APUNTES ABD

Author
B.L.B

Contents

1 Control de Acceso

Definicion 1.1. Administrador: persona responsable de garantizar la seguridad e integridad de los datos.

- Seguridad: previene la revelación, alteración y destrucción de los datos. Los usuarios solo pueden realizar operaciones autorizadas.
- Integridad: asegura la exactitud y la validez de los datos (consistentes)

Es el responsable principal de la **seguridad** del SGBD. Tiene una cuenta de súper-usuario (forma de acceso protegida) y especifica qué usuarios tienen acceso a qué datos y qué operaciones pueden realizar sobre esos datos.

El acceso se controla mediante privilegios (permisos para realizar operaciones).

1.1 Gestión de privilegios

En esta sección se explorará la forma de crear usuarios y concederles privilegios.

1.1.1 Crear cuenta de usuario

```
CREATE USER <usuario> IDENTIFIED BY 'contraseña';
```

https://dev.mysql.com/doc/refman/8.0/en/create-user.html

1.1.2 Borrar cuenta de usuario

```
DROP USER <usuario>;
```

1.1.3 Conceder privilegios

```
GRANT <privilegio> ON <objeto> TO <sujeto> [WITH GRANT OPTION]
```

La cláusula "With Grang Option" permite al usuario receptor de un permiso propagarlo a otros usuarios.

1.1.4 Retirar privilegios

```
REVOKE [GRANT OPTION FOR] <privilegio> ON <objeto> FROM <sujeto>
```

1.1.5 Tipos de objeto

```
*.* (Global)
<BD>.*
<BD>.<tabla>
<BD>.<tabla>(columna)
```

Mostrar permisos otorgados para nuestro usuario:

```
SHOW GRANTS;
```

Mostrar permisos para otro usuario:

```
SHOW GRANTS FOR <usuario>;
```

1.1.6 Ejercicio 1

Crear BD llamada “ej1DB”

```
CREATE DATABASE ej1DB;
```

Crear usuario llamado “ej1user” y darle permisos para crear tablas en ej1DB y leer, insertar y borrar filas en las tablas de ej1DB

```
CREATE USER ej1user IDENTIFIED BY '1234';
GRANT CREATE, SELECT, INSERT, DELETE ON ej1DB.* TO ej1user;
```

Verificar los permisos asignados

```
SHOW GRANTS;
```

1.2 Roles

Un rol es una colección de privilegios con un nombre.

1.2.1 Crear rol

```
CREATE ROLE <rol>;
```

1.2.2 Asignar privilegios a rol

```
GRANT <privilegio> ON <objeto> TO <rol>;
```

1.2.3 Asignar rol a usuario

```
GRANT <rol> TO <usuario>;
```

1.2.4 Activar y mostrar rol

```
SET ROLE <rol>;
SELECT current_role();
```

1.2.5 Definir rol por defecto

```
SET DEFAULT ROLE <rol> TO <usuario>;
```

1.2.6 Mostrar privilegios de rol

```
SHOW GRANTS FOR <usuario> USING <rol>;
```

1.2.7 Retirar privilegios

```
REVOKE <privilegio> ON <objeto> FROM <rol>;
```

1.2.8 Eliminar rol

```
DROP <rol>;
```

1.2.9 Ejercicio 2

Crear una BD "ej2DB" y usuario "ej2user"

```
CREATE DATABASE ej2DB;
CREATE USER ej2user IDENTIFIED BY '1234';
```

Crear un rol "ej2rol" con permisos para crear tablas e insertar y leer filas de las tablas.

```
CREATE ROLE ej2rol;
GRANT CREATE, INSERT, SELECT ON ej2DB.* TO ej2rol;
```

Asignar el rol al usuario

```
GRANT ej2rol TO ej2user;
```

1.3 Vistas

Definicion 1.2. Consultas almacenadas que producen un resultado al ser invocados y limitan la visibilidad de los datos.

Crear:

```
CREATE VIEW <BD>.<nombre> AS <select>;
```

Borrar:

```
DROP VIEW <nombre>;
```

No permiten borrar, actualizar o insertar, únicamente **SELECT**.

1.4 Rutinas almacenadas

Definicion 1.3. Porciones de código SQL reutilizables con nombre.

Existen de dos tipos:

1. Procedimientos: CALL, no devuelve valor.
2. Funciones: se invoca usando su nombre, devuelve valor.

No se recomienda implementar para lógica de negocio, dificulta el mantenimiento y depuración.

1.4.1 Procedimiento

```
DELIMITER //
CREATE PROCEDURE <nombre> (<parámetros>)
BEGIN
  <sentencias>
ENDE
DELIMITER ;
```

Parámetros: IN, OUT, INOUT. Se pueden asignar valores a variables:

```
DECLARE variable <tipo>;
SELECT columna INTO variable FROM .. WHERE ..
SET variable = 2;
```

Todo procedimiento pertenece a una DB. Si la DB se borra, el procedimiento también. Mostrar procedimientos:

```
SHOW PROCEDURE STATUS WHERE DB = '<nombre-BD>';
```

Llamar a procedimiento:

```
CALL proc();
```

Son necesarios los permisos **CREATE ROUTINE** y **EXECUTE** para crear y ejecutar los procedimientos.

