Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC3413 — Implementación de Sistemas de Base de Datos — 1' 2024

LABORATORIO 4

Laboratorio:	Recovery manager.
Publicación:	Lunes 10 de junio.
Ayudantía:	Viernes 14 de junio.
Entrega:	**Viernes 21 de junio hasta las 23:59 horas**.

Descripción del laboratorio

El objetivo de este laboratorio es implementar un mecanismo de recovery después de un crash. Para esto deben crear un programa `recovery` que puede consumir el log del sistema y reconstruir el `HeapFile` de acuerdo con la política de logging especificada por el sistema.

Formato del log

Log records. IIC3413DB usa un sistema de logging físico, quiere decir que logea cambios al nivel de una página completa, guardando los bytes cambiados, nuevos, o ambos. Para entender cómo se verá un log concreto en IIC3413DB, consideremos un update log de la política undo/redo de recovery. Abajo presentamos la estructura de un log de esta forma, explicando cada uno de los elementos que se guardarán en este log.

- `<WRITE-UR,TxId,TableId,pageno,offset,len,oldBytes,newBytes>`
  - `WRITE-UR` significa que estamos trabajando con un update undo-redo.
  - `TxId` representa el identificador de la transacción.
  - `TableId` representa el identificador de la transacción, de nuevo cómo un número entero. Este coincide con el identificador de la tabla en el catalogo del IIC3413DB.
  - `pageno` es el número de la página en el `HeapFile` cual almacena la tabla `TableId`.
  - `offset` es la posición en la página `pageno` del `HeapFile` de la tabla `TableId`.
  - `len` es el largo de bytes que cambiaron partiendo desde `offset`.
  - `oldBytes` es la secuencia de `len` bytes con los datos que estaban en estas posiciones de la página antes de realizar el update.
  - `newBytes` es la secuencia de `len` bytes con los datos que se deben guardar en estas posiciones de la página al realizar el update.
  - El largo de `oldBytes` es igual al largo de `newBytes`.

Los log records son escritos en el archivo de log cómo un *byte stream*. Quiere decir que el log será un archivo de raw bytes, y poder entender que hay en el log, debemos parsear los bytes del log y imprimir una representación lógica similar a cómo describimos nuestro undo/redo update arriba. Para poder hacer esto, primero explicaremos cómo se serializa cada uno de los log records usados por IIC3413DB. (Serializar significa cómo será escrito en formato bytes para que sepan que es lo que van a leer desde el log.)

- `<START,TxId>` – comienzo de la transacción `TxId`
  - 1 byte con número 0 (`START`)
  - 4 bytes con un número entero representando el `TxId`.
- `<COMMIT,TxId>` – commit de la transacción `TxId`
  - 1 byte con número 1 (`COMMIT`)
  - 4 bytes con un número entero representando el `TxId`.
- `<ABORT,TxId>` – abort de la transacción `TxId`
  - 1 byte con número 2 (`ABORT`)
  - 4 bytes con un número entero representando el `TxId`.
- `<END,TxId>` – finalización de la transacción `TxId`; se usa solo para las transacción que hicieron el `COMMIT` y cuyos datos se escribieron a disco con éxito. Comando usado solo para undo/redo logging.
  - 1 byte con número 3 (`END`)
  - 4 bytes con un número entero representando el `TxId`.
- `<WRITE-UR,TxId,TableId,pageno,offset,len,oldBytes,newBytes>` – update undo/redo
  - 1 byte con número 4 (`WRITE-UR`)
  - 4 bytes con un número entero representando el `TxId`.
  - 4 bytes con un número entero representando el `TableId`.
  - 4 bytes con un número entero representando el `pageno`.
  - 4 bytes con un número entero representando el `offset` en la página `pageno`.
  - 4 bytes con un número entero representando el `len` – largo de los bytes que cambiaron.
  - `len` bytes representando el contenido anterior escrito en la página `pageno` en posiciones entre `offset` y `offset+len-1`.
  - `len` bytes representando el contenido que fue escrito en la página `pageno` después de ejecutar el update en posiciones entre `offset` y `offset+len-1`.
- `<WRITE-U,TxId,TableId,pageno,offset,len,oldBytes>` – updates undo
  - 1 byte con número 5 (`WRITE-U`)
  - 4 bytes con un número entero representando el `TxId`.
  - 4 bytes con un número entero representando el `TableId`.
  - 4 bytes con un número entero representando el `pageno`.
  - 4 bytes con un número entero representando el `offset` en la página `pageno`.
  - 4 bytes con un número entero representando el `len` – largo de los bytes que cambiaron.
  - `len` bytes representando el contenido anterior escrito en la página `pageno` en posiciones entre `offset` y `offset+len-1`.
- `<START CHKP,n,TxId`$_1$`,...,TxId`$_n$`>`
  - 1 byte con número 6 (`START CHKP`)
  - 4 bytes con representando el número entero `n`
  - $4n$ bytes dónde los bytes $[4i, 4i+1, 4i+2, 4i+3]$ representan `TxId`$_i$, para $i = 0, \ldots, n-1$.

- `<END CHKP>`
  - 1 byte con número 7 (`END CHKP`)

La especificación supone representación little-endian de números! (Efectivamente, esto significa que si leen 4 bytes representando a un int, todo pasa piola porqué su computador debe usar este formato para representar las cosas.)

