueada por T1

S2 NO ve el UPDATE ni el INSERT de T1 porque T1 se confirmó después del inicio de T2 (SERIALIZABLE)

S1 y S2 ven todos los cambios hechos por T1 y T2 por estar confirmadas

S2 (T4) NO puede hacer el UPDATE porque S1 (T3) está haciendo UPDATE de esa misma fila

El UPDATE de S2 (T4) falla al infringir la serialización: S1 (T3) confirmó su UPDATE después de que T4 comenzara

Session 1	Session 2	Explanation

	`UPDATE employees SET salary = 7200 WHERE last_name = 'Hintz'`
	`*`
	`ERROR at line 1:`
	`ORA-08177: can't serialize access for this transaction`
		The commit that ends transaction 3 causes the Hintz update in transaction 4 to fail with the ORA-08177 error. The problem error occurs because transaction 3 committed the Hintz update *after* transaction 4 began.

	`SQL> ROLLBACK;` ← T4	Session 2 rolls back transaction 4, which ends the transaction.
	`SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;`	Session 2 begins transaction 5 and sets it to the SERIALIZABLE isolation level.

	`SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz');`
	
	`LAST_NAME SALARY`
	`---------- ----------`
	`Banda 7000`
	`Greene 9900`
	`Hintz 7100`
		Transaction 5 queries the salaries for Banda, Greene, and Hintz. The Hintz salary update committed by transaction 3 is visible.

	`SQL> UPDATE employees SET salary = 7200 WHERE last_name = 'Hintz';`
	
	`1 row updated.`
		Transaction 5 updates the Hintz salary to a different value. Because the Hintz update made by transaction 3 committed *before* the start of transaction 5, the serialized access problem is avoided.
		
		Note: If a different transaction updated and committed the Hintz row after transaction 5 began, then the serialized access problem would occur again.

	`SQL> COMMIT;` ← T5	Session 2 commits the update without any problems, ending the transaction.

S2 (T5) ve el UPDATE sobre Hintz hecho por S1 (T3) porque T3 se confirmó antes del inicio de T5 (SERIALIZABLE)

El UPDATE tiene éxito porque T3 se confirmó antes del inicio de T5
La tabla employees queda así:

LAST_NAME	SALARY
Banda	7000
Greene	9900
Hintz	7200

Anexo

Acceso a BD vía JDBC: **concurrencia**

- Establecer el modo de aislamiento:

```
conexion.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
```

- Consejos de uso:
  - Bloque con sólo actualizaciones:
    - TRANSACTION_READ_COMMITED
  - Bloque donde leamos varias veces el mismo registro:
    - TRANSACTION_REPEATABLE_READ
  - Bloque en el que leamos un valor para actualizarlo:
    - TRANSACTION_SERIALIZABLE
  - Bloque donde realicemos varias veces la misma consulta (varios registros):
    - TRANSACTION_SERIALIZABLE

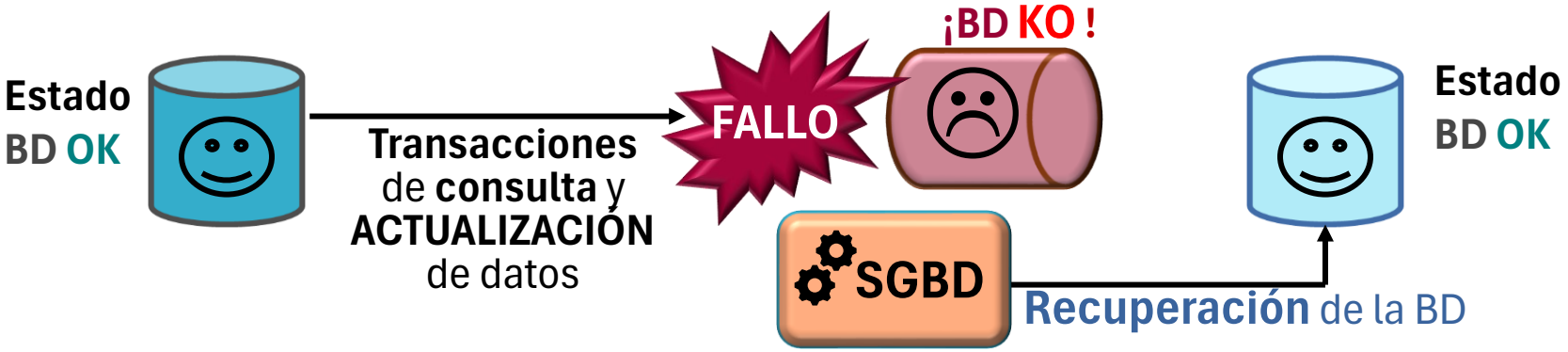
Anexo

Acceso a BD vía JDBC: **aislamiento**

- Niveles de aislamiento:
  - TRANSACTION_NONE
    - Sin soporte transaccional
  - TRANSACTION_READ_UNCOMMITED
    - Permite lecturas sobre datos no consolidados
  - TRANSACTION_READ_COMMITED
    - Permite lecturas sólo sobre datos consolidados
    - Nivel por defecto
  - TRANSACTION_REPEATABLE_READ
    - Bloquea los datos leídos
  - TRANSACTION_SERIALIZABLE
    - Sólo una transacción al mismo tiempo

11.3. Recuperación de fallos

- El SGBD ofrece un mecanismo para **recuperar la base de datos** en caso de que resulte **afectada** de alguna forma **por** un **fallo que afecte a su procesamiento**
  - La ocurrencia de **fallos** puede provocar la **pérdida de datos en los medios de almacenamiento**
  - El SGBD puede **devolver la BD a un estado coherente**



Tipos de fallos

Fallo local

- Sólo afecta a la transacción T fallida
- Pérdida de *datos correspondientes a T* en memoria principal

1. **Fallo local previsto** por la aplicación
   - Cancelación 'programada', excepción tratada
   - *«un saldo insuficiente cancela una transacción de reintegro»*
2. **Fallo local no previsto**
   - Error de programación (*bug*), división por cero, desbordamiento, interrupción provocada por el usuario
3. **Fallo por** imposición del control de **concurrencia**
   - El método de control de concurrencia decide abortar T porque incumple la serialización, o bien para romper un interbloqueo
     - Reiniciada más tarde

Tipos de fallos

Fallo global

- Afecta a todas las transacciones en ejecución
- Pérdida de datos en memoria principal, y quizá en el almacenamiento secundario

4. **Fallo del sistema** (caída suave)
   - Mal funcionamiento hardware, software (SGBD, SO), o red
   - No daña el disco: el contenido de la BD queda intacto
5. **Fallo de los medios de almacenamiento** (caídas duras)
   - 'Aterrizaje' de cabezas lectoras de disco, o soportes no legibles, etc.
   - Algunos bloques del disco pueden perder sus datos: BD corrupta
6. **Fallos físicos y catástrofes**
   - Desastre natural: inundación, terremoto, incendio, apagón...
   - Robo, sabotaje, destrucción o corrupción de datos, de hardware, de software o de las instalaciones (de modo intencionado o negligente)

Efectos de la ocurrencia de un fallo

- Sea cual sea la causa del fallo, hay 2 efectos principales que considerar
  - **Pérdida de la memoria principal**
    incluyendo la **caché de la base de datos**
    y otros **búferes** reservados por el SGBD
  - **Pérdida de datos** de la BD en el **almacenamiento** secundario
- Es necesario **minimizar estos efectos** mediante **conceptos, protocolos y técnicas** que permitan recuperar la BD tras un fallo del sistema
- *Recuperar la BD es restaurar un estado previo al fallo, desde el que se pueda reconstruir un estado consistente y cercano al momento del fallo*

La tarea del Gestor de Recuperación

- La función del subsistema ***Gestor de Recuperación*** del SGBD es garantizar la Atomicidad y Durabilidad (2 propiedades ACID de las transacciones) incluso cuando ocurran fallos
  - Debe asegurar que, aun en presencia de fallos, cada transacción T **ejecuta todas sus operaciones con éxito y su efecto queda permanente** en la BD,

    … o bien que **no tiene ningún efecto en la BD ni sobre otras transacciones**
  - Nunca deben ejecutarse sólo algunas operaciones de T
  - Ni siquiera por culpa de un **fallo** 'a mitad de T'
- Y todo ello teniendo en cuenta que **la escritura en la base de datos no es un proceso de un solo paso**, sino que los cambios primero se almacenan en memoria principal (caché de BD) y de forma asíncrona se vuelcan a disco

Estructuras de Memoria

- **Caché de Base de Datos**
  - Uno o más búferes en **memoria principal** que se utilizan para transferir (bloques de) datos **desde y hacia el disco**
  - Es un **área común a todas las transacciones**
    - Una vez que un bloque es copiado a la caché de BD, el bloque queda **accesible a todas las transacciones** en ejecución
    - Por tanto, si una T escribe un elemento X, el **nuevo valor** queda almacenado en la caché de BD en memoria y **el resto de transacciones pueden verlo**
    - De hecho, toda operación **leer** y **escribir busca primero el bloque en la caché de BD en memoria**, y si no está ahí, entonces lo busca en el disco
  - Sólo cuando se vuelcan los bloques de datos modificados desde la caché de BD a disco, se puede considerar permanente esa modificación
  - ¿Y **cuándo** se realiza un volcado de la caché de BD al disco?
    - Tras ejecutar un **comando específico** (p.ej. `COMMIT`), o
    - **Automáticamente** cuando la caché se llena
      - ★ Es el SGBD el que decide cuándo y qué volcar a disco