1.4.2 Ejercicio 3:

```
CREATE DATABASE BDvalores;
USE BDvalores;
CREATE TABLE valores(id INT NOT NULL PRIMARY KEY,
valor INT);
DELIMITER//
CREATE PROCEDURE suma()
BEGIN
DECLARE resultado INT;
SELECT sum(valor) INTO resultado FROM valores;
IF resultado < 10 THEN
  SELECT "Total < 10";
ELSE
  SELECT "Total >= 10";
END IF;
END//
DELIMITER ;
UPDATE valores SET valor=6 WHERE id=2;
CALL suma();
```

1.5 Otras técnicas

- **Ataques de denegación de servicio:** Limitar los recursos que usa cada cuenta de usuario.
- **Ataques SQL injection:** Usar parametrización en las consultas de código.

2 Auditoría

Definicion 2.1. Proceso que consiste en recoger, agrupar y evaluar evidencias para determinar si un sistema de información salvaguarda el activo empresarial, mantiene la integridad de los datos, cumple con las leyes y regulaciones establecidas y utiliza eficientemente los recursos.

Definicion 2.2. Proceso que permite medir, monitorizar y registrar los accesos a la información de una BD: quién y cuándo accede, desde donde, qué comando o sentencias SQL se ejecutaron y cuál fue el efecto.

2.1 Tipos

- **Rutinaria:** Garantiza la integridad de la información en la BD y verifica que hay controles para prevenir riesgos. Revisa los usuarios y los permisos y la configuración del SGBD.
- **Forense:** Trata de encontrar el motivo de un problema o fallo. Revisa los logs o registros.

2.2 Requisitos

- No puede comprometer el funcionamiento rutinario.
- Debe proveer información relevante relativa a *seguridad*, *apoyo en toma de decisiones*, *mejora en el funcionamiento* de la organización.
- En lo posible no debe ser *auto-aplicada* (debe hacerla una persona externa).

2.3 Acciones a auditar

- Usuarios: inicios/cierres de sesión (incluyendo fallidos) y propagación de permisos (incluyendo fallidos).
- Procesos: cambios en la configuración/tablas del SGBD y reinicio del servidor/proceso.
- Comandos: acciones con permisos de administrador, intentos fallidos de acceso por falta de autorización, ejecución de procedimientos almacenados.

2.4 Herramientas

- **Logs:** son los ficheros que contienen el registro de actividad del SGBD. Son flexibles, pero requieren vigilancia para evitar problemas de almacenamiento y su análisis puede ser una tarea compleja.
- **Aplicaciones de 3^{o}s:** Software específico que se conecta al SGBD y permite monitorizar características. Facilita la extracción de información y combinan información de diferentes sistemas, pero resulta en gasto económico y posible impacto en rendimiento.
- **Triggers:** son programas almacenados que se llaman automáticamente tras un evento concreto (tras insert/delete/update...). SQL soporta 2 tipos: a nivel de **fila** y a nivel de **sentencia**. MySQL solo soporta a nivel de fila. Es una forma de vigilar la integridad de los datos, pero no permiten hacer todo tipo de validaciones, pueden ser difíciles de depurar e implican mayor sobrecarga.

2.4.1 Cómo crear Triggers en MySQL

Cada trigger se asocia a una tabla concreta:

```
CREATE TRIGGER <nombre><cuándo><acción> on <tabla> FOR EACH ROW <acciones>

<cuándo>: Before/After
<acción>: INSERT, DELETE, UPDATE
```

El cuerpo del Trigger puede contener varias sentencias entre BEGIN y END. Mostrar triggers:

```
SHOW TRIGGERS;
```

Borrar triggers:

```
DROP TRIGGER <nombre>;
```

2.5 Ejercicio 1:

```
CREATE DATABASE AUDITADA;

CREATE TABLE Datos( id int NOT NULL PRIMARY KEY,
producto VARCHAR(50),
precio int);
CREATE TABLE Audit(gasto int,
fecha Date);

CREATE TRIGGER trig AFTER INSERT ON Datos FOR EACH ROW INSERT INTO
Audit VALUES (NEW precio, Now());
```

3 Recuperación (06-02-2024)

La SGBD debe asegurar la integridad y la seguridad de los datos (LOPD) ante los riesgos de que estos datos se pierdan. Dos situaciones:

- A nivel tabla/BD
- **A nivel de transacción**

Se hacen **transacciones** para agrupar varias operaciones, si algo va mal en alguna de las operaciones la transacción se interrumpe y no hay cambios en la BD.

Definicion 3.1. Transacción: Unidad lógica de procesamiento, de integridad y de recuperación.

```
START TRANSACTION;
\\OPERACIONES
COMMIT;(éxito) O  ROLLBACK;(fracaso)
```

Por defecto mySQL funciona en AutoCommit: crea una transacción por cada operación. Se puede desactivar para gestionar las transacciones de forma manual.

```
SET AUTOCOMMIT=0;
```

Para garantizar las validez de datos frente a fallos, los SGBD cumplen las propiedades ACID:

- Atomicity: unidad lógica de procesamiento, no puede ejecutarse a medias.
- Consistency: unidad lógica de integridad
- Isolation: ejecución de una transacción no afecta a otra ejecución.
- Durability: los cambios de una transacción confirmados deben persistir.