¿Cómo acceder al log? Para poder visualizar los comandos escritos en nuestro log en el archivo `print_log.cc` implementamos el método:

```
void print_log( )
```

cual permite procesar los bytes del un archivo del log y imprimir log records de manera descrita arriba. Aquí pueden asumir que los bytes del log son escritos de manera correcta y se leen desde el primero hasta el último. Por ejemplo, si el log solo contiene 5 bytes, y el primer byte contienen el número 0 (en little-endian), sabemos que hay que leer los próximos 4 bytes y recuperar `TxId`, escrito de nuevo, en formato little-endian, e imprimir texto `<START,TxId>`, seguido con un salto de linea, en el archivo especificado en `outFile`. Cuando se imprimen los `oldBytes` o `newBytes` estos vienen en el formato hex para poder ser visualizados. *Cuando se restituyen algunos datos en el recovery, allá el formato debe mantenerse cómo binario!*

Para ver cómo trabajar con un archivo de bytes (i.e. un archivo binario) en C++ revisen el siguiente link:

https://cplusplus.com/reference/istream/istream/read/

Cuando leen cosas desde un archivo binario, estas se guardan en un buffer local. Revisando los logs de arriba, pueden deducir cual debería ser el tamaño de este buffer, considerando que en un update log podemos pisar la página entera. Todas estas funcionalidades ya vienen implementadas en el archivo `print_log.cc` cual, al compilarse genera un ejecutable, dónde pueden pasar el archivo de log por linea de comando.

Este programa les permitirá realizar las siguientes tareas.

Problema 1: Undo recovery sin checkpoints [2 puntos] En este problema deberían implementar el método:

```
void simple_undo()
```

en el archivo `recovery_simple_undo.cc`. Este programa generará un ejecutable cual, al recibir el directorio dónde está guardada la bases de datos, y el archivo de su log, puede procesar sus comandos, y realizar un *undo recovery*, asumiendo que no hay checkpoints en el log. Esto significa que deben procesar el log entero, y correr el algoritmo de undo recovery explicado en las clases. Los únicos comandos que ocuren en este log serán con el código 0, 1, 2 y 5 de arriba. Cuando necesitan realizar un undo de un `WRITE-U` log, para acceder a la página y archivo del `HeapFile` para la tabla `TableId`, deben usar el siguiente método del catálogo:

```
FileId get_file_id(uint_32 TableId)
```

Usando el `FileId` y el `pageno` pueden conseguir la página a través del `BufferManager`. *Recuerden marcar la página cómo dirty si la van a modificar!*

El archivo `recovery_simple_undo.cc` ya viene con una implementación de su tarea hecha hasta la mitad, y ustedes deben completar el método para finalizar lo requerido.

Problema 2: Undo recovery con checkpoints [2 puntos] En este problema deberían implementar el método:

```
void real_undo()
```

Este método debe acceder al log, procesar sus comandos, y realizar un *undo recovery*, asumiendo la política de non-quiescent checkpoints explicada en las clases. Los comandos que ocuren en este log serán con el código 0, 1, 2, 5, 6 y 7 de arriba. Al finalizar, deben truncar el log hasta el último `START CHKP` cual sí tiene un `END CHKP`. Cuando eliminan records del log, deben eliminar solamente los log records *antes* (no inclusive) del `START CHKP` cual corresponde. Noten que puede ocurrir que el último `START CHKP` no tienen un `END CHKP`, pero hay más `START CHKP` válidos en el log anteriormente. Deben considerar todas estas situaciones.

La implementación debe ser realizada en el archivo `recovery_real_undo.cc`.

Problema 3: Undo-redo recovery sin checkpoints [2 puntos] En este problema deberían implementar el método:

```
void recoveryUR()
```

Este método debe acceder al log, procesar sus comandos, y realizar un *undo/redo recovery*, asumiendo que no hay checkpoints en el log. La política de recovery fue explicada en clases, y aplicará la optimización que considera el log del tipo `<END,TxId>`, el cual nos dice que `TxId` hizo su `COMMIT`, y todos los cambios que hizo esta transacción ya están escritos en el disco, y, por lo tanto, no es necesario hacer un redo de esta transacción.

La implementación debe ser realizada en el archivo `recovery_undo_redo.cc`.

Código

El código base está disponible en [1]. Asegúrense que están usando la última versión del sistema! Los tests se publicarán en el mismo repositorio.

Evaluación y entrega

El día límite para la entrega de esta tarea será el viernes 21 de junio a las 23:59 horas. Para ello se utilizará el formulario de canvas. *Su entrega debe contener todo el código de la carpeta `src` y nada más, y debe compilar cuando se les agregan otras carpetas en el repositorio. El código de la carpeta `src` debe ser entregado cómo un archivo `.zip`.* Por último, la evaluación será en base a varios test que su solución debe responder.

Ayudantía y preguntas

El día viernes 14 de junio se realizará una ayudantía donde se darán más detalles sobre IIC3413DB y el laboratorio. Para preguntas se pide usar el foro del curso.

Referencias

[1] Carlos Rojas, Diego Bustamante, Vicente Calisto, Tristan Heuer. IIC3413DB. https://github.com/DiegoEmilio01/IIC3413.