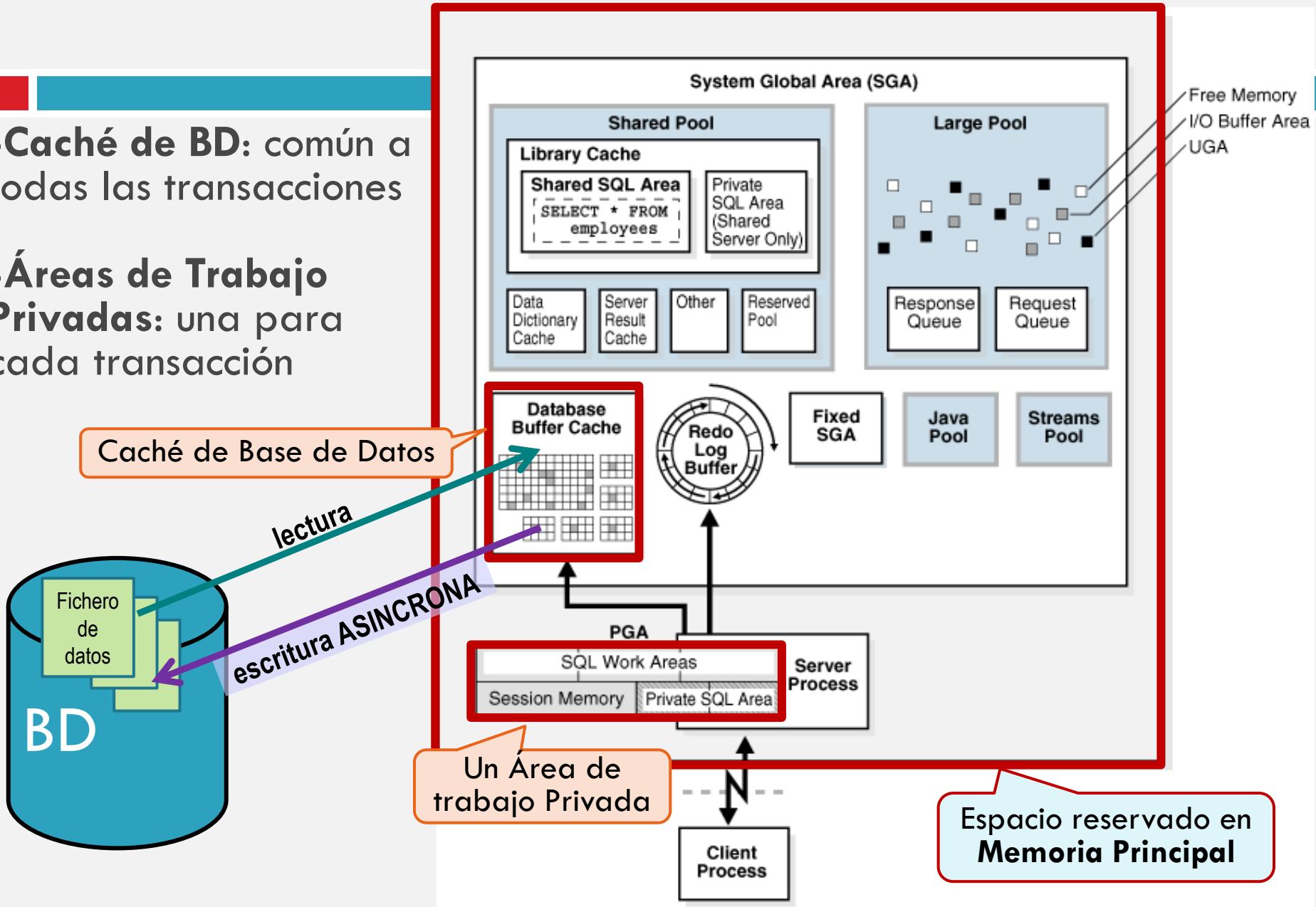
Estructuras de Memoria

- **Áreas de trabajo privadas**
  - Cada transacción T tiene asignada un área en memoria principal donde guarda todo elemento de datos que lee y/o escribe
  - Es un espacio en **memoria principal** y **local a la transacción T**
  - Se crea al iniciarse T y se elimina cuando T finaliza

Estructuras de memoria en SGBD Oracle

-**Caché de BD:** común a todas las transacciones

-**Áreas de Trabajo Privadas:** una para cada transacción



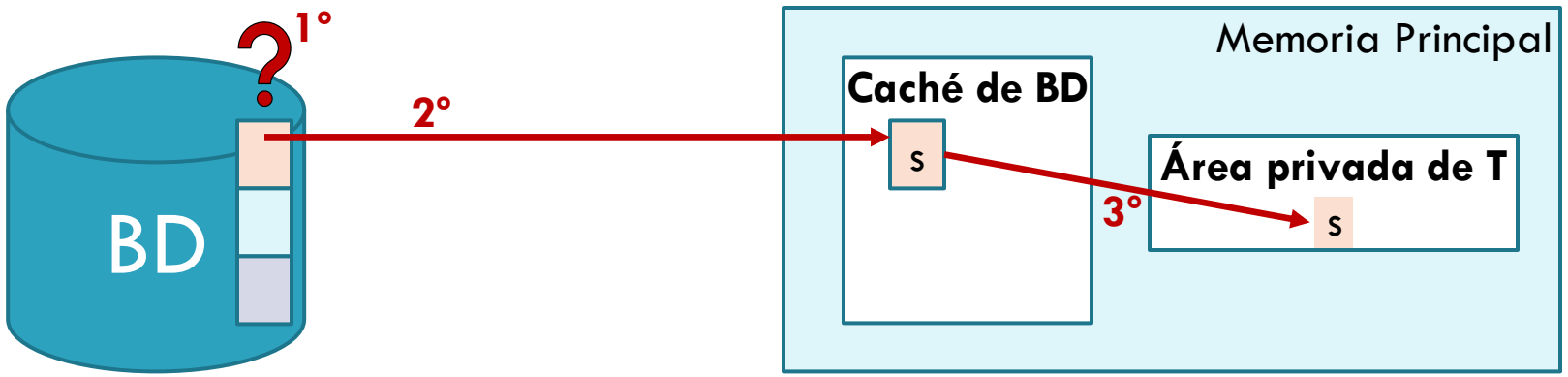
Lectura de la base de datos


```
UPDATE empleado
SET salario = salario * 1.1
WHERE nss = 111;
```

```
leer(nss=111, salario)
salario = salario * 1.1
escribir(nss=111, salario)
```

- Para implementar una operación **leer**, el SGBD realiza estos pasos:
  - Encuentra la dirección del bloque de **disco** que contiene el registro con valor 111 en el campo clave primaria
  - **Transfiere ese bloque a un búfer en memoria principal** (en la **caché de BD**)
  - Copia el valor del campo "salario" del registro desde el búfer de BD a la variable "salario" (en el **área privada** de la transacción en memoria principal)

Sólo si el bloque no está ya en la caché de BD



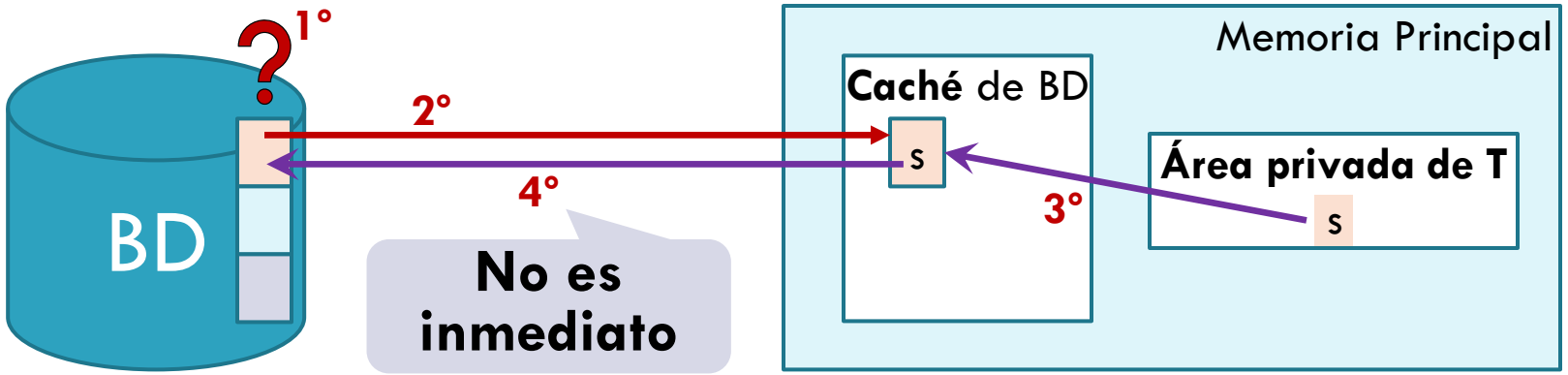
Escritura en la base de datos


```
UPDATE empleado
SET salario = salario * 1.1
WHERE nss = 111;
```

```
leer(nss=111, salario)
salario = salario * 1.1
escribir(nss=111, salario)
```

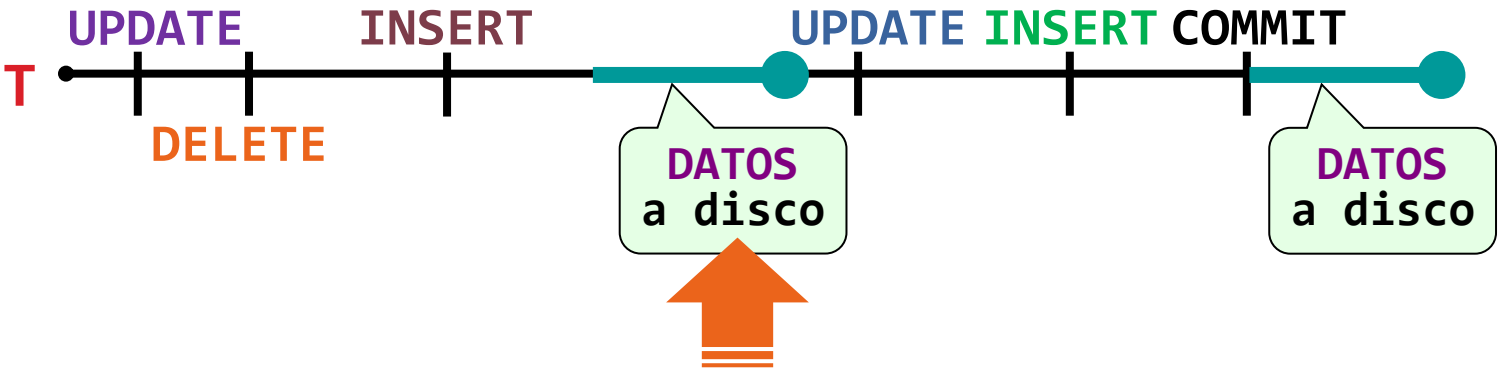
- Para implementar una operación **escribir**, el SGBD realiza estos pasos:
  - Encuentra la dirección del bloque de **disco** que contiene el registro con valor 111 en la clave primaria
  - **Transfiere el bloque a un búfer en memoria principal** (en la caché de BD)
  - Copia el valor de la variable "salario" (área privada de la transacción en memoria principal) en el campo "salario" del registro en el búfer (caché de BD)
  - **Transfiere el registro desde la caché de BD a** su localización en **disco**

Sólo si el bloque no está ya en la caché de BD



Escritura en la base de datos

- Una transacción T **puede modificar la BD al finalizar con éxito** su ejecución, tras emitir COMMIT…
- Y también **durante su ejecución**, antes de emitir COMMIT
  - Por ejemplo, porque sea necesario hacer hueco en la caché de datos
- **Por tanto, algunos cambios en los datos** realizados por T **pueden consolidarse en disco antes de confirmarse T**
  - Son '**modificaciones no comprometidas**'



‘Armas’ para la recuperación de fallos

¿Cómo se consigue recuperar una base de datos?

