Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
Segundo Semestre de 2018

IIC 2413 – Bases de Datos
Interrogación 3 Rúbrica

Pregunta 1: External Merge Sort y Transacciones

a) [2 ptos] Considere una relación de 3.000.000 páginas. Indique el número de fases y de I/O del External Merge Sort optimizado visto en clases cuando:

- En Buffer caben 11 páginas.

$$2 \cdot 3{,}000{,}000 \cdot \lceil 1 + log_{10}(\lceil 3000000/11 \rceil) \rceil = 4,2 \cdot 10^7 \quad (1)$$

- En Buffer caben 101 páginas.

$$2 \cdot 3{,}000{,}000 \cdot \lceil 1 + log_{100}(\lceil 3000000/101 \rceil) \rceil = 2,4 \cdot 10^7 \quad (2)$$

A medida que el buffer sigue creciendo el I/O decrece a $4 \cdot 3{,}000{,}000 = 1,2 \cdot 10^7$. para esto necesito aproximadamente $\sqrt{3{,}000{,}000} = 1733$ páginas en buffer. Si no escribo el último resultado es $3 \cdot 3{,}000{,}000 = 9 \cdot 10^7$.

b) [4 pts] Considerando el *schedule* del cuadro 1, tenemos que hay un ciclo entre T1 y T3, por lo tanto el *schedule* no es conflict serializable. En el caso de usar Strict-2PL tenemos dos casos:

- **Caso habitual**: Cuando llega T3, pide un *exclusive lock* sobre a y no se le otorga porque T1 tenía un *shared lock*. La transacción se congela y todo termina bien.
- Cuando llega T3 pasa lo mismo que lo anterior, pero si `W(a)` y `W(c)` no comparten recursos podría pasar que T3 siga con `W(c)`. En ese caso se toma un *exclusive lock* y luego cuando T1 pida un *shared lock* de C este no será otorgado. Así tendríamos un deadlock.

También, si T3 hubiese venido de la forma `W(c)` y después `W(a)`, sí o sí hubiesemos tenido un *deadlock*. Strict-2PL a veces puede generar *deadlocks* y esto es un problema para las bases de datos (que se resuelve mediante algoritmos que detectan y rompen los *deadlocks*). Cualquier respuesta entre las 3 es considerada correcta es correcta.

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

Cuadro 1: schedule problema 2.

Pregunta 2: Logging

Undo Logging

Considerando el *schedule*:

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>

- 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 `<T5, 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

Considerando el *schedule*:

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)>
<T3, d, 23>
<START T4>
<END CKPT>
<COMMIT T3>
<T4, e, 11>

- 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 d debe 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 hubiesemos 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.

Pregunta 3: MongoDB

Una respuesta a los procedimientos es:

- **[1.5 pts]** Entregue el número de "Me gusta" que suman todos los fotogramas que en su descripción tengan el texto "#Trekking" y que no tengan el texto "Cerro La Cruz".

```
var cursor = db.fi3.find({$text: {$search: "\"#Trekking\" -\"\Cerro La Cruz\""}});
var suma_likes = 0;
cursor.forEach(
    (fotograma) => {
        var fid = fotograma.fid;
        var cursor2 = db.ui3.find({"me_gusta": fid});
```

```
            cursor2.forEach(
                (usuarios_like) => {
                    suma_likes += 1;
                }
            )
        }
    )
    print(suma_likes);
```

- **[2 pts]** Imprima el identificador de cada usuario seguido de las URL de todos sus fotogramas.

```
    var cursor = db.ui3.find({});
    cursor.forEach(
        (usuario) => {
            var uid = usuario.uid;
            print("ID de Usuario: " + uid);
            var cursor2 = db.fi3.find({"posteada_por": uid});
            cursor2.forEach(
                (fotograma) => {
                    print(fotograma.fotograma);
                }
            )
        }
    )
```

- **[2.5 pts]** Imprima el identificador de cada usuario junto al número total de "Me gusta" que ha recibido. Esto es, sumar los "Me gusta" de cada uno de sus fotogramas.

```
var cursor = db.fi3.find({});
var users_likes = {};
cursor.forEach(
    (fotograma) => {
        var fid = fotograma.fid;
        var uid = fotograma["posteada_por"];
        var cursor2 = db.ui3.find({"me_gusta": fid});
        if (!users_likes[uid]) {
            users_likes[uid] = 0;
        }
        cursor2.forEach(
            (usuarios_like) => {
                users_likes[uid] += 1;
            }
        )
    }
)
printjson(users_likes);
```

Bonus [0.6 pts] Mencione 6 cerros que pertenecen a la Sierra de San Ramón.

- Cerro Provincia
- Cerro La Cruz

- Cerro San Ramón
- Cerro Punta Damas
- Cerro Morro del Tambor
- Cerro Minilla