Durante la ejecución de una base de datos, los cambios y cálculos ocurren en la memoria RAM. Una vez deban reflejarse, se envían al disco duro y se organizan en distintos ficheros. Para que la SGBD sea lo más eficiente posible, se trata de acceder lo menos posible al disco duro. Existen 3 tipos de fallos posibles:

- Fallos físicos
- Fallos de software
- Fallos al ejecutar la transacción

La SGBD debe asegurar que se cumplan las propiedades ACID a pesar de los fallos. Las SGBD disponen de un diario en el que se escriben las operaciones realizadas por las transacciones. Se escribe mediante "write ahead logging". Todo comando que pasa por la RAM, se escribe en un diario (fichero) del disco antes de realizar los cambios en la BD. Este diario es la base del proceso de recuperación y sirve para monitorizar la ejecución de las transacciones.

```
START_TRANSACTION, <T>, <time>
READ, <T>, <X>, <V>, <time>
WRITE, <T>, <X>, <oldV>, <newV>, <time>
COMMIT, <T>, <time>
ROLLBACK, <T>, <time>
```

En caso de fallo en medio de una transacción, el sistema de recuperación revisa el diario y aplica las operaciones REDO(T) o UNDO(T). En mySQL el diario se llama **Redo Log**. Por defecto se encuentra activado. Se puede repartir entre varios ficheros y se puede ver cuales se hallan activos. (Escrito en binario).

3.1 Proceso de recuperación

Tras el commit o el failed hay un paso más: terminated. Llega ahí tras pasar un punto de confirmación. En este tema se considerará un único diario. Cuando una transacción T realiza COMMIT: todas las operaciones de la transacción se ejecutaron con éxito, el efecto se ha anotado en el diario y T ha llegado a un punto de confirmación. Cuando una transacción T realiza ROLLBACK: las operaciones se han anotado en el diario, pero sus operaciones no deben escribirse en la BD. Si sucede un fallo durante una transacción, T no alcanza un punto de confirmación. El diario contiene alguna de sus operaciones, pero no está el COMMIT. El proceso de recuperación aplica UNDO(T): deshace las operaciones en orden inverso. Si sucede un fallo cuando una transacción ya ha sido confirmada, se debe rehacer. En este caso debe aplicarse REDO(T): rehace las operaciones en el orden original.
Frecuencia de la actualización del diario:

- Inmediata: por cada operación, la RAM envía dicha operación al diario. Mayor fiabilidad, menor rendimiento (más escrituras en disco).
- Diferida: las operaciones se envían al diario por bloques. Mayor rendimiento (menos escrituras), mejor jerarquía de memoria, mayor riesgo.

Cada cierto tiempo, el SGBD realiza el **Checkpoint**:
En *escritura diferida* de diario, el bloque de RAM se envía al diario aunque no haya alcanzado su tamaño total. Revisa el diario, y aquellas con COMMIT (finalizadas) las escribe en la BD. Crea una lista con las transacciones activas.

```
CHECKPOINT, <time>
```

Estos puntos permiten recorrer el diario para ignorar las confirmadas anteriores al checkpoint. En *escritura diferida*, hay 3 operaciones que fuerzan las escritura de los bloques: CHECKPOINT, ROLLBACK, COMMIT.

3.2 Actualización de BD

Hay dos maneras de gestionarlo: Inmmediata y diferida.
Actualización diferida: después del commit. **Algoritmo no-deshacer/rehacer**
Pueden suceder 2 tipos de fallo:

- Antes de realizar el COMMIT: no hay que deshacer nada. No ocurre nada.
- Después de ralizar el COMMIT: es necesario rehacer las operaciones.

Actualización **inmediata**: **Algoritmo deshacer/rehacer**

- Antes de realizar commit: es necesario deshacer las operaciones
- Después de realizar commit: es necesario rehacer las operaciones

```
ALGORITMO:
1) Crear listas vacías ACTIVAS y CONFIRMADAS
2) Llenar listas con transacciones activas previas al checkpoint.
3) Añadir transacciones que comienzan después del checkpoint.
```

```
3.1) Buscar posibles confirmaciones de las transacciones
    -COMMIT: Mover a confirmados
    -ROLLBACK: Borrar de activas
    ¿BD DIFERIDA?
      No: deshacer operaciones de transacciones activas
      Sí: rehacer WRITEs de transacciones confirmadas y reiniciar transacciones
      activas.
```

3.2.1 Ejercicio 3: (08-02-2024)

```
***** DATOS *****
DIARIO Nº 2
Instante de fallo: 405
Diario: inmediato
Actualización de diario: Diferida
Bloque: 3
***** RESOLUCIÓN *****
Mirar último comando que ha llegado al diario:
[COMMIT, T1, 390]
1) ACT ={} CONF={}
2)Mirar transacciones previas al último checkpoint:
ACT={T6,T3,T4,T2,T7,T8,T1,T5}
Eliminar las confirmadas:
ACT={T6,T2,T8,T1,T5}
3) Mismas activas.
3.1) Buscar confirmaciones:
ACT={T2,T8}
CONF={T5,T6,T1}
4) Operaciones: actualización inmediata. BD diferida? no:
   DESHACER (orden inverso): {T2, T8}
    WRITE: F:84; X:77; W:30; D:171
   REHACER (mismo orden): {T5, T6, T1}
    WRITE: Q:67; A:6; R:14; B:190; S:99; C:105; N:112; O:187; P:195
   REINICIAR ACTIVAS:
    T2, T8
```

3.3 Estructura de InnoDB

Conceptos importantes (examen):