- Pues con **Redundancia** + **Acciones** de Recuperación
- Un SGBD debe proporcionar estas facilidades para ayudar con la recuperación:

REDUNDANCIA

- Facilidades de ***backup***, para realizar copias periódicas de la base de datos
- Mecanismo de **bitácora**, para mantener la pista del estado actual de las transacciones y los cambios de los datos

ACCIONES DE RECUPERACIÓN

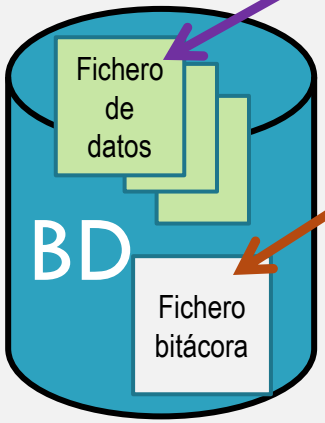
- Un protocolo de ***checkpoints***
- Un *Gestor de Recuperación* (*Recovery Manager*; subsistema del SGBD), que permita al sistema restaurar la BD a un estado consistente tras un fallo, mediante **técnicas y estrategias de recuperación**

Fichero Bitácora

- Fichero especial que **almacena detalles sobre las operaciones** efectuadas por **las transacciones**
  - También diario, *log*, *journal*, registro histórico…
- Se mantiene en el **almacenamiento secundario** (disco)
  - En un **área distinta** a donde se almacenan los datos de la BD
    - No le afecta ningún tipo de fallo, salvo los de tipo 5 y 6
  - Se puede duplicar o triplicar (mantener 2 o 3 copias independientes, en uno o varios discos)
    - Si una copia es dañada, se puede utilizar otra
  - Se suele realizar copias de seguridad periódicas
- Y en la memoria principal, obviamente, se mantiene el **búfer de bitácora** (~caché)
  - Se va rellenando con **entradas hasta que se llena**, momento en el que **se vuelca a disco**

Estructuras de memoria en el *SGBD Oracle*

- Caché de BD
- Áreas de trabajo Privadas
- Búfer de Bitácora



Instance

System Global Area (SGA)

Shared Pool

Library Cache

Shared SQL Area

SELECT * FROM employees

Private SQL Area (Shared Server Only)

Data Dictionary Cache

Server Result Cache

Other

Reserved Pool

Large Pool

Response Queue

Request Queue

Free Memory

I/O Buffer Area

UGA

Database Buffer Cache

Redo Log Buffer

Fixed SGA

Java Pool

Streams Pool

Búfer de Bitácora

PGA

SQL Work Areas

Session Memory

Private SQL Area

Server Process

Client Process

Espacio en Memoria Principal

Entradas en el fichero Bitácora

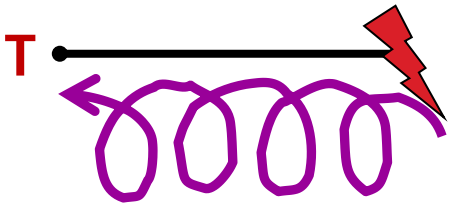
Cada registro del fichero se denomina **entrada**, que será de uno de estos tipos:

- **<INICIAR, T>**
  - Indica que la transacción T ha comenzado su ejecución
- **<LEER, T, x>**
  - Indica que T ha leído el valor del elemento x de la base de datos
- **<ESCRIBIR, T, x, valor_anterior, valor_nuevo>**
  - Indica que T ha modificado el valor del elemento x
- **<COMMIT, T>**
  - Indica que T ha finalizado con éxito y su efecto puede ser confirmado en la BD en disco: sus cambios pueden quedar permanentes
- **<ROLLBACK, T>**
  - Indica que la transacción T ha sido anulada de forma que ninguna de sus operaciones tendrá efecto sobre la BD: la transacción será revertida, todas sus operaciones serán deshechas
- **<PUNTO DE CONTROL, ...>** → La veremos más adelante

Recuperación de UNA transacción

- La **bitácora** permite al *Gestor de Recuperación* decidir **qué hacer con una transacción** T tras ocurrir un fallo:

- Si en la bitácora **NO está la entrada <COMMIT, T>** es porque **T** estaba **en curso de ejecución**



  - **Se debe DESHACER T**

```
...
<INICIAR,T>
<LEER,T,...>
<ESCRIBIR,T,...>
<ESCRIBIR,T,...>
<LEER,T,...>
<ESCRIBIR,T,...>
...
```

Fichero Bitácora

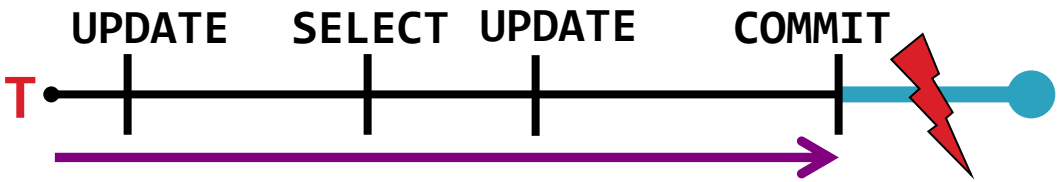
Recuperación de UNA transacción

- Si en la bitácora **SÍ está la entrada <COMMIT, T>** es que **T** había **finalizado correctamente**

```
...
<INICIAR,T>
<ESCRIBIR,T,X,10,20>
<LEER,T,Z...>
<ESCRIBIR,T,Z,1,3>
<COMMIT,T>
...
```

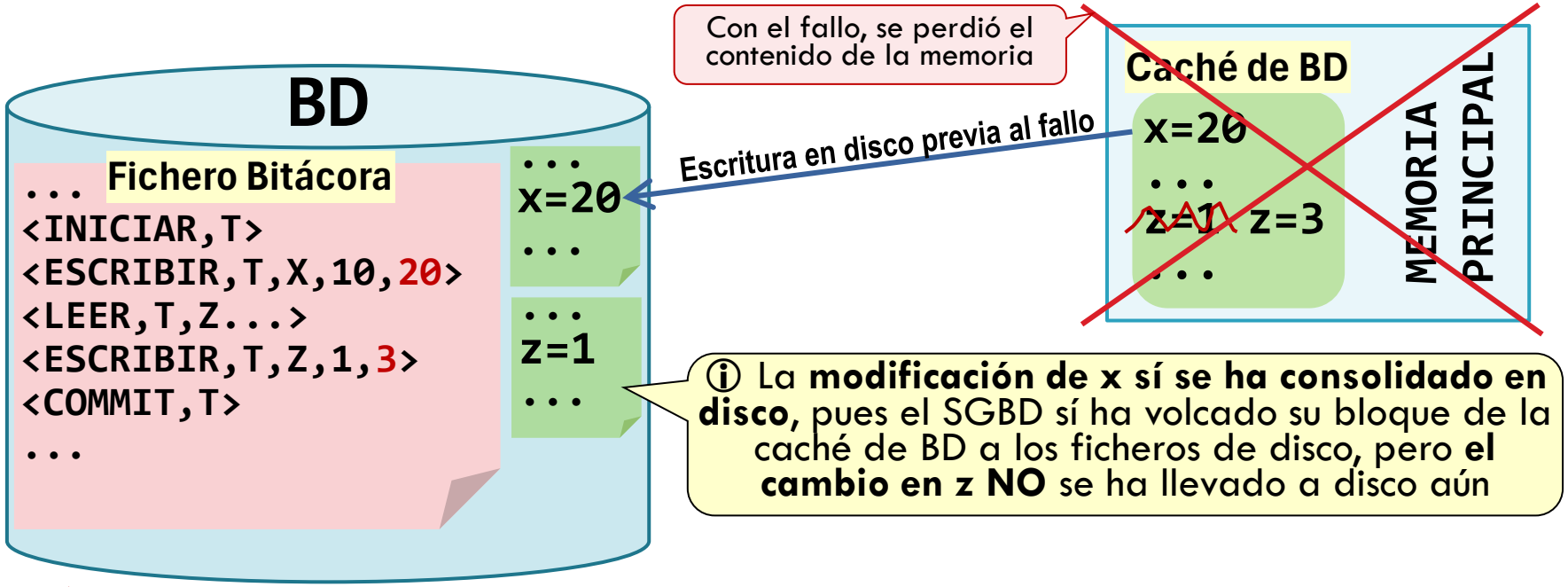
Fichero Bitácora

- **Se debe REHACER T**



Recuperación de UNA transacción

- **¿Por qué rehacer** una **T** que sabemos está **confirmada**?
  - Porque **no es seguro que todos sus cambios hayan sido llevados a la BD en disco**
  - Pueden haber quedado en memoria y haberse perdido con el fallo



El fallo ocurrió después de escribir el COMMIT de T en bitácora en disco

Recuperación de UNA transacción

- **DESHACER T** implica **deshacer** cada una de sus **operaciones de escritura**, utilizando las anotaciones en bitácora, empezando por la última (**orden inverso**)

  `<ESCRIBIR,T,X,`**`valor_anterior`**`,valor_nuevo>`

  `deshacer (<ESCRIBIR,T,X,`**`10`**`,20>)` ➡ `X=10` en la BD

- **REHACER T** implica **rehacer** cada una de sus **operaciones de escritura**, utilizando las anotaciones en bitácora, empezando por la primera (en el **mismo orden**)

