Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2413 - Bases de Datos
Primer Semestre 2025

# Ayudantía Repaso I2

Ayudantes: Olivia Llanos (olivia.llanos@uc.cl) - Paula Verastegui (paula.verastegui@uc.cl)

---

## Problema 1: Manejo de Transacciones y 2PL

**a)** Considere el Schedule del cuadro 1. Indique para cada valor de X (puede ser R o W) si el *schedule* es *conflict serializable* o no. Explique por qué.

| T1 | T2 | T3 | T4 |
|---|---|---|---|
| R(a) | | | |
| | W(a) | | |
| | R(b) | | |
| | | W(b) | |
| | | W(c) | |
| | | | W(c) |
| | | | R(d) |
| X(d) | | | |

Cuadro 1: Schedule problema a).

**Solución:** En el caso de $X = R$ estamos bien. Si $X = W$ hay un ciclo en el grafo de precedencia, por lo que el *schedule* no es *conflict serializable*:

$$T1 - T2 - T3 - T4 - T1$$

**b)** Considere el Schedule del cuadro 2. Diga como lo reordenaría/ejecutaría Strict-2PL. ¿Cambiaría el orden de algunas operaciones/transacciones? Para esto considera como se asignan shared/exclusive locks. ¿Puedes ver cómo cambiar algunas operaciones para generar un deadlock en 2PL?

| T1 | T2 | T3 |
|---|---|---|
| R(a) | | |
| | R(b) | |
| | | W(a) |
| | | W(c) |
| R(c) | | |
| | R(c) | |

Cuadro 2: Schedule problema b).

**Solución:** Orden sería siguiente (con locks):

- T1 – shared lock (A)
- T2 – shared lock (B)
- T3 – pide exclusive (A), sistema dice espera que termine T1
- T3 – está congelada, operación W(c) está igual en espera
- T1 – shared lock (C) – al ejecutar se liberan locks de T1
- T2 – shared lock (C) – al ejecutar se liberan los locks de T2
- Ahora va T3

**Otra solución válida:** Orden sería siguiente (con locks):

- T1 – shared lock (A)
- T2 – shared lock (B)
- T3 – pide exclusive (A), sistema dice espera que termine T1
- T3 – está congelada, operación W(c) está igual en espera
- T1 – shared lock (C) – al ejecutar se liberan locks de T1
- T3 - exclusive lock (A)
- T3 - exclusive lock (C) - al ejecutar se liberan todos los locks de T3
- T2 – shared lock (C) – al ejecutar se liberan los locks de T2

# Problema 2: Logging

## Undo Logging

Suponga que su sistema tuvo una falla. Al reiniciar el sistema, el sistema se encuentra con el log file que se muestra a continuación, en la tabla "Log Undo". Suponiendo que la política de recovery es la de Undo Logging, indique:

- Hasta qué parte del *log* debo leer.
- Qué variables deben deshacer sus cambios y cuál es el valor con el que quedarán.
- Qué variables (de las que aparecen en el *log*) no son cambiadas en el proceso.

| Log Undo |
|---|
| <START T1> |
| <START T2> |
| <T1, a, 4> |
| <T2, b, 5> |
| <T2, c, 10> |
| <COMMIT T1> |
| <START CKPT (T2)> |
| <START T3> |
| <START T4> |
| <T3, a, 10> |
| <T2, b, 7> |
| <T4, d, 5> |
| <COMMIT T2> |
| <END CKPT> |
| <START T5> |
| <COMMIT T3> |
| <T5, e, -3> |

Cuadro 3: Transacciones Undo Logging

**Solución:**

- Hasta qué parte del *log* debo leer.
  - Dado que existe un `END CKPT`, se debe leer hasta el `START CKPT`.
- Qué variables deben deshacer sus cambios y cuál es el valor con el que quedarán.
  - T5 no está marcada con commit, por lo que por la línea `<T5, e, -3>` indica que debemos hacer que e vuelva a ser -3. Por la misma razón, la línea `<T4, d, 5>` nos indica que **d** debe volver a ser 5.
- Qué variables (de las que aparecen en el *log*) no son cambiadas en el proceso.
  - No se debe hacer *undo* de ninguna otra transacción, por lo que las variables **a**, **b** y **c** no son tocadas.

## Redo Logging

Suponga que su sistema tuvo una falla. Al reiniciar el sistema, el sistema se encuentra con el *log file* que se muestra a continuación, en la tabla "Log Redo". Suponiendo que la política de *recovery* es la de *Redo Logging*, indique:

- Desde qué parte del *log* debo comenzar el proceso de *redo*.
- Qué variables deben rehacer sus cambios y cuál es el valor con el que quedarán.
- Qué variables (de las que aparecen en el *log*) no son cambiadas en el proceso.

- Si no hubiésemos encontrado la línea `<END CKPT>`, ¿desde qué parte del *log* debería comenzar el proceso de *redo*?.

| Log Redo |
|---|
| <START T1> |
| <T1, a, 1> |
| <COMMIT T1> |
| <START T2> |
| <T2, b, 2> |
| <T2, c, 3> |
| <COMMIT T2> |
| <START T3> <T3, a, 10> |
| <START CKPT (T3)> |
| <END T1> |
| <END T2> |
| <T3, d, 23> |
| <START T4> |
| <END CKPT> |
| <COMMIT T3> |
| <T4, e, 11> |

Cuadro 4: Transacciones Redo Logging

**Solución:**

- Desde qué parte del *log* debo comenzar el proceso de *redo*.
  - Dado que existe un `END CKPT`, debo leer hasta el `<START T3>`, que es la transacción más antigua que se señala en el `START CKPT`.
- Qué variables deben rehacer sus cambios y cuál es el valor con el que quedarán.
  - Tenemos la certeza de que T1 y T2 están guardadas en disco, porque tenemos la presencia de un `END CKPT`. Fuera de esas transacciones, debemos rehacer todas las otras transacciones que están marcadas con `COMMIT`. En este caso es sólo T3. Por lo que `<T3, a, 10>` nos indica que debemos rehacer **a** al valor 10 y `<T3, d, 23>` nos indica que debemos rehacer **d** a 23.
- Qué variables (de las que aparecen en el *log*) no son cambiadas en el proceso.
  - Continuando con el proceso anterior, T4 no está marcado con commit, por lo que **e** no se debe tocar. T4 debe ser marcada con `ABORT` en el proceso. **b** y **c** tampoco son cambiadas en el proceso. Lo cambios efectuados a la variable a debido a T1 tampoco deben ser realizados de nuevo.
- Si no hubiésemos encontrado la línea `<END CKPT>`, ¿desde qué parte del *log* debería comenzar el proceso de *redo*?.

- Debemos encontrar sí o sí un `<END CKPT>`. Así que como no hay otro se debe leer el *log* entero.