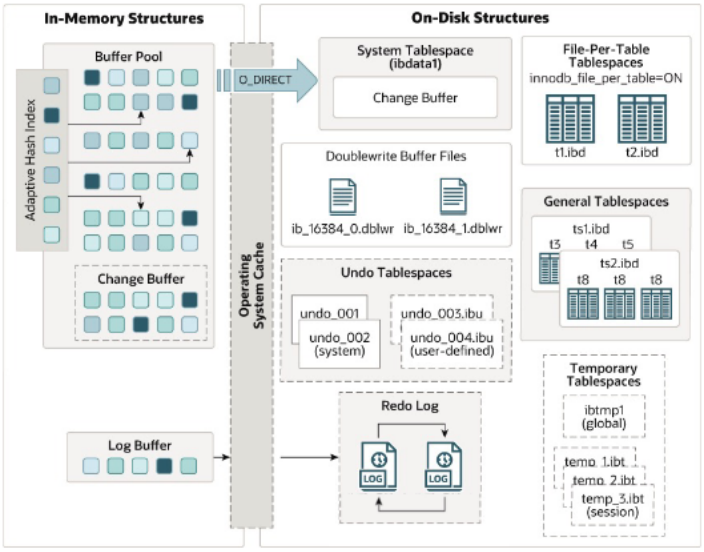
- Adaptive Hash Index: libreta de direcciones para el buffer pool.
- Buffer Pool: información en caché.
- Change Buffer: INSERTE, DELETE, UPDATE.
- Log Buffer: registro operaciones que se escribirán en el diario en la RAM.
- Redo Log: recibe el Log Buffer, diario de recuperación (más de un fichero).

- General Tablespaces: ficheros donde se guarda la informaciónd de las tablas persistentes de la DB.

Motor de almacenamiento de MySQL 8.0: InnoDB. Soporta propiedades ACID y control de concurrencia multi-usuario.
Mecanismos importantes:

- Flujo de operación de lectura: cliente - RAM y CPU. Consulta una tabla llamada **Buffer Pool**, que guarda las consultas más habituales en la RAM, de modo que no tiene que acceder al disco todas las veces que se haga esa petición. Consulta el **Change Buffer** por si ha habido algún cambio. Si no se encuetra allí, se accede al disco, **General Tablespaces**.
- Flujo de operación de escritura: cliente - RAM. Se registran los cambios en **Log Buffer**, y estos pasan al disco al **Redo Log** y estos pasarán a **General Tablespaces**.

Estructuras en memoria:



- *Buffer Pool*: caché de datos. Organizado con LRU. 128 MB. Se puede calcular su eficiencia con las variables de estado buffer.pool.reads y buffer.pool.read.requests. Para evitar problemas de rendimiento, al reiniciar MySQL se guarda un porcentaje del Buffer Pool en el disco, que cuando MySQL se inicia pasan a la RAM.
- *Log Buffer*: diario de la RAM que aún no se ha pasado a disco. Contiene los datos que deben escribirse en el Redo Log. 16 MB. La gestión de copia de datos se hace mediante frecuencia.

Los ajustes de estos parámetros pueden tener comportamientos no esperados en la DB. A mayor % de Buffer Pool a disco, mayor rendimento de consultas desde el comienzo, pero más tiempo de carga de MySQL.
En MySQL 8.0 hay una variables de sistema: innodb.dedicated.server. Sólo se debe activar en entornos donde mysql sea la única aplicación, puesto que gestiona la RAM automáticamente y puede afectar a otros procesos. Afecta a: buffer.pool.size, redo.log.capacity y a flush.method.

3.4 Examen:

- Ejercicios de cada tema (teoría)
- Ejercicios prácticos

3.5 Ejercicio 1 (12-02-2024)

a)

```
**********Datos**********
Fallo: 75
DB diferida
Diario inmediato
**********
0.1) último checkpoint: 0
1) iniciar listas:
Act={}
Conf={}
2) introducir las transacciones activas por delante del checkpoint y
si termina antes, quitarla.
3) todas las transacciones iniciadas después del CP y hasta el error:
ACT={T1, T2, T3}
Si terminan, al conf.
CONF={}
4) BD diferida? sí, no hay que deshacer.
Rehacer CONF : nada
Reiniciar: T1, T2, T3
```

b)

```
**********Datos**********
Fallo: 75
DB inmediata
Diario inmediata
**********
0.1) último checkpoint: 0
1) iniciar listas:
Act={}
Conf={}
2) introducir las transacciones activas por delante del checkpoint y
si termina antes, quitarla.
3) todas las transacciones iniciadas después del CP y hasta el error:
ACT={T1, T2, T3}
Si terminan, al conf.
CONF={}
4) BD diferida? no, hay que deshacer.
Deshacer:
```

```
 WRITE A3: 40
       B1: 45
       A2: 35
       A1: 30
Rehacer CONF : nada
Reiniciar: T1, T2, T3
```

c)

```
************Datos************
Fallo: 285
DB inmediata
Diario inmediato
*****************
0.1) último checkpoint: 230
1) iniciar listas:
Act={}
Conf={}
2) introducir las transacciones activas por delante del checkpoint y
si termina antes, quitarla.
ACT={T1, T2, T3, T4} -> ACT={T3, T4}
3) todas las transacciones iniciadas después del CP y hasta el error:
ACT={T4}
Si terminan, al conf.
CONF={T3}
4) BD diferida? no, hay que deshacer.
Deshacer:
 WRITE D4: 90
       C4: 120
       B4: 30
       A4: 45
Rehacer CONF :
 WRITE A3: 45
       B3: 25
       C3: 125
       D3: 95
Reiniciar: T4
```