  `<ESCRIBIR,T,X,valor_anterior,`**`valor_nuevo`**`>`

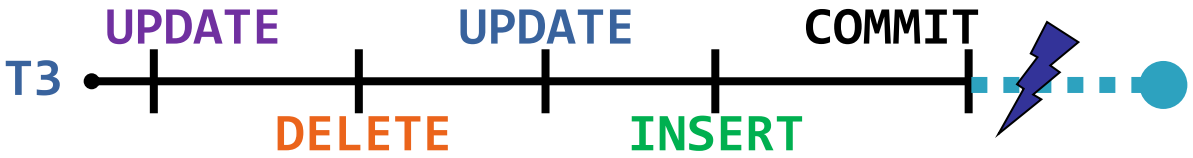
  `rehacer (<ESCRIBIR,T,X,10,`**`20`**`>)` ➡ `X=20` en la BD

Bitácora adelantada

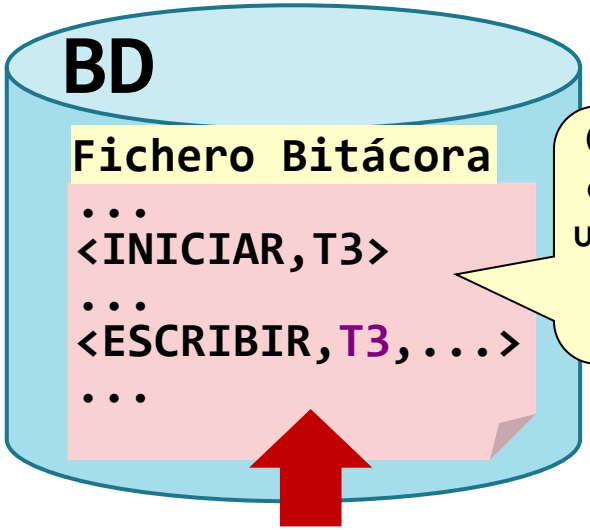
- Si ocurre un **fallo global cuando el búfer de bitácora está incompleto**, su contenido ***se pierde*** al igual que el resto de la memoria principal
- Contenía **entradas** que **aún no** habían sido **volcadas** en el fichero bitácora en disco
- Dichas entradas **no serán consideradas en el proceso de recuperación**, pues el SGBD acude al fichero bitácora para aplicar las acciones de recuperación
- Esto puede **impedir la restauración correcta** tras el fallo de una transacción

Veámoslo con un ejemplo…

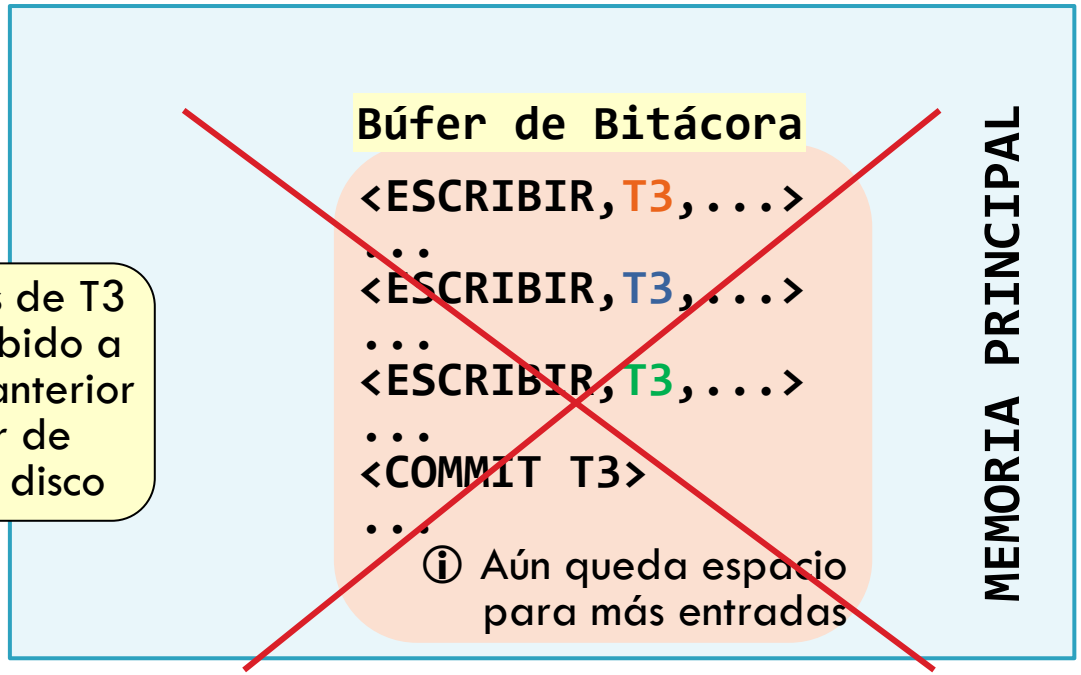
Bitácora adelantada



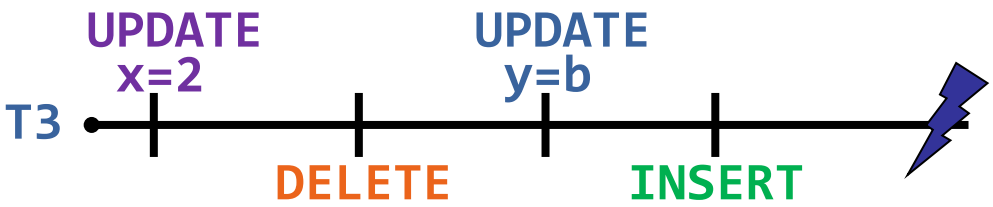
- ¿Qué pasa si ocurre un fallo global ***justo después*** *de que T3 emita el* COMMIT?
  - ¿Deshacer o Rehacer? No hay <COMMIT, T3> en el fichero bitácora
  - Así que ¡**Deshacer T3**!
    ¿Y eso es correcto? **NO**
    - ¡T3 había terminado bien!



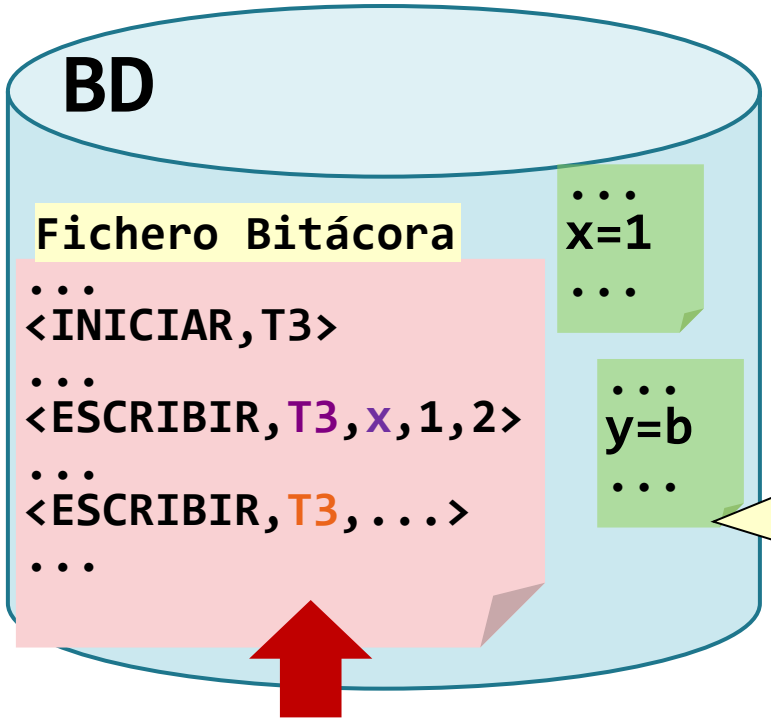
ⓘ Entradas de T3 en disco debido a un volcado anterior del búfer de bitácora a disco

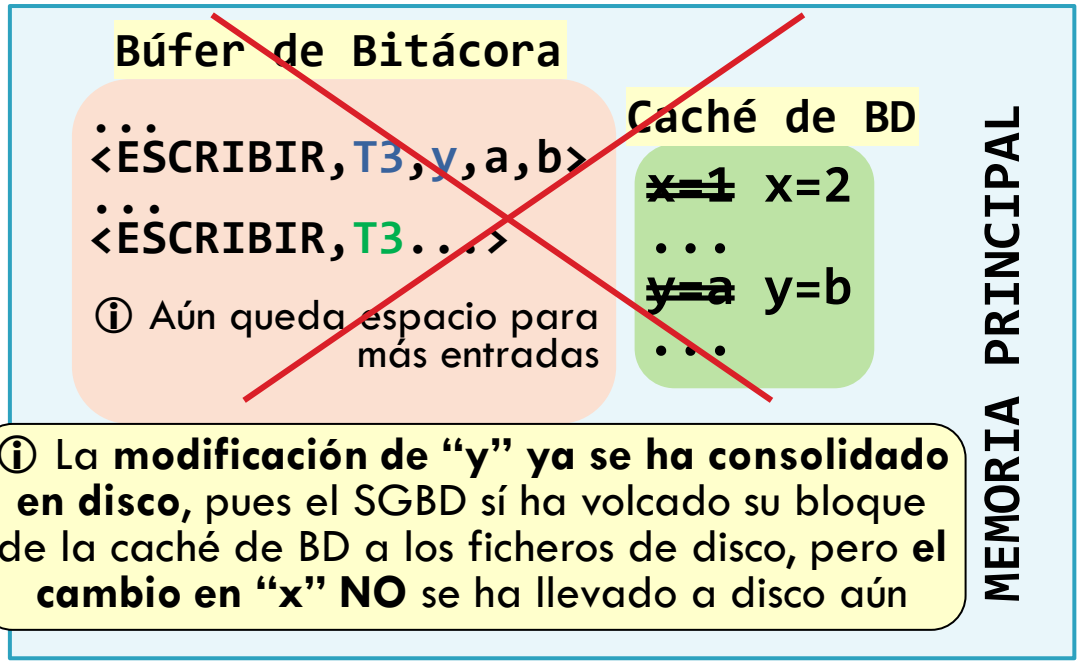


Bitácora adelantada



- ¿Y si ocurre un fallo del sistema cuando *T3* ***está en ejecución***...?
  - **Deshacer T3** ¡OK!
  - Pero... ¿Se puede? **NO**
    - ¡Faltan entradas de T3!





ⓘ La **modificación de "y" ya se ha consolidado en disco**, pues el SGBD sí ha volcado su bloque de la caché de BD a los ficheros de disco, pero **el cambio en "x" NO** se ha llevado a disco aún

Bitácora adelantada

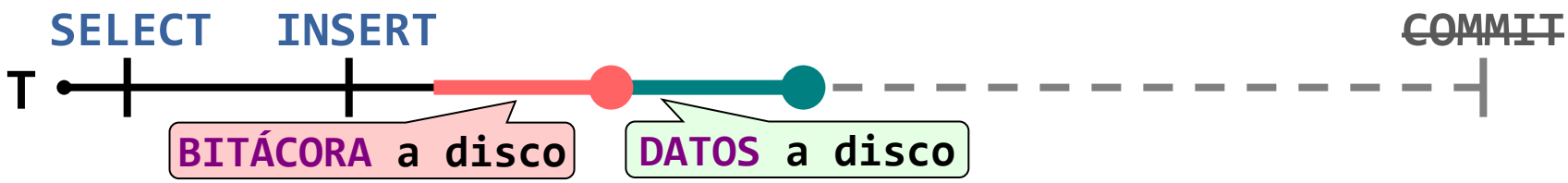
- Es necesario seguir un protocolo de **escritura anticipada en bitácora**, o **bitácora adelantada**:

> **No** se puede **guardar** en disco los **datos modificados por T hasta** que se haya **escrito en disco toda entrada de bitácora para T** hasta el momento actual

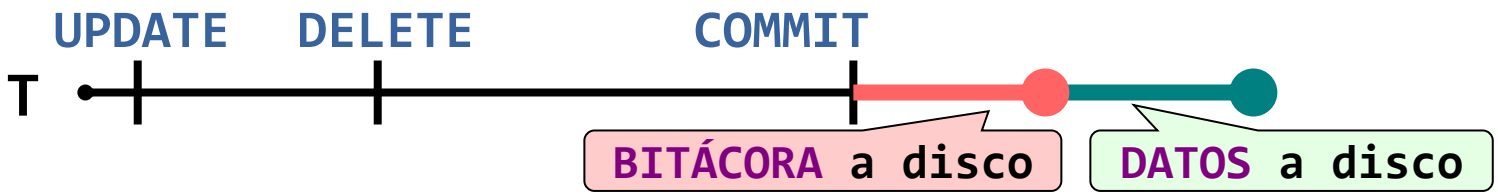
- Dicho de otro modo: **antes de guardar en disco datos modificados por T,** es necesario **escribir en** el fichero de **bitácora todas las entradas de bitácora para** T hasta el momento
- *Write-Ahead Logging* (WAL)

Bitácora adelantada

- Si el SGBD decide **llevar a disco cambios** realizados por T **mientras que T** aún **está en ejecución…**



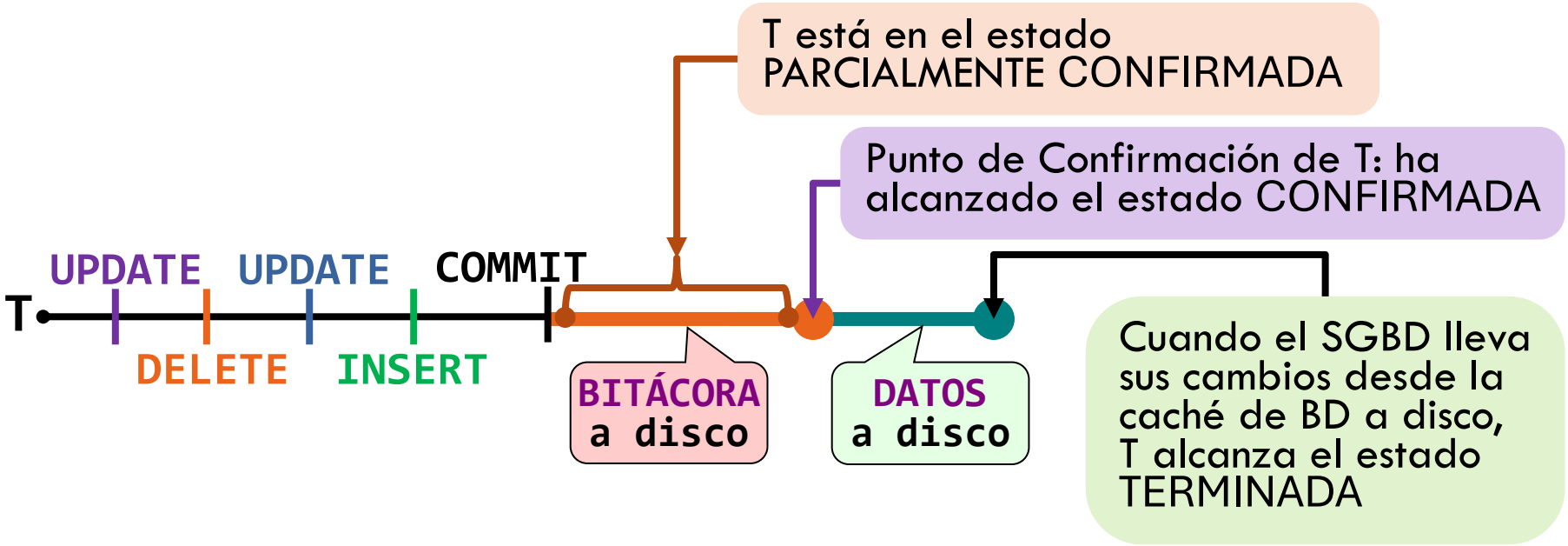
- Si el SGBD decide **llevar a disco cambios** realizados por T **cuando** T ya **ha ejecutado COMMIT…**



- En ambos casos…
  - 1° vuelca en el fichero de bitácora las entradas del buffer de bitácora correspondientes a T
  - 2° lleva los bloques de la caché de BD a los ficheros en disco

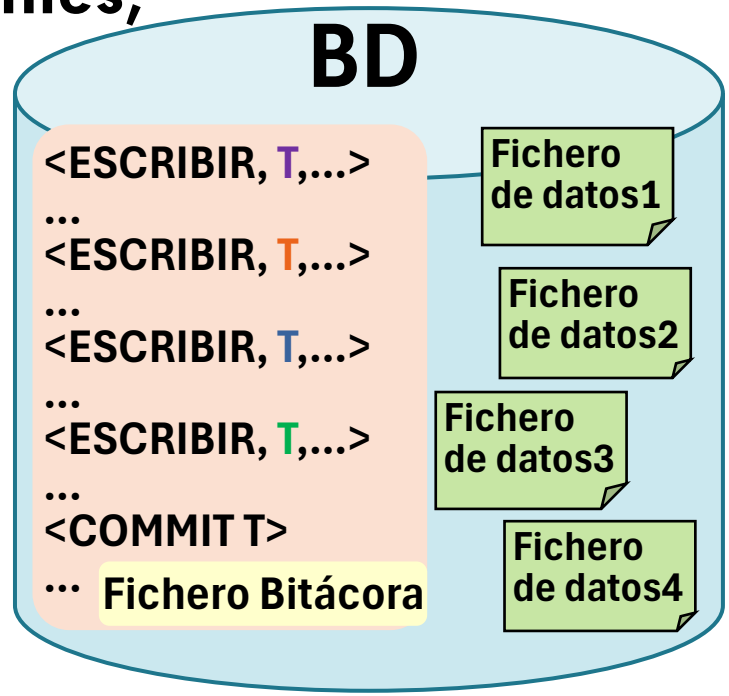
Bitácora adelantada

- Por tanto, el **COMMIT** de **T** se **completa** una vez que se ha **escrito en disco toda entrada de bitácora** pendiente **para T**
- En ese momento, T alcanza su '**punto de confirmación**'



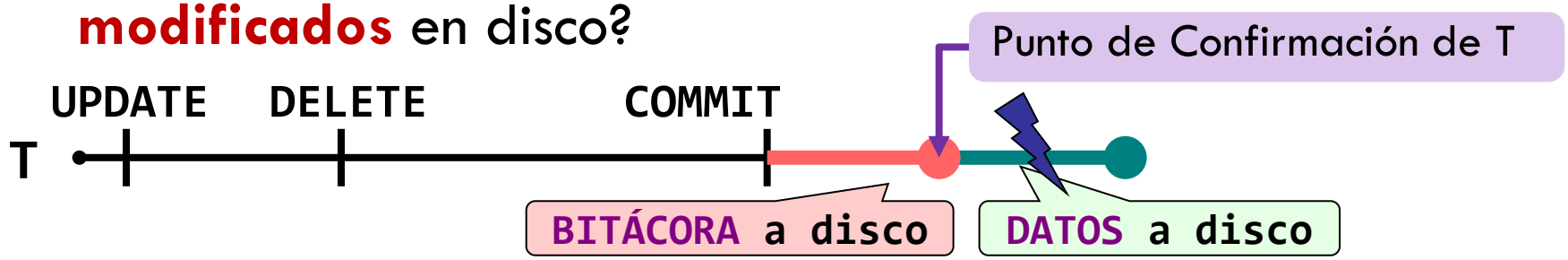
Bitácora adelantada y Punto de Confirmación

- ¿Qué significa
  "T ha alcanzado su punto de confirmación"?
  - T ha pasado al estado CONFIRMADA
  - Está <u>asegurado</u> que sus **modificaciones** sobre los datos **serán permanentes,** sea cual sea el momento en el que el SGBD vuelque dichos cambios al disco
    e incluso si no da tiempo a guardarlos debido a un fallo
    - Gracias a que el fichero de bitácora contiene todas las entradas correspondientes a T

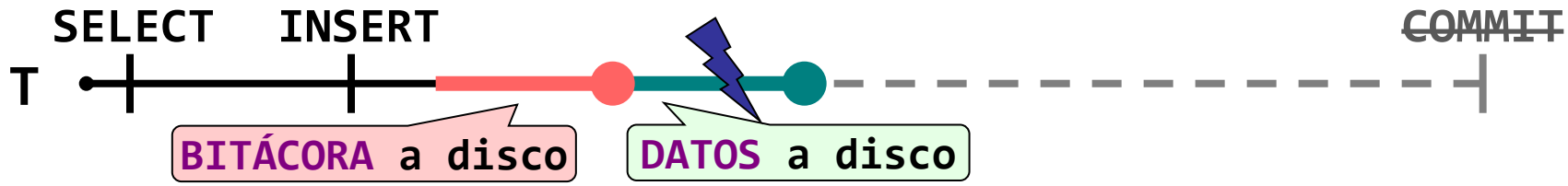


Bitácora adelantada y recuperación

- ¿Qué pasa si **el fallo ocurre al guardar los datos modificados** en disco?