d)

```
************Datos************
Fallo: 205
DB diferida
Diario diferido, bloque=9
*****************
0.1) última orden en diario: COMMIT, T3, 150
1) iniciar listas:
```

```
Act={}
Conf={}
2) introducir las transacciones activas por delante del checkpoint y
si termina antes, quitarla.
ACT={T6, T3, T4, T2, T7, T8}
3) todas las transacciones iniciadas después del CP y hasta el error:
ACT={T6, T4, T2, T7, T8}
Si terminan, al conf.
CONF={T3}
4) BD diferida? SÍ, no hay que deshacer.
Rehacer CONF :
 WRITE H: 36
       I: 96
       J: 19
Reiniciar: T6, T4, T2, T7, T8
```

4 Control de concurrencia (21-02-2024)

4.1 Introducción

Definicion 4.1. Concurrencia: un mismo procesador ejecuta varios procesos de forma simultánea.

Definicion 4.2. Paralelismo: varios procesadores ejecutan varios procesos de forma simultánea.

4.2 Plan de transacciones

Definicion 4.3. Serie intercalada de operaciones de distintas transacciones que se ejecutan de forma concurrente.

En un plan puede haber **conflictos**: situaciones que pueden derivar en problemas de concurrencia. Dos operaciones están en conflicto si:

- Las operaciones pertenecen a transacciones diferentes.
- Acceden a la misma variable/elemento X.
- Al menos una de las operaciones es write(X)

Una opción es utilizar planificaciones en serie: un plan donde las operaciones de cada transacción se ejecutan consecutivamente. Las operaciones de las transacciones no se intercalan y están libres de conflictos.

4.3 2PL

Definicion 4.4. Conservador/Estático: Cada transacción debe reservar todos los elementos a los que quiere acceder antes de comenzar a ejecutarse.

Definicion 4.5. Estricto: la transacción no libera ninguna reserva de escritura hasta finalizar.

Definicion 4.6. Riguroso: una transacción no libera ninguna reserva hasta que finaliza.

4.4 Marcas de tiempo de transacción

Esperar-Morir: Si t(Ti) > t(Tj), Ti espera. Si no, Ti aborta y reinicia.
Herir-Esperar: Si t(Ti) < t(Tj), Tj es abortada y se reinicia. Si no, Ti espera.

4.5 Ejercicios (04-03-2024)

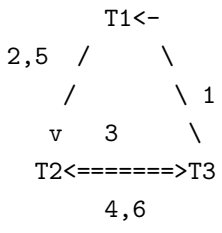
Ejercicios corregidos en clase
Ejercicios conflictos

```
[[1]]

R1(X), R2(Z), R1(Z), R3(X), R3(Y), W1(X), W3(Y), R2(Y), R1(W),
W2(Z), W1(W), R2(W), W2(Y)

Confictos:
1. <R1(Z), W2(Z)>
2. <R3(x), W1(X)>
3. <R3(Y), W2(Y)>
4. <W3(Y), R2(Y)>
5. <W3(Y), W2(Y)>
6. <W1(W), R2(W)>

Grafo:

         - T1 <-
  1,6   /        \
       /          \ 2
      v            \
     T2<------------T3
         3,4,5
No es cíclico, sí es serializable: T3, T1, T2

[[2]]
```

R1(X), R2(Z), R3(X), R1(Z), R2(Y), R3(Y), W1(X), R1(W),
W1(W), W2(Z), W3(Y), R2(W), W2(Y).

Conflictos:
1. <R3(X), W1(X)>
2. <R1(Z), W2(Z)>
3. <R2(Y), W3(Y)>
4. <R3(Y), W2(Y)>
5. <W1(W), R2(W)>
6. <W3(Y), W2(Y)>

Grafo:



Sí es cíclico, no serializable.

Ejercicios 2PL

[[3]] EJERCICIO 2 EXAMEN

TRANSACCIÓN A:

READ(A)
READ(B)
C = A + B
WRITE(C)
READ(E)
D = 7 + E
WRITE(D)
READ(D)
READ(A)
F = D - A
WRITE(F)

a)) Que no cumpla el protocolo 2PL:
RLOCK(A)
READ(A)
UNLOCK(A)
RLOCK(B)

```
READ(B)
UNLOCK(B)
C = A + B
WLOCK(C)
WRITE(C)
UNLOCK(C)
RLOCK(E)
READ(E)
UNLOCK(E)
D = 7 + E
WLOCK(D)
WRITE(D)
READ(D)
UNLOCK(D)
RLOCK(A)
READ(A)
UNLOCK(A)
F = D - A
WLOCK(F)
WRITE(F)
UNLOCK(F)
```

b)) 2PL NORMAL:

```
RLOCK(A)   -
READ(A)    |
RLOCK(B)   |
READ(B)    |
C = A + B  |
WLOCK(C)   |
WRITE(C)   |  FASE DE EXPANSIÓN
RLOCK(E)   |
READ(E)    |
D = 7 + E  |
WLOCK(D)   |
WRITE(D)   |
READ(D)    |
READ(A)    |
F = D - A  -
WLOCK(F)   -
UNLOCK(A)  |
UNLOCK(B)  |
UNLOCK(C)  | FASE DE CONTRACCIÓN
UNLOCK(E)  |
UNLOCK(D)  |
WRITE(F)   |
```

```
UNLOCK(F)   -
```

c)) 2PL RIGUROSO:
Realizar todas las operaciones antes de los desbloqueos.