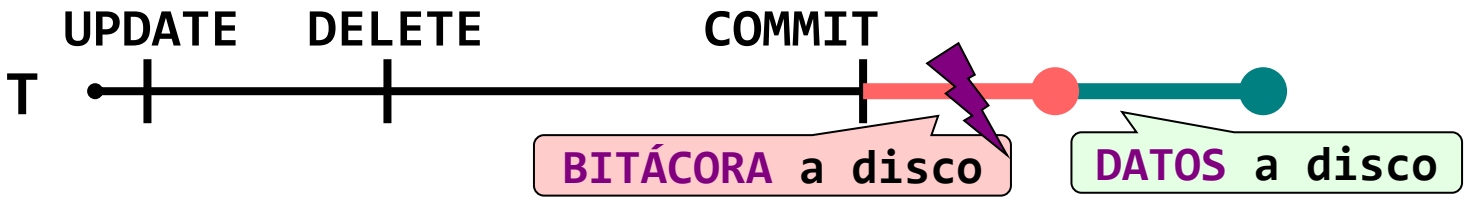
  - **REHACER** T (la entrada <COMMIT, T> está el fichero de bitácora)
  - Sin problemas, pues todas las entradas necesarias para REHACER están en el fichero de bitácora



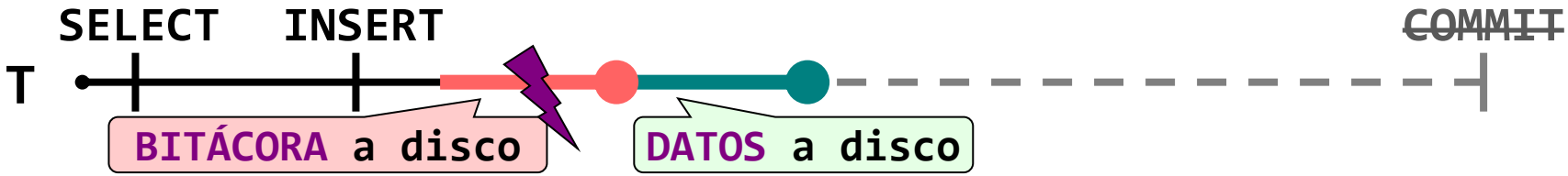
  - **DESHACER** T (no está <COMMIT, T> en el fichero de bitácora)
  - Sin problemas, porque toda entrada necesaria para DESHACER está en el fichero de bitácora

Bitácora adelantada y recuperación

- ¿Y si **el fallo ocurre al volcar el búfer de bitácora** en disco?



- Si NO ha dado tiempo a escribir la entrada <COMMIT, T> en la bitácora en disco, entonces el SGBD decide **DESHACER** T
- ¡Ok! Pues los datos modificados aún no habían sido llevados a disco



- El SGBD decide **DESHACER** T, pues no encuentra la entrada <COMMIT, T> en el fichero de bitácora
- Pero ¡Ok! Los datos modificados aún no habían sido llevados a disco

Estrategia de recuperación

- Recuperación tras un **fallo de tipo 5 o 6**, que produjo daños físicos en la BD…
  - **Restaurar** la última **copia de seguridad** de la BD, en un nuevo disco
  - Reconstruir un estado más actual usando la **bitácora** en disco: **Rehacer operaciones escribir** de las transacciones **confirmadas** hasta el momento de la caída
- Recuperación tras un **fallo de tipo 1 a 4**, que no ha dañó la BD físicamente, pero la dejó inconsistente…
  - **Deshacer operaciones escribir** de las **transacciones** que estaban **en curso** y no se confirmaron
  - **Rehacer**, si es necesario, **operaciones escribir** de las **transacciones confirmadas**, para asegurar que sus cambios están en disco
  - Para ello, usar la **bitácora** en disco y un algoritmo de recuperación

Protocolo de Checkpoints

- En el proceso de recuperación no se revisa TODA la bitácora en disco: sería muy costoso y poco práctico
- Para limitar la porción de la bitácora examinada y el coste de su procesamiento, se usan ***checkpoints*** (**puntos de control**)
- El SGBD marca automáticamente un punto de control…
  - Cada **m segundos**, o
  - Tras escribir **n** entradas < COMMIT, Ti > en la bitácora desde el último punto de control
    - "n" es un parámetro de configuración del sistema
  - … (Hay más acciones que provocan un *checkpoint*, que no veremos)

Protocolo de *Checkpoints*

- Marcar un punto de control significa …
  1. **Suspender** la **ejecución** de las **transacciones**
  2. **Forzar** la **escritura en disco del búfer de bitácora**
     - Volcado de todas las entradas **en el fichero bitácora**
  3. **Forzar** la **escritura en disco de todo bloque modificado de la caché de BD**
     - Volcado de todos los bloques modificados **a los ficheros de datos**
  4. **Escribir** una **entrada** <PUNTO DE CONTROL> en el fichero de bitácora en disco
  5. **Escribir** en un ***Fichero Especial de Arranque*** la dirección de la entrada < PUNTO DE CONTROL > dentro del fichero de bitácora
  6. **Reanudar** la **ejecución de las transacciones**

Protocolo de *Checkpoints*

- La entrada en el fichero de bitácora de tipo **<PUNTO DE CONTROL>** contiene:
  - Lista de identificadores de las **transacciones activas** (en curso de ejecución) en ese instante
  - Dirección en el fichero bitácora de la 1ª y última entradas para cada Ti activa
- Un punto de control **sincroniza memoria y disco**
  - búfer de bitácora → fichero de bitácora
  - caché de BD → ficheros de datos
    - Se transfiere al disco el efecto de las operaciones ESCRIBIR realizadas hasta ese instante por las transacciones

Protocolo de *Checkpoints*

Así, en el proceso de recuperación **el uso de puntos de control permite...**

- **Recorrer la bitácora a partir del último punto de control**
  - Y **no *desde el principio***
- **Ignorar las T_i confirmadas antes del último punto de control**
  - Es seguro que sus cambios se llevaron a disco en el *checkpoint*
  - Ya **no** es necesario **rehacer *todas las*** *transacciones* ***confirmadas***, sino sólo las que se confirmaron después del último *checkpoint*

Fichero Bitácora

```
...
<INICIAR,T4>
<ESCRIBIR,T4,...>
<INICIAR,T2>
<ESCRIBIR,T2,...>
<INICIAR,T1>
<COMMIT,T4>
<INICIAR,T3,...>
<PUNTO DE CONTROL...>
<ESCRIBIR,T1,...>
<ESCRIBIR,T3,...>
<LEER,T1,...>
<ESCRIBIR,T3,...>
<LEER,T2,...>
<ESCRIBIR,T1,...>
<COMMIT,T2>
<LEER,T3,...>
```

Estrategia de Recuperación: **Algoritmo Deshacer/Rehacer**

- Se accede a la última entrada <PUNTO DE CONTROL>
- Esta entrada contiene la **lista A de transacciones activas** **A = { T2, T1, T3 }**
  - T4 no está, porque hizo COMMIT y finalizó
- Se crea la **lista C de confirmadas** **C = ∅**
- Recorrer el fichero de bitácora desde ahí
  - Cada <INICIAR, T> añade T a **A**
  - Cada <COMMIT, T> mueve T de **A** a **C**
- Al terminar de recorrer la bitácora…
  - **A = {T1, T3}** y **C = {T2}**

▶ **Recuperación:**

- **Deshacer** las de **A**: **T1** y **T3**
- **Rehacer** las de **C**: **T2**
- **Ignorar** el resto: **T4**
  - Es seguro que sus cambios se llevaron a disco en el *checkpoint*

Fichero Bitácora

```
...
<INICIAR,T4>
<ESCRIBIR,T4,...>
<INICIAR,T2>
<ESCRIBIR,T2,...>
<INICIAR,T1>
<COMMIT,T4>
<INICIAR,T3,...>
<PUNTO DE CONTROL...>
<ESCRIBIR,T1,...>
<ESCRIBIR,T3,...>
<LEER,T1,...>
<ESCRIBIR,T3,...>
<LEER,T2,...>
<ESCRIBIR,T1,...>
<COMMIT,T2>
<LEER,T3,...>
```

Estrategia de recuperación

Siempre se debe **deshacer primero, y **rehacer después****

- En bitácora, las entradas <ESCRIBIR,...> contienen tanto valor_anterior como valor_nuevo porque pueden utilizarse para *deshacer* o para *rehacer* una modificación
- Hay que **deshacer** las operaciones **en el *orden inverso*** al de anotación en bitácora

  No se deshace cada T activa "en aislado", sino que **se va deshaciendo todas las activas** "a la vez", operación a operación, **de forma intercalada**
- Se debe **rehacer** las operaciones **en el *mismo orden*** en que aparecen en bitácora

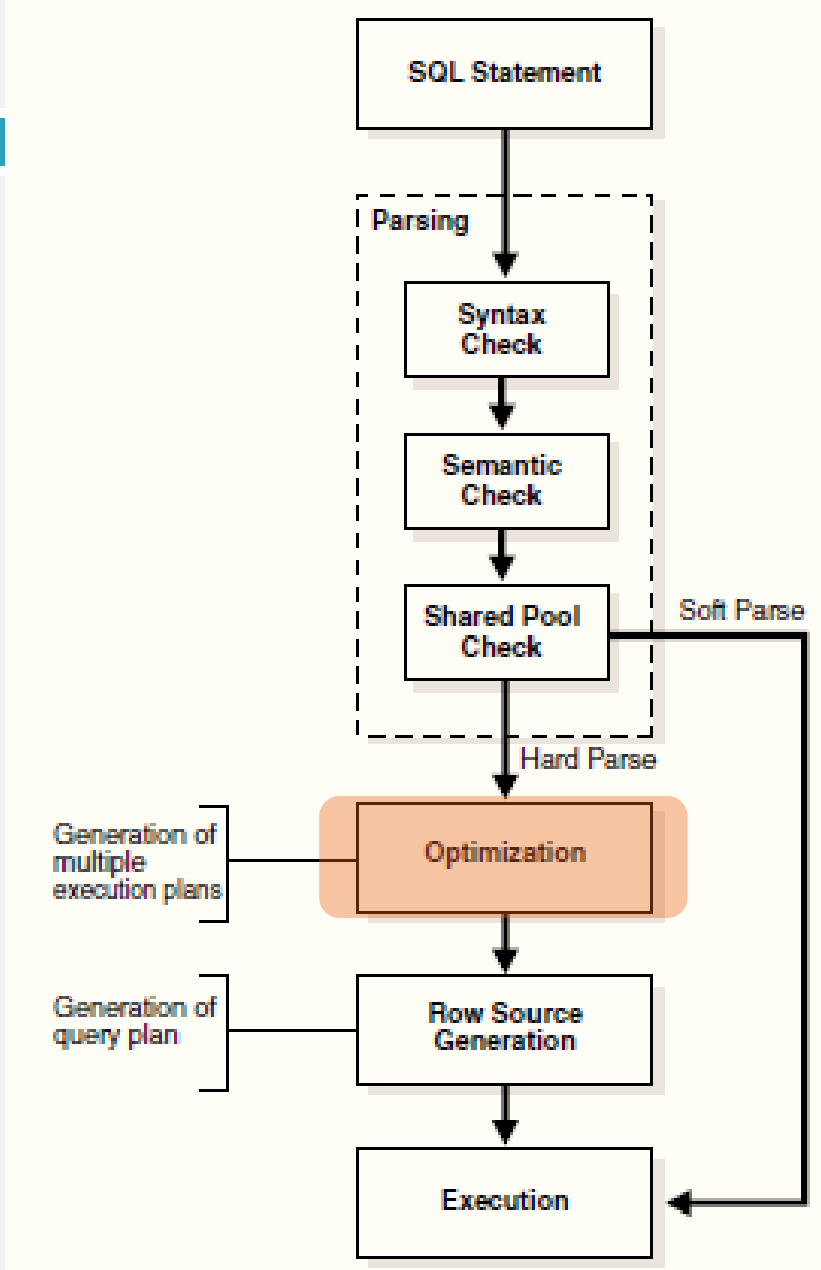
  No se rehace cada T confirmada "en aislado", sino que **se va rehaciendo todas las confirmadas** "a la vez", operación a operación, **de forma intercalada**

11.4. Procesamiento y optimización de consultas

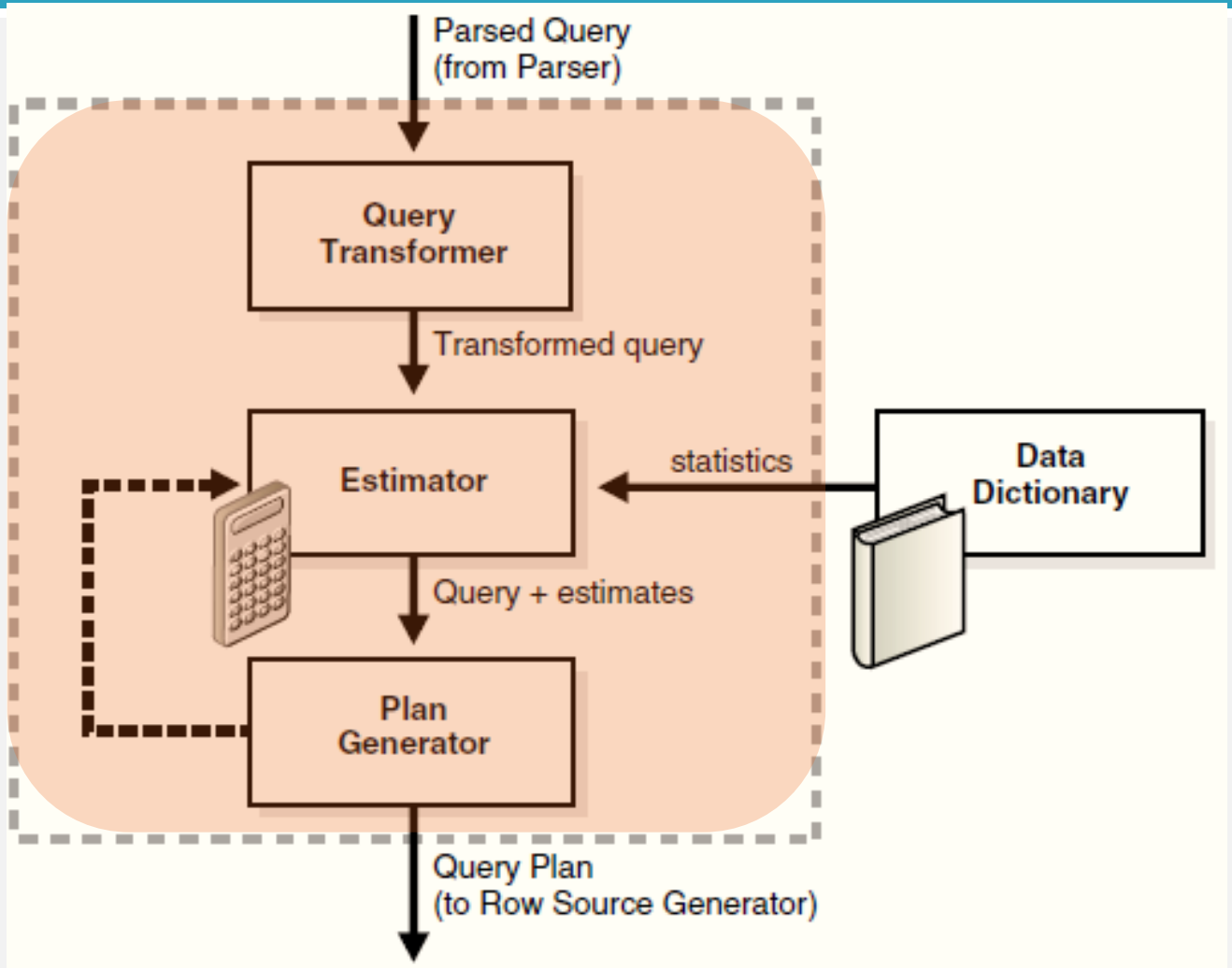
- El SGBD garantiza la **obtención** eficiente de datos
  - ***Búferes y cachés de datos***, para mantener en memoria y procesar datos extraídos de la BD en disco
  - *Estructuras de datos auxiliares y técnicas de búsqueda* para **acelerar la búsqueda en disco** de los registros deseados
    - **Índices** (ficheros auxiliares)
    - Elegir qué índices crear y mantener es parte de la etapa de Diseño Físico y Ajuste del esquema de la BD
  - *Técnicas de* ***optimización*** y ***estrategias de procesamiento*** para **acelerar la ejecución de las consultas**

- **El SGBD siempre ejecuta las sentencias SQL de la forma más eficiente posible**

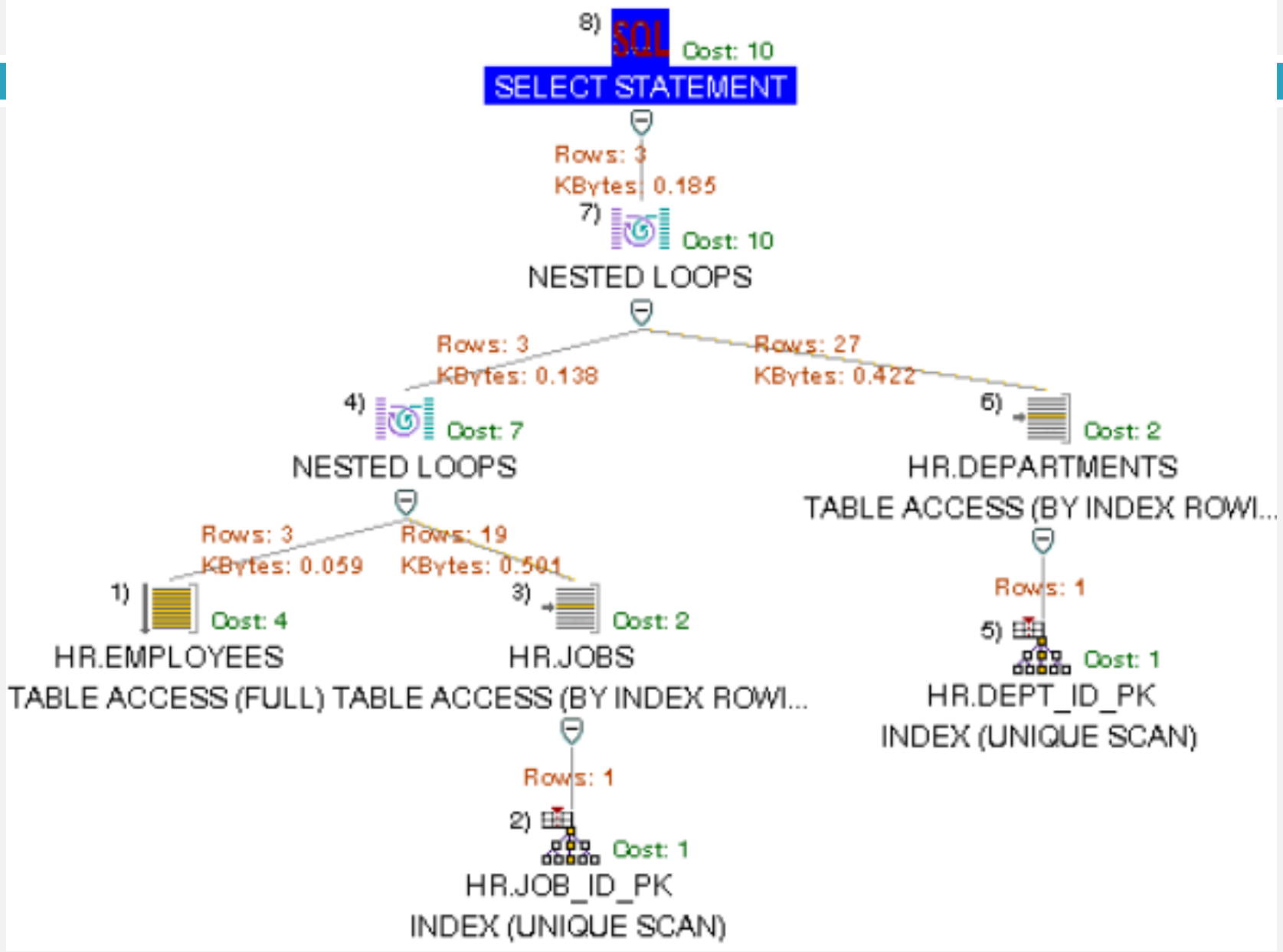
Etapas del procesamiento y optimización de consultas en Oracle