d)) 2PL ESTÁTICO

```
RLOCK(A)
RLOCK(B)
WLOCK(C)
WLOCK(D)
RLOCK(E)
WLOCK(F)

READ(A)
READ(B)
UNLOCK(B)
C = A + B
WRITE(C)
UNLOCK(C)
READ(E)
UNLOCK(E)
D = 7 + E
WRITE(D)
READ(D)
UNLOCK(D)
READ(A)
UNLOCK(A)
F = D - A
WRITE(F)
UNLOCK(F)
```

Consejo: desbloquear siempre que podamos.

Ejercicios Esperar-Morir, Herir-Esperar

```
+++ ESPERAR-MORIR: +++
t(T3) = 1
T3 -> WLOCK(Z)
t(T1) = 3
T1 -> wLOCK(X)
t(T2) = 5
T2 -> RLOCK(Y)
t(T1) > t(T3), T1 ROLLBACK y libera WLOCK(X)
T3 -> RLOCK(Y)
UNLOCK(Z), continuar
T2 -> WLOCK(X)
```

```
UNLOCK(Y), continuar
--
--
t(T3) < t(T2), T3 espera
--
--
UNLOCK(X), T3 -> WLOCK(X)
UNLOCK(Y), continuar
UNLOCK(X), continuar

+++ HERIR-ESPERAR: +++
t(T3) = 1
T3 -> WLOCK(Z)
t(T1) = 3
T1 -> WLOCK(X)
t(T2) = 5
T2 -> RLOCK(Y)
t(T1) > t(T3), T1 espera
T3 -> RLOCK(Y)
UNLOCK(Z), T1 -> WLOCK(Z)
t(T2) > t(T1), T2 espera
UNLOCK(Y), continuamos
t(T1) < t(T2), T2 es abortado: ROLLBACK, UNLOCK(Y), T1 -> WLOCK(Y)
UNLOCK(X), continua
T3 -> WLOCK(X)
UNLOCK(Y), continua
UNLOCK(Z), continua
--
--
UNLOCK(X), continua
**** cuando se hace un unlock, si alguien espera, se le concede, si no, se continúa ****

+++ ESPERAR-MORIR: +++
t(T1) = 1
T1 -> WLOCK(X)
t(T2) = 3
T2 -> RLOCK(Y)
t(T3) = 5
T3 -> WLOCK(Z)
t(T1) < t(T3), T1 espera
T3 -> RLOCK(Y)
UNLOCK(Z), T1 -> WLOCK(Z)
t(T1) < t(T2, t3), T1 espera
t(T2) > t(T1), T2 ROLLBACK, UNLOCK(Y), continua
UNLOCK(Y), T1 -> WLOCK(Y)
```

```
UNLOCK(X), continua
T3 -> WLOCK(X)
UNLOCK(Y), continua
UNLOCK(Z), continua
--
--
UNLOCK(X), continua
```

4.6 Protocolos de marcas de tiempo (05-03-2024)

Definicion 4.7. Protocolos que permiten cumplir la propiedad Isolation de ACID sin usar reservas o locks.

Cada transacción tiene asociada una marca de tiempo, asignada al comienzo de su ejecución. Cada elemento X tendrá 2 marcas de tiempo: TSREAD(X) y TSWRITE(X) (última lectura y última escritura).
Nos permite hacer una planificación de serie equivalente según las marcas de tiempo.
Situación: T realiza WRITE(X). Si una transacción más joven ha leído o escrito X antes que T, se aborta.

```
IF (TSREAD(X) > t(T)) OR (TSWRITE(X) > t(T))
   T se aborta
ELSE
   WRITE(X)
   TSWRITE(X) = t(T)
```

Situación: T realiza READ(X). Si una transacción más joven ha escrito X antes de que T pueda leerlo, se aborta.

```
IF (TSWRITE(X) > t(T))
   T se aborta
ELSE
   READ(X)
   TSREAD(X) = max(TSREAD(X), t(T))
```

Estos son **Protocolos optimistas**. Asumen que va a haber pocos problemas de concurrencia, tienen mejor rendimiento, durante la ejecución de la transacción no realizan comprobaciones ni modificaciones. Al finalizar comprueban y abortan o commitean según su éxito.