Componentes del Optimizador Oracle



Plan de Ejecución de una sentencia Oracle



Visualización del Plan de Ejecución en Oracle SQL Developer

- Icono “Explicación del Plan…” (F10)

Hoja de Trabajo | Generador de Consultas

```
1 SELECT num, des FROM COMPUESTO
2 WHERE num IN (SELECT compuesto FROM PROPIEDAD_EN_COMPUESTO
3               WHERE propiedad = (SELECT id FROM PROPIEDAD WHERE des LIKE 'ESTIMULANTE'))
4 ORDER BY des;
```

Explicación del Plan

SQL | 0,015 segundos

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			8
SORT		ORDER BY	8
MERGE JOIN		SEMI	7
TABLE ACCESS	COMPUESTO	BY INDEX ROWID	2
INDEX	COMPUESTO_PK	FULL SCAN	1
SORT		UNIQUE	5
Predicados de Acceso			
NUM=COMPUESTO			
Predicados de Filtro			
NUM=COMPUESTO			
VIEW	VW_NSO_1		4
INDEX	PRO_EN_COM_PK	RANGE SCAN	1
Predicados de Acceso			
PROPIEDAD= (SELECT ID FROM			
TABLE ACCESS	PROPIEDAD	FULL	3
Predicados de Filtro			

Anexo: Más sobre Concurrencia en Oracle

Anexo
Niveles de Aislamiento en Oracle

- Sentencia `SET TRANSACTION ISOLATION LEVEL nivel;`
- Implementa **2 de los 4** niveles del estándar SQL…
  - `READ COMMITTED` *(nivel por defecto)*
    - Cada sentencia dentro de la transacción sólo **ve los datos confirmados antes del inicio de la sentencia** (no de la transacción)
      - Así, los datos pueden ser modificados por otras transacciones entre una ejecución y otra de la misma sentencia dentro de la misma transacción, lo que permite *lecturas no repetibles* y *lecturas fantasma*
  - `SERIALIZABLE`
    - Cada sentencia dentro de la transacción sólo **ve los datos confirmados antes del inicio de la transacción** y sus propios cambios
- Y un **nivel no definido en el estándar** SQL
  - `READ-ONLY`
    - Cada sentencia dentro la transacción sólo **ve los datos confirmados antes del inicio de la transacción**, y **no puede modificar** datos

Anexo
Control de Concurrencia en Oracle

- ***Oracle* usa bloqueos** (cerrojos) **de forma implícita** para todas las sentencias SQL
- Por lo que el usuario/programador no necesita bloquear ningún recurso explícitamente
- Aun así, *Oracle* ofrece un mecanismo para **adquirir bloqueos manualmente**
  - Sentencia LOCK TABLE
  - Cláusula FOR UPDATE en la SELECT
- Los bloqueos manuales son **liberados** de forma automática **cuando la transacción finaliza** con COMMIT o con ROLLBACK
  - No existe sentencia para desbloquear

Anexo
Control de Concurrencia en Oracle

Bloqueo manual: sentencia LOCK TABLE

Ejemplos

- Bloquea la tabla EMPLEADO en *modo exclusivo*; si otro usuario ya había bloqueado la tabla, la sentencia no espera y devuelve el control inmediatamente, y un mensaje informando del bloqueo de la tabla

```
LOCK TABLE EMPLEADO
  IN EXCLUSIVE MODE
  NOWAIT;
```

- Bloquea la tabla EMPLEADO en *modo compartido*; si otro usuario ya había bloqueado la tabla, la sentencia espera 3 segundos para ver si puede bloquear la tabla entonces; si no, devuelve el control y el mensaje

```
LOCK TABLE EMPLEADO
  IN SHARE MODE
  WAIT 3;
```

Hay *5 modos de bloqueo*. Más información en los manuales de Oracle: https://docs.oracle.com/en/database/oracle/oracle-database/21/sqlrf/LOCK-TABLE.html

Anexo
Control de Concurrencia en Oracle

- Bloqueo manual: **SELECT...FOR UPDATE**
  - Bloquea las filas seleccionadas para que otros usuarios no las puedan bloquear o actualizar hasta que finalice la transacción que contiene tal SELECT... FOR UPDATE
  - No se puede especificar la cláusula FOR UPDATE en subconsultas

Ejemplos

- Bloquea las filas de EMPLEADO que viven en Murcia y también bloquea las filas en DEPARTAMENTO correspondientes a dichos empleados

```
SELECT nss, E.nombre, apellido, salario
 FROM EMPLEADO E JOIN DEPARTAMENTO ON dep=coddep
 WHERE E.ciudad = 'MURCIA'
 FOR UPDATE;
```

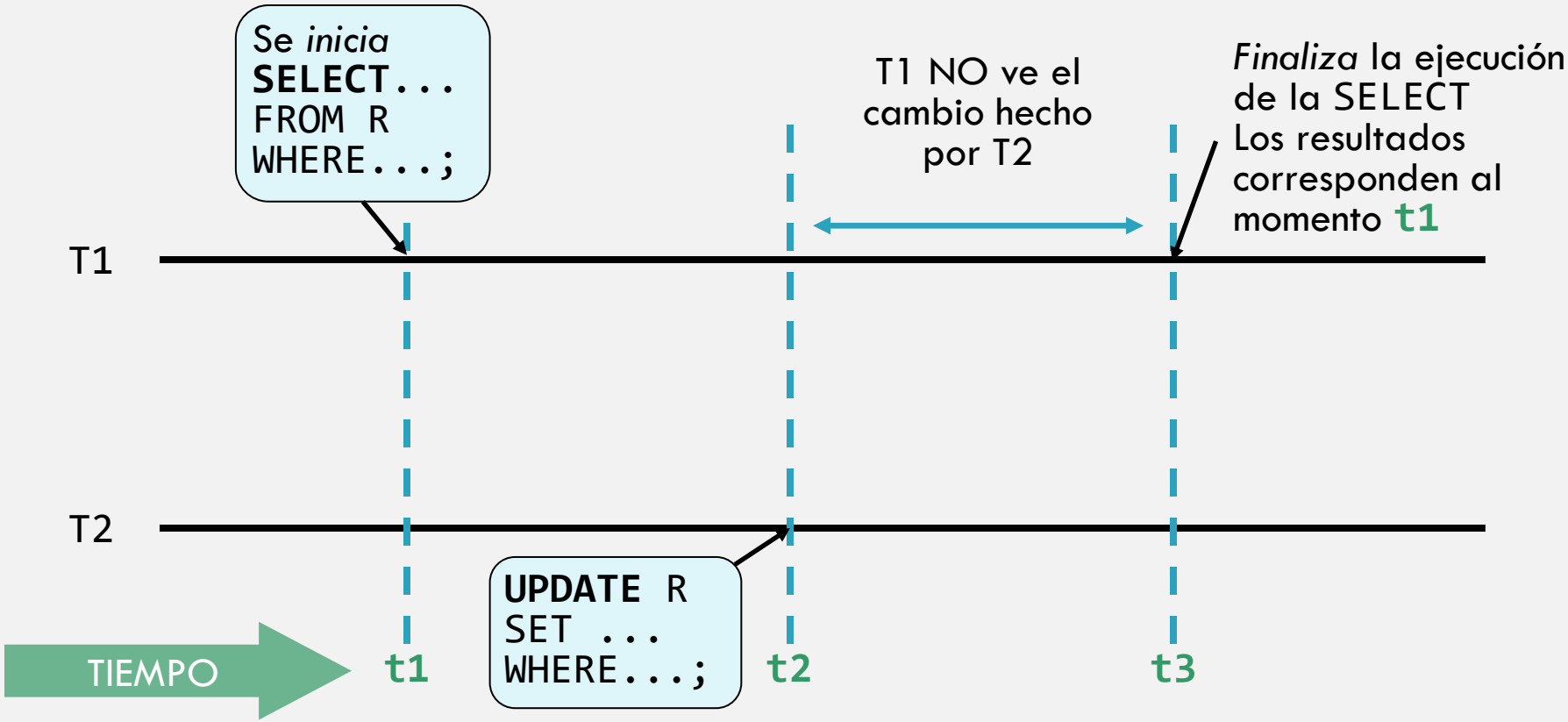
- Bloquea las filas de EMPLEADO de Murcia, pero **no** bloquea filas en DEPARTAMENTO

```
SELECT nss, E.nombre, apellido, salario
 FROM EMPLEADO E JOIN DEPARTAMENTO ON dep=coddep
 WHERE E.ciudad = 'MURCIA'
 FOR UPDATE OF salario; --columna de EMPLEADO
```

Anexo
Consistencia Multiversión en ORACLE

- *Oracle Database* no sólo usa **bloqueos** para garantizar la concurrencia y consistencia de datos
- También utiliza lo que denomina **Modelo de consistencia Multiversión**
- Cada sentencia en una transacción concurrente tiene una 'vista de los datos' **consistente con un punto en el tiempo**
  - Ese punto será el **inicio de la transacción** o el inicio **de la sentencia**
    - *Transaction-Level Read Consistency*
    - *Statement-Level Read Consistency*
  - Por ejemplo, si cuando comienza una consulta (SELECT) hay otras transacciones **no confirmadas** ejecutándose, *Oracle* garantiza que **la consulta no ve los cambios** hechos por esas otras transacciones

Anexo.
Consistencia Multiversión en ORACLE



El efecto es el mismo que si se hubiera ejecutado primero T1 y luego T2

Anexo
Consistencia Multiversión en ORACLE

- Veamos de **forma intuitiva** **cómo Oracle consigue implementar este modelo** (es avanzado):
  - Se consigue mediante lo que se conoce como **datos 'undo'**
  - Cada vez que se modifican datos, se crean 'entradas undo', que se almacenan en los segmentos "undo" de la BD en disco
  - Estas entradas contienen los valores antiguos de los datos que han sido modificados por transacciones no confirmadas, o confirmadas recientemente
  - Por lo tanto, en la BD pueden existir **múltiples versiones de los mismos datos,** todas **en diferentes puntos del tiempo**
  - La BD puede utilizar **instantáneas** de datos en diferentes puntos en el tiempo para proporcionar *vistas consistentes de lectura* de los datos y **permitir *consultas (lecturas) sin bloqueo***

+info: https://docs.oracle.com/en/database/oracle/oracle-database/21/cncpt/data-concurrency-and-consistency.html#GUID-E8CBA9C5-58E3-460F-A82A-850E0152E95C