4.7 Ejercicios (11-03-2024)

Ejercicios corregidos en clase
Ejercicios protocolos optimistas

```
[[1]]
t(T1) = 1
TSWRITE(X) < t(T1): READ(X) & TSREAD(X) = 1
```

```
t(T2) = 3
TSWRITE(Y) < t(T1): READ(Y) & TSREAD(Y) = 1
TSWRITE(Z) < t(T2): READ(Z) & TSREAD(Z) = 3
TSWRITE(Y) < t(T2): READ(Y) & TSREAD(Y) = 3
--
--
TSREAD(Z) = t(T2), TSWRITE(Z) < t(T2): WRITE(Z) & TSWRITE(Z) = 3
TSWRITE(X) < t(T2): READ(X) & TSREAD(X) = 3
TSWRITE(X) < t(T1), TSREAD(X) > t(T1): T1 ROLLBACK
--
--
--
TSWRITE(X) < t(T2), TSREAD(X) = t(T2): WRITE(X) & TSWRITE(X) = 3
--

[[2]]
t(T1) = 1
TSWRITE(Z) < t(T1): READ(Z) & TSREAD(Z) = 1
--
t(T2) = 4
TSWRITE(X) < t(T2): READ(X) & TSREAD(X) = 4
--
TSWRITE(Z) < t(T1), TSREAD(Z) = t(T1): WRITE(Z) & TSWRITE(Z) = 1
TSWRITE(W) < t(T2), TSREAD(W) < t(T2): WRITE(W) & TSWRITE(W) = 4
TSWRITE(Y) < t(T2): READ(Y) & TSREAD(Y) = 4
TSWRITE(X) < t(T1): READ(X) & TSREAD(X) = 4
TSWRITE(Z) < t(T2): READ(Z) & TSREAD(Z) = 4
TSWRITE(Y) < t(T1): READ(Y) & TSREAD(Y) = 4
--
--
TSWRITE(X) < t(T2), TSREAD(X) = t(T2): WRITE(X) & TSWRITE(X) = 4
TSWRITE(Y) < t(T1), TSREAD(Y) > t(T1): T1 ROLLBACK

[[3]]
t(T2) = 1
TSWRITE(X) < t(T2): READ(X) & TSREAD(X) = 1
t(T1) = 3
TSWRITE(Z) < t(T1): READ(Z) & TSREAD(Z) = 3
TSWRITE(Z) < t(T2): READ(Z) & TSREAD(Z) = 3
--
t(T3) = 7
TSWRITE(Y) < t(T3): READ(Y) & TSREAD(Y) = 7
TSWRITE(X) < t(T2), TSREAD(X) = t(T2): WRITE(X) & TSWRITE(X) = 1
TSWRITE(X) < t(T3): READ(X) & TSREAD(X) = 7
TSWRITE(Y) < t(T1): READ(Y) & TSREAD(Y) = 7
--
```

```
TSWRITE(Y) < t(T3), TSREAD(Y) = t(T3): WRITE(Y) & TSWRITE(Y) = 7
TSWRITE(W) < t(T2): READ(W) & TSREAD(W) = 1
--
TSWRITE(Z) < t(T1), TSREAD(Z) = t(T1): WRITE(Z) & TSWRITE(Z) = 3
--
TSWRITE(W) < t(T1): READ(W) & TSREAD(W) = 3
--
TSWRITE(Y) > t(T1), TSREAD(Y) > t(T1): T1 ROLLBACK
TSWRITE(W) < t(T2), TSREAD(W) > t(T2): T2 ROLLBACK
```

4.8 Repaso segundo parcial (ejercicios)

```
[[1]] Conflictos
R2(Y); R3(Y); R1(X); R2(W); W1(X); W3(X); W2(Y); R1(W); R3(Z);
W1(W); W3(W); W2(Z);
Lista de conflictos:
1. <R3(Y), W2(Y)>
2. <R1(X), W3(X)>
3. <R2(W), W1(W)>
4. <R2(W), W3(W)>
5. <W1(X), W3(X)>
6. <R1(W), W3(W)>
7. <R3(Z), W2(Z)>
8. <W1(W), W3(W)>

Grafo:
     ---->T1----
     |         | 2,5,6,8
     |    4    v
    T2<=======>T3
          1,7
No es serializable.

[[2]] 2PL
READ(A)
READ(B)
X = A + B
WRITE(X)
READ(A)
W = A - 5
WRITE(W)
READ(Y)
Y = Y * 7
WRITE(Y)

A) 2PL NORMAL: bloquear cuando necesitemos, desbloquear cuando podamos.
```

```
RLOCK(A)    _
READ(A)      |
RLOCK(B)     |
READ(B)      |
X = A + B    |
WLOCK(X)     | FASE DE EXPANSIÓN
WRITE(X)     |
READ(A)      |
W = A - 5    |
WLOCK(W)     |
WRITE(W)     |
WLOCK(Y)    _|
UNLOCK(A)   _
UNLOCK(B)    |
UNLOCK(X)    |
UNLOCK(W)    |
READ(Y)      | FASE DE CONTRACCIÓN
Y = Y * 7    |
WRITE(Y)     |
UNLOCK(Y)   _|
```

B) 2PL ESTRICTO: como un 2PL normal y 1.desbloqueo R 2.desbloqueo W

```
RLOCK(A)
READ(A)
RLOCK(B)
READ(B)
X = A + B
WLOCK(X)
WRITE(X)
READ(A)
W = W - 5
WLOCK(W)
WRITE(W)
WLOCK(Y)
UNLOCK(A)
UNLOCK(B)
READ(Y)
Y = Y * 7
WRITE(Y)
UNLOCK(X)
UNLOCK(W)
UNLOCK(Y)
```

[[3]] Herir-Esperar

****

IF (t(Ti) < t(Tj)): herir (Tj ROLLBACK) ELSE: esperar

```
**************************************************
1. t(T3) = 1
2. T3 -> RLOCK(W)
3. t(T1) = 3
4. t(T2) = 4
5. T3 -> WLOCK(Z)
6. T1 -> WLOCK(X)
7. T2 -> RLOCK(Y)
8. t(T2) > t(T1): T2 ESPERA
9. UNLOCK(X): T2 -> WLOCK(X)
10. T2 -> WLOCK(Y)
11. UNLOCK(W): CONTINUA
12. t(T1) < t(T2): T2 ROLLBACK; UNLOCK(X, Y); T1-> RLOCK(Y)
13. T3 -> WLOCK(X)
14. --
15. UNLOCK(Z): CONTINUA
16. T1 -> WLOCK(W)
17. --
18. UNLOCK(X): CONTINUA
19. UNLOCK(W): CONTINUA
20. --
21. UNLOCK(Y): CONTINUA
22. --
```

4.9 Ejercicios repaso (12-03-2024)

```
[[1]] CONFLICTOS
R1(X), R3(Z), R2(W), W1(X), R3(Y), R1(W), W2(W), R1(Y), R2(Z),
W3(Z), R2(X), W1(Y), W2(Z)

1.<R3(Z), W2(Z)>
2.<W1(X), R2(X)>
3.<R3(Y), W1(Y)>
4.<R1(W), W2(W)>
5.<R2(Z), W3(Z)>

      ----T1<----
   2,4|         |3
      v   5     |
     T2<=======>T3
           1

No es serializable

[[2]] 2PL
```

```
READ(X)
X = X + 5
WRITE(X)
READ(Y)
READ(X)
Y = Y * X
WRITE(Y)
READ(W)
Z = W - 7
WRITE(Z)

A) 2PL NORMAL: fases diferenciadas
WLOCK(X)
READ(X)
X = X + 5
WRITE(X)
WLOCK(Y)
READ(Y)
READ(X)
Y = Y * X
WRITE(Y)
RLOCK(W)
READ(W)
Z = W - 7
WLOCK(Z)
UNLOCK(X)
UNLOCK(Y)
UNLOCK(W)
WRITE(Z)*
UNLOCK(Z)

B) 2PL CONSERVADOR/ESTÁTICO: todos los bloqueos al principio
WLOCK(X)
WLOCK(Y)
RLOCK(W)
WLOCK(Z)
READ(X)
X = X + 5
WRITE(X)
READ(Y)
READ(X)
UNLOCK(X)
Y = Y * X
WRITE(Y)
UNLOCK(Y)
```

```
READ(W)
UNLOCK(W)
Z = W - 7
WRITE(Z)
UNLOCK(Z)

[[3]] Herir-Esperar
*****************************************************
IF (t(Ti) < t(Tj)): herir (Tj ROLLBACK) ELSE: esperar
*****************************************************
1. t(T2) = 1
2. T2 -> WLOCK(X)
3. t(T3) = 3
4. T3 -> WLOCK(Y)
5. t(T1) = 5
6. T1 -> RLOCK(Z)
7. t(T3) > t(T2): T3 esperar
8. T2 -> RLOCK(Z)
9. UNLOCK(X); T3 -> WLOCK(X)
10. t(T1) > t(T3): T1 esperar
11. UNLOCK(Y): T1 -> RLOCK(Y)
12. t(T1) > t(T3): T1 espera
13. UNLOCK(X); T1 -> WLOCK(X)
14. UNLOCK(X): CONTINUA
15. t(T2) < t(T1): T1 ROLLBACK; UNLOCK(Y); UNLOCK(Z); T2-> WLOCK(Y)
16. t(T3) > t(T2): T3 esperar
17. --
18. UNLOCK(Z); T3 -> WLOCK(Z)
19. --
20. UNLOCK(Z): CONTINUA
21. UNLOCK(Y): CONTINUA

[[4]] - timestamps
*******************************************************
READ: IF (TSWRITE(X) > t(T)): T ROLLBACK
      ELSE: READ(X) & TSREAD(X) = max{t(T), TSREAD(X)}
WRITE: IF (TSWRITE(X), TSREAD(X) > t(T)): T ROLLBACK
       ELSE: WRITE(X) & TSWRITE(X) = t(T)
*******************************************************

1. t(T3) = 1
2. TSWRITE(X) < t(T3): T3->READ(X) & TSREAD(X) = 1
3. t(T1) = 3
4. TSWRITE(Z) < t(T1): T1->READ(Z) & TSREAD(Z) = 3
5. TSWRITE(Z) < t(T3): T3->READ(Z) & TSREAD(Z) = 3
6. t(T2) = 6
```

7. TSWRITE(Y) < t(T2): T2->READ(Y) & TSREAD(Y) = 6
8. --
9. TSWRITE(W) < t(T1): T1->READ(W) & TSREAD(W) = 3
10. TSWRITE(X) < t(T2): T2->READ(X) & TSREAD(X) = 6
11. --
12. --
13. TSWRITE(Z)<t(T1), TSREAD(Z)=t(T1):WRITE(Z) & TSWRITE(Z)=3
14. TSWRITE(X)<t(T3), TSREAD(X)>t(T3): T3 ROLLBACK
15. --
16. TSWRITE(Y)<t(T2), TSREAD(Y)=t(T2): T2->WRITE(Y) & TSWRITE(Y)=6
17. TSWRITE(W)<t(T1), TSREAD(W)=t(T1): T1->WRITE(W) & TSWRITE(W)=3
18. TSWRITE(Z)<t(T2): T2->READ(Z) & TSREAD(Z)=6
19. --
20. --
21. --
22. TSWRITE(Y)>t(T1): T1 ROLLBACK
23. --
24. --
25. TSWRITE(Z)<t(T2), TSREAD(Z)<t(T2): WRITE(Z) & TSWRITE(Z) = 6
26. --